Advanced Supervised Learning in Multi-layer Perceptrons - From Backpropagation to Adaptive Learning Algorithms

Martin Riedmiller
Institut für Logik, Komplexität und Deduktionssyteme
University of Karlsruhe
W-76128 Karlsruhe
FRG
riedml@ira.uka.de

***Abstract*— Since the presentation of the backpropagation algorithm [1] a vast variety of improvements of the technique for training the weights in a feed-forward neural network have been proposed. The following article introduces the concept of supervised learning in multi-layer perceptrons based on the technique of gradient descent. Some problems and drawbacks of the original backpropagation learning procedure are discussed, eventually leading to the development of more sophisticated techniques.**

This article concentrates on adaptive learning strategies. Some of the most popular learning algorithms are described and discussed according to their classification in terms of global and local adaptation strategies.

The behavior of several learning procedures on some popular benchmark problems is reported, thereby illuminating convergence, robustness, and scaling properties of the respective algorithms.

I. Introduction

At present, supervised learning is probably the most frequently used technique in the field of neural networks. A teacher provides training examples of an arbitrary mapping which the network is to learn. Learning in this context means incremental adaptation of connection weights that transport information between simple processing units.

In fact, this sort of learning can be expressed as a minimization problem over a many dimensional parameter space, namely the vector space spanned by the weights.

A typical technique to perform this kind of optimization is gradient descent. The learning rule of the most popular supervised learning procedure, the backpropagation algorithm [1], follows the principle of gradient descent. Section II outlines supervised learning in multi-layer perceptrons, and describes the backpropagation algorithm.

After a short discussion of possible problems and pitfalls of the basic algorithm, a selection of more elaborate learning techniques is presented. Section III is dedicated to the introduction and discussion of global adaptive learning algorithms, especially the class of conjugate gradient methods. Several local adaptive learning rules are introduced in Section IV. The last part of the article discusses the performance of backpropagation and several adaptive variations on a couple of benchmark problems. Some important properties of learning procedures are then examined and compared.

II. Foundations

A. Multi-layer Perceptrons

A multi-layer perceptron is a feed-forward neural network, consisting of a number of units (neurons) which are connected by weighted links. The units are organized in several layers, namely an input layer, one or more hidden layers, and an output layer. The input layer receives an external activation vector, and passes it via weighted connections to the units in the first hidden layer. These compute their activations and pass them to neurons in succeeding layers (Figure 1).

From a distal point of view, an arbitrary input vector is propagated forward through the network, finally causing an activation vector in the output layer. The entire network function, that maps the input vector onto the output vector is determined by the connection weights of the net.

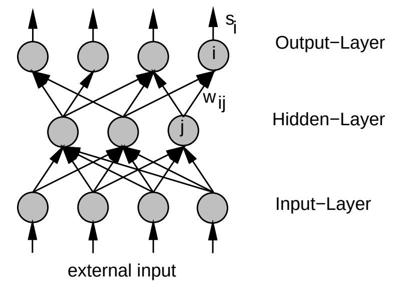

Figure 1: *Topology of a typical feed-forward network with one hidden layer. The external input is presented to the input layer, propagated forward through the hidden layer(s) and yields an output activation vector in the output layer.*

Each neuron i in the network is a simple processing unit that computes its activation s_i with respect to its incoming excitation, the so-called net input net_i:

$$net_i = \sum_{j \in pred(i)} s_j w_{ij} - \theta_i$$

where $pred(i)$ denotes the set of predecessors of unit i, w_{ij} denotes the connection weight from unit j to unit i, and θ_i is the unit's bias value. For the sake of a homogeneous representation, θ_i is often substituted by a weight to a 'bias unit' with a constant output 1. This means that biases can be treated like weights, which is done throughout the remainder of the text.

The activation of unit i, s_i, is computed by passing the net input through a non-linear activation-function. Usually, the sigmoid logistic function

$$s_i = f_{log}(net_i) = \frac{1}{1 + e^{-net_i}}$$

is used. A nice property of this function is its easily computable derivative:

$$\frac{\partial s_i}{\partial net_i} = f'_{log}(net_i) = s_i * (1 - s_i)$$

B. Supervised Learning

In supervised learning, the objective is to tune the weights in the network such that the network performs a desired mapping of input to output activations. The mapping is given by a set of examples of this function, the so-called pattern set $\mathcal{P}$.

Each pattern pair p of the pattern set consists of an input activation vector x^p and its target activation vector t^p. After training the weights, when an input activation x^p is presented, the resulting output vector s^p of the net should equal the target vector t^p. The distance between the target and the actual output vector, in other words the fitness of the weights, is measured by the following energy or cost function E:

$$E := \frac{1}{2} \sum_{p \in \mathcal{P}} \sum_{n} (t_n^p - s_n^p)^2 \qquad (1)$$

where n is the number of units in the output layer. Fulfilling the learning goal now is equivalent to finding a global minimum of E. The weights in the network are changed along a search direction $d(t)$, driving the weights in the direction of the estimated minimum:

$$\begin{aligned} \triangle w(t) &= \epsilon * d(t) \\ w(t+1) &= w(t) + \triangle w(t) \end{aligned}$$

where the learning parameter ϵ scales the size of the weight-step. To determine the search direction $d(t)$, first order derivative information, namely the gradient $\nabla E := \frac{\partial E}{\partial w}$ is commonly used.

The backpropagation algorithm, introduced in the next section, performs successive computation of ∇E by propagating the error back from the output layer towards the input layer.

C. The Backpropagation Algorithm

The basic idea, used to compute the partial derivatives $\frac{\partial E}{\partial w_{ij}}$ for each weight in the network, is to repeatedly apply the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i} \frac{\partial s_i}{w_{ij}} \qquad (2)$$

where

$$\frac{\partial s_i}{\partial w_{ij}} = \frac{\partial s_i}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}} = f'_{log}(net_i)\, s_j \qquad (3)$$

To compute $\frac{\partial E}{\partial s_i}$, or the influence of the output s_i of unit i on the global error E, the following two cases are distinguished:

- If i is an output unit, then

$$\frac{\partial E}{\partial s_i} = \frac{1}{2} \frac{\partial (t_i - s_i)^2}{\partial s_i} = -(t_i - s_i) \qquad (4)$$

- If i is not an output unit, then the computation of $\frac{\partial E}{\partial s_i}$ is a little more complicated. Again, the chain rule is applied:

$$\begin{aligned} \frac{\partial E}{\partial s_i} &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial s_i} \\ &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial net_k} \frac{\partial net_k}{\partial s_i} \\ &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k} f'_{log}(net_k)\, w_{ki} \qquad (5) \end{aligned}$$

where $succ(i)$ denotes the set of all units k in successive layers (successive means closer to the output layer) to which unit i has a non-zero weighted connection w_{ki}.

Equation (5) assumes knowlegde of the values $\frac{\partial E}{\partial s_k}$ for the units in successive layers to which unit i is connected. This can be provided by starting the computation at the output layer (4) and then successively computing the derivatives for the units in preceding layers, applying (5).

In other words, the gradient information is successively moved from the output-layer back towards the input-layer. Hence the name 'backpropagation algorithm'.

D. Gradient Descent

Once the partial derivatives are known, the next step in backpropagation learning is to compute the resulting weight update. In its simplest form, the weight update is a scaled step in the opposite direction of the gradient, in other words the negative derivative is multiplied by a constant value, the learning-rate ϵ. This minimization technique is commonly known as 'gradient descent':

$$\triangle w(t) = -\epsilon * \nabla E(t) \qquad (6)$$

or, for a single weight:

$$\triangle w_{ij}(t) = -\epsilon * \frac{\partial E}{\partial w_{ij}}(t) \qquad (7)$$

Although the basic learning rule is rather simple, it is often a difficult task to choose the learning-rate appropriately. A good choice depends on the shape of the error-function, which obviously changes with the learning task itself. A small learning-rate will result in long convergence time on a flat error-function, whereas a large learning-rate will possibly lead to oscillations, preventing the error to fall below a certain value. Moreover, although convergence to a (local) minimum can be proven under certain circumstances, there is no guarantee that the algorithm finds a *global* minimum of the error-function.

Another problem with gradient descent is the 'contra intuitive' influence of the partial derivative on the size of the weight-step. If the error-function is shallow, the derivative is quite small, resulting in a small weight step. On the other hand, in the presence of steep ravines in the energy landscape, where cautious steps should be taken, large derivatives lead to large weight steps, possibly taking the algorithm to a completely different region of weight space (Figure 2).

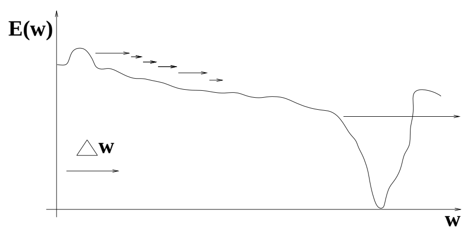

Figure 2: *Problem of gradient descent: The weight-step is dependent on both the learning parameter and the size of the partial derivative* $\frac{\partial E}{\partial w_{ij}}$

An early idea, introduced to make learning more stable, was to add a momentum term:

$$\triangle w_{ij}(t) = -\epsilon \frac{\partial E}{\partial w_{ij}}(t) + \mu \, \triangle w_{ij}(t-1) \qquad (8)$$

The momentum parameter μ scales the influence of the previous weight-step on the current one. It should be noted that although this technique works well on many learning tasks, this is not a general technique for gaining stability or speeding up convergence. Sometimes, comparable or even better results can be achieved by using no momentum term at all. Usually, when using gradient descent with momentum, the learning-rate should be decreased to avoid unstable learning.

E. Learning by pattern versus. Learning by epoch

Basically there are two possible methods for computing and performing weight-update, depending on *when* the update is performed.

In the 'learning by pattern' method, a weight-update is performed after each presentation of a pattern pair and the computation of the respective gradient. This is also known as 'online learning' or 'stochastic learning', because one tries to minimize the overall error by minimizing the error for each individual pattern pair, and these are not actually the same. This method works especially well for large pattern sets containing substantial amounts of redundant information.

An alternative method, known as 'learning by epoch', first sums gradient information for the *whole* pattern set, then performs the weight-updates. This method is also known as 'batch learning'. Each weight-update tries to minimize the summed error of the pattern set, in other words the error-function defined in (1). The adaptive procedures described in the following section use the latter type of learning, because the summed gradient information for the whole pattern set contains more reliable information regarding the shape of the entire error-function.

F. Adaptive Techniques

Many techniques have been proposed to date to deal with the above mentioned, inherent problems of gradient descent. Most of these have their roots in the well-explored domain of optimization theory.

These techniques can roughly be divided into two categories. Algorithms that use global knowledge of the state of the entire network, such as the direction of the overall weight-update vector, are referred to as 'global' techniques. There are many examples where adaptive learning algorithms make use of global knowledge [2], [3]. One class of global algorithms, the conjugate gradient method, is discussed in the following section.

By contrast, local adaptation strategies are based on weight-specific information only, such as the temporal behavior of the partial derivative of this weight. The local approach is more closely related to the neural network concept of distributed processing in which computations can be made in parallel. Furthermore, it appears that for many applications local strategies work far better than global techniques, although they use less information and are often much easier and faster to compute [4].

III. Global Adaptive Techniques

The following presents a short review of some global adaptation techniques. A good introduction to the foundation of several optimization approaches can be found in [5].

A. Steepest Descent

While the gradient descent technique used in the standard backpropagation algorithm performs weight-update by a *constant* scaling ϵ of a search direction $d(t) = -\nabla E(t)$, the 'steepest descent' procedure tries to take an optimal weight-step by finding an individual scaling parameter $\epsilon(t)$ each iteration.

Determining such an optimal parameter can be regarded as a one-dimensional optimization problem known as 'line search'. In the simplest case, a small initial learning-rate is used, which is iteratively increased until the error-function no longer decreases.

Unfortunately, for every iteration the evaluation of the error-function E is required, which means a costly forward propagation of the whole pattern set to compute the new value of E. In general, more elaborate methods for line search must be used, such as the false position method, which typically converges in 2-3 iterations [6].

When applying the method of steepest descent, it can be shown that two successive weight-steps are necessarily perpendicular. Assume that an ϵ has been found that yields an optimal weight-step, which means that $\frac{\partial E(w(t+1))}{\partial \epsilon} = 0$. Then,

$$\begin{aligned} \frac{\partial E(w(t+1))}{\partial \epsilon} &= \frac{\partial E(w(t+1))}{\partial w(t+1)} \frac{\partial (w(t) + \epsilon * d(t))}{\partial \epsilon} \\ &= \nabla E(t+1)\, d(t) \\ &= 0 \end{aligned} \qquad (9)$$

This means that the new gradient $\nabla E(t+1)$, which determines the new direction $d(t+1)$, and the old search direction $d(t)$ are perpendicular. This relation is finally used to improve the performance of the steepest descent procedure, as it is done by conjugate gradient methods described in the following section.

B. The Conjugate Gradient Method

Finding an optimal learning-rate is a costly iterative procedure, so we do not want to completely destroy this effort in succeeding steps. Accordingly, the condition found in equation (9) should also hold for the following weight-step, namely

$$d(t)\, \nabla E(t+2) \overset{!}{=} 0 \qquad (10)$$

It can be shown that condition (10) is fulfilled, if

$$d(t)\, \mathcal{H}\, d(t+1) = 0 \qquad (11)$$

where $\mathcal{H}$ denotes the Hessian matrix, containing the second order derivatives of the weights. Two vectors fulfilling condition (11) are called 'conjugate'.

To determine the new search direction $d(t+1)$ that fulfills (11) we set:

$$d(t+1) := -\nabla E(t+1) + \beta * d(t)$$

This means that the new search direction is a combination of both the direction indicated by the gradient and by the previous search direction.

The parameter β is computed for example according to the Polak-Ribiere rule:

$$\beta = \frac{(\nabla E(t+1) - \nabla E(t))\, \nabla E(t+1)}{(\nabla E(t))^2}$$

As in the steepest descent procedure, a line search technique has to be applied to find an optimal learning-rate that minimizes the error along the new search direction $d(t+1)$.

According to the reported results on several small binary classification tasks [6], the higher expense of the conjugate gradient computation (line search + Polak Ribiere) is compensated very well by a much faster convergence compared with backpropagation learning. However, as shown in a recent comparison on a real world benchmark [4], global optimization techniques can experience severe problems with convergence when applied to larger learning tasks.

IV. Local Adaptive Techniques

A. The Delta-Bar-Delta Rule

To overcome the drawbacks of the simple backpropagation weight update, Jacobs [7] proposed weight-specific learning-rates, since the error-function may have a different shape with respect to the one-dimensional view of each weight in the network. Because of this, Jacobs introduced a second learning law, which determines the evolution of a learning-rate according to a local estimation of the shape of the error-function.

This estimation is based on the observed behavior of the partial derivative during two successive weight-steps. If the derivatives have the same sign, the learning-rate is linearly increased by a small constant to accelerate learning in shallow regions. On the other hand, a change in sign of the two derivatives indicates that the procedure has overshot a local minimum; the previous weight-step was too large. As a consequence, the learning-rate is exponentially decreased by multiplying it with a decreasing-factor smaller than unity:

$$\epsilon_{ij}^{(t)} = \begin{cases} \kappa + \epsilon_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- * \epsilon_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \epsilon_{ij}^{(t-1)} & , \text{ else} \end{cases} \qquad (12)$$

$$\text{with } 0 < \eta^- < 1$$

The weight-update itself is the same as with backpropagation learning, except that the fixed global learning-rate ϵ is replaced by a weight-specific, dynamic learning-rate $\epsilon_{ij}(t)$:

$$\triangle w_{ij}(t) = -\epsilon_{ij}(t) \frac{\partial E}{\partial w_{ij}}(t) + \mu \triangle w_{ij}(t-1) \qquad (13)$$

As reported in [7], the Delta-Bar-Delta converges faster than backpropagation and is more robust with respect to choice of parameters.

B. SuperSAB

SuperSAB [8] is also based on the idea of sign-dependent learning-rate adaptation, as just described with the Delta-Bar-Delta method. A very similar approach can be found in [9].

The basic change is to increase the learning-rate exponentially instead of linearly as with the Delta-Bar-Delta method. This is done to take the wide range of temporarily suited learning-rates into account.

$$\epsilon_{ij}^{(t)} = \begin{cases} \eta^+ * \epsilon_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- * \epsilon_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \epsilon_{ij}^{(t-1)} & , \text{ else} \end{cases} \qquad (14)$$

$$\text{with } 0 < \eta^- < 1 < \eta^+$$

Moreover, in case of a change in sign of two successive derivatives, the previous weight-step is reverted.

SuperSAB has shown to be a fast converging algorithm, that is often considerably faster than ordinary gradient descent. One possible problem of SuperSAB is the large number of parameters that need to be determined in order to achieve good convergence times, namely the initial learning-rate, the momentum factor, and the increase (decrease) factors.

Another drawback, inherent to all learning-rate adaptation algorithms, is the remaining influence of the size of the partial derivative on the weight-step:

$$\triangle w_{ij}(t) := -\epsilon_{ij}(t) * \frac{\partial E}{\partial w_{ij}}(t) + \mu \triangle w_{ij}(t-1)$$

Despite careful adaptation of the learning-rate, the derivative itself can have an un-foreseeable influence on the size of the weight-step. For example consider the situation, where a very shallow error-function leads to a permanent increase of the learning-rate. Although the learning-rate grows rather large, the resulting weight-step remains small, due to the small partial derivative. When suddenly a region of steep descent is reached, probably indicating the presence of a minimum, the resulting large derivative is scaled by the large learning-rate, pushing the weight in a region far away from the previous (promising) position (Figure 2).

C. *Quickprop*

A completely different approach to local adaptive learning is that of Fahlman [10], in which the local error-function for each weight is assumed to be a 'parabola whose arms are opened upward', and that the slope of the curve is not affected by changing all other weights in the network. Estimates of the position of the minimum for each weight are obtained by solving the following equation for the two following partial derivatives $\frac{\partial E}{\partial w_{ij}}(t-1)$ and $\frac{\partial E}{\partial w_{ij}}(t)$:

$$\triangle w_{ij}(t) := \frac{\frac{\partial E}{\partial w_{ij}}(t)}{\frac{\partial E}{\partial w_{ij}}(t-1) - \frac{\partial E}{\partial w_{ij}}(t)} \triangle w(t-1) \tag{15}$$

It can be shown that this weight update is equivalent to a local application of Newton's approximation method, which can be derived from the first order Taylor series expansion for the approximation of the error. The objective is to find a minimum of $f(x)$, and this is done by searching for an x for which $f'(x) = 0$. Under the assumption that $f'(x)$ is convex, Newton's method iteratively computes updates of x according to the following equation:

$$x(t+1) = x(t) + \triangle x(t) \tag{16}$$

where

$$\triangle x(t) = -\frac{f'(x(t))}{f''(x(t))} \tag{17}$$

If the second order information $f''(x)$ is not easily available (as it is the case for the weights in a neural network), an approximation is made using the first order derivatives:

$$\begin{aligned} f''(x(t)) &= \frac{f'(x(t)) - f'(x(t-1))}{x(t) - x(t-1)} \\ &= \frac{f'(x(t)) - f'(x(t-1))}{\triangle x(t-1)} \end{aligned} \tag{18}$$

Substituting (18) in (17) then yields:

$$\begin{aligned} \triangle x(t) &= -\frac{f'(x(t))}{f'(x(t)) - f'(x(t-1))} \triangle x(t-1) \\ &= \frac{f'(x(t))}{f'(x(t-1)) - f'(x(t))} \triangle x(t-1) \end{aligned} \tag{19}$$

This corresponds exactly to the expression given in equation (15).

Although the main formula for the weight-update (15) is straightforward and easy to compute, there are a few modifications necessary, due to violation of the above assumptions. Firstly, the actual update-rule is composed of both the application of (15) and a small gradient descent step. Moreover, in order to avoid arbitrary large weight-steps resulting from a possibly very small denominator in (15), the present weight-step is restricted to be at most ν times as large as the previous step.

Thus the Quickprop algorithm has two parameters, these being a learning rate ϵ for gradient descent, and a second parameter ν which limits the step-size (the default value for ν is 1.75).

There is a marked improvement of learning time compared with standard backpropagation, and indeed Quickprop is one of today's most frequently used adaptive learning paradigms.

D. *Rprop*

Rprop stands for '**R**esilient back**prop**agation' and is a local adaptive learning scheme [11]. The basic principle of Rprop is to eliminate the harmful influence of the size of the partial derivative on the weight step. As a consequence, only the sign of the derivative is considered to indicate the direction of the weight update. The size of the weight change is exclusively determined by a weight-specific, so-called 'update-value' $\triangle_{ij}$:

$$\triangle w_{ij}(t) = \begin{cases} -\triangle_{ij}(t) & , \text{ if } \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ +\triangle_{ij}(t) & , \text{ if } \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ 0 & , \text{ else} \end{cases} \tag{20}$$

It should be noted, that by replacing the $\triangle_{ij}$ by a constant update-value $\triangle$, equation (20) yields the so-called 'Manhattan'-update rule.

The second step of Rprop learning is to determine the new update-values $\triangle_{ij}(t)$. This is based on a sign-dependent adaptation process, similar to the learning-rate adaptation of equation (14).

$$\triangle_{ij}^{(t)} = \begin{cases} \eta^{+} * \triangle_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^{-} * \triangle_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \triangle_{ij}^{(t-1)} & , \text{ else} \end{cases} \tag{21}$$

where $0 < \eta^- < 1 < \eta^+$

At the beginning, all update-values are set to an initial value $\triangle_0$, which is one of two parameters of Rprop. Since $\triangle_0$ directly determines the size of the first weight step, it should be chosen according to the initial values of the weights themselves, for example $\triangle_0 = 0.1$. The choice of this value is rather uncritical, for it is adapted as learning proceeds.

In order to prevent the weights from becoming too large, the maximum weight-step determined by the size of the update-value, is limited. The upper bound is set by the second parameter of Rprop, $\triangle_{max}$. The default upper bound is set somewhat arbitrarily to $\triangle_{max} = 50.0$. Usually, convergence is rather insensitive to this parameter as well. Nevertheless, for some problems it can be advantageous to allow only very cautious (namely small) steps, in order to prevent the algorithm getting stuck too quickly in suboptimal local minima.

The increase and the decrease factor are fixed to $\eta^+ = 1.2$ and $\eta^- = 0.5$. These values are based on both theoretical considerations and empirical evaluations. This reduces the number of free parameters to two, namely $\triangle_0$ and $\triangle_{max}$.

To summarize, the basic principle of Rprop is the direct adaptation of the weight update-values $\triangle_{ij}$. In contrast to the learning-rate based algorithms described earlier, Rprop modifies the size of the weight-step directly by introducing the concept of resilient update-values. As a result, the adaptation effort is not blurred by un-foreseeable gradient behaviour. Due to the clarity and simplicity of the learning laws, there is only a slight expense in computation compared with ordinary backpropagation.

Rprop suffers from the same problem as does any of the above mentioned adaptive learning algorithms. Because the adaptation is based on an estimation of the topology of the error-function, both adaptation and weight update can be first performed after the whole gradient information is available, in other words after each pattern has been presented and the gradient of the sum of pattern errors is known. Accordingly, adaptive learning procedures are typically based on 'learning by epoch'. This possibly reduces their efficiency on redundant training sets compared to a simple stochastic gradient descent and poses problems on their use with variable training sets.

Moreover, a restricted local adaptation scheme inherently lacks the overall view that global techniques may have. If for example, an optimal search direction for the minimum lies along the diagonal, a local scheme will try to decrease the error in each dimension, by carefully searching the local minimum with small weight-steps; it will not increase the composite weight-step along the diagonal, which would be the more appropriate approach in this case.

Nevertheless, the results reported in the following section show the favorable properties of local adaptation strategies in practical applications.

V. Comparative Studies

A good learning algorithm should fulfill at least the following requirements:

- fast convergence
- easy parameter choice
- good generalization ability on unknown inputs

Due to the wide variety of different learning problems with different requirements and different goals it is not easy to establish a fair comparison between the many variants of supervised learning techniques. Nearly as many benchmark problems are reported in the literature as new learning algorithms. This is not surprising, since every new variant solves a specific learning problem faster than most other techniques, and certainly faster than backpropagation.

One of the most famous benchmark problems is the 'exclusive or' (XOR) problem, or in its more general form, the N-parity problem. Following the argumentation of Fahlman [10], this is not a typical benchmark for the real world problems solved with neural networks. The highly desired ability of a network to generalize, that is to map similar input patterns to similar output activations, doesn't apply with XOR. A single change of a bit in the input vector requires a complementary classification. The reason why we include N-parity problems here is that they are often used in the literature to benchmark new learning algorithms.

A better class of benchmarks is the family of the N-M-N encoder problems. The network consists of N units each in the input and output layers, and M neurons in the hidden layer. The input vector comprises N bits, one of which is set to '1', and the remaining bits set to '0'. The output (target) vector is identical to the input, so the task of the network is to perform an auto-association between input and output vectors. The objective is to learn a mapping of N input units to M hidden units (encoding) and a mapping of M hidden units to N output units (decoding), where in general $M < N$. If $M \leq \log_2 N$ we refer to this mapping as a 'tight encoder'.

A. Testing Conditions

In the following experiments the performance of several algorithms was tested in twenty runs, each with a different initial weight setting. The weights were chosen randomly within a certain range. Learning time is reported as the average number of epochs[1] required until the tasked was learned. If a run failed to converge, its convergence time is set to a benchmark-dependent maximum number of epochs. The number of converged runs is reported in the 'success' row of each table.

Following Fahlman's suggestions as to how results should be reported, learning of binary tasks is complete, if a '40-20-40' criterion is fulfilled: an output is considered to be a logical zero if it is in the lower 40% of the output

[1] An epoch is defined as the period during which every pattern of the training set is presented once

range, a one if it is in the upper 40%, and indeterminate (and therefore incorrect) if it is in the middle 20% of the range.

A wide variety of parameter values was tested in order to find a correspondingly good choice for each learning algorithm. However, in practice it is often undesirable or even impossible to perform large parameter test series, due to time or hardware constraints. Moreover, the easier it is to find a parameter setting that allows fast and robust convergence, the better the algorithm will be suited for practical application. The sensitivity of the average number of required epochs on a good choice of the initial learning parameter is shown in the figures which follow.

The tables show the average number of epochs required using the best parameter setting. The comparison was performed for the following learning procedures: Backpropagation by epoch (BP), SuperSAB (SSAB), Quickprop (QP) and Rprop.

In the following, ϵ denotes the (initial) learning-rate (BP, SSAB, QP), $\triangle_0$ denotes the initial update-value (RPROP), $\triangle_{max}$ is the maximum step size (RPROP), μ is the momentum (BP, SSAB), and ν denotes the maximal growth factor (QP).

B. 3 Bit Parity

This is the 3 bit version of the 'XOR'-problem. The three-layer-network consists of 3 input, 3 hidden and 1 output neuron. The target for the output is 'one', if the number of 'one' bits in the input is odd, and 'zero' otherwise. For the symmetric nature of the problem we used symmetric activation functions with a range of $[-1, +1]$. The maximum learning time was set to 100 epochs. The weights were randomly initialized within the range $[-1.0, +1.0]$. In summary:

Task:	3 bit parity
Network:	3-3-1 (3 input, 3 hidden, 1 output)
No. of patterns:	$2^3 = 8$
Activation:	symmetric
Max. epochs:	100
Weight initialization:	[-1.0,+1.0]

B.1. Learning Time

Table 1 shows the results for the different learning algorithms on the 3 bit parity task.

3 Bit Parity				
Algorithm	$\epsilon/\triangle_0$	$\mu/\nu/\triangle_{max}$	# epochs	success
BP by ep.	0.2	0.9	17.7	20/20
SSAB	1.0	0.5	19.2	20/20
QP	0.1	$*$	18.3	20/20
RPROP	0.07	$*$	17.6	20/20

Table 1: *3 Bit Parity: Results for the different learning procedures*

As is clear from this table, all algorithms converge rather fast when the corresponding best parameter setting was used. Due to the very short convergence time in general, the adaptive algorithms have no chance to prove their superiority over pure gradient descent when optimally tuned parameters are used. The '$*$'-mark in the second row of both Rprop and Quickprop means that the respective default value was used, and that no further tuning was needed for this parameter - in fact both algorithms only needed tuning of one parameter to achieve their best result.

B.2. Sensitivity

In the following, we regard the influence of the choice of learning parameter on the average number of epochs required. For convenience, we consider the (initial) learning-rate ϵ for backpropagation, SuperSAB and Quickprop and the initial update-value $\triangle_0$ for Rprop. The remaining parameters for each algorithm are set to the values that can be found in Table 1.

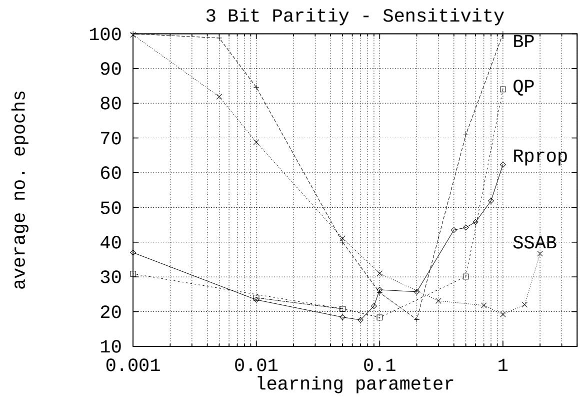

Figure 3: *3 Bit Parity: Sensitivity of the different learning procedures to choice of initial learning parameter*

As demonstrated in Figure 3, standard backpropagation is very sensitive to the choice of its learning-rate parameter. A slight deviation from the optimal value causes the algorithm to consume considerably more learning time. A little surprising is the obvious sensitivity of the SuperSAB algorithm, despite its learning-rate parameter being adapted during learning. This is possibly due to the highly nonlinear nature of the parity problem. Both Quickprop and Rprop are rather robust with respect to choice of initial learning parameter. This is a notable result, since their second parameters have been set to their default values (see Table 1).

C. 6 Bit Parity

C.1. Description

Task:	6 bit parity
Network:	6-12-1 (6 input, 12 hidden, 1 output)
No. of patterns:	$2^6 = 64$
Activation:	symmetric
Max. epochs:	1000
Weight initialization:	[-1.0,+1.0]

The results of the 6 bit parity problem are reported to illuminate the scaling properties of the algorithms, in other words their convergence behavior when the difficulty of the learning task is increased.

C.2. Learning Time and Sensitivity

Table 2 shows the results obtained using the different learning algorithms to solve the 6 bit parity problem.

6 Bit Parity				
Algorithm	$\epsilon/\triangle_0$	$\mu/\nu/\triangle_{max}$	# epochs	success
BP by ep.	0.3	0.0	279.4	16/20
SSAB	0.01	0.9	82.6	20/20
QP	0.005	$*$	50.5	20/20
RPROP	0.05	$*$	52.8	20/20

Table 2: *6 Bit Parity: Results for the different learning procedures*

The 6 bit parity task is considerably more difficult to learn than the 3 bit version described previously. This is reflected in the drastically increased number of epochs required by the backpropagation algorithm (about fifteen times as high). Moreover, backpropagation failed to converge in 4 of the 20 trials. The use of a momentum term did not improve convergence on this learning task. In contrast to this, all the adaptive algorithms converged in all twenty runs, being more than 3 times (SuperSAB) or even more than 5 times as fast (Quickprop, Rprop) compared with pure gradient descent.

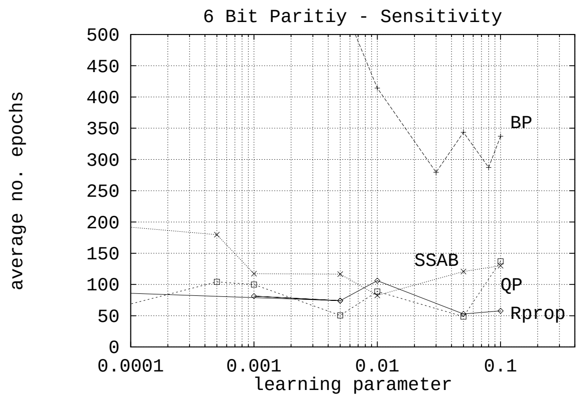

Figure 4: *6 Bit Parity: Sensitivity of the different learning procedures to choice of initial learning parameter*

Figure 4 shows the superiority of the adaptive algorithms even more impressively. For a wide range (several magnitudes) of initial values of learning parameter, both Quickprop and Rprop converge at least 3 times faster than standard backpropagation. Again, the second parameter of Quickprop and Rprop could remain set to its default value, which again simplifies their use.

D. 10-5-10 Encoder

D.1. Description

The 10-5-10 encoder task is a typical benchmark problem of the N-M-N encoder family described earlier. The network consists of 10 input units, 5 hidden units and 10 output units. The pattern set contains 10 pattern pairs. In summary:

Task:	10-5-10 Encoder
Network:	10-5-10 (10 input, 5 hidden, 10 output)
No. of patterns:	10
Activation:	logistic
Max. epochs:	500
Weight initialization:	[-1.0,+1.0]

D.2. Learning Time and Sensitivity

Table 3 shows the average learning times of the different procedures.

10-5-10 Encoder				
Algorithm	$\epsilon/\triangle_0$	$\mu/\nu/\triangle_{max}$	# epochs	success
BP by ep.	1.7	0.0	137.1	20/20
SSAB	2.0	0.8	49.2	20/20
QP	1.0	$*$	21.0	20/20
RPROP	0.7	$*$	19.0	20/20

Table 3: *10-5-10 Encoder: Results for the different learning procedures*

This again is an example of a learning task where the best backpropagation result was achieved using no momentum. This demonstrates the need to both alter learning-rate and momentum in order to find a good parameter setting for standard backpropagation.

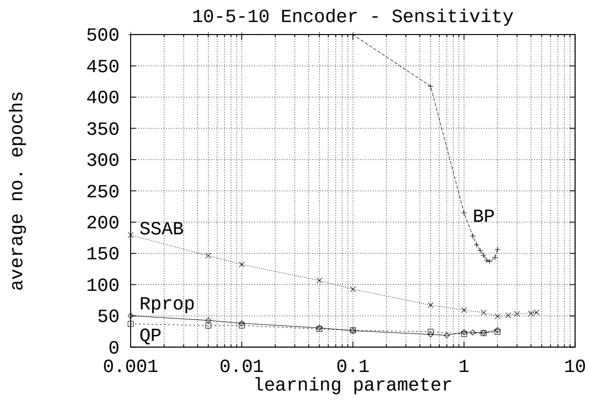

Figure 5: *10-5-10 Encoder Task: Sensitivity of the different learning procedures to choice of initial learning parameter*

On the 10-5-10 encoder task, Quickprop and Rprop clearly outperform the other algorithms, needing less than one sixth of the number of epochs required by backpropagation and being more than 2 times faster than SuperSAB. As far as the influence of choice of learning parameter on learning time is concerned, this task again confirms the particular robustness of adaptive learning algorithms against variation of their parameters (Figure 5).

E. 12-2-12 Encoder

E.1. Description

In the next experiment, the learning task was made more difficult by reducing the number of hidden units, in order to investigate the algorithms' ability to find sophisticated solutions in weight space. The number of input and output units was increased to 12, while the width of the hidden layer was reduced to two neurons.

Task:	12-2-12 'Tight' Encoder
Network:	12-2-12 (12 input, 2 hidden, 12 output)
No. of patterns:	12
Activation:	logistic
Max. epochs:	15000
Weight initialization:	[-1.0,+1.0]

E.2. Learning Time and Sensitivity

Table 4 shows the results of the 12-2-12 Encoder problem.

12-2-12 'Tight Encoder'				
Algorithm	$\epsilon/\triangle_0$	$\mu/\nu/\triangle_{max}$	# epchs	success
BP	div.	div.	> 15000	0/20
SSAB	1.0	0.95	536.0	20/20
QP	1.0	1.2	221.0	20/20
RPROP	0.5	$*$	210	20/20

Table 4: *12-2-12 Encoder: Results for the different learning procedures*

Interestingly, backpropagation was not able to learn the task in under $15,000$ epochs, despite many different parameter settings being tested. On the other hand, all adaptive procedures converged rather fast, although both SuperSAB and Quickprop needed some fine tuning of their second parameters to do so. Moreover, Quickprop exhibited quite sensitive behavior, depending on the choice of its first learning parameter. Rprop converged very fast, using the default setting for its second parameter, and again was rather insensitive to choice of its first learning parameter (Figure 6).

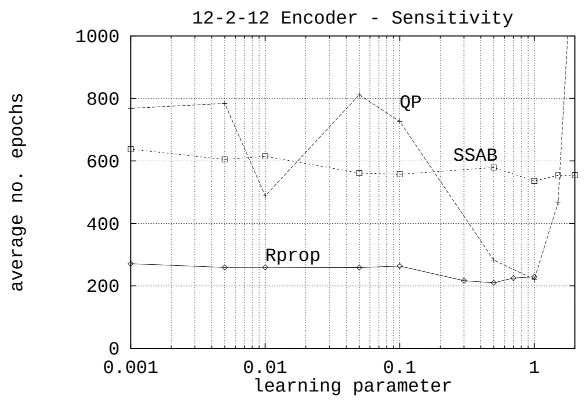

Figure 6: *12-2-12 Encoder Task: Sensitivity of the different learning procedures to choice of initial learning parameter*

The fast and robust convergence of adaptive learning algorithms, and the failure of pure gradient descent, demonstrates the ability of the advanced techniques to exploit their adaptability to solve very complex learning tasks in situations, where a suitable solution in weight space is difficult to find. We may even conclude that, by using adaptive methods, smaller networks with fewer weights can be used. This will on one the one hand lead to less computational effort, and on the other hand to better generalization ability promoted by a less complex network topology.

F. Two Spirals

The task of this difficult benchmark problem is to discriminate between two spirals which coil three times around the origin of the x-y plane. The training set consists of 194 points in the plane belonging either to one class (spiral) or to the other. The network consists of three hidden layers with 5 units (nodes) per layer. Each unit is connected to every unit in previous layers (the network uses so called 'short-cut connections' [12]).

Task:	Two Spirals
Network:	2-5-5-5-1 (+shortcut connections)
No. of patterns:	194
Activation:	symmetric
Max. epochs:	15000
Weight initialization:	[-1.0,+1.0]

G. Learning time and sensitivity

Table 5 shows the results on the two spirals problem.

Two Spirals				
Algorithm	$\epsilon/\triangle_0$	$\mu/\nu/\triangle_{max}$	# epchs	success
BP	0.0008	0.9	8830	9/20
SSAB	0.01	0.9	10015	8/20
QP	0.00005	1.3	8415	12/20
RPROP	0.001	0.1	2605	19/20

Table 5: *Two Spirals Task: Results for the different learning procedures*

The difficulty of this benchmark is reflected not only in the large number of epochs but also in the high failure rates. In subsequent experiments we further discovered that convergence is highly dependent on the range of weight initialization.

When a careful parameter tuning was applied, the performance of standard backpropagation was comparable with both SuperSAB and Quickprop. Convergence was reached in 9 out of 20 runs for backpropagation, and in 8 out of 20 runs for SuperSAB. Quickprop worked more reliably and converged in 12 runs. The best result for Quickprop was achieved using a cautious setting for its second parameter ($\nu = 1.3$). Rprop achieved the best result on this benchmark, converging in 19 out of 20 runs, and thereby being more than 3 times faster than the other algorithms. For this difficult and highly nonlinear problem, Rprop's second parameter was chosen considerably smaller than its default value. Again, the tuning of the first parameter was rather non critical (Figure 7).

VI. Conclusion

This article gives an overview over past and recent developments in algorithms for supervised learning in multilayer perceptrons.

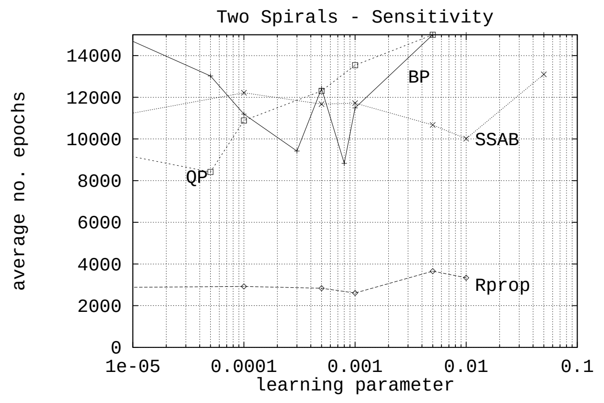

Figure 7: *Two Spirals Task: Sensitivity of the different learning procedures to choice of initial learning parameter*

All approaches described here make use in some manner of the first order partial derivative of each weight with respect to the overall network error. This gradient information can easily be computed by the backpropagation algorithm. The variety of proposed weight learning rules ranges from a simple gradient descent (commonly referred to as 'backpropagation learning') to more sophisticated global and local adaptation techniques.

Many of the proposed procedures, especially the global techniques, are based on ideas from many dimensional optimization theory, and often require increased computation.

As demonstrated on a couple of representative benchmark problems the local adaptive algorithms, especially Quickprop and Rprop, converge considerably faster than the ordinary gradient descent algorithm. What is probably even more significant from a practical perspective is the drastically improved robustness of the adaptive algorithms with respect to choice of initial parameters.

It appears, that their very simple and straightforward local adaptation rules are very effective for the type of function minimization required in the context of multilayer neural networks.

References

[1] D. E. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing, Vol. I Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[2] R. Salomon. Improved convergence rate of backpropagation with dynamic adaptation of the learning rate. In H.-P. Schwefel and R. Männer, editors, *Lecture Notes in Computer Science, PPSN 1*, pages 269–273, Dortmund, 1990. Springer-Verlag.

[3] M. F. Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(3):525–533, 1993.

[4] W. Schiffmann, M. Joost, and R. Werner. Optimization of the backpropagation algorithm for training multilayer perceptrons. Technical report, University of Koblenz, Institute of Physics, 1993.

[5] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the theory of neural computation.* Addison-Wesley, Redwood City, CA 94065, 1991.

[6] A. Kramer and S. Vincentelli. Efficient parallel learning algorithms for neural networks. In D.Touretzky, editor, *Advances in Neural Information Processing*, volume I, San Mateo, 1989. Morgan Kauffman.

[7] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4), 1988.

[8] T. Tollenaere. Supersab: Fast adaptive backpropagation with good scaling properties. *Neural Networks*, 3(5), 1990.

[9] Fernando M. Silva and Luis B. Almeida. Speeding up backpropagation. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 151–158. North-Holland, Amsterdam, 1990.

[10] S. E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, CMU-CS-88-162, Carngie-Mellon University, 1988.

[11] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In H. Ruspini, editor, *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pages 586 – 591, San Francisco, 1993.

[12] K. Lang and M. Witbrock. Learning to tell two spirals apart. In *Proceedings of 1988 Connectionist Models Summer School.* Morgan Kaufmann, 1988.