To be published in:
Neurocomputing

Empirical Evaluation of the Improved Rprop Learning Algorithms

Christian Igel and Michael Hüsken
Institut für Neuroinformatik
Ruhr-Universität Bochum
44780 Bochum, Germany
Tel: +49 234 32 25558 Fax: +49 234 32 14209
{Christian.Igel,Michael.Huesken}@neuroinformatik.ruhr-uni-bochum.de

Abstract

The Rprop algorithm proposed by Riedmiller and Braun is one of the best performing first-order learning methods for neural networks. We discuss modifications of this algorithm that improve its learning speed. The new optimization methods are empirically compared to the existing Rprop variants, the conjugate gradient method, Quickprop, and the BFGS algorithm on a set of neural network benchmark problems. The improved Rprop outperforms the other methods; only the BFGS performs better in the later stages of learning on some of the test problems. For the analysis of the local search behavior, we compare the Rprop algorithms on general hyperparabolic error landscapes, where the new variants confirm their improvement.

Keywords: supervised learning; resilient backpropagation (Rprop); gradient-based optimization

1 Yet Another Learning Algorithm?

Much research in the field of neural networks is concerned with developing new learning algorithms, where learning is used as a synonym for data-driven optimization of network parameters. Each year many new learning methods are presented, so why should one care for yet another one? The learning algorithms proposed in this article are not totally new approaches, but modifications of the two established Rprop (*r*esilient back*prop*agation) algorithms [17, 18, 6]. The improved versions maintain the advantageous properties of the originals:

1. Rprop as proposed by Riedmiller and Braun is (already) very fast and accurate. The reader is referred to [8, 17, 18, 19] for comparisons of Rprop with other supervised learning techniques and to the review of learning methods in [15].

2. The Rprop algorithms are known to be very robust with respect to their internal parameters [16, 17, 18]. In addition, these parameters are comparatively intuitive and therefore easy to adjust.

3. Rprop is a first-order method. The time and space complexity scales only linearly with the number of parameters to be optimized.

4. The algorithms are general methods for gradient-based optimization. In particular, they do not rely on special properties of a certain class of network topologies.

5. The update rule depends only on the sign of the derivative and not on its amount. So Rprop is very suitable for applications where the gradient is numerically estimated or the error is noisy.

6. Rprop is easy to implement and not susceptible to numerical problems. A hardware implementation is described in [11].

In this study, strong empirical evidence is given that the new methods outperform the original ones in terms of speed. The empirical evaluation makes use of four neural network benchmark problems (two classification and two regression tasks) and artificial error landscapes. The proposed new algorithms are compared to widely used general gradient-based optimization techniques, namely the two original Rprop variants, Fahlman's Quickprop, the BFGS (Broyden, Fletcher, Goldfarb, and Shanno) algorithm, and the conjugate gradient method.

In the next section, we describe the Rprop algorithm as proposed by Riedmiller and Braun [17, 18] and our modifications firstly suggested in [6]. In Sec. 3 we evaluate the learning schemes on neural network benchmark problems and in the succeeding section we analyze the behavior of the Rprop algorithms on artificial error surfaces.

2 The Rprop Algorithms

Gradient-based optimization techniques are the most widely used class of algorithms for supervised learning in neural systems. Adaptive gradient based algorithms with individual step-sizes try to overcome the inherent difficulty of the choice of the right learning rates. This is done by controlling the weight update for every single connection during the learning process in order to minimize oscillations and to maximize the update step-size. Let w_{ij} denote the weight in a neural network from neuron j to neuron i and E be an arbitrary error measure that is differentiable with respect to the weights. Bias parameters are regarded as being weights from an extra constant input. Superscripts indicate the learning iteration. In each iteration the new weights are given by

$$w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)} \quad . \tag{1}$$

The learning algorithm stops when a certain termination criterion is met (e.g., t exceeds a predefined value). In the Rprop learning algorithm the direction of each weight update is based on the sign of the partial derivative $\partial E/\partial w_{ij}$. A step-size Δ_{ij}, i.e., the update amount of a weight w_{ij}, is adapted for each weight individually. The main difference to other techniques is that the step-sizes are independent of the absolute value of the partial derivatives: If no *weight-backtracking* (see Sec. 2.1) is used the step-sizes are computed as

$$\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)} \quad , \tag{2}$$

where the sign(.) operator returns $+1$ if its argument is positive, -1 if the argument is negative, and 0 otherwise. The Δ_{ij} are initialized to a constant Δ_0. The benefits of this update scheme are described in [17, 18]. One iteration of the original Rprop algorithm can be divided into two parts. The first part, the adjustment of the step-sizes, is basically the same for all Rprop algorithms employed in this study. For

each weight w_{ij} an individual step-size Δ_{ij} is adjusted using the following rule:

$$\Delta_{ij}^{(t)} := \begin{cases} \min\left(\eta^+ \cdot \Delta_{ij}^{(t-1)}, \Delta_{\max}\right) & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \max\left(\eta^- \cdot \Delta_{ij}^{(t-1)}, \Delta_{\min}\right) & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \Delta_{ij}^{(t-1)} & , \text{ otherwise } , \end{cases} \tag{3}$$

where $0 < \eta^- < 1 < \eta^+$. If the partial derivative $\partial E/\partial w_{ij}$ possesses the same sign for consecutive steps, the step-size is increased, whereas if it changes sign, the step-size is decreased (the same principle is also used in other learning methods, e.g., [7, 24]). The step-sizes are bounded by the parameters $\Delta_{\min}$ and $\Delta_{\max}$. Note that by (1) and (2) the following holds:

$$\frac{\partial E^{(t)}}{\partial \Delta_{ij}^{(t-1)}} = \frac{\partial E^{(t)}}{\partial w_{ij}^{(t)}} \frac{\partial w_{ij}^{(t)}}{\partial \Delta_{ij}^{(t-1)}} = -\frac{\partial E}{\partial w_{ij}}^{(t)} \operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t-1)}\right) . \tag{4}$$

Hence, the direction of the change of Δ_{ij} following (3) is in accordance with a gradient-based optimization of the network error with respect to Δ_{ij}.

In the following, we describe the second part of the algorithm, the update of the weights, for the different Rprop versions. For convenience, we use the same notation for the partial derivatives and the variables holding the values of the partial derivatives in the implementation of the algorithms. Two of the algorithms described later reset these variables to zero indicated by $\partial E^{(t)}/\partial w_{ij}{}^{(t)} := 0$ in order to affect the scheme described by (3).

2.1 Rprop with Weight-Backtracking

The first version of the Rprop algorithm as proposed in [18] implements a general concept for improving network training termed weight-backtracking, which means retracting a previous update for some or all weights, cf. [22, 24, 25]. Whether to take back a step or not is decided by means of a heuristic.

After adjusting the step-sizes according to (3), the weight updates Δw_{ij} are determined. Two cases are distinguished. If the sign of the partial derivative has not changed, a regular weight update is executed:

$$\textbf{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} \geq 0 \textbf{ then } \Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)} . \tag{5}$$

In case of a change of sign of the partial derivative, the previous weight update is reverted:

$$\textbf{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \textbf{ then } \left\{ \Delta w_{ij}^{(t)} := -\Delta w_{ij}^{(t-1)} ; \frac{\partial E}{\partial w_{ij}}^{(t)} := 0 \right\} . \tag{6}$$

Setting the stored derivative to zero avoids an update of the learning rate in the next iteration, because the otherwise branch in (3) becomes active. This can be regarded as an implementation trick; the same effect could be achieved by adding a flag to the algorithm. A similar rule for partial weight-backtracking can be found in [24].

We refer to this original algorithm as Rprop$^+$ throughout this article, where the superscript indicates the use of weight-backtracking. The left column of Fig. 1 describes Rprop$^+$ in pseudo-code.

for each w_{ij} **do** {
  if $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} > 0$ **then** {
    $\Delta_{ij}^{(t)} := \min\left(\Delta_{ij}^{(t-1)} \cdot \eta^+, \Delta_{\max}\right)$
    $\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$
    $w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$
  }
  elseif $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$ **then** {
    $\Delta_{ij}^{(t)} := \max\left(\Delta_{ij}^{(t-1)} \cdot \eta^-, \Delta_{\min}\right)$
    $w_{ij}^{(t+1)} := w_{ij}^{(t)} - \Delta w_{ij}^{(t-1)}$
    $\frac{\partial E}{\partial w_{ij}}^{(t)} := 0$
  }
  elseif $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} = 0$ **then** {
    $\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$
    $w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$
  }
}

for each w_{ij} **do** {
  if $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} > 0$ **then** {
    $\Delta_{ij}^{(t)} := \min\left(\Delta_{ij}^{(t-1)} \cdot \eta^+, \Delta_{\max}\right)$
  }
  elseif $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$ **then** {
    $\Delta_{ij}^{(t)} := \max\left(\Delta_{ij}^{(t-1)} \cdot \eta^-, \Delta_{\min}\right)$
  }
  $w_{ij}^{(t+1)} := w_{ij}^{(t)} - \operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$
}

Figure 1: Pseudo-code of one iteration of the Rprop$^+$ algorithm with weight-backtracking [18] (left column) and of the Rprop$^-$ algorithm without weight-backtracking scheme [17] (right column).

2.2 Rprop without Weight-Backtracking

In [1, 17] a different version of the Rprop algorithm is described. The weight-backtracking is omitted and the right hand side of (5) is used in all cases. Hence, there is no need to store the previous weight updates. We denote this version as Rprop$^-$. The right column of Fig. 1 shows the corresponding pseudo-code.

2.3 Improved Rprop with Weight-Backtracking

Our first modification of Rprop is based on the consideration that a change of sign of the partial derivative implies that the algorithm has jumped over a local minimum, but does *not* indicate whether the weight update has caused an increase or a decrease of the error. Thus, the decision made in (6) to undo such a step is somewhat arbitrary. Even worse, it appears to be counterproductive to take back a step though the overall error has decreased. The idea of the modification of Rprop$^+$ is to make the step reversal dependent on the evolution of the error.

There is a reason not to revert a step that has not caused a change to the sign of the corresponding partial derivative: Suppose that the weights influence the network error independently of each other and that the error function has only one local optimum within a hypercube $U \subseteq \mathbb{R}^n$, where n is the number of weights, and $\boldsymbol{w}^{(t)} \in U$. Then every change of $w_i^{(t)}$ in the direction of $-\partial E^{(t)}/\partial w_i^{(t)}$ takes the weight vector closer to the local optimum in U if $\boldsymbol{w}^{(t+1)} \in U$ and $\operatorname{sign}\left(\partial E^{(t)}/\partial w_i^{(t)}\right) =$

$\text{sign}\left(\partial E^{(t+1)}/\partial w_i^{(t+1)}\right)$.

These considerations lead to the rule, that those weight updates that have caused changes to the signs of the corresponding partial derivatives are reverted, but only in case of an error increase: We replace (6) by

$$\begin{aligned}
&\textbf{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \textbf{ then } \{ \\
&\quad \Delta w_{ij}^{(t)} = \begin{cases} -\Delta w_{ij}^{(t-1)} & , \text{ if } E^{(t)} > E^{(t-1)} \\ 0 & , \text{ otherwise} \end{cases} \\
&\quad \frac{\partial E}{\partial w_{ij}}^{(t)} := 0 \\
&\}
\end{aligned} \tag{7}$$

In (7) we combine "individual" information about the error surface (sign of the partial derivative of the error function with respect to a weight) with more "global" information (network error) in order to decide for each weight individually whether or not to revert a step. This combines the strictly "global" approach, where the complete previous update for *all* weights is reversed if $E^{(t)} > \gamma E^{(t-1)}$ ($\gamma = 1.0$ in [22]; $1.0 < \gamma \leq 1.05$ in [25]), with the ideas in [18, 24].

Compared to Rprop$^+$ only one additional variable, the previous error $E^{(t-1)}$, has to be stored. We refer to this modified algorithm as iRprop$^+$ throughout the remainder of this paper. Figure 2 summarizes iRprop$^+$ in pseudo-code.

2.4 Improved Rprop without Weight-Backtracking

In case of a change of sign of the derivative, the iRprop$^+$ algorithm described in the previous section does two things in addition to the reduction of the step-size: First, it performs weight-backtracking in the (few) cases where the overall error increases. Second, it always sets the derivative $\partial E^{(t)}/\partial w_{ij}{}^{(t)}$ to zero. In order to analyze the effects of these two different actions, we came up with the algorithm described in Fig. 3, which is the same as iRprop$^+$, but without weight-backtracking. We denote this algorithm as iRprop$^-$. If the sign of a partial derivative changes, iRprop$^-$ reduces the corresponding step-size but does not modify the corresponding weight. Additionally, it is ensured that in the next step the weight is modified using the reduced step-size. Furthermore, a step-size cannot be altered if it has been reduced in the previous step. The only difference compared to Rprop$^-$ is that the derivative is set to zero.

3 Experimental Evaluation

In this section, we compare the new algorithms with the original Rprop versions, the BFGS algorithm, Quickprop, and the conjugate gradient method (CG) on four neural network benchmark problems. Two classification tasks (the *cancer1* and *diabetes1* data sets, both from the UCI repository of machine learning database, as given in the PROBEN1 benchmark collection [12]) and two regression problems (predicting the sunspots and Lorenz time series) are considered. As models, we employ feed-forward neural networks for classification and the modeling of the sunspots time series, whereas we apply a recurrent neural network for the Lorenz time series prediction.

for each w_{ij} **do** {

if $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} > 0$ **then** {

$\Delta_{ij}^{(t)} := \min\left(\Delta_{ij}^{(t-1)} \cdot \eta^+, \Delta_{\max}\right)$

$\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$

$w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$

}

elseif $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$ **then** {

$\Delta_{ij}^{(t)} := \max\left(\Delta_{ij}^{(t-1)} \cdot \eta^-, \Delta_{\min}\right)$

if $E^{(t)} > E^{(t-1)}$ **then** $w_{ij}^{(t+1)} := w_{ij}^{(t)} - \Delta w_{ij}^{(t-1)}$

$\frac{\partial E}{\partial w_{ij}}^{(t)} := 0$

}

elseif $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} = 0$ **then** {

$\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$

$w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$

}

}

Figure 2: The iRprop$^+$ algorithm with improved weight-backtracking scheme. The proposed algorithm differs only in the highlighted line from the original Rprop$^+$.

In all experiments, we use the same parameters for the four Rprop algorithms. These parameters are set to $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_0 = 0.5$ (the initial value of the Δ_{ij}), $\Delta_{\min} = 0$ and $\Delta_{\max} = 50$ (cf. [18, 12, 17]). The only exception is the training of the recurrent neural network, where we set $\Delta_0 = 0.0125$, see Sec. 3.4.

All feed-forward network architectures used have sigmoidal hidden and linear output units, whereas in the recurrent network all activation functions are sigmoidal. In the following descriptions, the notation 8–2–2–2 stands for a network architecture with 8 inputs, 2 hidden layers with 2 units each and 2 output neurons.

The mean squared error

$$E_{\mathrm{MSE}}(\boldsymbol{w}) = \frac{1}{d \cdot N} \sum_{n=1}^{P} ||y(\boldsymbol{x}^n; \boldsymbol{w}) - \boldsymbol{t}^n||^2 \tag{8}$$

is employed for network training. Here $\boldsymbol{t}^n$ denotes the desired output for a particular input $\boldsymbol{x}^n$ and $y(\boldsymbol{x}^n; \boldsymbol{w})$ the output of the neural network given $\boldsymbol{x}^n$ and the weight vector $\boldsymbol{w}$; $||.||$ is the Euclidean norm. The dimension of the output is denoted by d, the number of training patterns (or the length of time series in the case of the recurrent network) by N. The error percentage [12] defined as $100 \cdot E_{\mathrm{MSE}}(\boldsymbol{w})/(t_{\max} - t_{\min})^2$, where $t_{\max}$ and $t_{\min}$ denote the maximum and minimum target value, respectively, is used to display the results in this section. However, cross-entropy based error functions may be more suitable for classification tasks and work also well in combination with Rprop [8].

We compare the Rprop algorithms with three common optimization techniques,

for each w_{ij} **do** {

  if $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} > 0$ **then** {

    $\Delta_{ij}^{(t)} := \min\left(\Delta_{ij}^{(t-1)} \cdot \eta^+, \Delta_{\max}\right)$

  }

  elseif $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$ **then** {

    $\Delta_{ij}^{(t)} := \max\left(\Delta_{ij}^{(t-1)} \cdot \eta^-, \Delta_{\min}\right)$

    $\frac{\partial E}{\partial w_{ij}}^{(t)} := 0$

  }

  $w_{ij}^{(t+1)} := w_{ij}^{(t)} - \operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$

}

Figure 3: The iRprop$^-$ algorithm without weight-backtracking. The proposed algorithm differs only in one line from Rprop$^-$.

Fahlman's Quickprop (cf. [2, 15]), the BFGS (cf. [14]) algorithm, a quasi-Newton method, which iteratively estimates the inverse of the Hessian, and the conjugate gradient method (cf. [14, 20]).

We employ the nonlinear CG algorithm using the Polak-Ribière method. The search direction is reset to the negative gradient direction whenever a search direction is computed that is not a descent direction, i.e., the scalar product of the actual gradient and the computed search direction is not positive. Different line search methods (zeroth- and first-order) have been tested. An elaborated direct (zeroth-order) line search performed best and is therefore applied throughout this investigation for both CG and BFGS.

The performance of Quickprop strongly depends on the fixed learning rate parameter ϵ (denoted η in [15]). Hence, for every test problem, we first performed 10 trials for each $\epsilon \in \{0.00001, 0.0001, 0.001, 0.1, 0.5, 1, 2, 3, 4, 5\}$. The ϵ value that gave the best results is used in the comparison of the learning algorithms: $\epsilon = 1$ for the *cancer1* task, $\epsilon = 2$ for the *diabetes1* task, $\epsilon = 1$ for the sunspots problem, and $\epsilon = 0.01$ for the Lorenz time series prediction. The maximum growth factor μ is set to 1.75 as recommended in [15].

For a fair comparison, the computational costs for optimization are measured in *propagations* (the computational complexity of the optimization algorithms is neglected): evaluating the error function (e.g., during line search) of a neural network corresponds to one ("forward") propagation, calculating the error and the gradient corresponds to two propagations ("forward" and "backward"). Hence, one iteration of a Rprop algorithm needs two propagations.

All results presented are based on 100 independent trials for each problem. To display the results, the median is preferred to the less robust average. To achieve a fair comparison, the 100 random weight initializations were the same for all the learning algorithms. In order to analyze whether the differences between the error trajectories are significant, every 10 propagations a Wilcoxon rank sum test [26] has been performed (all statements refer to a significance level of 5 %; however, most differences are highly significant). In the following, only the most important results are reported.

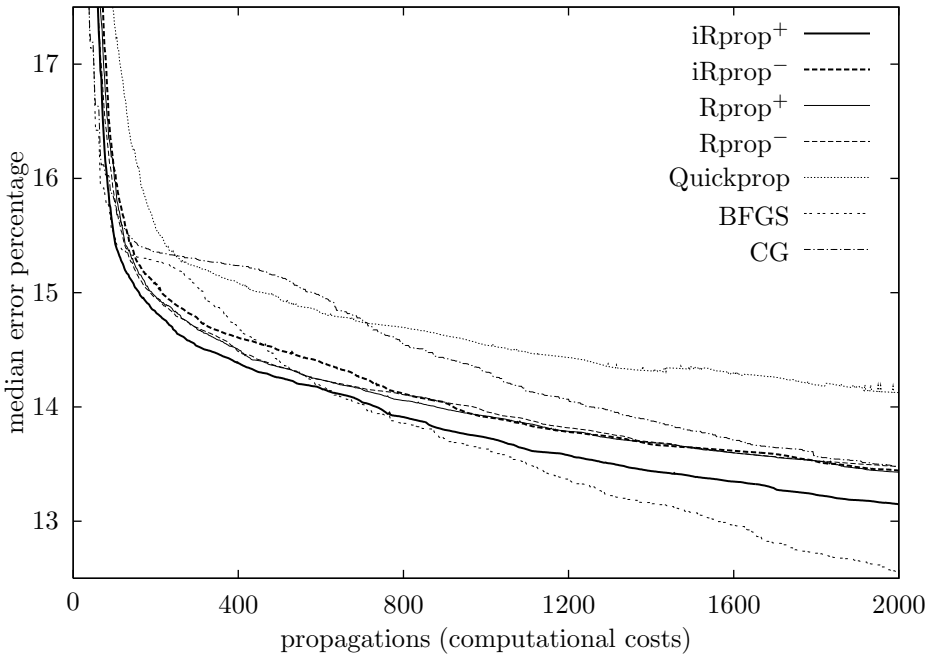

Figure 4: Medians of the training errors for the diabetes task.

3.1 Diabetes Classification

3.1.1 Problem and Model Description

The goal of this real-world classification task is to decide whether a Pima Indian individual is diabetes positive or not. There are 8 inputs representing personal data and results from a medical examination. A 1-of-2 encoding is used for the output, which is a binary attribute that is correlated with the question whether the person suffers from diabetes or not. The training set consists of 384 patterns. Some of the patterns are disturbed by noise: they contain nonsensical zero entries. An 8–2–2–2 feed-forward neural network with all shortcut connections (i.e., all feed-forward connections including those that skip intermediate layers) is used, which gave good results in [12].

3.1.2 Results

The trajectories of the error percentage are shown in Fig. 4. In the diabetes task, iRprop$^+$ performs significantly better than all the other Rprop variants, CG, and—as in all other tasks—Quickprop. It also significantly outperforms the BFGS in the first approx. 600 propagations. After more than 1200 propagations the BFGS shows significantly better results than iRprop$^+$. Behind approx. 100 propagations, Rprop$^+$, Rprop$^-$, and iRprop$^-$ do not differ significantly from each other.

After the (predominantly stochastic) initial phase the BFGS reaches significantly lower training errors than the CG method and Quickprop. This holds for all test problems in this study.

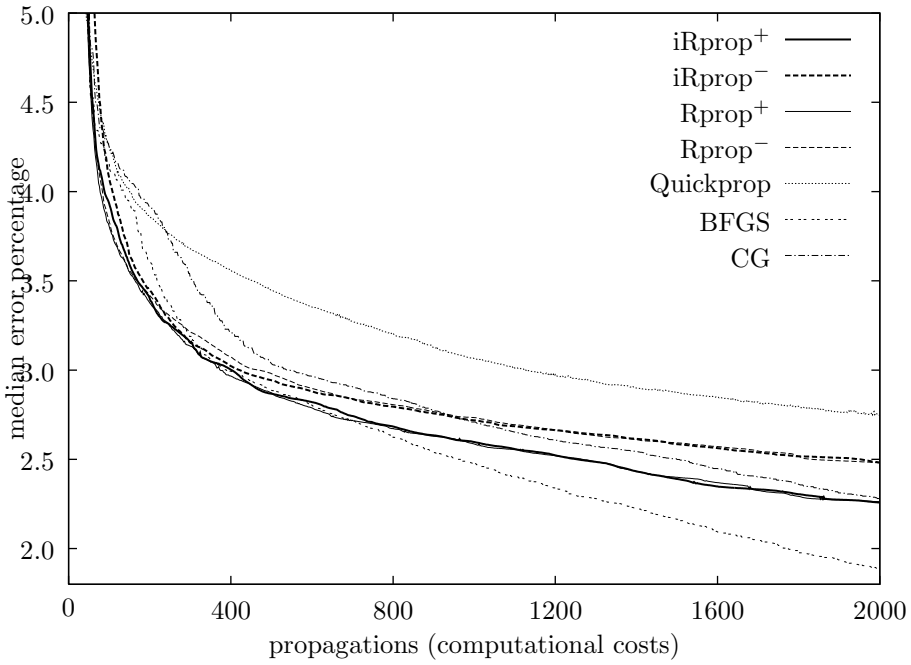

Figure 5: Medians of the training errors for the cancer problem.

3.2 Cancer Classification

3.2.1 Problem and Model Description

The *cancer1* problem [27] is also a real-world classification task: Based on 9 inputs describing a tumor, the task is to classify it as either benign or malignant. The data set consists of 350 patterns. Again a 1-of-2 encoding is used for the output. The architecture is a 9–4–2–2 feed-forward neural network with all shortcut connections, which performed well in [12].

3.2.2 Results

The results are summarized in Fig. 5. The differences between iRprop$^+$ and Rprop$^+$ are not significant. Both algorithms perform significantly better than CG, Quickprop, and the Rprop variants without weight-backtracking[1]. For more than 400 propagations, iRprop$^+$ significantly outperforms BFGS, then the BFGS algorithm gets better. The Rprop variants without weight-backtracking do not differ significantly from each other.

In all the four tasks, iRprop$^+$ performs significantly better than CG with the only exception of the final iterations in this problem, where the differences do not become significant.

3.3 Sunspots

3.3.1 Problem and Model Description

The goal of this regression task is to reproduce the time series of the average number of sunspots observed per year. The data from the time steps $t-1$, $t-2$, $t-4$,

[1]The better performance compared to iRprop$^-$ is in contrast to the findings in [6], where a different initial step size Δ_0 was used.

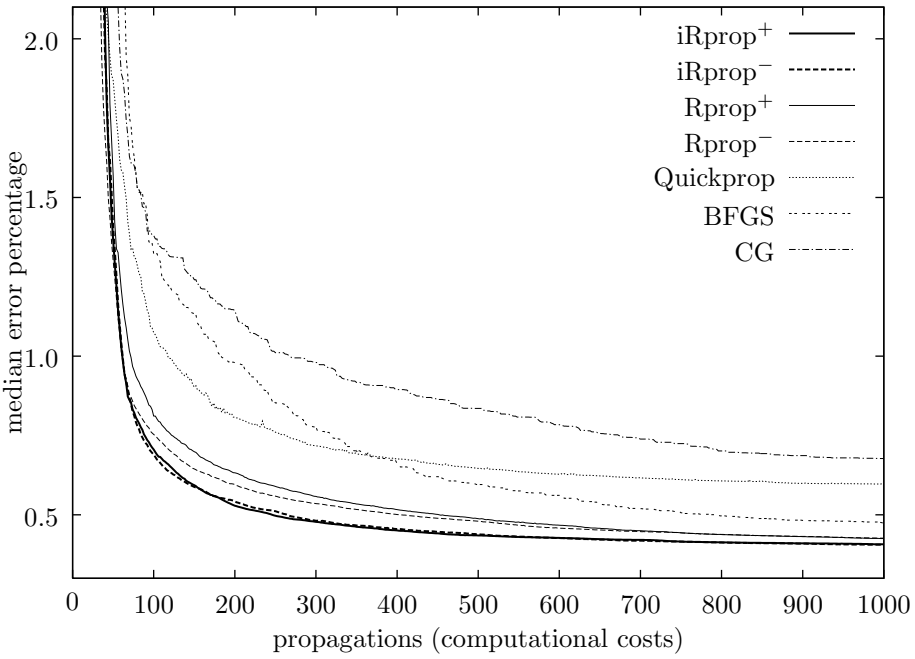

Figure 6: Medians of the training errors for the sunspot problem.

and $t-8$ are used to predict the average number of spots at time t. The input data are normalized between 0.2 and 0.8. 289 patterns are used, the first pattern to predict is from year 1708. A 4–5–1 feed-forward neural network without shortcut connections is used.

3.3.2 Results

iRprop$^+$ and iRprop$^-$ perform better than the other learning schemes, see Fig. 6 for the trajectories of the error percentage. The better performance of iRprop$^+$ compared to iRprop$^-$ is not significant most of the time.

3.4 Lorenz Time Series

3.4.1 Problem and Model Description

The task is the modeling of the Lorenz attractor [10], which is defined by the three coupled differential equations

$$\begin{aligned} \dot{x}(t) &= \sigma\,(y(t) - x(t)) \\ \dot{y}(t) &= -x(t)\,z(t) + r\,x(t) - y(t) \\ \dot{z}(t) &= x(t)\,y(t) - b\,z(t) \quad . \end{aligned} \tag{9}$$

For the chosen parameter values $\sigma = 16$, $r = 45.92$ and $b = 4$ the time series shows chaotic behavior. We solved these equations by means of a fourth order Runge-Kutta method with time step $0.01 = \Delta t/2$. The task during the presentation of the training sequence was to predict one time step ahead, only using the x coordinate of the data: $x(t) \to x(t + \Delta t)$. We generated a training sequence of 2020 data points, including 20 data points for the warm up of the internal states; these 20 points were not used for the calculation of the training error.

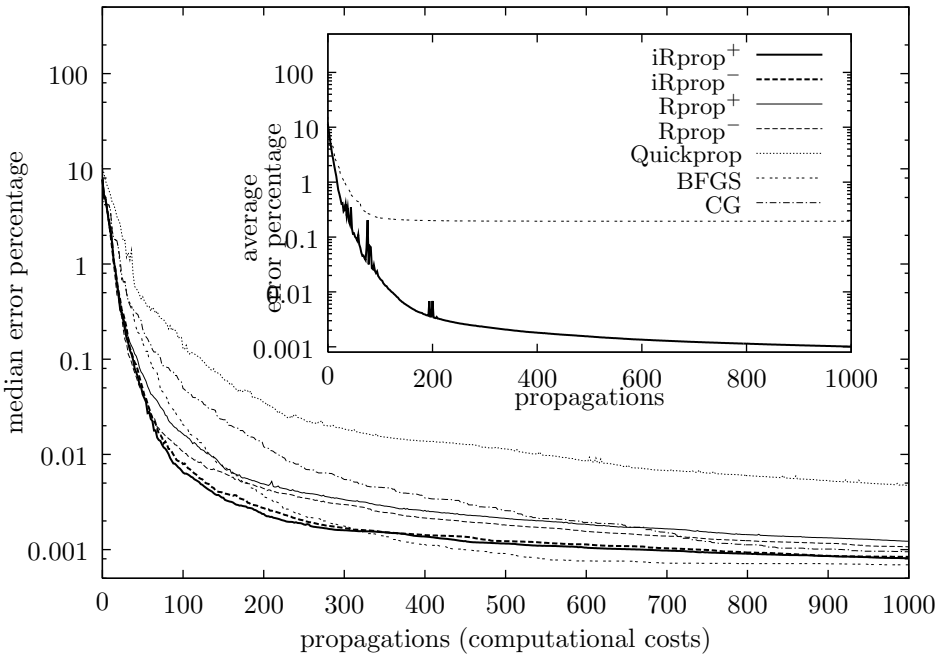

Figure 7: Training errors for the Lorenz time series modeling task on logarithmic scale. The outer plot shows the medians of the error percentages, whereas the inset plot shows the average for iRprop$^+$ and the BFGS algorithm..

For this task, a recurrent neural network is chosen. More precisely, we use an extended Elman network [23] with a single memory layer. For all activation functions the hyperbolic tangent is employed. Hence, input and output data are normalized to lie between -0.7 and 0.7. Because the weights in recurrent neural networks are more sensitive than in feed-forward structures, a smaller initial Δ_0 (0.0125) is chosen.

3.4.2 Results

The statistics of the error percentage for 100 trials are shown in Fig. 7. iRprop$^+$ and iRprop$^-$ show better performance than Rprop$^+$, Rprop$^-$, and CG. Before the approx. 200$^{\text{th}}$ propagation, iRprop$^+$ and iRprop$^-$ perform significantly better, after approx. 400 propagations significantly worse than the BFGS. As the trajectories of the averaged error percentage—see inset plot in Fig. 7—behave differently, we can conclude that there are some outliers, e.g., trials where algorithms get stuck in bad local optima, that dominate the average errors. This lack of robustness can be interpreted as a drawback of the BFGS algorithm. The discrepancy between median and average does not occur that clearly in the three feed-forward neural network learning tasks reflecting that learning is more robust in the feed-forward than in the recurrent networks.

3.5 Notes on Generalization

A common issue addressed when discussing neural network learning algorithms is generalization, i.e., the ability of the trained network to process previously unseen data in a desired way. In this study we restrict our analysis of the algorithms to their performance on the presented training set, because to our minds generaliza-

tion is mainly a problem of an adequate formulation of the learning task, whereas the learning algorithm's concern is the efficient optimization on the presented error surface. Nevertheless, there may be a tradeoff between learning speed and generalization *if* generalization is judged by means of an extra validation data set (cf. [15], p. 153). To see this, let us assume an optimal learning algorithm in terms of learning speed and an error function that considers solely the performance on a training data set. The algorithm would find a weight configuration that results in the minimal training error in a single step. Obviously, this solution is not likely to generalize well. The weight configurations that correspond to networks that produce a low error on a validation set are usually found by evaluating the solutions along the trajectory in the weight space the learning algorithm generates. With increasing speed of the algorithm, the number of intermediate steps decreases, and therefore so does the chance of evaluating a weight configuration that shows good results on the validation data set. This problem does not occur if generalization is achieved by incorporating a regularization term into the error function.

With regard to the new algorithms, it has been shown in [6] that the improved Rprop algorithms does not seem to converge to different minima than the original ones—they only converge faster.

3.6 Discussion

In the previous paragraphs, we have compared the new variants of Rprop with the two original ones and three widely spread optimization techniques, on three real world neural network learning problems, plus one realistic one [13]. Basically, the outcomes of all of our experiments are comparable: iRprop$^+$ has turned out to be superior in the initial phase of learning, whereas in some experiments BFGS performs better in a later phase. In all test problems Quickprop and the CG method perform rather badly. The iRprop$^-$ algorithm appears to be worse than iRprop$^+$, but it is very compact and needs less memory; it performs better than the comparable Rprop$^-$.

In the later phase of learning, the BFGS might benfit from two facts: the estimation of the Hessian and the implicit assumption of a quadratic error surface become more accurate after a while. However, one has to keep two important things in mind: First, BFGS has quadratic time and space complexity, which adversely affects the application to large neural networks. Second, if generalization is judged by means of extra data sets, the later phase of learning is likely to correspond to already *overfitted* models, because networks that generalize well are usually found much earlier in the learning process [6, 12]. Therefore, the powerful behavior of BFGS may not be relevant in certain scenarios.

4 Rprop on Artificial Test Functions

In order to get more general results and ideas as to why our modifications increase the learning speed, we analyze the behavior of the Rprop algorithms on artificial error surfaces.

4.1 Method

In the vicinity of an optimum the error surface can be approximated by a hyperparaboloid, i.e., the contours of constant error are elliptical. Let $\boldsymbol{w}$ denote the n-dimensional vector of parameters to be optimized, e.g., n weights of a neural

network. Then the general hyperparabolic error surface is given by

$$E_a(\boldsymbol{w}) = \sum_{i=1}^{n} \left(a^{\frac{i-1}{n-1}} \langle \boldsymbol{w}, \boldsymbol{o}_i \rangle \right)^2, \tag{10}$$

where $\langle ., . \rangle$ denotes the scalar (dot) product. The vectors $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n \in \mathbb{R}^n$ form a normalized orthogonal basis with random orientation, i.e., for each $1 \leq i, j \leq n$ they satisfy $\langle \boldsymbol{o}_i, \boldsymbol{o}_j \rangle = 0$ if $i \neq j$ and $||\boldsymbol{o}_i|| = 1$. The parameter a controls the condition of the problem. This test function is a generalization of the artificial error surface proposed in [21]. It is used in [3, 4] for analyzing the local search properties of evolutionary algorithms.

Rprop is not invariant under rotation of the coordinate system; its performance strongly depends on the choice of $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n$ (not rotating the coordinate system results in an unrealistic special case, where the weights in the neural network influence the error independently of each other). In our experiments, we use a different basis in each trial. The *Gram-Schmidt orthogonalization procedure* is used to construct $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n$ from randomly drawn basis vectors.

The parameter a in Eq. (10) corresponds to the relation between the longest and the shortest axis of the elliptical contours; it is equal to the square root of the condition of the problem defined as the ratio between the largest and the smallest eigenvalue of the Hessian of $E_a(\boldsymbol{w})$. According to [3], a choice of $a = 10^3$ is reasonable for real world optimization problems. An idea about the magnitude of a for neural networks comes from the analysis of the Hessian matrix. It is argued that a typical Hessian has few small, many medium, and few very large eigenvalues [9]. This leads to a ratio between the longest and the shortest axis that is much larger than 10^3. As the Rprop algorithms depend only on the sign of the derivative and the ranking of the error values, the results obtained with test functions generated by Eq. (10) do not only hold for the parabolic error surface, but also for any monotone increasing transformation.

In the following, we compare the four Rprop algorithms on the hyperparabolic error surfaces with different values for a. We perform 100 trials per setting and Rprop variant. The randomly chosen initial values of $\boldsymbol{w}$ and the basis vectors are the same for each Rprop, but different for each trial. During optimization, we computed characteristic features of the steps on the error surface for subsequent analysis [5].

4.2 Results

The performance of the four Rprop algorithms depends on the condition of the test function, but only for very small and unrealistic values of a ($a \lesssim 3$) the original methods converge faster than the modified algorithms. The larger a is the more iRprop$^-$ and iRprop$^+$ outperform the original algorithms as shown in [6] for dimension $n = 100$. However, the qualitative behavior appears to be independent of the problem's dimension. See Tab. 1 for the results for $n = 2$, where for different values of a the number of steps required for the error to fall below 10^{-6} are given. We can conclude that the proposed modifications improve the local optimization properties of Rprop for all relevant scenarios.

Table 1 shows interesting properties of the (improved) Rprop algorithms. We measured the probability of improvement (PI, the percentage of steps that lead to an error decrease) the probability of worsening (PW), the average improvement (AI, the average amount of an error decrease), the average worsening (AW, the average amount of an error increase), and the amount of the average change of the error ΔE. Furthermore, we calculated the average step-size $||\widehat{|\boldsymbol{\Delta}|}||$, where we define

$||\boldsymbol{\Delta}|| := \sqrt{\sum_{ij} \Delta_{ij}^2}$. For the considered parameterization of the error function and the algorithms, we have:

1. For large values of a, the new algorithms are about 2.5 times faster than the original ones.

2. The new Rprop versions do not necessarily have a higher fraction of beneficial iterations, but take larger steps. As indicated by the average improvement (AI), this leads to larger steps towards the optimum (the also larger deteriorations can be neglected).

3. For large values of a, AI, AW, PI, PW, and therefore ΔE appear to be independent of a. It is striking that the number of steps are approximately proportional to a^2, i.e., the condition of the problem: The dominating term in the error function $E_2(\boldsymbol{w})$ is $(a \langle \boldsymbol{w}, \boldsymbol{o}_2 \rangle)^2$. Thus the error function scales with a^2. As the initial $\boldsymbol{w}^{(0)}$ have been chosen independently of a and the error threshold to be reached is kept constant, this scaling of the error function determines the number of steps needed.

As the step-sizes are an important factor, the question arises how the choice of η^+ and η^- influences the results. Is the improved learning speed a mere consequence of the parameterization, i.e., do the original versions perform as well as the improved Rprops when choosing different values for η^+ and η^-? To answer these questions, we performed 100 trials for each of the following settings with condition $a = 10$ and dimension $n = 2$. We fixed η^- to the standard value of 0.5, and varied $\eta^+ \in \{1.05, 1.1, 1.15, \ldots, 1.6\}$, i.e., even beyond the sensible choices [16]. In a second set of experiments, we fixed η^+ to the standard value of 1.2 and varied $\eta^- \in \{0.4, 0.45, 0.5, \ldots, 0.8\}$.

In all experiments, the new algorithms clearly outperformed the original ones. To our knowledge, there has been no investigation of the Rprop performance for $\eta^- \neq 0.5$. However, the best results were achieved using the parameters $\eta^- = 0.7$ and the commonly used $\eta^+ = 1.2$.

5 Conclusions

The Rprop algorithm is one of the best performing first-order learning algorithms for neural networks with arbitrary topology. As experimentally shown, its learning speed can be significantly improved by small modifications without increasing the complexity of the algorithm. The new methods, in particular iRprop$^+$, perform better than the original algorithms on all of the neural network test problems, including feed-forward and recurrent models and real world and realistic data sets. Moreover, iRprop$^+$ outperforms the efficient and widely spread BFGS, Quickprop, and CG algorithms in the relevant domain. The experiments confirm the robustness of the Rprop learning methods with respect to their internal parameters.

Our analysis of the local search ability of the different versions of Rprop on a hyperparabolic error surface, i.e., a surface comparable to the vicinity of a minimum, gives a deeper insight into the differences and common features between the different algorithms. These investigations show that the behavior of Rprop depends on the condition of the surface; for realistic settings, the improved algorithms outperform the original ones. Moreover, one can see that the modifications enable Rprop not to make *more* successful steps, i.e., steps decreasing the error, but *longer* ones, i.e., steps with larger step-sizes.

Based on these results, we conclude that the improved Rprop with weight-backtracking iRprop$^+$ should be considered as the first choice for first-order batch learning of (large) neural networks.

Acknowledgments

The authors acknowledge support by the BMBF under grant LEONET 01 IB 802 C4. We would like to thank Arne Heizmann, Marc Toussaint, and the two anonymous reviewers for their helpful suggestions.

References

[1] H. Braun. *Neuronale Netze: Optimierung durch Lernen und Evolution.* Springer-Verlag, 1997.

[2] S. E. Fahlman. Faster-learning variations of back-propagation: An empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, San Mateo, 1988. Morgan Kaufmann.

[3] N. Hansen and A. Ostermeier. Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The $(\mu/\mu, \lambda)$-CMA-ES. In *EUFIT'97, 5th European Congress on Intelligent Techniques and Soft Computing*, pages 650–654, Aachen, 1997. Verlag Mainz, Wissenschaftsverlag.

[4] N. Hansen, A. Ostermeier, and A. Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 57–64, Pittsburgh, 1995. Morgan Kaufmann.

[5] C. Igel and K. Chellapilla. Fitness distributions: Tools for designing efficient evolutionary computations. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming*, volume 3, chapter 9, pages 191–216. MIT Press, 1999.

[6] C. Igel and M. Hüsken. Improving the Rprop learning algorithm. In H. Bothe and R. Rojas, editors, *Proceedings of the Second International Symposium on Neural Computation, NC 2000*, pages 115–121. ICSC Academic Press, 2000.

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

[8] M. Joost and W. Schiffmann. Speeding up backpropagation algorithms by using cross–entropy combined with pattern normalization. *International Journal of Uncertainty, Fuzziness and Knowledge–based Systems*, 6(2):117–126, 1998.

[9] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G. B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, number 1524 in LNCS, chapter 1, pages 9–50. Springer-Verlag, 1998.

[10] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963.

[11] L. M. Patnaik and K. Rajan. Target detection through image processing and resilient propagation algorithms. *Neurocomputing*, 35(1-4):123–135, 2000.

[12] L. Prechelt. PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, 1994.

[13] L. Prechelt. A quantitative study of experimental evaluations of neural network learning algorithms: Current research practice. *Neural Networks*, 9(3):457–462, 1996.

[14] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1994.

[15] R. D. Reed and R. J. Marks II. *Neural Smithing*. MIT Press, 1999.

[16] M. Riedmiller. Untersuchungen zu Konvergenz und Generalisierungsfähigkeit überwachter Lernverfahren im SNNS. In A. Zell, editor, *Workshop SNNS-93: Simulation Neuronaler Netze mit SNNS*, Stuttgart, 1993. Universität Stuttgart, Fakultät Informatik.

[17] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons – from backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces*, 16(5):265–278, 1994.

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

[19] W. Schiffmann, M. Joost, and R. Werner. Comparison of optimized backpropagation algorithms. In Verleysen, editor, *Proceedings of the European Symposium on Artificial Neural Networks, ESANN '93*, pages 97–104, Brussels, 1993.

[20] J. R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, $1\frac{1}{4}$. edition, 1994.

[21] F. M. Silva and L. B. Almeida. Acceleration techniques for the backpropagation algorithm. In L. B. Almeida and C. J. Wellekens, editors, *Neural Networks – EURASIP Workshop 1990*, number 412 in LNCS, pages 110–119. Springer-Verlag, 1990.

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

[23] P. Stagge and B. Sendhoff. An extended Elman net for modeling time series. In W. Gerstner, editor, *International Conference on Artificial Neural Networks*, LNCS. Springer-Verlag, 1997.

[24] T. Tollenaere. Supersab: Fast adaptive backpropagation with good scaling properties. *Neural Networks*, 3:561–573, 1990.

[25] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon. Accelerating the convergence of the backpropagation method. *Biological Cybernetics*, 59:257–263, 1988.

[26] F. Wilcoxon. Individual comparison by ranking methods. *Biometrics Bulletin*, 1:80–83, 1945.

[27] W. H. Wolberg and O. L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proc. Natl. Acad. Sci. USA*, 87:9188–9192, 1990.

Table 1: Results for two-dimensional artificial error surfaces, averaged over 100 trials per setup. The parameter a controls the condition of the problem. The number of steps refers to the iterations required for the error to fall below 10^{-6}. In addition, the probability of improvement (PI), the probability of worsening (PW), average improvement (AI, i.e., the average amount of an error decrease), the average worsening (AW, i.e., the average amount of an error increase), the amount of the average change of the error ΔE, and the average step-size $||\widehat{\boldsymbol{\Delta}}||$ are given.

	a	steps	PI	PW	AI	AW	ΔE	$\|\|\widehat{\boldsymbol{\Delta}}\|\|$
Rprop$^+$	3	46.7	0.809	0.191	0.109	0.00966	0.0863	0.0338
Rprop$^-$	3	36.5	0.897	0.103	0.124	0.0114	0.11	0.0376
iRprop$^-$	3	42.7	0.768	0.137	0.124	0.00742	0.0943	0.0361
iRprop$^+$	3	38.3	0.82	0.148	0.13	0.00758	0.105	0.0373
Rprop$^+$	4	64	0.835	0.165	0.131	0.0143	0.107	0.0254
Rprop$^-$	4	52.9	0.931	0.0686	0.141	0.0218	0.13	0.0267
iRprop$^-$	4	48.5	0.791	0.122	0.181	0.0143	0.142	0.0323
iRprop$^+$	4	42.3	0.838	0.134	0.196	0.0134	0.162	0.0343
Rprop$^+$	5	84.7	0.855	0.145	0.149	0.0184	0.124	0.0198
Rprop$^-$	5	74.8	0.942	0.0578	0.151	0.0287	0.141	0.0196
iRprop$^-$	5	55.1	0.809	0.107	0.239	0.0236	0.191	0.0293
iRprop$^+$	5	50.4	0.849	0.126	0.249	0.0209	0.209	0.0295
Rprop$^+$	10	272	0.9	0.0999	0.171	0.0393	0.15	0.0066
Rprop$^-$	10	264	0.977	0.023	0.161	0.0983	0.155	0.00583
iRprop$^-$	10	123	0.886	0.0599	0.381	0.0839	0.333	0.0136
iRprop$^+$	10	125	0.918	0.0693	0.363	0.0858	0.327	0.0126
Rprop$^+$	50	$6.24{\cdot}10^3$	0.933	0.0665	0.179	0.0674	0.163	0.000307
Rprop$^-$	50	$6.61{\cdot}10^3$	0.99	0.0105	0.158	0.231	0.154	0.000246
iRprop$^-$	50	$2.73{\cdot}10^3$	0.953	0.0263	0.396	0.224	0.372	0.000636
iRprop$^+$	50	$2.68{\cdot}10^3$	0.971	0.0263	0.398	0.327	0.378	0.000625
Rprop$^+$	100	$2.48{\cdot}10^4$	0.935	0.0649	0.18	0.0699	0.164	$7.8{\cdot}10^{-5}$
Rprop$^-$	100	$2.63{\cdot}10^4$	0.99	0.0104	0.158	0.237	0.154	$6.23{\cdot}10^{-5}$
iRprop$^-$	100	$1.07{\cdot}10^4$	0.957	0.0249	0.403	0.243	0.379	0.000162
iRprop$^+$	100	$1.06{\cdot}10^4$	0.973	0.0241	0.401	0.366	0.382	0.000159
Rprop$^+$	500	$6.19{\cdot}10^5$	0.935	0.065	0.18	0.07	0.164	$3.14{\cdot}10^{-6}$
Rprop$^-$	500	$6.58{\cdot}10^5$	0.99	0.00988	0.158	0.249	0.154	$2.5{\cdot}10^{-6}$
iRprop$^-$	500	$2.65{\cdot}10^5$	0.959	0.0238	0.406	0.257	0.383	$6.59{\cdot}10^{-6}$
iRprop$^+$	500	$2.65{\cdot}10^5$	0.975	0.0229	0.402	0.387	0.383	$6.39{\cdot}10^{-6}$
Rprop$^+$	1000	$2.48{\cdot}10^6$	0.935	0.0648	0.18	0.0701	0.164	$7.86{\cdot}10^{-7}$
Rprop$^-$	1000	$2.63{\cdot}10^6$	0.99	0.00988	0.158	0.249	0.154	$6.25{\cdot}10^{-7}$
iRprop$^-$	1000	$1.06{\cdot}10^6$	0.959	0.0237	0.406	0.258	0.384	$1.64{\cdot}10^{-6}$
iRprop$^+$	1000	$1.06{\cdot}10^6$	0.975	0.0229	0.401	0.384	0.382	$1.59{\cdot}10^{-6}$