Speeding - up Backpropagation – A Comparison of Orthogonal Techniques

Marcus Pfister Raúl Rojas

Freie Universität Berlin
Fachbereich Mathematik – Institut für Informatik
Takustr. 9, 1000 Berlin 30.

May 10, 1993

Abstract

In recent years much effort has been spent trying to develop more efficient variations of the backpropagation learning algorithm. This has led to a combinatorial explosion of learning methods of which no detailed evaluation exists. We have analyzed the most important algorithms and extracted their minimal building blocks. By arranging these building blocks in different forms, and testing the resulting algorithms, we obtained new combinations which were benchmarked in a commercial workstation. Our results show which factors are responsible for the increased speed-up of the tested algorithms. These results could lead to better learning methods for neural networks.

1 Introduction

The basic backpropagation correction step for a weight w_i in a multilayer neural network is given by

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} \tag{1}$$

where E denotes the quadratic error function of the network and γ a learning constant. Backpropagation is thus just a gradient descent method for the optimization of the networks weights. By taking into account previous correction steps, the k-th correction can be rewritten as

$$\Delta w_i^{(k)} = -\gamma \frac{\partial E}{\partial w_i} + \alpha \Delta w_i^{(k-1)}. \tag{2}$$

Since in the backpropagation algorithm the partial derivative $\partial E/\partial w_i$ is computed as $\partial E/\partial w_i = -\delta_j o_i$, where $\delta_j o_i$ is the backpropagated error up to neuron j, and o_i is the output of neuron i in the feedforward step, (2) can be rewritten as:

$$\Delta w_i^{(k)} = \gamma \delta_j o_i + \alpha \Delta w_i^{(k-1)}.$$

Different learning methods modify this equation in different ways. Figure 1 shows a diagram of the most common approaches.
Since neural networks are massively parallel learning models, we concentrated our attention on learning algorithms that preserve the network flow computational model. Some researchers have proposed algorithms, which they claim are superior to standard backpropagation, but that use weight-updating strategies based on non-local information. For all backpropagation variations

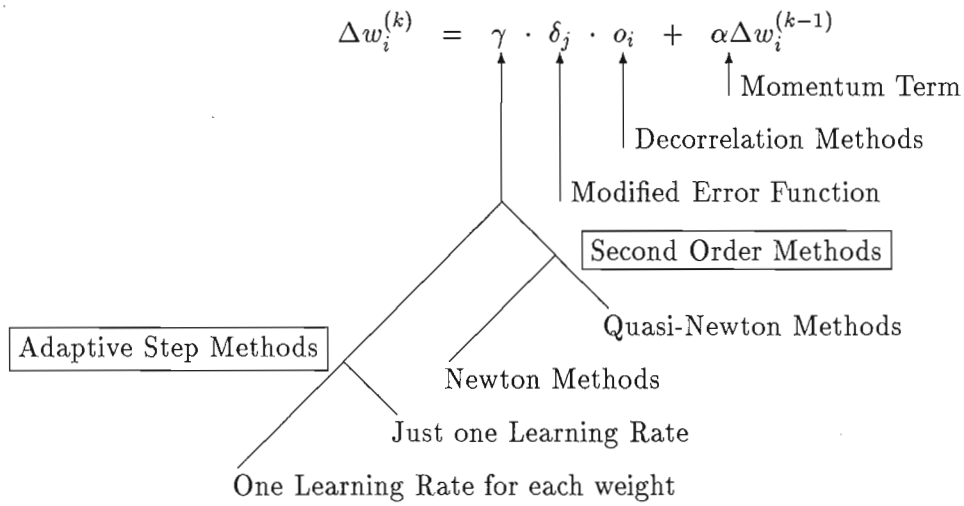


Figure 1: Common approaches for the acceleration of Backpropagation

examined in this paper, the neurons need no more information than the one they can get through their input- or output channels. No global information is involved.
After the discussion of the learning algorithms, a run-time comparison of the algorithms on the basis of selected benchmarks is made. This includes an estimation of the 'optimal' parameters of standard backpropagation for each benchmark and the comparison of this 'optimal standard' algorithm with other backpropagation variations.

2 Accelerating the Backpropagation Algorithm

In this section, various approches to improve backpropagation will be described. We divided them in the following classes, between which there is of course no sharp distinction:

1. Standard-Variations (Sec. 2.1)
2. Adaptive-Step Algorithms (Sec. 2.2)
3. Second-Order Algorithms (Sec. 2.3)

2.1 Standard – Variations

•*More than one pattern to learn: Batching vs. On-Line*
The first decision to be made is how to update the weights, if there is more than one pattern to learn. We usually have the choice of using *batch* or *on-line* backpropagation.

• *Introduction of a Momentum Term*
This is a method for increasing the learning rate while simultaneously avoiding oscilations. It works similar to a phisical *momentum*. The weight correction is modified to

$$\Delta W^{(k)} := -\gamma \nabla E(W^{(k)}) + \alpha \Delta W^{(k-1)}. \tag{3}$$

where k denotes the current iteration and $\alpha < 1$ a new real parameter.

• *Using bipolar- instead of binary vectors*
Another simple approach is to train the network with bipolar vectors, which consist of elements equal to -1 or 1. Two arbitrary bipolar vectors are orthogonal, and thus decorrelated, with a probability, that increases with the dimension n of the vectors. This has a positive effect on the convergence of learning procedures for multilayer neural networks.

• *Handling flat spots of the error function*
The first derivative of the sigmoid goes to zero, as the output of the neuron goes to zero or one. Since the correction of the weights is proportional to these derivatives, they will then also go to zero. These regions of the error function are called the *flat spots*. The presence of these flat spots is one of the main reasons for the slow convergence of standard backpropagation. Several proposals have been made to solve this problem.

1. *Adding an offset to the derivative of the sigmoid*
   S. E. Fahlmann [Fah88] proposed making sure, that the sigmoid's derivative will never get too close to zero. His idea was to add a small constant $\epsilon > 0$ to the first derivative of the sigmoid, so that this expression is always greater than zero (at least ϵ). Good results were obtained with $\epsilon \approx 0.1$.

2. *Modification of the error function*
   K. Balakrishnan and V. Honavar [BH92] proposed a modification of the error function, to eliminate at least the flat spots in the output layer. Their approach is not to evaluate the error between the *output* of the output neurons $o_{pi}^{(2)}$ and the desired output t_{pi}, but between the *input* of the output neurons and their desired input, which is easy to compute as $s^{-1}(t_{pi})$. This avoids the calculation of the derivatives of the last layer's sigmoids.

2.2 Adaptive-step algorithms

The idea behind this kind of methods is to use *variable step sizes* instead of a constant learning rate γ for the weight-corrections. This stepsize changes with the shape of the error function.

• *The gradient reuse algorithm*
This algorithm was proposed by D. R. Hush and J. M. Salas [HS88]. The search direction $-\nabla E(W^{(k)})$ is followed in discrete steps $W^{(k_{i+1})} := W^{(k_i)} - \gamma^{(k)} \nabla E(W^{(k)})$, as long as the error function decreases. Then a new search direction $-\nabla E(W^{(k+1)})$ is computed. The stepsize $\gamma^{(k)}$ is tuned that a reuse rate $i \approx 10$ can be expected.

• *The dynamic adaption algorithm*
R. Salomon suggested a similar procedure, but taking only two points along the search direction

and adjusting the learning parameter $\gamma^{(k)}$ dynamically [Sal92]. This means that for a given search direction $d = \nabla E(W)$ and a given learning parameter $\gamma^{(k)}$ the points $W^{(k-1)} - d \cdot \gamma^{(k)}\zeta$ and $W^{(k-1)} - d \cdot \gamma^{(k)}/\zeta$ are examined. The point $W^{(k+1)}$, that causes the smallest error and the corresponding new learning rate are chosen.

• *The Delta-Bar-Delta algorithm*
This algorithm, developed by R. A. Jacobs [Jac88], uses different learning rates for every single weight, which are adapted at each iteration. This approach reflects the idea that the slope of the error surface might differ considerably, depending on the weight directions.

• *The extended Delta-Bar-Delta algorithm*
A. A. Minai and R. D. Williams found that the Delta-Bar-Delta algorithm has several drawbacks [MW90]. The most important one is, that the introduction of a momentum term, which is a rather elegant method of speeding standard backpropagation up, sometimes causes the Delta-Bar-Delta algorithm to diverge. So an adaptivly changing momentum is introduced.

2.3 Second-order algorithms

The idea behind these methods is that the error function $E(W)$ is approximated locally by a quadratic function, its truncated Taylor series

$$E(W^{(k)} + h) \approx E(W^{(k)}) + \nabla E(W^{(k)})^T h + \frac{1}{2} h^T \nabla^2 E(W^{(k)}) h,$$

where $\nabla^2 E(W)$ is the second derivative of $E(W)$, the *Hessian matrix.*

• *Quickprop*
Quickprop is an improvement to backpropagation, proposed by S. E. Fahlmann [Fah88]. Quickprop is based on two assumptions: First, the error function $E(W)$ is a parabola, whose arms open upward and second, the change in the slope of the error curve, as seen by each weight, is not affected by all the other weights that are changing at the same time. Now if the error function is quadratic in each direction and is not affected by other weights, the first derivative has to be linear, and the minimum along this direction can be found easily.

• *Extended Quickprop*
M. Fombelida and J. Destiné merged the Extended Delta-Bar-Delta and the Quickprop Algorithm [FD92]. The idea was to introduce the adaptive learning rate of the Extended Delta-Bar-Delta Algorithm into Quickprop, which uses just an adaptive momentum rate.

3 Orthonormalisation and Decorrelation of the Training-Set

The positive effect of uncorrelated input data may be visualized as a change of the error surface, which has to be climbed down. Since there is a relation (or *duality*, see [Roj93]) between input- and weight space, orthogonalization of the input data has also some kind of orthogonalizing effect on the error function in weight space. This means that small angles between different steps of the error function are increased, which has a 'rounding' effect on the error surface. Small narrow oval valleys become more symmetric now and much easier to descend.

• *Principal component ananlysis: Sanger's and Oja's Rule*
Principal component analysis (PCA) is in fact some kind of data compression. The objective is to find m orthogonal vectors out of a set of n-dimensional input vectors, that account for as much as possible of the data variance [Roj93]. The input data is projected into an m-dimensional subspace. Because of the possible reduction in dimensionality (and thus, reduction in the number of weights) this makes the data much easier to handle.

• *Adaptive data-decorrelation*
F. M. Silva and L. B. Almeida have proposed a different mrthod of data decorrelation and orthonormalisation [AS91]. They employ layers of linear associators after the input- and hidden layer, which have as much neurons as the input- rsp. hidden layer. These layers perform a linear transformation, such that the output is uncorrelated and normalized. These additional layers are trained with an unsupervised learning rule, either in batch or in on-line mode.

4 Simulation Results

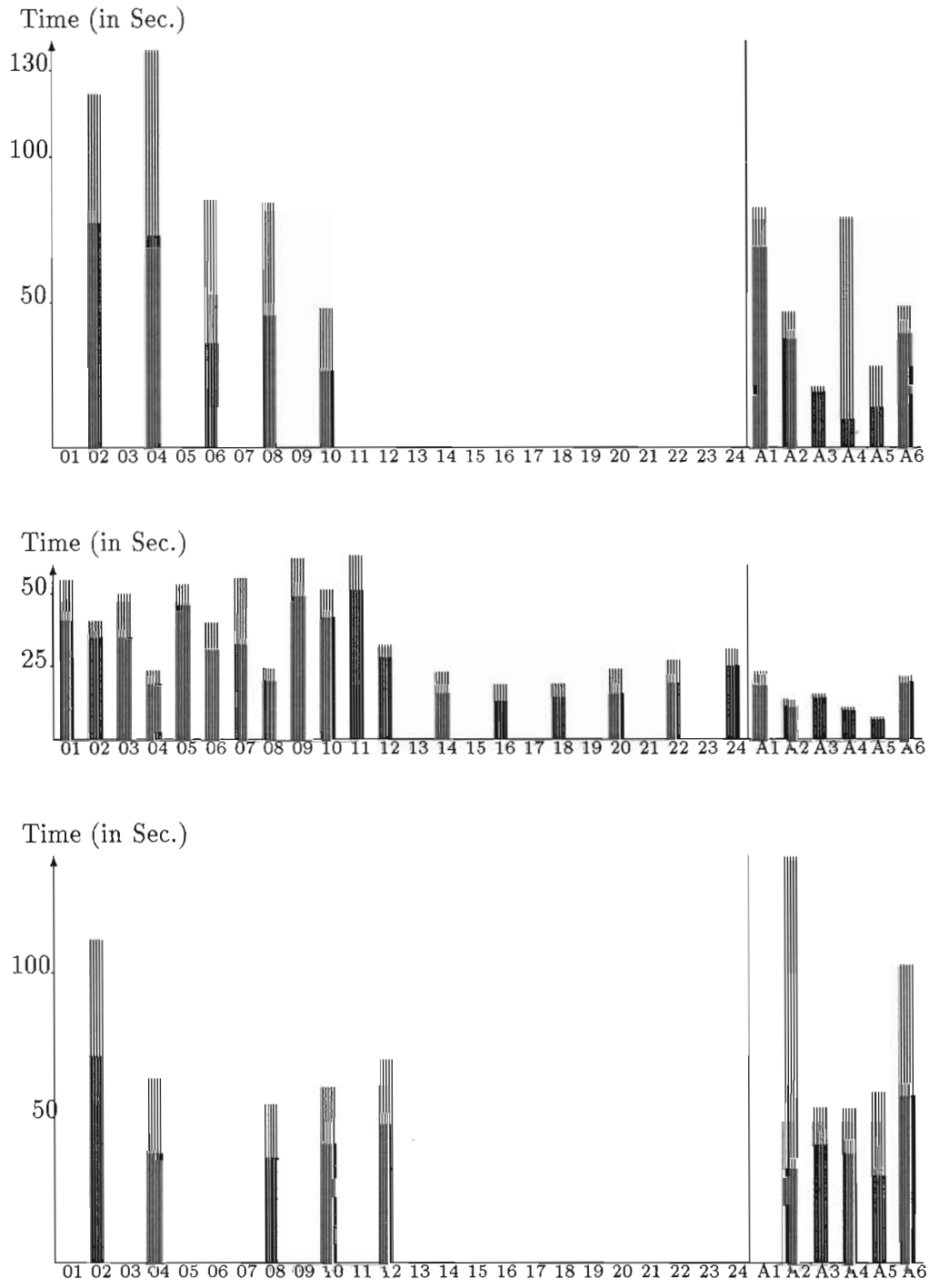
4.1 The benchmarks used in this paper

The problem of selecting adequate benchmarks to measure the convergence speed of a learning algorithm is a rather difficult one. A learning algorithm, that performes very fast on a certain problem may completely fail in another. We need a set of carefully choosen benchmarks, hopefully representative enough of 'real world' applications, to investigate what a certain algorithm is able to do. We chose the following ones:

	Benchmarks	Netsize
1	A 4-bit parity problem.	4-4-1
2	A 10-5-10 encoder problem.	10-5-10
3	A 16-4-16 encoder problem.	16-4-16
4	Clustering of the 16×16-square.	8-10-3
5	A classification task whith highly correlated input data.	5-10-3

4.2 Runtime Comparison

A runtime comparison with all the algorithms described in section 2.2 and 2.3 was made. We also tested the behavior of all algorithms when the standard modifications described in section 2.1 were applied. This means that each algorithm was run in 24 different modes.
Each algorithm was started 10 times. If no solution was reached within 1000 iterations (rsp. 2000 for benchmark 1 and 4), the algorithm was declared to have failed to converge. The average runtime (grey bars) as well as the fastest results obtained are (black bars) reported for those algorithms, which converged in more than 50% of then trials. The diagram in Figure 2 shows the reults for the last three benchmarks. The numbers 01 – 24 denote the different standard variations. For the other algorithms, the results for the best combination of the standard variations were reported.



The numbers 01–24 denote the different standard Variations		
A1: Grad. Reuse	A2: Delt.-B.-Delt.	A3: Ex. Delt.-B.-Delt.
A4: Dynam. Adapt.	A5: Quickprop	A6: Ex. Quickprop

Figure 2: Results for the benchmarks three, four and five.

4.3 Conclusions

- **Standard variations**

  • **Bipolar vectors** seem to have a major influence on the convergence speed of all learning procedures tested. Almost all benchmarks (except for the encoder problems, since their input is already orthonormal), were learned faster, sometimes significantly, when the network was trained with bipolar vectors, no matter which algorithm was used.

  • **Adding an offset to the sigmoid's derivative** speeds the algorithms up in most cases. Exceptions are those problems, which require a fine tuning of the weights. In these cases, in which the error function seems to have rather steep slopes anyway, the offset can cause the weight updating algorithms to perform rather wild jumps.

  • **Decorrelation Algorithms** also seem to have a major influence on the convergence of the learning procedures. Uncorrelated data is in fact much easier to learn than correlated data. The speedup obtained with those decorrelation algorithms may even be bigger, if for example a network has to learn large sets of highly correlated experimental data. This can easily happen in real world applications.

  • **The use of the modified error function.** Although it sounds like a good idea, the modified error function only seems to provide speedups for smaller, less complex tasks, such as the 10-5-10 encoder or the recognition of the shading of the 16×16-square. For more complex tasks, the modified error function rathe slows down the convergence, or even causes divergence of the algorithm.

- **Weight updating strategies**

  • **The gradient reuse algorithm** was only faster than the standard algorithm for small problems, like the encoder problems or the recognition of the shading of the 16×16 square. For more complex problems it was either slower than standard backpropagation (like for the 4-bit parity problem) or did not converge at all (like for the correlated cluster problem).

  • **The delta-bar-delta algorithm** only performed well for the 4-bit parity problem. It seems, that a fine tuning of the parameters $\kappa\phi$ and ψ, on which the weight-updating depends, is required. This requires, on the other hand, quite a few tests to tune them, which again takes a lot of time.

  • **The extended delta-bar-delta algorithm** has similar problems as the delta bar delta algorithm, although it performes better in general, and significantly better for the recognition of the correlated cluster. But here as well, there are a lot of parameters which have to be chosen carefully.

  • **The dynamic adaption algorithm.** Although the algorithm performs very well for all benchmarks, it seems that the normalization of the gradient, which the original algorithm requires, has a bigger influence on the convergence of the algorithm than it was reported by Salomon. Theoretically it makes no difference, but in practice, if a large learning parameter and a steep descent come together, the algorithm may overshoot any minimum and get stuck in very flat spots far away from any solution. Besides this problem, the dynamic adaption has the big advantage, that there is no parameter, which needs to be tuned.

• **Quickprop** also performed very well for all benchmarks. The algorithm has just one important parameter involved, the 'maximum growth factor μ', which almost needs no tuning. Experiments show, that it is rather to be choosen too small than too big, values of about 1.3 to 1.7 will do well for any problem.

• **Extended Quickprop** was almost a complete failiure. It was often slower than the standard algorithm (or at least not significantly faster) and always slower than any other algorithm, except for the gradient reuse algorithm. This may again have the reason in the fact that there are quite a lot of parameters involved, which need to be tuned. In our opinion, an algorithm which has many parameters of great influence, which have to be tuned finely and which are different for each problem, is not very good anyway.

5 Future Work

We are now working on an implementation of the faster variations discovered in this study on a neural computer with 256 processors (CNAPS). We expect to have first results in a few weeks.

References

[AS91] L. B. Almeida and F. M. Silva. Speeding-up backpropagation by data orthonormalisation. In T. Kohonen, K. Mäkisara, O. Simula and J. Kangas, editor, *Artificial Neural Networks*, pages 1503–1507, Amsterdam, 1991. North Holland.

[BH92] K. Balakrishnan and V. Honavar. Improving convergence of backpropagation by handling flat spots in the output layer. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks*, pages 1003–1009, Amsterdam, 1992. North Holland.

[Fah88] S. E. Fahlmann. Faster-learning variations on back-propagation. In D. Touietzky and T. Sejnowski G. Hinton, editors, *Proceedings of the '88 Connectionist Models Summer School*, pages 38–51. Carnegie-Mellon-University, 1988.

[FD92] M. Fombellida and J. Destine. The extended quickprop. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks*, pages 973–977, Amsterdam, 1992. North Holland.

[HS88] D. R. Hush and J. M. Salas. Improving the learning rate of back-propagation with the gradient reuse algorithm. In *Proceedings of the IEEE 1st International Conference on Neural Networks*, volume 1, pages 441–447, 1988.

[Jac88] R. A. Jacobs. Increased rates of convergence through learning rate adaption. *Neural Networks*, 1:295–307, 1988.

[MW90] A. A. Minai and R. D. Williams. Backpropagation heuristics: A study of the extended delta-bar-delta algorithm. In *Proceedings of the IEEE 1st International Conference on Neural Networks*, volume 1, pages 595–600, 1990.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing*. MIT Press, 1986.

[Roj93] R. Rojas. *Theorie der Neuronalen Netze – Eine systematische Einführung*. Springer, Berlin, March 1993.

[Sal92] R. Salomon. *Verbesserung konnektionistischer Lernverfahren, die nach der Gradientenmethode arbeiten*. PhD thesis, TU Berlin, November 1992.