Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons *

W. Schiffmann, M. Joost, R. Werner
University of Koblenz
Institute of Physics
Rheinau 1
56075 Koblenz
e-mail: evol@infko.uni-koblenz.de

September 29, 1994
(First edition published in 1992)

Contents

*This work is supported by the *Deutsche Forschungsgemeinschaft* (DFG) as part of the project *FE–generator* (grant Schi 304/1–1)

1 Introduction

This is the revised edition of our technical report, which has been published in 1992. There is no new stuff in this revision, but some minor bug fixes are helpful for implementing the described algorithms. The results of this report also have been published on ESANN '93 [Schiffmann et al., 1993]. The dataset used in this comparision is available by anonymous ftp (FTP server: ics.uci.edu, files: pub/machine-learning-databases/thyroid-disease/ann∗).

Backpropagation is one of the most popular training algorithms for multilayer perceptrons. Unfortunately it can be very slow for practical applications. Over the last years many improvement strategies have been developed to speed up backpropagation. It's very difficult to compare these different techniques, because most of them have been tested on very special data sets. The reported results are based on some kind of tiny and artificial training sets like XOR, encoder or decoder. It's very doubtful if this results hold for a much more complicate practical application. In these report an overview of many different speedup techniques is given. All of them are tested on a very hard practical classification task, which consists of a big medical data set. As you will see many of these optimized algorithms fail in learning the data set.

2 Application

We have used measurements of the thyroid gland for testing different approaches. Each measurement vector consists of 21 values – 15 binary and 6 analog. Three classes are assigned to each of the measurement vectors which correspond to the hyper-, hypo- and normal function of the thyroid gland. Since over 92% of all patients have a normal function, a useful classifier must be significantly better than 92% correct classifications. The training set consists of 3772 measurement vectors and again 3428 measurements are available for testing. The training period was limited to 5000 epochs using a fixed 3 layer network architecture with 21 input- , 10 hidden- and 3 output units. The Network was fully interconnected. Using a SPARC2 CPU training takes from 12 to 24 hours. The weights of the network have been randomly chosen by a normal distribution ($\mu = 0.0, \sigma = 0.1$). The bias of each unit has been computed as follows. First the average input pattern of the whole learning set has been calculated. While propagating this averaged pattern through the network the bias of each unit is tuned to half activate every hidden or output unit. By this means the gradient of the sigmoid activation function of every unit is maximized, which has some benefits on the gradient descent during the training.

3 Mathematical Notation

Many different mathematical notations are used to describe training algorithms for neural networks. In order to compare different techniques a uniform notation is necessary:

a_i : Activation of unit i
w_{ij} : Connection strength from unit i to unit j
o_i : Desired activation of output unit i
(i-th's component of the output vector)
net_i : Netinput to unit i

$$net_i = \sum_{\forall j w_{ji} \neq 0} w_{ji} * a_j$$

$$a_i = \frac{1}{1 + e^{-net_i}}$$

E_p : Quadratic Error for pattern p

$$E_p = \sum_{i \in Outputunits} (a_i - o_i)^2$$

E : Total quadratic error on the training set

$$E = \sum_p E_p$$

$\frac{\partial E_p}{\partial w_{ij}}$: Partial derivative for pattern p with respect to w_{ij}

$\frac{\partial E}{\partial w_{ij}}$: Partial derivative for the whole training set with respect to w_{ij}

$$\frac{\partial E}{\partial w_{ij}} = \sum_p \frac{\partial E_p}{\partial w_{ij}}$$

∇E : Gradient with respect to the whole trainings set

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n})$$

∇E_p : Gradient with respect to pattern p

$$\nabla E_p = (\frac{\partial E_p}{\partial w_1}, \frac{\partial E_p}{\partial w_2}, \ldots, \frac{\partial E_p}{\partial w_n})$$

$\Delta w_{ij}(n)$: Weight update of w_{ij} in the n-th learning step

$$w_{ij}(n+1) = \Delta w_{ij}(n) + w_{ij}(n)$$

4 Backpropagation

Basically, Backpropagation [Rumelhart, 1986] is a gradient descent technique to minimize some error criteria E. In the batched mode variant the descent is based on the gradient ∇E for the total training set :

$$\Delta w_{ij}(n) = -\epsilon * \frac{\partial E}{\partial w_{ij}} + \alpha * \Delta w_{ij}(n-1)$$

ϵ and α are two non negative constant parameters called learning rate and momentum. The momentum can speed up training in very flat regions of the error surface and suppresses weight oscillation in step valleys or ravines. Unfortunately it is necessary propagate the whole training set through the network for calculating ∇E . This can slow down training for bigger training sets. For some tasks (e.g. neural controllers) [Schiffmann and Geffers, 1993] no finite training set is available. Therefore the update is based just on the gradient for the actual training pattern ∇E_p:

$$\Delta w_{ij}(n) = -\epsilon * \frac{\partial E_p}{\partial w_{ij}} + \alpha * \Delta w_{ij}(n-1)$$

A good choice of ϵ and α is very essential for training success and speed. Adjusting these parameters by hand can be very difficult and might take a very long time for more complicated tasks.

Results of the training with backpropagation and update after every pattern presentation heavily depend on a proper choice of the parameters (see Figure 1 and Table 1, respectively). Nevertheless good results can be achieved by carefully adjusting learning rate and momentum. Total error and recognition rate with respect to the training set and for the testing set (rightmost 2 columns) are presented in every table of this report. In order to compare the results more easily the best result in every column is underlined. The recognition rate specifies the percentage of correct classified patterns. A pattern is called correct classified, if the euclidian distance between the network output vector and the desired output vector is smaller than to any other possible output vector.

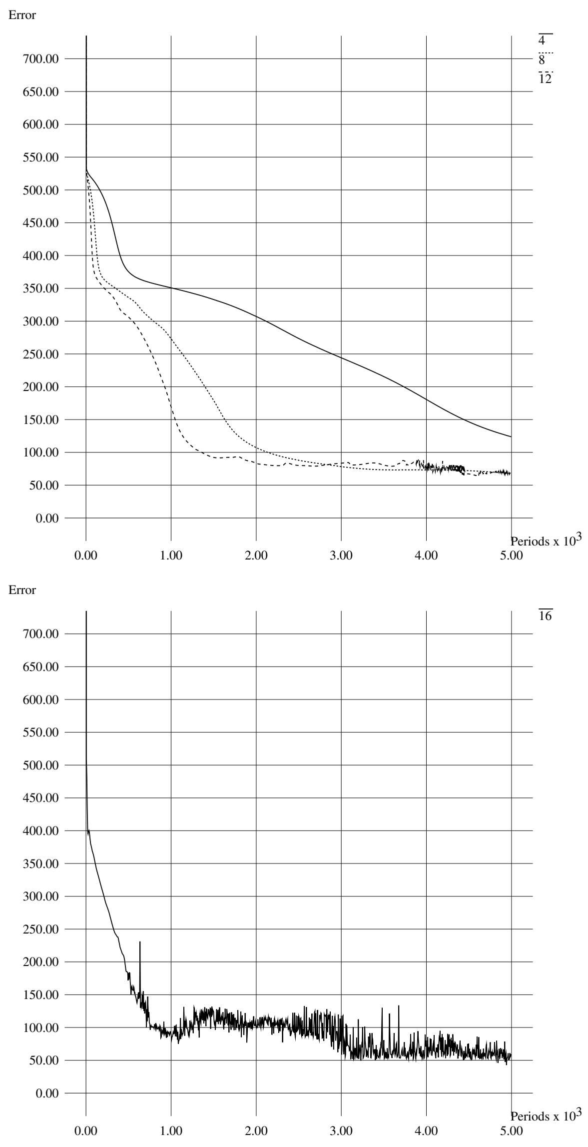

Figure 1: Backprop updated after every pattern for run 4, 8, 12 and 16 of Table 1

			Training Set		Testing Set	
	ϵ	α	Error	Recog. rate	Error	Recog. rate
1	0.01	0.0	267.3	95.07	293.14	94.52
2	0.01	0.1	250.9	95.47	279.14	94.69
3	0.01	0.2	231.8	96.24	261.44	95.19
4	0.01	0.5	123.8	98.25	163.95	97.14
5	0.01	0.9	61.3	98.91	151.07	97.37
6	0.05	0.0	70.7	98.94	141.86	97.64
7	0.05	0.1	72.0	98.94	145.83	97.52
8	0.05	0.2	68.3	98.94	150.04	97.49
9	0.05	0.5	80.9	98.75	173.97	96.85
10	0.05	0.9	$\underline{50.5}$	$\underline{99.13}$	139.11	97.55
11	0.1	0.0	74.9	98.81	168.92	97.05
12	0.1	0.1	67.0	98.83	158.50	97.17
13	0.1	0.2	64.5	98.91	171.50	96.94
14	0.1	0.5	103.7	98.14	208.52	96.30
15	0.1	0.9	77.0	98.67	170.02	97.05
16	0.5	0.0	59.3	98.89	$\underline{137.63}$	$\underline{97.58}$
17	0.5	0.1	69.8	98.75	149.26	97.32
18	0.5	0.5	55.0	98.94	148.45	97.52
19	0.5	0.9	77.5	98.59	150.63	97.26

Table 1: Backprop updated after every pattern

Results for a batched update after every training epoch are much worse (see Figure 2). No suitable parameters have been found for training a useful network.

5 Global learning rate adaptation

One way to optimize the backpropagation algorithm is to find proper values for the learning rate automatically. The following techniques try to adjust this parameters during the training. Most of them also adjust the momentum parameter, so that step size and search direction are altered.

5.1 Fixed calculating of the learning rate

Harry A. C. Eaton and Tracy L. Olivier have suggested a calculation of the learning rate for backpropagation using batched updates [Eaton et al., 1992]. This calculation is based on the assumption that similar training patterns result in similar gradients. So it is desirable to reduce the learning rate if there are many similar training patterns. Therefore the training set must be divided in m subsets of similar patterns. Let $N_1, N_2, \ldots, N_m$ be the sizes of these subsets. Learning rate and momentum can now be set in the following manner:

$$\epsilon = \frac{1.5}{\sqrt{N_1^2 + N_2^2 + \ldots + N_m^2}}$$

$$\alpha = 0.9$$

For our application the training set was divided in 3 subsets representing the 3 different output classes:

$$\epsilon = \frac{1.5}{\sqrt{93^2 + 191^2 + 3488^2}} \approx 0.00043$$

As one can see (see Figure 3) the results are very disappointing. The learning rate is to small while the momentum is to big (compare with Figure 2). No useful network could be trained with this technique.

5.2 Decreasing learning rate

Christian Darken and John Moody decrease the learning rate during the training [Darken et al., 1990]. This so called "Search-Then-Converge" strategy is suggested for back-propagation using updates for every training pattern. Starting with a big learning rate $\epsilon(0)$ the value is decreased during the training to approx $\epsilon(0)/(1+n)$ later on:

$$\epsilon(n) = \frac{\epsilon(0)}{1 + \frac{n}{r}}$$

	ϵ	α	Training Set Error	Training Set Recog. rate	Testing Set Error	Testing Set Recog. rate
1	0.0001	0.0	525.5	92.47	462.63	92.71
2	0.0001	0.1	524.8	92.47	461.98	92.71
3	0.0001	0.9	478.5	92.58	426.07	92.85
4	0.001	0.0	473.3	92.60	422.32	92.85
5	0.001	0.1	<u>461.8</u>	<u>92.63</u>	<u>414.34</u>	<u>92.85</u>
6	0.001	0.9	568.0	92.47	500.00	92.71
7	0.01	0.0	568.0	92.47	500.00	92.71
8	0.01	0.1	568.0	92.47	500.00	92.71
9	0.01	0.9	568.0	92.47	500.00	92.71

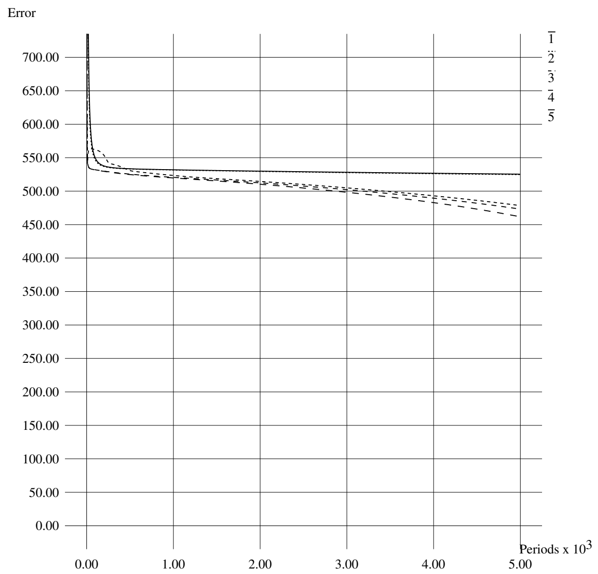

Figure 2: Backprop using batchmode

			Training Set		Testing Set	
	ϵ	α	Error	Recog. rate	Error	Recog. rate
1	0.00043	0.9	568.0	92.47	500.00	92.71
2	0.00043	0.0	511.0	92.47	450.10	92.71

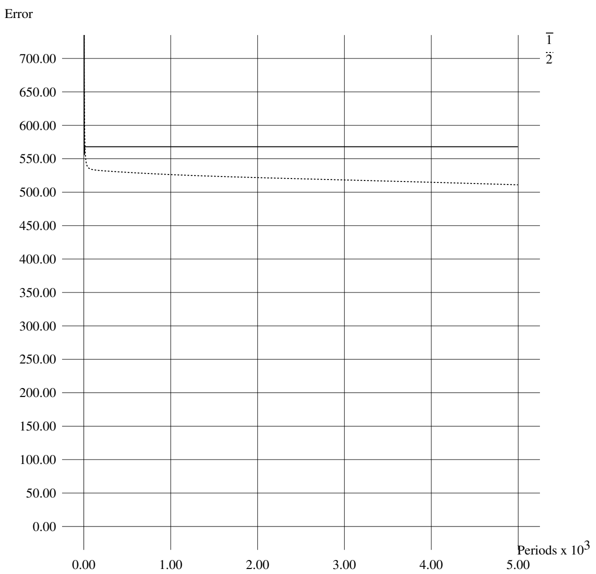

Figure 3: Fixed calculating of the learning rate

The constant parameter r can be used to adjust this learning rate schedule with respect to the total training period. After the first r learning steps the learning rate is halved by this update rule.

As you see in Figure 1 big learning rates are useful in the early training phase but result in oscillation later on. By using a decreasing learning rate during the training the advantages of big values (fast learning in the early learning phase) and small values (good asymptotic behavior) can be combined by a proper value for r (see Figure 4). Unfortunately a good r can only be found by trial and error. Nevertheless networks with a moderate performance have been trained by this algorithm.

				Training Set		Testing Set	
	ϵ	α	r	Error	Recog. rate	Error	Recog. rate
1	0.5	0.0	500	57.7	98.99	141.78	97.46
2	0.5	0.0	1000	51.8	99.15	135.62	97.67
3	0.5	0.0	2000	44.2	99.20	126.84	97.90

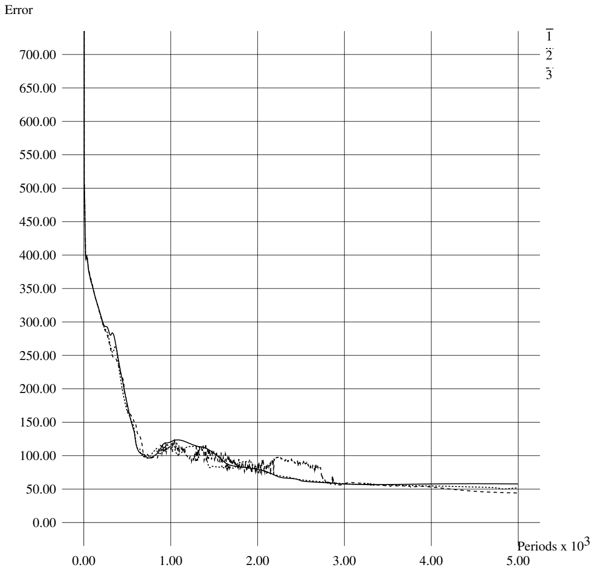

Figure 4: Decreasing learning rate

5.3 Learning rate adaptation for each training pattern

J. Schmidhuber also uses updates for every training pattern [Schmidhuber, 1989]. He is calculating a new learning rate for every update and doesn't use any momentum. Therefore the tangent in the error surface of the actual training pattern at the current position is used. The new values for every connection are found by calculating the point of intersection with the zero plane. For practical reasons it is necessary to define an upper limit for a single learning step. There also may be some error surfaces which never reach the zero plane. For those surfaces a small constant value E_{offset} is subtracted to make sure that zero points exists:

$$\epsilon(n) = \text{min}(\frac{E_p - E_{offset}}{\parallel \nabla E_p \parallel^2}, \epsilon_{max})$$

$$\Delta w_{ij}(n) = -\epsilon(n) * \frac{\partial E_p}{\partial w_{ij}}$$

20.0 is the recommend value for ϵ_{max}. Schmidhuber emphasized, that his algorithm is able to escape from a local minimum. Undoubtedly his strategy can escape from such a local minimum ($E_p \neq 0 \wedge \parallel \nabla E_p \parallel \approx 0$). Nevertheless this may result in very big updates, which might corrupt the whole network in one learning step. It's doubtful if this is desirable especially for networks which already classify most of the training patterns correctly. This strategy also can't handle very big training sets where some wrong classified patterns are likely to exist.

The achieved results (see Figure 5) support this theoretical disadvantages. Without any offset the system shows a chaotic behavior which results in infinite updates. By using a small offset useful networks can be trained. Nevertheless training with fixed learning rates is superior to this approach (compare with Figure 1).

5.4 Evolutionarily adapted learning rate

R. Salomon uses a simple evolution strategy to adjust the learning rate [Salomon, 1989 and 1990]. Starting with some ϵ the next update is done by using an increased and a decreased learning rate. The one which results in better performance is used as a starting point for the next update:

1. Create two equal networks and an initial learning rate .

2. Adjust the weights of both networks as follows:

$$\Delta w_{ij}(n) = -\epsilon(n) * \frac{\frac{\partial E}{\partial w_{ij}}}{\parallel \nabla E \parallel}$$

	ϵ_{max}	E_{offset}	Training Set Error	Training Set Recog. rate	Testing Set Error	Testing Set Recog. rate
1	20	0.00	-	-	-	-
2	20	0.03	91.5	98.36	163.54	97.23

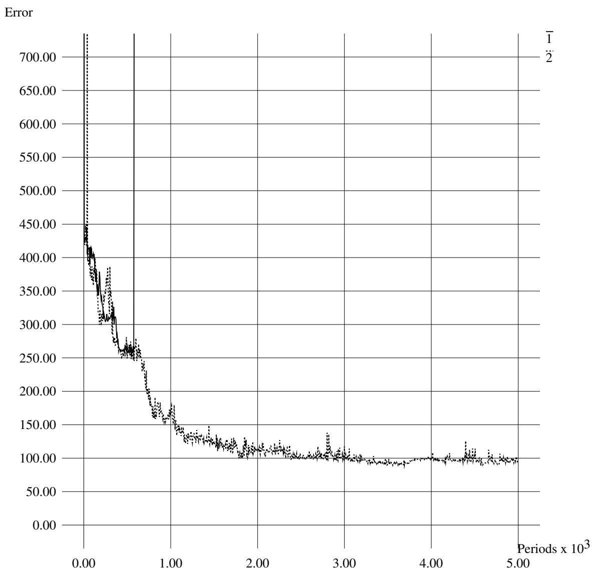

Figure 5: Learning rate adaptation for each training pattern

3. Discard the networks and restart with the former network and the initial learning rate, if both total errors have been increased (Backtracking).
   In the case of decreasing total errors use the network with a smaller total error with learning rates $\epsilon(n) * \beta$ and $\epsilon(n) * \frac{1}{\beta}$ to start the next learning step.

This simple strategy can handle almost any initial learning rate and greatly improves the performance of backpropagation using batched updates (see Figure 6). To compare the results one has to take into consideration the doubled calculation time. Nevertheless no useful network can be trained.

			Training Set		Testing Set	
	$\epsilon(0)$	β	Error	Recog. rate	Error	Recog. rate
1	0.001	1.1	332.1	94.64	347.32	<u>94.14</u>
2	0.01	1.1	<u>331.1</u>	<u>94.64</u>	346.79	94.08
3	0.1	1.1	331.3	94.64	<u>346.73</u>	94.11
4	0.5	1.1	535.2	92.47	472.46	92.71
5	1.0	1.1	536.1	92.47	472.85	92.71

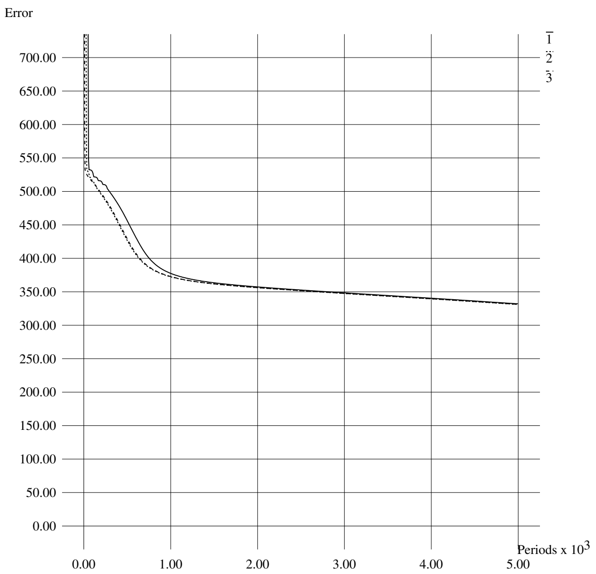

Figure 6: Evolutionary adapted learning rate

5.5 Angle driven learning rate adaptation

L.-W. Chan and F. Fallside adapt learning rate and momentum during the training [Chan et al., 1987]. Therefore the angle between $\nabla E(n)$ and $\Delta w(n-1)$ is calculated. The adaptation tries to adjust this angel at 90^o. As long as the angle is less than 90^o the learning rate is increased otherwise it is decreased:

1. Calculate:

$$\cos\Theta(n) = \frac{-\nabla E(n) \cdot \Delta w(n-1)}{\| \nabla E(n) \| * \| \Delta w(n-1) \|}$$

2. Adapt the learning rate:

$$\epsilon(n) = \epsilon(n-1) * (1 + 0.5 * cos\Theta(n))$$

3. Adapt the momentum:

$$\alpha(n) = \alpha(0) * \frac{\| \nabla E(n) \|}{\| \Delta w(n-1) \|}$$

4. Adjust the weights:

$$\Delta w_{ij}(n) = \epsilon(n) * (\frac{\partial E}{\partial w_{ij}} + \alpha(n) * \Delta w_{ij}(n-1))$$

Unfortunately the learning rate was adapted much to rapidly which results in very big learning rates. So we tried to modify the adaptation rule:

$$\epsilon(n) = \epsilon(n-1) * (1 + 0.1 * cos\Theta(n))$$

In addition we use a backtracking strategy, which restarts a learning step using a halved learning rate in the case of increasing total error. Results (see Figure 7) are almost similar to the evolution strategy.

5.6 Nearly optimal learning rate adjust using line search

On principle it's possible to calculate the optimal learning rate for an update direction. A learning rate is called optimal for a given update direction if it minimizes E with respect to that search direction. If one uses the negative gradient for the search a "steepest descent" is performed. A simple way to approximate the optimal learning rate is to start with a small value, perform a weight update and calculate the total new error. As long as the total error decreases the update is redone using an increased learning rate [Hertz et al., 1991]. Unfortunately E has to be calculated for every new iteration of the learning rate. This may be computational intensive if many iterations are necessary. It's also difficult to define a proper start value for the learning rate iteration.

	$\epsilon(0)$	$\alpha(0)$	Training Set Error	Training Set Recog. rate	Testing Set Error	Testing Set Recog. rate
1	0.0001	0.0	332.1	94.64	347.46	94.14
2	0.0001	0.001	<u>317.6</u>	<u>94.67</u>	<u>335.47</u>	<u>94.17</u>
3	0.0001	0.01	568.0	92.47	500.00	92.71
4	0.0001	0.1	568.0	92.47	500.00	92.71

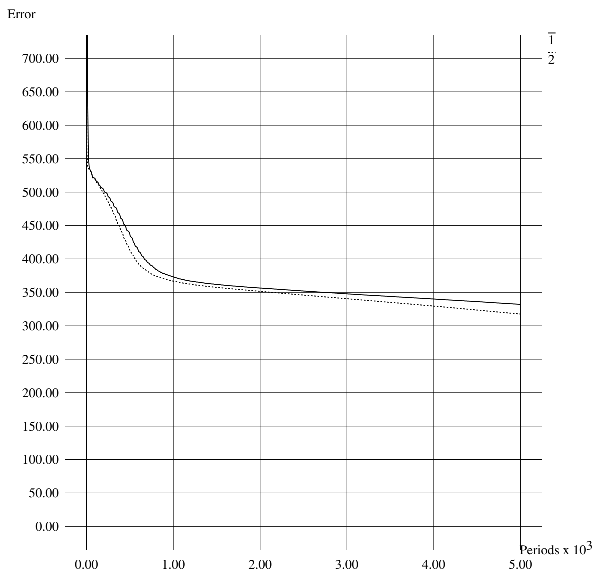

Figure 7: Angle driven learning rate adaptation

Finding the optimal learning rate for a given search update direction means to minimize a one dimensional function. Therefore standard strategies like Newton's method can be applied. Unfortunately Newton's method requires the calculation of the second derivative. Another possibility is to use a Newton like iteration like the method of "False Position" [Luenberger, 1973]. Instead of using the second derivative an approximation by calculating a difference quotient is used:

$$f''(x_k) \approx \frac{f'(x_{k-1}) - f'(x_k)}{x_{k-1} - x_k}$$

The minimum can now be approximated by calculating the following iteration:

$$x_{k+1} = x_k - f'(x_k) * \frac{x_{k-1} - x_k}{f'(x_{k-1}) - f'(x_k)}$$

Finally $\frac{\delta E}{\delta \epsilon}$ has to be calculated and some criteria for stopping the iteration is necessary to apply the "False Position" method. In our simulations the search is terminated by the following criteria:

$$\frac{|x_k - x_{k-1}|}{|x_k|} \leq 0.01$$

Starting values for the iteration have been calculated as follows:

$$\begin{aligned} x_0(n+1) &= \epsilon(n) \\ x_1(n+1) &= \epsilon(n) * 1.5 \qquad \text{, if } f'(x_0) \leq 0.0 \\ x_1(n+1) &= \epsilon(n)/1.5 \qquad \text{, else} \end{aligned}$$

Typically the line search requires about 3 iterations when used with the Polak–Ribiere rule (chapter 5.6.1). Using a conjugate gradient method (chapter 5.6.2) a more conservative setting for the start values is necessary, resulting in some more iterations:

$$\begin{aligned} x_0(n+1) &= \epsilon(n) \\ x_1(n+1) &= 0.0 \end{aligned}$$

5.6.1 Polak–Ribiere method and line search

A. H. Kramer and A. Sangiovanni-Vincentelli are using the Polak–Ribiere method for calculating the momentum [Kramer et al., 1989]. The learning rate is adjusted by a line search to minimize E with respect to the current search direction:

$$\Delta w(n) = \epsilon(n) * (-\nabla E(n) + \frac{(\nabla E(n) - \nabla E(n-1))^T * \nabla E(n)}{\nabla E(n-1)^T * \nabla E(n-1)} * \Delta w(n-1))$$

This algorithm is free from any adjustable parameters. Results (see Figure 8) are similar to the evolution strategy and the angle driven adaptation. Unfortunately some more calculation time is necessary due to some iterations needed to adjust the learning rate by a line search (typically 2 - 3 iterations). Nevertheless no useful networks can be trained by this algorithm.

	Training Set		Testing Set	
	Error	Recog. rate	Error	Recog. rate
1	322.0	94.70	339.33	94.17

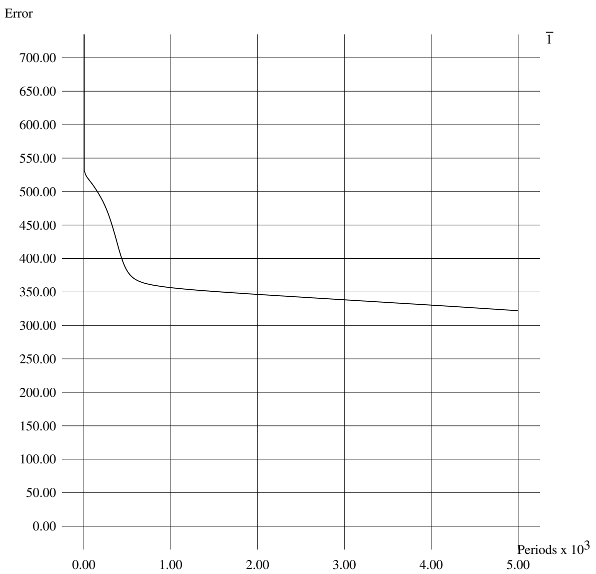

Figure 8: Polak–Ribiere method with line search

5.6.2 Conjugate gradient method and line search

J. Leonard and M. A. Kramer combine a conjugate gradient method and a line search strategy [Leonard et al., 1990]:

1. Calculate the exponential averaged gradient:

$$\nabla \bar{E}(n) = \nabla E(n) + \frac{\| \nabla E(n) \|^2}{\| \nabla E(n-1) \|^2} * \nabla \bar{E}(n-1)$$

For to start this calculation and every r updates simply use the actual gradient:

$$\nabla \bar{E}(n) = \nabla E(n) \text{ , if } n \bmod r = 0$$

2. Update the connections:

$$\Delta w(n) = \epsilon(n) * \nabla \bar{E}(n)$$

$\epsilon(n)$ is calculated by a line search to minimize E with respect to the actual search direction.

This algorithm is the most powerful one using global adaptations and batch mode updates (see Figure 9). Nevertheless results are very poor.

6 Local learning rate adaptations

Local adaptations are using independent learning rates for every adjustable parameter (every connection). Therefore they are able to find optimal learning rates for every weight.

6.1 Learning rate adaptation by sign changes

F. M. Silva and L. B. Almeida are using separate learning rates ϵ_{ij} for each connection. The adaptation of these learning rates is done by observing the signs of the last two gradients. As long as no change in sign is detected the corresponding learning rate is increased. If the sign changes the learning rate is decreased. This is the exact algorithm:

1. Choose some small initial value for every $\epsilon_{ij}(0)$.
2. Adapt the learning rates:

$$\begin{aligned} \epsilon_{ij}(n) &= \epsilon_{ij}(n-1) * u \qquad \text{, if } \frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) \geq 0 \\ \epsilon_{ij}(n) &= \epsilon_{ij}(n-1) * d \qquad \text{, else} \end{aligned}$$

3. Update the connections:

$$\Delta w_{ij}(n) = -\epsilon_{ij}(n) * (\frac{\partial E}{\partial w_{ij}} + \alpha * \Delta w_{ij}(n-1))$$

According to Silva and Almeida the choice of proper parameters u and d is easy as long as $u \approx \frac{1}{d}$ holds. The recommend values are $1.1 - 1.3$ or $0.7 - 0.9$ respectively. They also use a backtracking strategy which restarts an update step if the total error increases. For this restart all learning rates are halved.

Results (see Figure 10) are very impressing. Most of the runs result in better performance and reduced learning time.

		Training Set		Testing Set	
	r	Error	Recog. rate	Error	Recog. rate
1	50	244.9	94.57	267.92	93.84
2	100	252.6	94.54	278.28	93.84
3	1000	267.5	94.57	299.24	93.73
4	5000	269.1	94.54	297.83	93.70

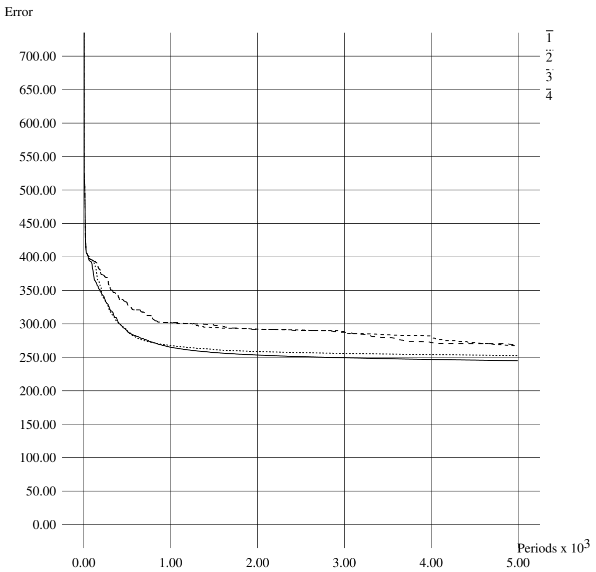

Figure 9: Conjugate gradient method and line search

	$\epsilon_{ij}(0)$	u	d	α	Training Set Error	Training Set Recog. rate	Testing Set Error	Testing Set Recog. rate
1	0.0001	1.1	$1/u$	0.0	25.9	99.60	105.54	98.45
2	0.0001	1.1	$1/u$	0.1	38.4	99.42	108.73	98.28
3	0.0001	1.1	$1/u$	0.9	536.7	92.47	473.87	92.71
4	0.001	1.1	$1/u$	0.0	<u>21.5</u>	<u>99.60</u>	97.40	98.37
5	0.001	1.1	$1/u$	0.1	33.9	99.34	<u>96.65</u>	<u>98.45</u>
6	0.01	1.1	$1/u$	0.0	568	92.47	500.00	92.71

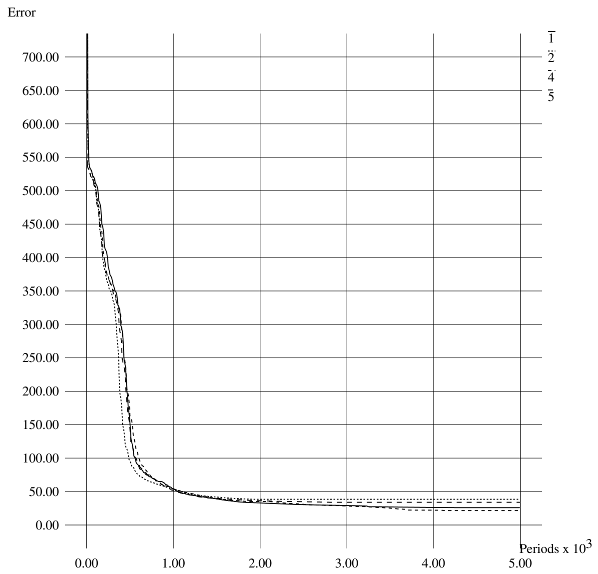

Figure 10: Learning rate adaptation by sign changes (Silva and Almeida)

T. Tollenaere's SuperSAB algorithm [Tollenaere, 1990] is quite similar to Silva and Almeida's approach. He has modified the update rule, so that updates which result in sign changes are undone:

3. Update the connections:

   If $\frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) \geq 0$:

   $$\Delta w_{ij}(n) = -\epsilon_{ij}(n) * \frac{\partial E}{\partial w_{ij}} + \alpha * \Delta w_{ij}(n-1)$$

   Else:

   $$w_{ij}(n+1) = w_{ij}(n-1)\ ;\ \Delta w_{ij}(n) = 0.0$$

Recommend values for u and d are 1.05 or 0.5 respectively. Instead of using global backtracking only some kind of local backtracking is used. Therefore weight updates which result in sign changes in the corresponding gradients are undone.

Results for the first 1000 training epoch are superior to Silva and Almeida's approach. Nevertheless the training became chaotic later on. In Figure 11 a typical training run is shown (momentum = 0.0, learning rate = 0.001).

Therefore we tried to combine SuperSAB with Silva and Almeida's backtracking strategy. Nevertheless to many backtracking steps are performed. One reason may be the local backtracking technique in the SuperSAB algorithm, which may result in an increased total error.

Our second try was to perform some weight decay in every update:

3. Update the connections:

   If $\frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) \geq 0$:

   $$\Delta w_{ij}(n) = -\epsilon_{ij}(n) * \frac{\partial E}{\partial w_{ij}} + \alpha * \Delta w_{ij}(n-1) - decay * w_{ij}(n)$$

   Else:

   $$w_{ij}(n+1) = -\Delta w_{ij}(n)\ ;\ \Delta w_{ij}(n) = 0.0$$

Unfortunately no suitable $decay$ factor could be found. The $decay$ factor just slightly changes the time, where the chaotic behaviour starts. Therefore one may suggest that it's not a problem with to big connection strength rather the learning rates may grow to much. So we used an upper limit for the learning rates:

2. Adapt the learning rates:

   $$\begin{aligned} \epsilon_{ij}(n) &= \epsilon_{ij}(n-1) * u \qquad &&\text{, if } \frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) \geq 0 \wedge \\ & &&\epsilon_{ij}(n-1) \leq \epsilon_{max} \\ \epsilon_{ij}(n) &= \epsilon_{ij}(n-1) * d \qquad &&\text{, else} \end{aligned}$$

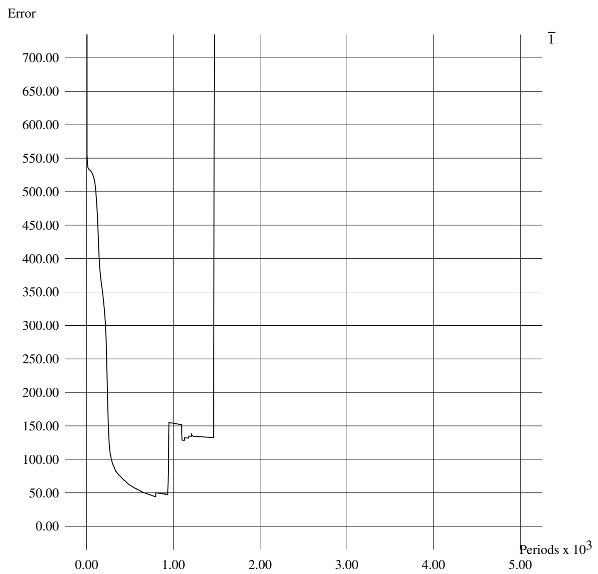

Figure 11: SuperSAB

By using a proper upper limit ϵ_{max} the algorithm behaves perfectly all over the training period. Results are similar to Silva and Almeida's approach but even somewhat better in the early training phase (see Figure 12). Figure 13 illustrates dependencies on the momentum factor.

						Training Set		Testing Set	
	$\epsilon_{ij}(0)$	ϵ_{max}	u	d	α	Error	Recog. rate	Error	Recog. rate
1	0.001	0.1	1.05	0.5	0.0	57.9	99.15	111.12	98.02
2	0.001	1.0	1.05	0.5	0.0	40.4	99.39	108.72	98.25
3	0.001	10.0	1.05	0.5	0.0	32.9	99.47	<u>105.31</u>	<u>98.42</u>
4	0.001	100.0	1.05	0.5	0.0	<u>29.2</u>	<u>99.55</u>	122.44	98.19
5	0.001	1000.0	1.05	0.5	0.0	107.8	98.49	153.94	97.70

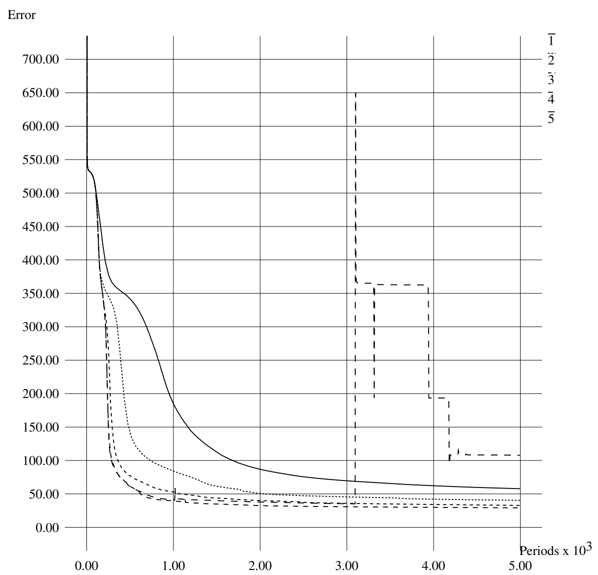

Figure 12: SuperSAB with limited learning rates

6.2 Delta–Bar–Delta Technique

Robert A. Jacobs also uses a local learning rate adaptation [Jacobs, 1988]. In contrary to the former approaches his delta–bar–delta algorithm controls the learning rates by observing the sign changes of an exponential averaged gradient. He increases the

	$\epsilon_{ij}(0)$	ϵ_{max}	u	d	α	Training Set Error	Training Set Recog. rate	Testing Set Error	Testing Set Recog. rate
1	0.001	100.0	1.05	0.5	0.0	29.2	99.55	122.44	98.19
2	0.001	100.0	1.05	0.5	0.1	30.6	99.47	123.59	98.07
3	0.001	100.0	1.05	0.5	0.9	41.0	99.31	122.73	98.28
4	0.001	10.0	1.05	0.5	0.0	32.9	99.47	<u>105.31</u>	<u>98.42</u>
5	0.001	10.0	1.05	0.5	0.1	<u>27.9</u>	<u>99.55</u>	107.71	98.13
6	0.001	10.0	1.05	0.5	0.9	30.0	99.47	128.30	97.72

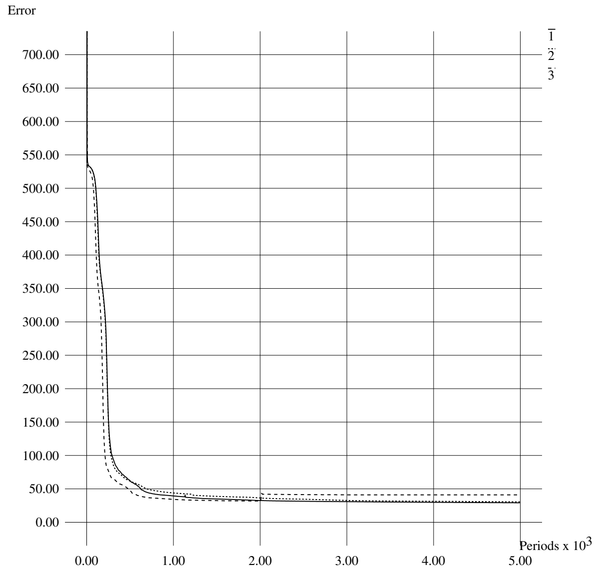

Figure 13: SuperSAB with limited learning rates

learning rates by adding a constant value instead of multiplying it:

1. Choose some small initial value for every $\epsilon_{ij}(0)$.

2. Adapt the learning rates:

$$\epsilon_{ij}(n) = \epsilon_{ij}(n-1) + u \qquad , \text{if } \frac{\partial E}{\partial w_{ij}}(n) * \bar{\bar{\delta}}_{ij}(n-1) > 0$$

$$\epsilon_{ij}(n) = \epsilon_{ij}(n-1) * d \qquad , \text{if } \frac{\partial E}{\partial w_{ij}}(n) * \bar{\bar{\delta}}_{ij}(n-1) < 0$$

$$\epsilon_{ij}(n) = \epsilon_{ij}(n-1) \qquad , \text{else}$$

   $\bar{\bar{\delta}}(n)$ denotes the exponential averaged gradient:

$$\bar{\bar{\delta}}_{ij}(n) = (1-\phi) * \frac{\partial E}{\partial w_{ij}}(n) + \phi * \bar{\bar{\delta}}_{ij}(n-1)$$

3. Update the connections:

$$\Delta w_{ij}(n) = -\epsilon_{ij}(n) * \frac{\partial E}{\partial w_{ij}}$$

Very different values are recommend for u $(5.0, 0.095, 0.085, 0.035)$. Jacobs uses $(0.9, 0.85, 0.666)$ for d and 0.7 for ϕ. Using $\phi = 0$ the algorithm becomes quite similar to Silva and Almeida's approach.

Results (see Figure 14) are quite good but worse than using Siva and Almeida's algorithm. In particular it's difficult to find a proper u. Small values may result in slow adaptations while big ones endanger the learning process.

6.3 RPROP

M. Riedmiller and H. Braun are using an adaptive version of the "Manhattan-Learning" rule [Riedmiller et al., 1992]. In contrast to all other described algorithms the "Manhattan-Learning" rule uses a fixed update step size not influenced by the magnitude of the gradient. Only the sign of the derivative is used to find the proper update direction. RPROP uses independent update step sizes Δ_{ij} for every connection. Further more these step sizes are adapted with respect to the sign of the actual and the last derivative. The step sizes are bound by upper and lower limits in order to avoid oscillation and arithmetic underflow of floating point values. Finally local backtracking is applied to those connections where sign changes of the derivative are detected:

1. Choose some small initial value for every update step size $\Delta_{ij}(0)$.

	$\epsilon_{ij}(0)$	u	d	ϕ	Training Set Error	Training Set Recog. rate	Testing Set Error	Testing Set Recog. rate
1	0.0001	0.0001	0.9	0.7	<u>51.6</u>	<u>99.20</u>	<u>110.64</u>	<u>98.37</u>
2	0.0001	0.001	0.9	0.7	295.4	94.90	316.59	94.31
3	0.0001	0.01	0.9	0.7	568.0	92.47	500.00	92.71
4	0.0001	0.1	0.9	0.7	568.0	92.47	500.00	92.71
5	0.001	0.001	0.9	0.7	317.8	94.70	335.71	94.19
6	0.001	0.01	0.9	0.7	505.1	92.50	445.34	92.71
7	0.001	0.1	0.9	0.7	568.0	92.47	500.00	92.71
8	0.01	0.01	0.9	0.7	568.0	92.47	500.00	92.71

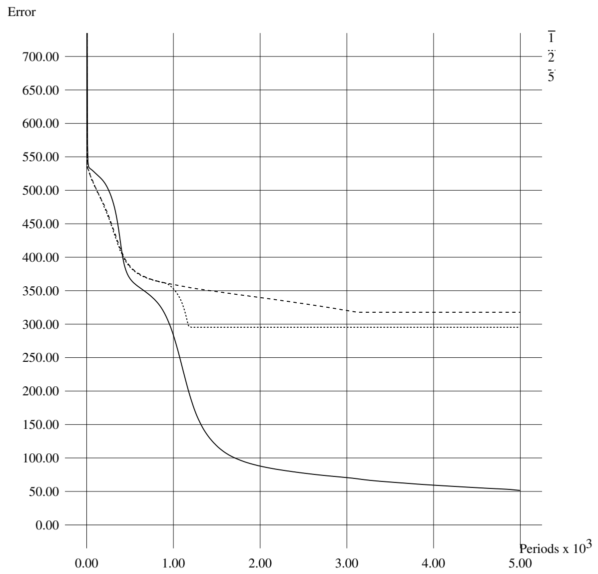

Figure 14: Delta–Bar–Delta

2. Adapt the step sizes:

$$
\begin{aligned}
\Delta_{ij}(n) &= \Delta_{ij}(n-1) * u && \text{, if } \frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) > 0 \\
\Delta_{ij}(n) &= \Delta_{ij}(n-1) * d && \text{, if } \frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) < 0 \\
\Delta_{ij}(n) &= \Delta_{max} && \text{, if } \Delta_{ij}(n) \geq \Delta_{max} \\
\Delta_{ij}(n) &= \Delta_{min} && \text{, if } \Delta_{ij}(n) \leq \Delta_{min}
\end{aligned}
$$

3. Update the connections:

   If $\frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) \geq 0$:

$$
\begin{aligned}
\Delta w_{ij}(n) &= -\Delta_{ij}(n) && \text{, if } \frac{\partial E}{\partial w_{ij}}(n) > 0 \\
\Delta w_{ij}(n) &= +\Delta_{ij}(n) && \text{, if } \frac{\partial E}{\partial w_{ij}}(n) < 0
\end{aligned}
$$

   Else:

$$
w_{ij}(n+1) = w_{ij}(n-1) \; ; \; \frac{\partial E}{\partial w_{ij}}(n) := 0.0
$$

Recommend values for the parameters are:

$$
\begin{aligned}
\Delta_{max} &= 50.0 \\
\Delta_{min} &= 0.000001 \\
u &= 1.2 \\
d &= 0.5
\end{aligned}
$$

RPROP is fastest training algorithm tested in this report (see Figure 15). Results are almost independent from the initial setting of the update step size. Networks with very good performance have been trained. According to training speed only Quickprop is comparable to RPROP.

6.4 Quickprop

This algorithm is a collection of different heuristics for optimizing backpropagation [Fahlman, 1988]. Having a closer look at the derivative $\frac{\partial E}{\partial w_{ij}}$ we have to notice that it's necessary to calculate $o_i * (1 - o_i)$. By using a sigmoid activation function this value is limited to the range [0.0 ... 0.25]. Unfortunately it tends to become very small if the output approaches 0.0 or 1.0. This may slow down a gradient descent. Therefore Fahlman modifies the calculation to $o_i * (1 - o_i) + 0.1$. The modified gradient based on this calculation is further denoted by $\frac{\partial E^*}{\partial w_{ij}}$. Further a new error function is used. If the absolute unit error becomes less than 0.1 the error is simply set to zero. By using

	$\Delta_{ij}(0)$	Training Set Error	Training Set Recog. rate	Testing Set Error	Testing Set Recog. rate
1	0.0001	52.0	98.81	155.12	97.02
2	0.001	25.99	99.58	123.42	97.93
3	0.01	26.0	99.52	129.13	97.93
4	0.1	29.1	99.47	120.73	98.02
5	1.0	28.4	99.50	129.84	97.93

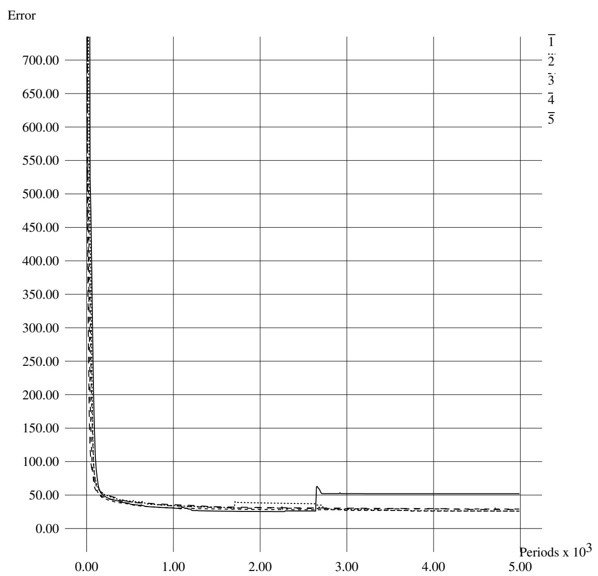

Figure 15: RPROP

this error criteria units having a quadratic error less than 0.01 aren't trained anymore. For to prevent weights from growing very large a small weight decay is used by further modifying the gradient:

$$\frac{\partial E^{**}}{\partial w_{ij}}(n) = \frac{\partial E^{*}}{\partial w_{ij}}(n) + decay * w_{ij}(n)$$

$$\Delta w_{ij}(n) = -\epsilon_{ij}(n) * \frac{\partial E^{**}}{\partial w_{ij}}(n) + \alpha_{ij}(n) * \Delta w_{ij}(n-1)$$

Basically the connection updates is calculated by the method of “False Position” (chapter 5.6), which is applied independently to every connection. The second term of the sum exactly calculates this value if the momentum in chosen in the following manner:

$$\widetilde{\alpha}_{ij}(n) = \frac{\frac{\partial E^{**}}{\partial w_{ij}}(n)}{\frac{\partial E^{**}}{\partial w_{ij}}(n-1) - \frac{\partial E^{**}}{\partial w_{ij}}(n)}$$

Nevertheless the calculated update has to be limited if the step computed by this formula is too large, infinite or uphill on the current gradient $\frac{\partial E^{**}}{\partial w_{ij}}(n)$:

$$\begin{aligned} \alpha_{ij}(n) = \alpha_{max} \qquad & \text{, if } \widetilde{\alpha}_{ij}(n) \text{ infinite} \\ & \vee\, \widetilde{\alpha}_{ij}(n) > \alpha_{max} \\ & \vee\, \widetilde{\alpha}_{ij}(n) * \Delta w_{ij}(n-1) * \frac{\partial E^{**}}{\partial w_{ij}}(n) > 0.0 \\ \alpha_{ij}(n) = \widetilde{\alpha}_{ij}(n) \qquad & \text{, else} \end{aligned}$$

A learning rate is still necessary to start the training or restart it after a 0.0 update. If the gradient and the last update have the same sign the learning rate is also used:

$$\begin{aligned} \epsilon_{ij}(n) = \epsilon_0 \qquad & \text{, if } \frac{\partial E^{**}}{\partial w_{ij}}(n) * \Delta w_{ij}(n-1) < 0.0 \\ & \vee\, \Delta w_{ij}(n-1) = 0.0 \\ \epsilon_{ij}(n) = 0 \qquad & \text{, else} \end{aligned}$$

Recommend values for the parameters are:

$$\alpha_{max} = 1.75$$

$$\epsilon_0 = 0.55$$

$$decay = 0.0001$$

Almost perfect results (see Figure 16) have produced by this algorithm. Quickprop's performance is close to rprop's results.

	ϵ_0	Training Set Error	Training Set Recog. rate	Testing Set Error	Testing Set Recog. rate
1	0.0001	33.8	99.5	123.38	97.99
2	0.001	<u>29.2</u>	<u>99.6</u>	119.28	98.10
3	0.01	36.8	99.4	<u>110.91</u>	<u>98.25</u>
4	0.1	568.0	92.5	500.00	92.71

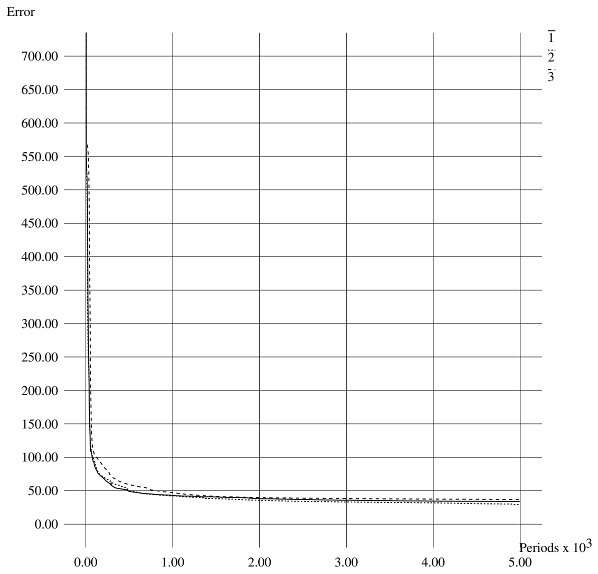

Figure 16: Quickprop

6.5 Cascade Correlation

Scott E. Fahlman and Christian Lebiere have presented a new learning architecture called cascaded correlation algorithm [Fahlman et al., 1990]. This algorithm differs in many ways from all other approaches. It begins with a minimal network, then automatically trains and adds new hidden units one by one, creating a multi-layer structure. Once a new hidden unit has been added to the network, its input-side weights are frozen. The hidden units are trained in order to maximize the correlation between the unit output and the output error. So a training cycle is divided into two phases. First the output units are trained to minimize the total output error. This training is done up to out_{max} epochs. In phase two $ncand$ candidate units are inserted having connections to every output unit and every previously inserted hidden unit. This units are trained up to hid_{max} epochs in order to correlate with the output error. The best of this candidate units became a new hidden unit, whereas the other ones are deleted. Next weights of the new unit get frozen. Now a new training cycle starts. Weights are adjusted by the quickprop update rule. Because all input connections of a hidden unit are frozen after the training of this unit, it's possible to store the activations of the hidden units over the entire training set. This can speed up the training. For further details of this algorithm see Fahlman et al. 1990 . This algorithm was included in our report, because it seems to be interesting to compare the results. Nevertheless comparison is quite difficult, because networks grow during the training and start with very little connections. Although hidden units connections get frozen after training. Figure 17 shows results in terms of training epochs. Nevertheless results became more impressing if we compare it in terms of updated connections. Therefore we have calculated the number epochs in conventional training having nearly the same number of updated weights (see Figure 18). As one can see this algorithm is superior to training algorithms using fixed topologies. Nevertheless the network performance could not be further improved. The algorithm was able to train networks with 100 % recognition rate with respect to the training set, if sufficient hidden units are used ($nhidden = 20$). Nevertheless generalization ability with respect to the testing set could not be improved. Most of the networks using 10 hidden units generalize as good or even better as those having 20 hidden units.

7 Conclusion

Table 2 shows a comparison of the best results of every training algorithm.

Many "optimized" algorithms failed in training the considered task, although most authors promised a algorithm superior to standard backpropagation. Many of these algorithms have only been tested by training tiny artificial tasks. These results cannot be transferred to more complicated training sets. Especially for these kind of training sets optimization is necessary, whereas it's of little importance to speed up XOR learning. Nevertheless most of the algorithms are superior to standard backpropagation running in batched mode. On the other hand backpropagation updating the connections after every pattern presentation outperforms all global adaptive learning algorithms. Algorithms using local adaptation strategies greatly reduce the training time and also improve the network performance. In terms of learning speed RPROP and Quickprop seems to be

					Training Set		Testing Set	
	$ncand$	out_{max}	hid_{max}	$nhidden$	Error	Recog. rate	Error	Recog. rate
1	8	50	50	10	38.44	99.47	99.56	98.25
2	8	50	50	20	14.20	99.81	123.79	98.10
3	8	50	100	10	23.51	99.71	91.50	98.42
4	8	50	100	20	0.83	100.00	104.84	98.25
5	8	100	50	10	40.79	99.36	99.36	98.34
6	8	100	50	20	0.82	100.00	101.36	98.48
7	8	100	100	10	10.37	99.84	122.32	97.96
8	8	100	100	20	6.55	99.89	142.69	97.84

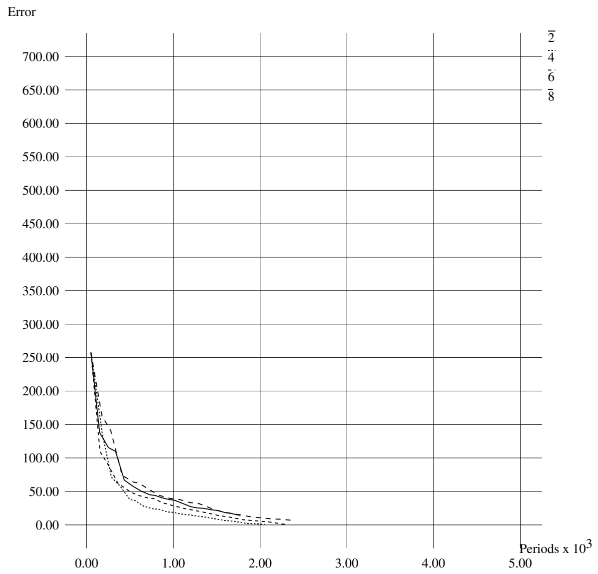

Figure 17: Cascade Correlation

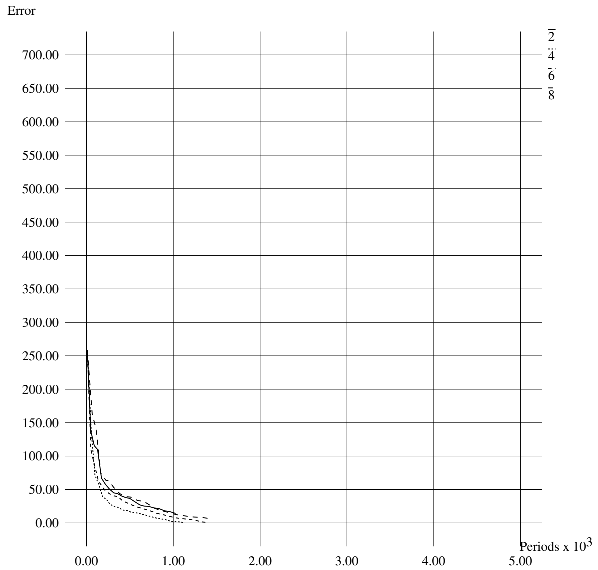

Figure 18: Cascade Correlation with rescaled x-axis

	Training Set		Testing Set	
Algorithm	Error	Recog. rate	Error	Recog. rate
Backprop	50.3	99.13	137.63	97.58
Backprop (batch mode)	461.8	92.63	414.34	92.85
Backprop (batch mode) + Eaton and Oliver	511.0	92.47	450.10	92.71
Backprop + Darken and Moody	44.2	99.20	126.84	97.90
J. Schmidhuber	91.5	98.36	163.54	97.23
R. Salomon	331.1	94.64	346.73	94.14
Chan and Fallside	317.6	94.67	335.47	94.17
Polak-Ribiere + line search	322.0	94.70	339.33	94.17
Conj. gradient + line search	244.9	94.57	267.92	93.84
Silva and Almeida	21.5	99.60	96.65	98.45
SuperSAB	27.9	99.55	105.31	98.42
Delta-Bar-Delta	51.6	99.20	110.64	98.37
RPROP	25.99	99.58	120.73	98.02
Quickprop	29.2	99.60	110.91	98.25
Cascade correlation 10 units	10.37	99.84	91.50	98.42
Cascade correlation 20 units	0.82	100.00	101.36	98.48

Table 2: Best results

superior to all other training algorithms using fixed topologies. Nevertheless Silva and Almeida's approach and SuperSAB have trained networks, which generalize a little better. The cascade correlation algorithm clearly outperforms all other approaches but is not directly comparable with them.

Most algorithms are using batched updates. Very little optimization is done on back-propagation updating connections with respect to ∇E_p. Further research in needed on this topic. There seems to be little influence on the generalization ability. Nevertheless generalization depends on the network topology, as the cascade correlation algorithm shows. Training a network with more and more hidden units just increases the approximation quality with respect to the learning set but doesn't improve the generalization behaviour.

8 References

Chan, L. W. and Fallside, F. : *An adaptive training algorithm for backpropagation networks*, Computer Speech and Language, Vol. 2, page 205-218,1987

Darken, C. and Moody, J. : *Note on Learning Rate Schedules for Stochastic Optimization*, Neural Information Processing Systems , Lippmann R. P. and Moody J. E. and Touretzky D. S. (Editors), page 832-838, 1991

Eaton, Harry A. C. and Olivier, Tracy L. : *Learning Coefficient Dependence on Training Set Size*, Neural Networks, Vol. 5, page 283-288, 1992

Fahlman, Scott E. : *An Empirical Study of Learning Speed in Back-Propagation Networks*, Technical Report CMU-CS-88-162, 1988

Fahlman, Scott E. and Lebiere, Christian : *The Cascade-Correlation Learning Architecture*, Neural Information Processing Systems 2, page 524-532, 1990

Hertz, John and Krogh, Anders and Palmer, Richard G. : *Introduction to theory of neural computation*, Addison-Wesley Publishing Company, ISBN 0-201-51560-1, 1991

Jacobs, Robert A. : *Increased Rates of Convergence Through Learning Rate Adaption*, Neural Networks, Vol. 1, page 295-307,1988

Joost, Merten and Werner, Randolf : *Algorithmen zur Optimierung neuronaler Merkmalsfilter*, Diplomarbeit, Universität Koblenz, 1991

Kramer, Alan H. and Vincentelli, A. Sangiovanni : *Efficient parallel learning algorithms for neural networks*, Advances in Neural Information Processing Systems I, Touretzky D. S. (Editor), page 40-48, 1989

Leonard, J. and Kramer, M. A. :*Improvement of the Backpropagation Algorithm for Training Neural Networks*, Computers Chem. Engng., Volume 14, No 3, page 337-341, 1990

Luenberger, David G. : *Introduction to linear and nonlinear programming*, Addison-Wesley, 1973

Rumelhart D.E., Hinton G.E. and Williams R.J., 1986 : *Learning internal representations by error propagation*, in Parallel Distributed Processing: *Explorations in the Microstructures of Cognition*, Vol.I, MIT Press, pp. 318–362

Robinson, T. and Fallside, F. : *A recurrent error propagation network speech recognition system*, Computer Speech and Language, Vol. 5, No 3, page 259-274, 1991

Riedmiller, Martin and Braun, Heinrich : *RPROP - A Fast Adaptive Learning Algorithm*, Technical Report (To appear in: Proc. of ISCIS VII), Universität Karlsruhe, 1992

Salomon, Ralf : *Adaptive Regelung der Lernrate bei back-propagation*, Forschungsberichte des Fachbereichs Informatik, Technische Universität Berlin, Bericht 1989/24, 1989

Salomon, Ralf : *Improved convergence rate of back-propagation with dynamic adaption of the learning rate*, Lecture Notes in Computer Science, PPSN I, Dortmund, page 269-273, 1990

Scalero, Robert S. and Tepedelenlioglu, Nazif : *A Fast Algorithm for Training Feedforward Neural Networks*, IEEE Transactions on Signal Processing, Vol. 40, No 1, page 202-210, January 1992

Schiffmann, W. and Joost, M. and Werner, R. : *Comparison of optimized backpropagation algorithms*, Proc. of the European Symposium on Artificial Neural Networks, ESANN '93, Brussels, page 97-104, 1993

Schmidhuber, Jürgen : *Accelerated Learning in Back-Propagation Nets*, Connectionism in perspective, Elsevier Science Publishers B.V. (North-Holland), page 439-445, 1989

Silva, Fernando M. and Almeida, Luis B. : *Speeding up Backpropagation*, Advanced Neural Computers, Eckmiller R. (Editor), page 151-158, 1990

Tollenaere, Tom : *SuperSAB: Fast Adaptive Backpropagation with Good Scaling Properties*, Neural Networks, Vol. 3, page 561-573, 1990