4 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 partial derivatives in each dimension.

The number of required epochs is drastically reduced in comparison to the original backpropagation procedure, whereas the expense of computation is only slightly increased in contrast to many other adaptive learning algorithms, e.g. SuperSAB or Quickprop.

Considering the examined learning tasks, RPROP outperformed all other algorithms with respect to the epochs required to learn the task and the robustness of the choice of its parameters.

When choosing the standard parameter values for the increase and decrease factor $\eta^+ = 1.2, \eta^- = 0.5$, which actually produce the best results independent of the learning problem, there remains only one parameter to be adjusted, namely the initial update-value $\triangle_0$, which is also found to be uncritical.

RPROP is currently beeing tested on further learning tasks. The results obtained so far are very promising and seem to confirm the quality of the new algorithm with respect to both convergence time and robustness.

References

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

[2] C. Anderson, *"Learning and Problem Solving with Multilayer Connectionist Systems"*, Technical Report COINS TR 86-50, University of Massachusetts, Amherst, MA, 1986

[3] T. Tollenaere, *"SuperSAB: Fast Adaptive Backpropagation with good Scaling Properties"*, Neural Networks, Vol 3, pp. 561 - 573, 1990

[4] R. Jacobs, *"Increased Rates of Convergence Through Learning Rate Adaptation"*, Neural Networks, Vol 1, No 4, 1988

[5] S. E. Fahlman, *"An Empirical Study of Learning Speed in Back-Propagation Networks"*, CMU-CS-88-162, September 1988

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

[7] J. Feulner, V. Ullrich, *"On Learning Strategies for Deterministic Games Using Backpropagation: A Case Study"*, in: Baray/Özgüç (eds.), Proceedings of ISCIS VI, Elsevier 1991

3.4 Nine Men's Morris

To show the performance of the learning procedures on bigger networks, a network was trained to play the endgame of Nine Men's Morris [6], [7].

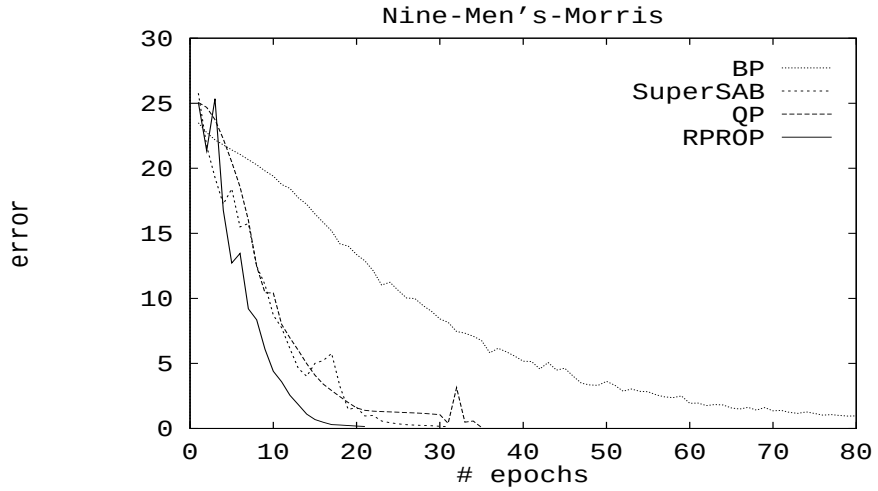

Figure 4: Decrease of the error over learning time for the Nine Men's Morris task

The entire network is built up of two indentical 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. 4):

Nine Men's Morris							
Algorithm	$\epsilon/\triangle_0$	μ/ν	η^+	η^-	# epochs	σ	WR($\epsilon/\triangle$)
Original BP	0.2	0.5	-	-	98	34	[0.03,0.2]
SuperSAB	0.05	0.9	1.7	0.5	34	4	[0.01,0.3]
Quickprop	0.005	1.75	-	-	34	12	[0.001,0.02]
RPROP	0.05	-	1.2	0.5	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 algorithm. Quickprop also took 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 standard choice of η^+ and η^- was used, and the choice of the initial update-value $\triangle_0$ was found to be fairly uncritical.

3.3 The 12-2-12 Encoder Problem

The task of the 12-2-12 Encoder 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'), demonstrating the capabilities of the learning algorithm to solve difficult tasks, where a solution in weight-space is hard to find:

12-2-12 'Tight Encoder'							
Algorithm	$\epsilon/\triangle_0$	μ/ν	η^+	η^-	# epochs	σ	WR($\epsilon/\triangle$)
Original BP	div.	div.	-	-	> 5000		
SuperSAB	1.0	0.95	1.05	0.5	534	90	[0.01,4.5]
Quickprop	0.05	1.3	-	-	405	608	[0.05,1.1]
RPROP	1.0	-	1.2	0.5	322	112	[0.0001,5.0]

Although a wide variety of parameter values for ϵ and μ has been tested, original backpropagation was not able to find a solution for the tasks in less than 5000 epochs. SuperSAB without the momentum-term did its job, but only when trying a higher moment, were acceptable learning times achieved. So despite the adaptivity of the learnig-rates, several experiments were needed to find an optimal parameter set.

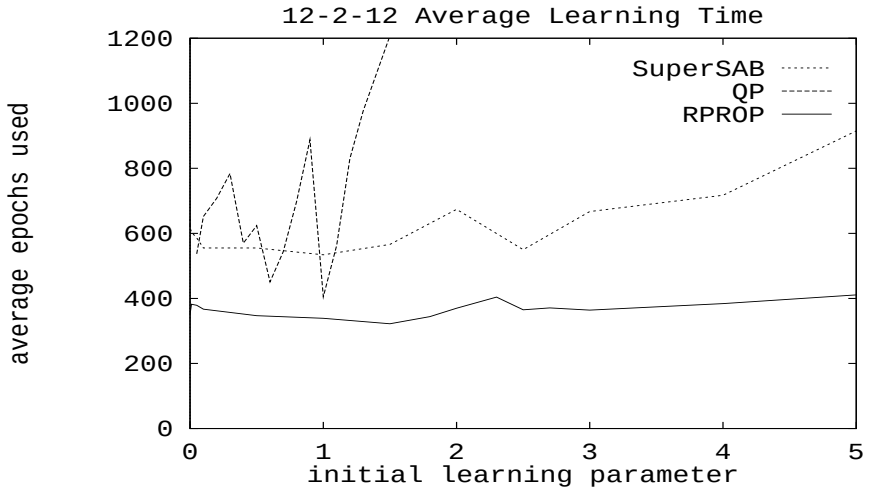

Figure 3: Behaviour of the average learning time for the 12-2-12 encoder task when varying the parameter ϵ rsp. $\triangle_0$. The figure shows the dependency of the several adaptive learning algorithms on a good estimate for their initial parameter values.

Quickprop converges fast, but a number of trials were needed to find a good value for ϵ (note the very small WR). Again, the best result was obtained using RPROP. On the one side, the algorithm converges very fast, and on the other side, a good parameter choice can be easily found (standard values for η^+, η^-, very broad WR). Figure 3 demonstrates the (averaged) behaviour of the adaptive algorithms on the 12-2-12 encoder task.

in which every pattern of the training set is presented once. For each algorithm, a wide variety of parameter values was tested to find an optimal choice to allow a fair comparison between the several learning procedures Backpropagation, SuperSAB [3], Quickprop [5], and RPROP.

Learning is complete, if a binary criterion is reached. That is, the 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.

In the following, ϵ denotes the (initial) learning-rate (BP, SuperSAB, Quickprop), $\triangle_0$ denotes the initial update-value (RPROP), μ is the momentum (BP, SuperSAB), ν the maximal growth factor (Quickprop), and η^+, η^- denote the increase/decrease factor (SuperSAB, RPORP).

In order to get a measure of the difficulty to find an optimal parameter set, a 'Well-working Region' (WR) of the parameter ϵ (respectively $\triangle_0$ for RPROP) is defined. If the value of ϵ rsp. $\triangle_0$ is selected within that intervall WR, convergence is guaranteed within at most 1.5 times the minimum learning time, achieved with the optimal paramter choice.

3.2 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							
Algorithm	ϵ	μ/ν	η^+	η^-	# epochs	σ	WR(ϵ)
Original BP	1.9	0.0	-	-	121	30	[1.1,2.6]
SuperSAB	2.0	0.8	1.05	0.5	55	11	[0.1,6.0]
Quickprop	1.5	1.75	-	-	21	3	[0.1,3.0]
RPROP	2.0	-	1.2	0.5	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 using the standard choice of $\eta^+ = 1.2$ and $\eta^- = 0.5$. As shown in the width of the WR, the choice of the initial weight update-value $\triangle_0$ isn't critical either.

Note that the update-value is not influenced by the *magnitude* of the derivatives, but only by the behaviour of the *sign* of two succeeding derivatives. 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 (fig. 2), 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 (fig. 1).

The update-rule for the weights is the same as in eq. (3) with one exception: if the partial derivative changes sign, the previous update-step, leading to a jump over the minimum, is reverted:

$$\triangle w_{ij}(t) = -\triangle w_{ij}(t-1) \text{ , if } \frac{\partial E}{\partial w_{ij}}(t) * \frac{\partial E}{\partial w_{ij}}(t-1) < 0 \tag{5}$$

When a change of sign occured, the adaptation process is 'restarted', which means, that in the succeeding step no adaptation of the update-value is performed (in practice this can be done by setting $\frac{\partial E}{\partial w_{ij}}(t-1) := 0$ in eq (4))

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

At first glance, this approach resembles a little to the well-known SuperSAB algorithm [3],[4]. However, there are a few important differences. SuperSAB tries to adapt the learning-rate which scales the partial derivative to compute the weight-update $\triangle w_{ij}(t) = \epsilon_{ij}(t) \frac{\partial E}{\partial w_{ij}}(t)$. So, the weight-update is still strongly dependent on the magnitude of the partial derivative. This is possibly leading to either emphasizing or compensating effects, which means that the adaptation is partly unworthy because of the unforseeable influence of the magnitude of the partial derivative.

To avoid this problem of 'double adaptivity', RPROP changes the value of the weight-update $\triangle_{ij}$ directly, only depending on the sign of the partial derivative without reference to its magnitude. As a further effect of the only sign-dependent weight-update, learning is spread equally all over the entire network, whereas with value-sensible learning-rules, weight-update is a function of the distance between the weight and the output-layer.

At the beginning, all update-values $\triangle_{ij}$ are set to an initial value $\triangle_0$. The choice of this value is not critical, for it is adapted as learning proceeds (see the results in the next section). In all experiments, the decreasing and increasing factors were held constant to $\eta^+ = 1.2$ and $\eta^- = 0.5$. The range of the update-values was restricted to an upper limit of 50.0 and a lower limit of $1e^{-6}$ to avoid underflow problems of floating point variables.

3 Results

3.1 Methodology

In the following experiments, learning time was measured as the average number of epochs required until the task was learned in ten different runs. An epoch is defined as the period

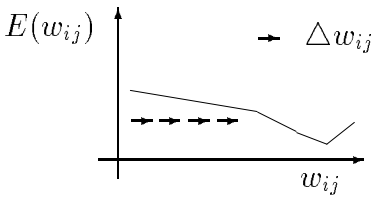

Figure 1: In shallow regions a small learning-rate leads to long convergence times.

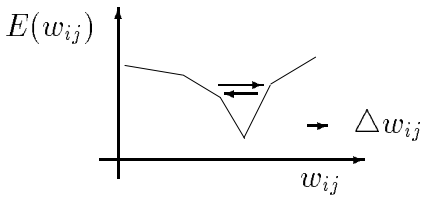

Figure 2: Large learning-rates lead to oscillation in the proximity of local minima.

Many algorithms have been proposed so far to deal with the above problems, e.g. by introducing a momentum term or doing some sort of parameter adaptation during learning. The following section presents a new adaptive learning scheme combining both easy computation and powerful adaptation for the sake of faster convergence.

2 The RPROP-algorithm

RPROP stands for 'resilient propagation' and is a new adaptive learning algorithm that considers the local topology of the errorfunction to change its behaviour. It is based on the so-called 'Manhattan-Learning'-rule, described by Sutton in [2]:

$$\triangle w_{ij} = \begin{cases} -\triangle_0 & , \text{ if } \frac{\partial E}{\partial w_{ij}} > 0 \\ +\triangle_0 & , \text{ if } \frac{\partial E}{\partial w_{ij}} < 0 \\ 0 & , \text{ else} \end{cases} \tag{3}$$

where $\triangle_0$, the 'update-value', is a problem-dependent constant.

Due to its simplicity, this is a very coarse way to adjust the weights, and so it is not surprising that this method doesn't work satisfactory on difficult problems, where it is hard to find an acceptable solution (e.g strong nonlinear mappings).

The basic idea for the improvement realized by the RPROP algorithm was to achieve some more information about the topology of the errorfunction so that the weight-update can be done more appropriately. For each weight we introduce its own 'personal' update-value $\triangle_{ij}$, which evolves during the learning process according to its local sight of the errorfunction E. So we get a second learning-rule for the update-values themselves:

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

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

To appear in: Proceedings of ISCIS VII

RPROP - A Fast Adaptive Learning Algorithm

Martin Riedmiller and Heinrich Braun

Institut für Logik, Komplexität und Deduktionssysteme, Universität Karlsruhe, Kaiserstrasse 12, 7500 Karlsruhe, FRG

Abstract

In this paper, a new learning algorithm, RPROP, is proposed. To overcome the inherent disadvantages of the pure gradient-descent technique of the original backpropagation procedure, RPROP performs an adaptation of the weight update-values according to the behaviour of the errorfunction. The results of RPROP on several learning tasks are shown in comparison to other well-known adaptive learning algorithms.

1 Introduction

Backpropagation is the most widely used algorithm for supervised learning with multilayered feed-forward networks. The basic idea of the backpropagation learning algorithm 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 [1]:

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

where w_{ij} is the weight from neuron j to neuron i, a_i is the activation value 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:

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

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 (fig. 1); on the contrary a large learning rate will possibly lead to oscillation, preventing the error to fall below a certain value (fig. 2).