A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm

Martin Riedmiller Heinrich Braun
Institut für Logik, Komplexität und Deduktionssyteme
University of Karlsruhe
W-7500 Karlsruhe
FRG
riedml@ira.uka.de

***Abstract*— A new learning algorithm for multilayer feedforward networks, RPROP, is proposed. To overcome the inherent disadvantages of pure gradient-descent, RPROP performs a local adaptation of the weight-updates according to the behaviour of the errorfunction. In substantial difference to other adaptive techniques, the effect of the RPROP adaptation process is not blurred by the unforseeable influence of the size of the derivative but only dependent on the temporal behaviour of its sign. This leads to an efficient and transparent adaptation process. The promising capabilities of RPROP are shown in comparison to other well-known adaptive techniques.**

I. Introduction

A. Backpropagation Learning

Backpropagation is the most widely used algorithm for supervised learning with multi-layered feed-forward networks. The basic idea of the backpropagation learning algorithm [1] is the repeated application of the chain rule to compute the influence of each weight in the network with respect to an arbitrary errorfunction E:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}} \tag{1}$$

where w_{ij} is the weight from neuron j to neuron i, s_i is the output, and net_i is the weighted sum of the inputs of neuron i. Once the partial derivative for each weight is known, the aim of minimizing the errorfunction is achieved by performing a simple gradient descent:

$$w_{ij}(t+1) = w_{ij}(t) - \epsilon \frac{\partial E}{\partial w_{ij}}(t) \tag{2}$$

Obviously, the choice of the learning rate ϵ, which scales the derivative, has an important effect on the time needed until convergence is reached. If it is set too small, too many steps are needed to reach an acceptable solution; on the contrary a large learning rate will possibly lead to oscillation, preventing the error to fall below a certain value.

An early way proprosed to get rid of the above problem is to introduce a momentum-term:

$$\triangle w_{ij}(t) = -\epsilon \frac{\partial E}{\partial w_{ij}}(t) + \mu \triangle w_{ij}(t-1) \tag{3}$$

where the momentum parameter μ scales the influence of the previous step on the current. The momentum-term is believed to render the learning procedure more stable and to accelerate convergence in shallow regions of the errorfunction.

However, as practical experience has shown, this is not always true. It turns out in fact, that the optimal value of the momentum parameter μ is equally problem dependent as the learning rate ϵ, and that no general improvement can be accomplished.

B. Adaptive Learning Algorithms

Many algorithms have been proposed so far to deal with the problem of appropriate weight-update by doing some sort of parameter adaptation during learning. They can roughly be separated into two categories: global and local strategies. Global adaptation techniques make use of the knowledge of the state of the entire network (e.g. the direction of the previous weight-step) to modify global parameters, whereas local strategies use only weight-specific information (e.g. the partial derivative) to adapt weight-specific parameters. Besides the fact, that local adaptation strategies are more closely related to the concept

of neural learning and are better suited for parallel implementations, their superiority over global learning algorithms has been impressively demonstrated in a recently published technical report [2].

The majority of both global and local adaptive algorithms performs a modification of a (probably weight-specific) learning-rate according to the observed behaviour of the errorfunction. Examples of such algorithms are the Delta-Bar-Delta technique [3] or the SuperSAB algorithm [4]. The adapted learning rate is eventually used to calculate the weight-step.

What is often disregarded, is, that the size of the actually taken weight-step $\triangle w_{ij}$ is not only dependend on the (adapted) learning-rate, but also on the partial derivative $\frac{\partial E}{\partial w_{ij}}$. So the effect of the carefully adapted learning-rate can be drastically disturbed by the unforseeable behaviour of the derivative itself. This was one of the reasons that lead to the development of RPROP: to avoid the problem of 'blurred adaptivity', RPROP changes the size of the weight-update $\triangle w_{ij}$ directly, i.e. without considering the size of the partial derivative.

C. Other Acceleration Techniques

A great variety of further modifications of the backpropagation procedure has been proposed (e.g. the use of modified errorfunctions, or sophisticated weight initialization techniques), which all promise to accelerate the speed of convergence considerably. In our experience, some of them worked slightly better on several problems, but for other problems they did not improve convergence or even gave worse results.

So the only modification we found useful and uncritical in use, was to simply 'clip' the logistic activation function at a value, that could be reasonably distinguished from the assymptotic boundary value. This results in an always non-zero derivative, preventing the unit of getting stuck. Especially in more difficult problems, this technique worked far more stable than adding a small constant value to the derivation of the activation function, as proposed in [5].

II. RPROP

A. Description

RPROP stands for '**r**esilient **prop**agation' and is an efficient new learning scheme, that performs a direct adaptation of the weight step based on local gradient information. In crucial difference to previously developped adaptation techniques, the effort of adaptation is not blurred by gradient behaviour whatsoever.

To achieve this, we introduce for each weight its individual update-value $\triangle_{ij}$, which solely determines the size of the weight-update. This adaptive update-value evolves during the learning process based on its local sight on the errorfunction E, according to the following learning-rule:

$$\triangle_{ij}^{(t)} = \begin{cases} \eta^+ * \triangle_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- * \triangle_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \triangle_{ij}^{(t-1)} & , \text{ else} \end{cases} \quad (4)$$

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

Verbalized, the adaptation-rule works as follows: Every time the partial derivative of the corresponding weight w_{ij} changes its sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update-value $\triangle_{ij}$ is decreased by the factor η^-. If the derivative retains its sign, the update-value is slightly increased in order to accelerate convergence in shallow regions.

Once the update-value for each weight is adapted, the weight-update itself follows a very simple rule: if the derivative is positive (increasing error), the weight is decreased by its update-value, if the derivative is negative, the update-value is added:

$$\triangle w_{ij}^{(t)} = \begin{cases} -\triangle_{ij}^{(t)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\triangle_{ij}^{(t)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0 & , \text{ else} \end{cases} \quad (5)$$

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \triangle w_{ij}^{(t)} \quad (6)$$

However, there is one exception: If the partial derivative changes sign, i.e. the previous step was too large and the minimum was missed, the previous weight-update is reverted:

$$\triangle w_{ij}^{(t)} = -\triangle w_{ij}^{(t-1)} \text{ , if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \quad (7)$$

Due to that 'backtracking' weight-step, the derivative is supposed to change its sign once again in the following step. In order to avoid a double punishment of the update-value, there should be no adaptation of the update-value in the succeeding step. In practice this can be done by setting $\frac{\partial E}{\partial w_{ij}}^{(t-1)} := 0$ in the $\triangle_{ij}$ adaptation-rule above.

The update-values and the weights are changed every time the whole pattern set has been presented once to the network (learning by epoch).

B. Algorithm

The following pseudo-code fragment shows the kernel of the RPROP adaptation and learning process. The **minimum** (**maximum**) operator is supposed to deliver the

minimum (maximum) of two numbers; the **sign** operator returns $+1$, if the argument is positve, -1, if the argument is negative, and 0 otherwise.

For all weights and biases{

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

$\triangle_{ij}(t) = \mathbf{minimum}\ (\triangle_{ij}(t-1) * \eta^+, \triangle_{max})$

$\triangle w_{ij}(t) = -\ \mathbf{sign}\ (\frac{\partial E}{\partial w_{ij}}(t)) * \triangle_{ij}(t)$

$w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t)$

}

else if $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0)$ **then** {

$\triangle_{ij}(t) = \mathbf{maximum}\ (\triangle_{ij}(t-1) * \eta^-, \triangle_{min})$

$w_{ij}(t+1) = w_{ij}(t) - \triangle w_{ij}(t-1)$

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

}

else if $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) = 0)$ **then** {

$\triangle w_{ij}(t) = -\ \mathbf{sign}\ (\frac{\partial E}{\partial w_{ij}}(t)) * \triangle_{ij}(t)$

$w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t)$

}

}

C. Parameters

At the beginning, all update-values $\triangle_{ij}$ are set to an initial value $\triangle_0$. For $\triangle_0$ directly determines the size of the first weight-step, it is preferably chosen in a reasonably proportion to the size of the initial weights. A good choice may be $\triangle_0 = 0.1$. However, as the results in the next section show, the choice of this parameter is not critical at all. Even for much larger or much smaller values of $\triangle_0$ fast convergence is reached.

With exception of the Spiral Learning Task, the range of the update-values was restricted to an upper limit of $\triangle_{max} = 50.0$ and a lower limit of $\triangle_{min} = 1e^{-6}$ to avoid overflow/underflow problems of floating point variables. In several experiments we observed, that by setting the maximum update-value to a considerably smaller value (e.g. $\triangle_{max} = 1.0$), we could reach a smoothened behaviour of the decrease of error.

The choice of the decrease factor η^- and increase factor η^+ was lead by the following considerations: if a jump over a minimum occured, the previous update-value was too large. For it is not known from gradient information how much the minimum was missed, in average it will be a good guess to halve the update-value, i.e. $\eta^- = 0.5$. The increase factor η^+ has to be large enough to allow fast growth of the update-value in shallow regions of the errorfunction, on the other side the learning process can be considerably disturbed, if a too large increase factor leads to persistent changes of the direction of the weight-step. In all our experiments, the choice of $\eta^+ = 1.2$ gave very good results, independend of the examined problem. Slight variations of this value did neither improve nor deteriorate convergence time. So in order to get parameter choice more simple, we decided to constantly fix the increase/decrease parameters to $\eta^+ = 1.2$ and $\eta^- = 0.5$.

One of the main advantages of RPROP lies in the fact, that for many problems no choice of parameters is needed at all to obtain optimal or at least nearly optimal convergence times.

D. Discussion

The main reason for the success of the new algorithm roots in the concept of 'direct adaptation' of the size of the weight-update. In contrast to all other algorithms, only the sign of the partial derivative is used to perform both learning and adaptation. This leads to a transparent and yet powerful adaptation process, that can be straight forward and very efficiently computed with respect to both time and storage consumption.

Another often discussed aspect of common gradient descent is, that the size of the derivative decreases exponentially with the distance between the weight and the output-layer, due to the limiting influence of the slope of the sigmoid activation function. Consequently, weights far away from the output-layer are less modified and do learn much slower. Using RPROP, the size of the weight-step is only dependend on the sequence of signs, not on the magnitude of the derivative. For that reason, learning is spread equally all over the entire network; weights near the input layer have the equal chance to grow and learn as weights near the output layer.

III. Results

A. Testing Methodology

For our study we implemented several learning procedures: Ordinary gradient descent by backpropagation (BP), SuperSAB (SSAB) [4], Quickprop (QP) [5] and RPROP.

To allow a fair comparison between the several learning procedures, a wide variety of parameter values was tested for each algorithm. Learning time is reported as the average number of epochs[1] required in ten different runs, and the respective standard deviation σ. For every algorithm the parameter setting was used that gave the best result.

For the binary tasks described in the following, learning is complete, if a binary criterion is reached. That is, the

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

activation of each unit in the output layer is smaller than 0.4 if its target value is 0.0, and bigger than 0.6 if its target value is 1.0.

Having large pattern sets, it is often not possible in practice, to test many different parameter settings until a good solution is found. So the simplicity of parameter choice is an important criterion for a learning procedure that should be well considered.

In order to get a rough measure of the simplicity to find an optimal parameter set, we define the 'Wellworking Region' (WR) of the parameter ϵ (respectively $\triangle_0$ for RPROP) as follows: If the value of ϵ respectively $\triangle_0$ lies within that intervall WR, the task shall be learned on average within at most 1.5 times the learning time that the algorithm achieved with the optimal parameter setting. (E.g. if a task is learned in minimum 200 epochs, then for all the values in the wellworking region WR convergence is reached within at most 300 epochs).

It should be noted clearly, that this is only a very rough measure for the robustness of an algorithm for the following reasons: Firstly, if an algorithm converges very fast, 1.5 times the minimum epoch number used is a much smaller region than that of a slow converging algorithm. Secondly, many algorithms use more than one parameter. While the 'Wellworking Region' only describes the behaviour of the algorithm in one parameter dimension, there is often a lot of expense neglected that arise when searching for the other parameter values.

Although this measure is rather disadvantageous for our RPROP-algorithm, which converges very fast *and* has only one parameter to adjust, the results on the Wellworking regions are still worth noting.

In the following experiments, ϵ denotes the (initial) learning-rate (BP, SSAB, QP), $\triangle_0$ denotes the initial update-value (RPROP), μ is the momentum (BP, SSAB), and ν denotes the maximal growth factor (QP).

B. The 10-5-10 Encoder Problem

The first problem to be described is the 10-5-10 Encoder task, for it is also discussed largely in [5]. The task is to learn an autoassociation between 10 binary input/output patterns. The network consists of 10 neurons in both the input and the output layer, and a hidden layer of 5 neurons. The following table shows the average learning times used by the diffent learning procedures:

10-5-10 Encoder					
Algo.	$\epsilon/\triangle_0$	μ/ν	# epochs	σ	WR(ϵ)
BP	1.9	0.0	121	30	[1.1,2.6]
SSAB	2.0	0.8	55	11	[0.1,6.0]
QP	1.5	1.75	21	3	[0.1,3.0]
RPROP	2.0	-	19	3	[0.05,2.0]

As can be seen, the adaptive procedures RPROP, Quickprop and SuperSAB do much better than the original backpropagation algorithm with respect to the convergence time as well as the robustness of the choice of their parameter values (indicated by the width of the WR). The Quickprop algorithm still outperforms SuperSAB by a factor 2.5, and is about 6 times as fast as original backpropagation.

The best result is achieved using RPROP, which learned the task in an average time of only 19 epochs. As shown in the width of the WR, the choice of the initial update-value $\triangle_0$ is not critical either.

C. The 12-2-12 Encoder Problem

The 12-2-12 Encoder task is to learn an autoassociation of 12 input/output patterns. The network consists of both 12 neurons in the input and the output layer, and a hidden layer of only 2 neurons ('Tight Encoder'). The difficulty of this task is to find a sensitive encoding/decoding of a 12-bit input/output vector in only 2 hidden neurons. The family of the 'Tight Encoder'-tasks demonstrates the capability of the learning algorithm to find a difficult solution in weight-space. The table shows the results of the several learning algorithms, and the respective best parameter setting:

12-2-12 'Tight Encoder'					
Algo.	$\epsilon/\triangle_0$	μ/ν	# epchs	σ	WR($\epsilon/\triangle_0$)
BP	div.	div.	> 15000		
SSAB	1.0	0.95	534	90	[0.01,4.5]
QP	0.05	1.3	405	608	[0.05,1.1]
RPROP	1.0	-	322	112	[0.0001,5.0]

Although a wide variety of parameter values has been tested, original backpropagation was not able to find a solution for the task in less than 15000 epochs. SuperSAB finally did its job, but only when varying the initial momentum parameter considerably, were acceptable learning times achieved. So despite the adaptivity of the learnigrates, several experiments were needed to find an optimal parameter set. Quickprop converges fast, but also a number of trials were needed to find a good value for each of its two parameters.

The best result was obtained using RPROP. On the one side, the algorithm converges considerably faster than all others, on the other side, a good parameter choice can be easily found (very broad WR). Figure 1 demonstrates the (averaged) behaviour of the adaptive algorithms on the 12-2-12 encoder task with respect to the variation of their initial learning parameter.

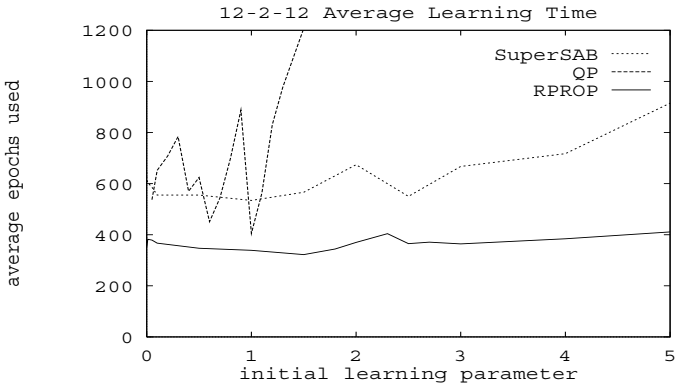

Figure 1: Behaviour of the average learning time for the 12-2-12 encoder task when varying the parameter ϵ rsp. $\triangle_0$. The figure shows the dependency of the several adaptive learning algorithms on a good estimate for their initial parameter values.

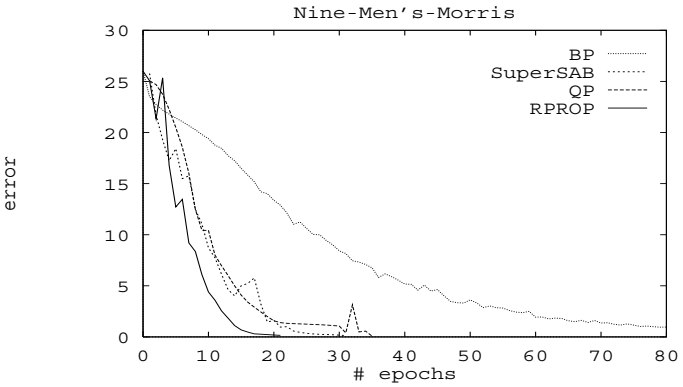

Figure 2: Decrease of the error over learning time for the Nine Men's Morris task

D. Nine Men's Morris

To show the performance of the learning procedures on more realistic problems, characterized by bigger networks and larger pattern sets, a network was trained to play the endgame of Nine Men's Morris [6].

The entire network is built up of two identical networks, linked by a 'comparator neuron'. Two alternative moves are presented to the respective partial network and the network is to decide, which move is the better one. Each partial network has an input layer with 60 units, two hidden layers with 30 and 10 neurons respectively, and a single output unit.

The pattern set consists of 90 patterns, each encoding two alternative moves and the desired output of the comparator neuron. The results of the Nine Men's Morris problem are listed below (see also Fig. 2):

Nine Men's Morris					
Algo.	$\epsilon/\triangle_0$	μ/ν	# epochs	σ	WR($\epsilon/\triangle_0$)
BP	0.2	0.5	98	34	[0.03,0.2]
SSAB	0.05	0.9	34	4	[0.01,0.3]
QP	0.005	1.75	34	12	[0.001,0.02]
RPROP	0.05	-	23	3	[0.05,0.3]

After several trials to find good parameter values, SuperSAB is able to learn the task in approximately 1/3 of the time used by the original backpropagation algorithm. Quickprop also took in average 34 epochs to learn the task, but the choice of its parameters was much easier compared to SuperSAB.

A further improvement was achieved using RPROP, which only took 23 epochs to learn. Again, the choice of the initial update-value $\triangle_0$ was found to be fairly uncritical.

E. The Two Spirals Problem

The difficulty of the 'Two Spirals Problem' has been demonstrated in many attempts to solve the problem with backpropagation and several elaborated modifications. The pattern set consists of 194 patterns, describing the points of two distinct spirals in the x-y-Plane. The network is built up of 2 input units, three hidden layers with 5 units each, and 1 output unit with symmetric activation functions. Each unit is connected to every unit in earlier layers (short-cut connections [7]).

The results reported so far are an average of 20.000 epochs on three(!) different runs for Backpropagation (using both an increasing learnig-rate and momentum-factor), and an average of 11.000 epochs using a nonlinear 'cross-entropy' error function.

Using Quickprop, an average of 7900 epochs on three different runs is reported, but it should be noted that a (non-standard) arctan-errorfunction and (non-standard) weight decay were needed to obtain this result.

In order to get statistically relevant results, we tested RPROP on 20 different runs. A run was considered successful, if a solution was found in less than 15.000 epochs. Weights were initialized randomly within $[-0.25, 0.25]$. The maximal update-value $\triangle_{max}$ was chosen considerably small, i.e. $\triangle_{max} = 0.001$, to avoid an early occurence of stuck units. The parameters η^+, η^- were set to their standard values. The result is listed below:

Two Spirals			
Algorithm	$<$ 15.000 ep.	Min.	average
RPROP	19/20	1500	4987

In comparison to the reported results, RPROP converges 4 times faster than backpropagation, and about 1.5 times faster than Quickprop, although both backpropagation and Quickprop needed non-standard extension to converge at all. The fact, that the pure RPROP algorithm was used and no further modifications had to be applied, again demonstrates the advantage of the new algorithm with respect to the simplicity of use.

F. Summary

In the following, the best results of the several learning algorithms are listed in an overview. The figures show the average number of epochs used on the learning tasks described above plus the results on an additional figure recognition task. The second row from below shows the results of the RPROP-algorithm using the standard parameter choice ($\triangle_0 = 0.1, \triangle_{max} = 50.0$). As can be seen, even without annoying parameter tuning very good results can be achieved.

Average number of required epochs				
Problem	10-5-10	12-2-12	9 Men's Morris	Figure Rec.
BP (best)	121	> 15000	98	151
SSAB (best)	55	534	34	41
QP (best)	21	405	34	28
RPROP (std)	30	367	30	29
RPROP (best)	19	322	23	28

IV. Conclusion

The proposed new learning algorithm RPROP is an easy to implement and easy to compute local learning scheme, which modifies the update-values for each weight according to the behaviour of the sequence of signs of the partial derivatives in each dimension of the weight-space.

The number of learning steps is significantly reduced in comparison to the original gradient-descent procedure as well as to other adaptive procedures, whereas the expense of computation of the RPROP adaptation process is held considerably small.

Another important feature, especially relevant in practical application, is the robustness of the new algorithm against the choice of its initial parameter.

RPROP is currently beeing tested on further learning tasks, for example on pattern sets containing continuous target values. The results obtained so far are very promising and confirm the quality of the new algorithm with respect to both convergence time and robustness.

References

[1] D. E. Rumelhart and J. McClelland. *Parallel Distributed Processing.* 1986.

[2] 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.

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

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

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

[6] H.Braun, J. Feulner, and V. Ullrich. Learning strategies for solving problem of planning using backpropagation. In *Proceedings of NEURO Nimes*, 1991.

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