Published in:
Proceedings of the Second International Symposium on Neural Computation, NC'2000, pp. 115–121, ICSC Academic Press, 2000

Improving the Rprop Learning Algorithm

Christian Igel and Michael Hüsken*

Institut für Neuroinformatik
Ruhr-Universität Bochum, 44780 Bochum, Germany
{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 introduce modifications of the algorithm that improve its learning speed. The resulting speedup is experimentally shown for a set of neural network learning tasks as well as for artificial error surfaces.

Keywords: supervised learning, individual step-size adaptation, weight-backtracking, Rprop

1 Introduction

Gradient descent techniques are the most widely used class of algorithms for supervised learning in neural networks. 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. The best of these techniques known to the authors in terms of convergence speed, accuracy and robustness with respect to its parameters is the Rprop (**r**esilient back**prop**agation) algorithm. The reader is referred to [6, 12, 13, 15] for comparisons of Rprop with other supervised learning techniques and to the review of learning methods in [11]. We would like to stress that the algorithm is a general method for gradient based optimization; in particular, it does not rely on special properties of a certain class of network topologies. Further, its memory requirements scale only linearly with the number of free parameters to optimize.

A common and quite general method for improving network training is weight-backtracking. Weight-backtracking means retracting a previous weight update for some or all weights, cf. [19, 17, 18, 13]. Whether to take back a step or not is decided by means of a heuristic. Rprop has been proposed in literature with and without backtracking. In this paper, we suggest modifications of both versions and experimentally compare the two new methods and the original Rprop algorithms. We make use of artificial test functions and a set of learning problems for feed-forward neural networks.

In the next section, we describe the Rprop algorithm as proposed by Riedmiller and Braun [13, 12]. In Sec. 3, we introduce our modifications and in Sec. 4, we show the improved learning speed by means of different learning tasks. The article ends with a conclusion.

2 The Rprop Algorithm

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 input; superscripts indicate the learning epoch (iteration).

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, i.e. the update amount of a weight, 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 derivative. The benefits of this update scheme are described in [13, 12].

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 algorithms employed in this study. For each weight w_{ij} an individual step-size Δ_{ij} is adjusted using the

*This work was supported from the BMBF under grant LEONET 01IB802C4.

following rule:

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

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. [5, 18]). The step-sizes are bounded by the parameters $\Delta_{\min}$ and $\Delta_{\max}$. In the following, we describe the second part of the algorithm, the update of the weights, for the different Rprop versions.

2.1 Rprop with Weight-Backtracking

After adjusting the step-sizes, 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:

$$\textbf{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} \geq 0 \textbf{ then}$$
$$\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)} \ , \tag{2}$$

where the sign operator returns +1 if its argument is positive, −1 if the argument is negative, and 0 otherwise. In case of a change of sign of the partial derivative, the previous weight update is reverted:

$$\textbf{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \textbf{ then}$$
$$\Delta w_{ij}^{(t)} := -\Delta w_{ij}^{(t-1)} \textbf{ and } \frac{\partial E}{\partial w_{ij}}^{(t)} := 0 \ . \tag{3}$$

Setting the stored derivative to zero avoids an update of the learning rate in the next iteration, because the else branch in (1) becomes active. This can be regarded as an implementation trick; the same effect could be reached by adding a flag to the algorithm. A similar rule for partial weight-backtracking can be found in [18]. Finally, the new weights are given by

$$w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)} \ . \tag{4}$$

We refer to this original algorithm as Rprop$^+$ throughout this article, where the superscript indicates the use of weight-backtracking. Algorithm 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**
    $\Delta w_{ij}^{(t)} := -\operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$
    $w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$
  fi
od

Algorithm 1: The Rprop$^+$ algorithm with weight-backtracking as introduced in [13].

2.2 Rprop without Weight-Backtracking

In [1, 12] a different version of the Rprop algorithm is described. The weight-backtracking is omitted and the right hand side of (2) is used in all cases. Hence, there is no need to store the previous weight updates. We denote this version as Rprop$^-$. Algorithm 2 shows the corresponding 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**
    $\Delta_{ij}^{(t)} := \min\left(\Delta_{ij}^{(t-1)} \cdot \eta^+, \Delta_{\max}\right)$
  elseif $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$ **then**
    $\Delta_{ij}^{(t)} := \max\left(\Delta_{ij}^{(t-1)} \cdot \eta^-, \Delta_{\min}\right)$
  fi
  $w_{ij}^{(t+1)} := w_{ij}^{(t)} - \operatorname{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$
od

Algorithm 2: The Rprop$^-$ algorithm without weight-backtracking scheme as introduced in [12].

3 Modifications

3.1 A New Weight-Backtracking Scheme

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 decrease of the error. Thus, the decision made in (3) 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.

Suppose that the network is close to a (local) optimum, so that locally the error surface can be approximately regarded as quadratic. If the algorithm does not leave the basin of attraction of the optimum, i.e. the step-sizes are adapted to sufficiently small values, then each weight update not leading to a change of sign of the corresponding partial derivative is always a step taking the weight closer to its value at the optimum. This fact leads to the rule to revert only weight updates that have caused changes of the corresponding partial derivatives in case of an error increase. Hence, we replace (3) by

$$\textbf{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \textbf{ and } E^{(t)} > E^{(t-1)} \quad \textbf{then } \Delta w_{ij}^{(t)} := -\Delta w_{ij}^{(t-1)} \ . \tag{5}$$

In (5) 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) to make the decision whether to revert a step or not for each weight individually. This combines the strictly 'global' approach, where the complete previous update for *all* weights is reversed if $E^{(t)} > \gamma E^{(t-1)}$ ($\gamma = 1.0$ [17]; $1. < \gamma \leq 1.05$ [19]), with the ideas in [13, 18]. The adaptation of the step-sizes is still independent of an error decrease or increase.

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. Algorithm 3 summarizes iRprop$^+$ 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**

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

    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**

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

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

  fi

od

Algorithm 3: The iRprop$^+$ algorithm with improved weight-backtracking scheme. The proposed algorithm differs only in one line from the original Rprop$^+$.

3.2 A New 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 increased. Second, it always sets the derivative $\frac{\partial E}{\partial w_{ij}}^{(t)}$ to zero. In order to analyze the effects of these two different actions, we came up with Algorithm 4, which is the same as iRprop$^+$ without weight-backtracking. We denote this algorithm as iRprop$^-$. The only difference between Rprop$^-$ and iRprop$^-$ is that the derivative is set to zero.

If the sign of a partial derivative changes, iRprop$^-$ reduces the corresponding step-size and does not modify the corresponding weight. Additionally, it is ensured that in the next step the weight is modified using the reduced step-size, but the actual gradient direction.

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**

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

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

fi

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

od

Algorithm 4: The iRprop^{-} algorithm without weight-backtracking. The proposed algorithm differs only in one line from Rprop^{-}.

4 Experimental Evaluation

In order to compare the four algorithms, we use artificial test functions with known properies. Additionally, we compare the performance of the algorithms on feed-forward neural network learning problems.

In all the experiments, we use the same parameters for all four algorithms, $\eta^{+} = 1.2$, $\eta^{-} = 0.5$, $\Delta_0 = 0.0125$ (the initial value of the Δ_{ij}), $\Delta_{\min} = 0$ and $\Delta_{\max} = 50$, see [13, 10, 12].

All presented results are averaged over 1000 independent trials for each problem, where the four learning algorithms started from the same (weight) initializations.

4.1 Artificial Test Functions

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 optimize, e.g. n weights of a neural network. Then the hyperparabolic error surface is given by

$$E_a(\boldsymbol{w}) = \sum_{i=1}^{n} \left(a^{\frac{i-1}{n-1}} \langle \boldsymbol{w}, \boldsymbol{o}_i \rangle \right)^2 , \qquad (6)$$

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 $1 \leq i, j \leq n$ it holds $\langle \boldsymbol{o}_i, \boldsymbol{o}_j \rangle = 0$ if $i \neq j$ and $||\boldsymbol{o}_i|| = 1$. This test function is a generalization of the artificial error surface proposed in [16]. It is used in [3, 2] for analyzing the local search properties of evolutionary algorithms.

Rprop is not invariant against 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 a not realistic special case, where the weights in the neural network influence the error independent 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 corresponds to the relation between longest and shortest axis of the elliptical contours; it is equal to the square root of the conditioning of the problem defined as the ratio between largest and smallest eigenvalue of the Hessian of Eq. (6). According to [2], 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 [8]. This leads to a ratio between longest and shortest axis much larger than 10^3.

As the Rprop algorithms (except iRprop^{+}) depend only on the sign of the derivative, results obtained with test functions generated by Eq. (6) do not only hold for parabolic error surfaces, but for a larger class of basins of attraction.

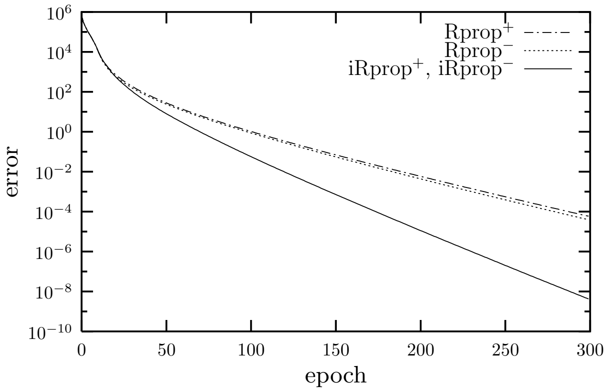

Figure 1: Averaged error curves for E_{10}, the iRprop^{-} and iRprop^{+} trajectories coincide. The modified algorithms perform significantly better (Wilcoxon rank sum test in the last epoch, $p < .001$).

We found the performance of the four algorithms to be dependent on the conditioning of the test functions. However, only for very small and unrealistic values of a ($a \lesssim 3$) the original methods converge faster than the modified algorithms. The larger a the more outperform iRprop$^-$ and iRprop$^+$ the original algorithms. Figure 1 shows the results for $a = 10$ and a problem dimension of 100 as an example. We can conclude that the proposed modifications improve the local optimization properties of the Rprop algorithms for all relevant scenarios.

4.2 Neural Network Benchmark Problems

The first two test problems are classification tasks taken from the PROBEN1 benchmark collection [10], namely the *diabetes1* and *cancer1* training (test) data sets with 384 (193) and 351 (175) samples, respectively. We use the same network architectures as the best networks in [10], see Tab. 1. A 8–2–2–2 topology means 8 inputs, two hidden layers with two units each and 2 output neurons. The output units were linear and the hidden units sigmoid-like. The networks had all possible feed-forward connections, including all shortcut connections. The sum-of-squares error function was employed for network training as in [10], but cross-entropy based error functions may be more suitable for classification tasks and work also well in combination with Rprop [6].

The third problem is the *T-C* classification problem inspired by [14] as described in [1], where the goal is to distinguish between five pixel 'T' and 'C' patterns on a 4×4 pixel grid, see also [9]. The complete set of possible patterns is used for training.

As we are concerned with the approximation of solutions of differential equations (DEs) by neural networks (see [4]), we employed the Rprop algorithm for this task. As an example, we present *problem3* and *problem5*, both described in [7]. The first one is a second order ordinary DE, the latter is a two dimensional second order partial DE, both with boundary conditions. The basic approach, the network structure (fully connected layers without shortcuts, sigmoidal hidden units and linear output units) and the error measure were the same as in [7, 4].

Figures 2 to 6 show the averaged learning trajectories. For the classification tasks taken from the PROBEN1 benchmark collection, the error percentage as defined in [10] is given. Table 1 summarizes the averaged final training error and the minimum error reached by each learning method.

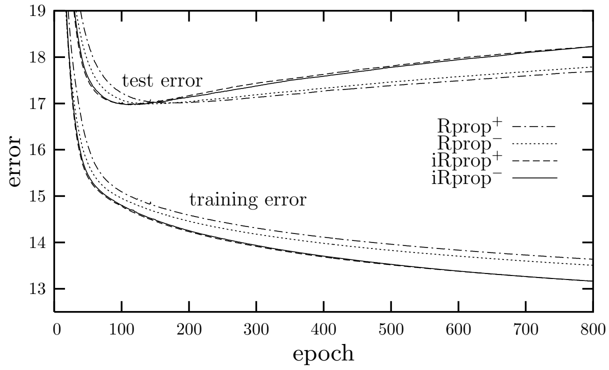

Figure 2: Averaged error percentage for the diabetes problem. The iRprop$^-$ and iRprop$^+$ trajectories can hardly be distinguished.

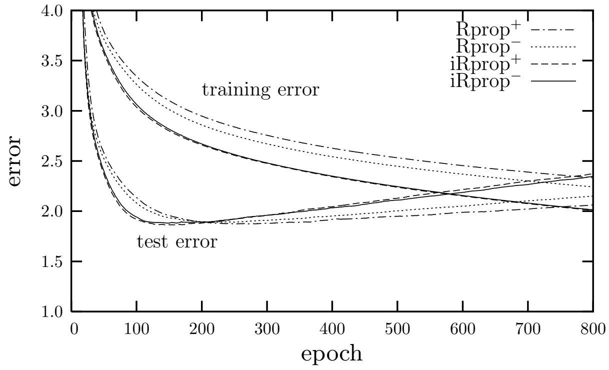

Figure 3: Averaged error percentage for the cancer problem. The iRprop$^-$ and iRprop$^+$ trajectories can hardly be distinguished. The fact that the training error is larger than the test error is characteristic for this test problem.

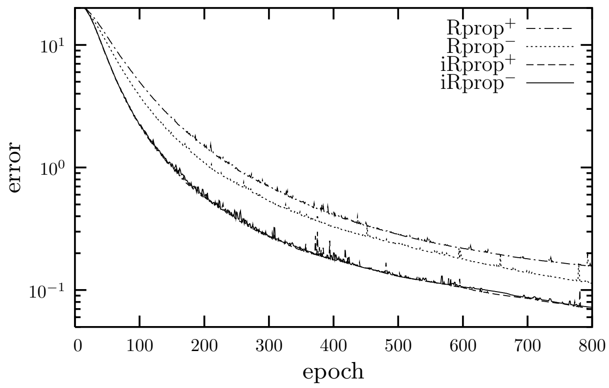

Figure 4: Averaged error percentage for the *T-C* problem. The iRprop$^-$ and iRprop$^+$ trajectories can hardly be distinguished.

The new algorithms learned faster in all experiments. iRprop$^+$ utilizing the new backtracking scheme performs slightly better than iRprop$^-$. The averaged errors between the new and the original learning algorithms in the last epochs differ significantly (Wilcoxon rank sum test, $p < .001$). Also in the early stages of the learning processes the new methods are superior. The Rprop$^+$ with the standard weight-backtracking scheme often performs worse than Rprop$^-$ without weight reversals in our experiments. This explains why the original backtracking was removed from the Rprop algorithm by the inventors.

This investigation focuses on improving the process of minimizing the training error. However, in practice the learning algorithms have to be used in conjunction with mechanisms that ensure the network's ability to generalize, i.e. to predict the correct outputs for input patterns not previously used by the learning algorithm. Overfitting is neither a problem in case of the *T-C* task, where the complete data set is presented, nor in the case of the DE examples, where overfitting is not observed. For the two other problems, we have additionally calculated the error on a test set not involved in network training. The results in Figs. 2 and 3 show that the modified algorithms do not converge to (local) optima with worse generalization properties than the other methods. However, the good generalizing networks are found faster due to the increased learning speed.

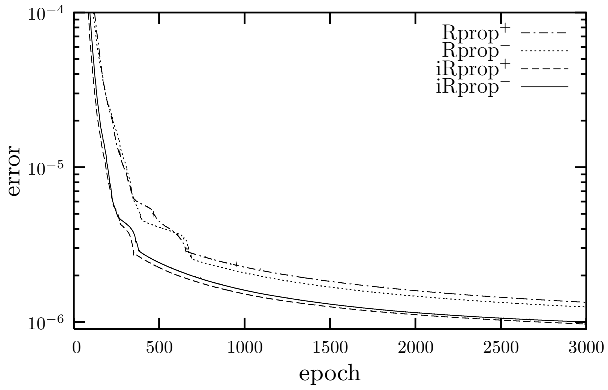

Figure 5: Averaged learning curves for DE solving (*problem3*).

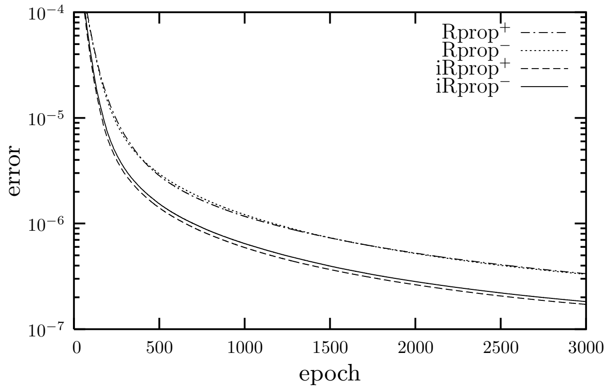

Figure 6: Averaged learning curves for *problem5*. The Rprop$^-$ and Rprop$^+$ trajectories can hardly be distinguished.

5 Conclusion

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 perform better than the original algorithms on all the neural network test problems. For the relevant parameterizations, the new algorithms also converge faster on artificial hyperparabolic test functions. The iRprop$^-$ algorithm appears to be slightly worse than iRprop$^+$, but it is very compact and needs less memory.

The three algorithms Rprop$^+$, iRprop$^-$ and iRprop$^+$ differ only in the used weight-backtracking scheme. In the context of the basic Rprop algorithm, the original backtracking decreases the learning speed, whereas the modified backtracking increases the learning speed. Setting the derivative to zero strongly influences the learning speed of the Rprop algorithm, as shown by comparison of the Rprop$^-$ and iRprop$^-$ results. Combined with the improved weight-backtracking this yields the best performing algorithm in our investigation, iRprop$^+$.

The experiments in this article are restricted to artificial test functions and feed-forward multilayer neural networks. However, we have employed the improved Rprop algorithms for other tasks including gradient based optimization of recurrent and radial basis function networks. In all experiments, they outperformed the original algorithms.

Future work will be directed toward an analysis of the advantages of the proposed methods applied to mathematically tractable error surfaces.

Table 1: Errors in the last epoch. For the PROBEN1 tasks the error percentage is given. The differential equation (*problem3* and *problem5*) results have to be multiplied with 10^{-6} and 10^{-7}, respectively.

data set	architecture	averaged (best) error in the last epoch			
		Rprop$^+$	Rprop$^-$	iRprop$^+$	iRprop$^-$
E_{10}	—	$5.914 \cdot 10^{-5}$ ($6.355 \cdot 10^{-7}$)	$3.925 \cdot 10^{-5}$ ($5.559 \cdot 10^{-7}$)	$4.236 \cdot 10^{-9}$ ($2.125 \cdot 10^{-11}$)	$4.236 \cdot 10^{-9}$ ($2.125 \cdot 10^{-11}$)
diabetes1	8–2–2–2	13.64 (12.37)	13.50 (12.18)	13.16 (11.86)	13.16 (11.72)
cancer1	9–4–2–2	2.333 (1.459)	2.242 (1.273)	2.009 (0.958)	2.014 (0.910)
T-C	16–5–1	0.1637 ($2.65 \cdot 10^{-7}$)	0.1137 ($1.32 \cdot 10^{-7}$)	0.0701 ($0.043 \cdot 10^{-7}$)	0.0723 ($0.025 \cdot 10^{-7}$)
problem3	1–10–1	1.340 (0.0097)	1.251 (0.0100)	0.970 (0.0094)	0.999 (0.0101)
problem5	2–10–1	3.352 (0.247)	3.309 (0.193)	1.717 (0.200)	1.825 (0.194)

References

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

[2] 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.

[3] 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.

[4] M. Hüsken, C. Goerick, and A. Vogel. Fast adaptation of the solution of differential equations to changing constraints. In *Proceedings of the Second International Symposium on Neural Computation, NC'2000*. ICSC Academic Press, 2000.

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

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

[7] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Tansactions on Neural Networks*, 9(5):987–1000, 1998.

[8] 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. Springer-Verlag, 1998.

[9] J. R. McDonnell and D. E. Waagen. Neural network structure design by evolutionary programming. In D. B. Fogel, editor, *Proceedings of the Second Annual Conference on Evolutionary Programming, EP93*, pages 79–89, San Diego, CA, USA, 1993. Evolutionary Programming Society.

[10] 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.

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

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

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

[14] D. E. Rummelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error backpropagation. In D. E. Rummelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

[15] 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.

[16] 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.

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

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

[19] 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.