Hybrid Learning Algorithms for Neural Networks – The adaptive Inclusion of Second Order Information

Marcus Pfister Raúl Rojas

Abstract

Since the introduction of the backpropagation algorithm as a learning rule for neural networks much effort has been spent trying to develop faster alternatives. This is e.g. done by using adaptively changing learning rates or exploiting second order information of the error surface. These optimization strategies are fixed once chosen, so if the heuristic does not fit the actual shape of the error surface, the computed weight changes will be far from the optimal ones.

In this paper we propose two *hybrid learning algorithms*, which dynamically switch between different optimization strategies. The algorithms basically use adaptive step sizes for the weight changes, but adaptively include second order information if a valley of the error function is reached.

The proposed hybrid algorithms, as well as standard backpropagation and three other known fast learning algorithms, were implemented on a SIMD neurocomputer, Adaptive Solutions CNAPS, and benchmarked against the Carnegie-Mellon benchmarks.

1 Introduction

Backpropagation, the learning algorithm for feed-forward neural networks rediscovered by Rumelhart *et al* [RHW86], updates the networks weights w_{lij} in each iteration k using $\Delta w_{lij}^{(k)} = \gamma \cdot \nabla E(w_{lij}^{(k)})$, where γ is a learning constant and $\nabla E(W)$ is the gradient of the quadratic error function $E(W)$. Improvements to the basic gradient descent method have been the subject of much research. A summary can be found in [PR93, SJW92]. These approaches can be classified into three classes, depending on their basic strategy. **Standard variations** include simple modifications of the algorithm, like the use of a momentum term [RHW86], adding an offset to the sigmoid derivative [Fah88] or preconditioning methods like decorrelation of the input data [PR93]. **Adaptive step methods** use a *variable* stepsize which is heuristically adapted to the error surface. There may either be one global learning rate for all weights, or individual learning rates for each single weight. **Second order methods** compute a quadratic approximation of the error surface, which is then minimized in order to reach the minimum of the actual error function iteratively.

The problem with these algorithms is that the chosen heuristic to approximate the error surface may not correspond to the *actual* shape. So, if a weight correction is computed on the basis of a second order approximation in nonquadratic regions, or if in locally quadratic regions second order information is neglected, only poor improvements can be expected for this step.

In this paper we propose learning algorithms which dynamically include second order information. These algorithms, which we call *hybrid algorithms* are based on Rprop [RB94], a 'Manhattan Learning' algorithm. In minimum regions, were the error surface is assumed to be locally quadratic, second order information is introduced by using either one dimensional secant steps (as in Quickprop [Fah88]), or by evaluating the diagonal terms of the hessian matrix.

The hybrid algorithms, as well as three other fast learning algorithms, were implemented on Adaptive Solutions CNAPS, a SIMD computer with 256 processors. On this machine we benchmarked the algorithms against four of the Carnegie-Mellon 'real-wold' benchmarks.

2 The RPROP Learning Algorithm

The Rprop (*resilient backpropagation*) algorithm, developed by M. Riedmiller and H. Braun [RB94], is an adaptive step algorithm using independent learning rates for each weight. The weights are updated by just taking care of the *sign* of the gradients components $\nabla E(w_{lij})$, not of its *magnitude* (*Manhattan Learning*), which very well handles the problem of flat spots or very steep descents of the error function. To start the process, a gradient step is taken, then the learning rates are increased, as long as the projection of the gradient in each coordinate axis points to the same direction as the previous one, since then a minimum lies ahead. If the sign of the gradient changes, which indicates that a minimum has been overlooked, the learning rate is cut to half and the weight w_{lij} for which the sign of the gradient has changed, is not changed in this iteration. This gives the following pseudocode for Rprop:

Step 1: Update the individual learning rates

$$\gamma_{lij}^{(k)} = \begin{cases} MIN(u \cdot \gamma_{lij}^{(k-1)},\ \gamma_{max}) & \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) > 0 \\ MAX(d \cdot \gamma_{lij}^{(k-1)},\ \gamma_{min}) \quad \text{if} & \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) < 0 \\ \gamma_{lij}^{(k-1)} & \text{otherwise} \end{cases}$$

Step 2: Update the weights

$$w_{lij}^{(k+1)} = \begin{cases} w_{lij}^{(k)} - \gamma_{lij}^{(k)} \cdot SIGN(\nabla E(w_{lij}^{(k)})) & \text{if} \quad \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) \geq 0 \\ w_{lij}^{(k)} & \text{otherwise} \end{cases}$$

IF $(\nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) < 0)$ SET $\nabla E(w_{lij}^{(k)}) := 0$.

M. Riedmiller and H. Braun propose $u = 1.2$ and $d = 0.5$ as default values.

3 Secant Methods and Quickprop

Secant methods can be used to approximate second derivatives by using only first order gradient information. For a one dimensional function $\hat{E}(w)$, the second derivative $\hat{E}''(w)$ can be approximated with the slope of the secant through the values of the first derivative in two near points w_{k-1} and w_k by a finite difference step $\hat{E}''(w) \cdot (w_k - w_{k-1}) \approx \hat{E}'(w_k) - \hat{E}'(w_k)$. We can now minimize the one dimensional function $\hat{E}(w)$ by iteratively taking the steps

$$w_k = w_k + \frac{\hat{E}'(w_{k-1})}{\hat{E}'(w_k) - \hat{E}'(w_{k-1})}(w_k - w_{k-1}). \tag{1}$$

A method which updates the weights using individual one-dimensional secant steps in each weight direction w_{lij} is *Quickprop*, proposed by S.E. Fahlman [Fah88]. The idea is to minimize the error function independently in each weight direction by updating each individual weight w_{lij} by taking a step as described by (1). To start the process, or restart it for those weights which have previously taken a step of size zero and are now facing nonzero gradient components, a backpropagation step with a fixed learning rate γ is taken.

4 The Pseudo-Newton Diagonal Estimation Algorithm

As Quasi-Newton methods we denote those methods, which are *directly* derived from Newton's method and thus involve analytical evaluations of the Hessian matrix $\nabla^2 E(W^{(k)})$ or parts of it. One of these methods, the *The Pseudo-Newton Diagonal Estimation Algorithm*, has been proposed by S. Becker and Y. leCun [BlC88]. Their idea was simply to neglect the off-diagonal elements of the Hessian matrix, that is $\nabla^2 E(W) \approx diag(\ldots, \partial^2 E/\partial^2 w_{lij}, \ldots)$. This leads to the following learning rule:

$$\Delta w_{lij}^{(k)} := -\frac{1}{|\partial^2 E/\partial^2 w_{lij}| + \mu} \cdot \nabla E(w_{lij}^{(k)}) \tag{2}$$

where $\mu > 0$ is a small constant introduced to avoid that the denominator gets too close to zero.
The diagonal elements $\partial^2 E/\partial^2 w_{lij}$ of the Hessian matrix can be computed by an extended backpropagation algorithm (second order backpropagation) which needs, as usual backpropagation, only local information [Roj93b].

5 Hybrid Learning Algorithms

This new class of learning algorithms are designed by combining the best properties extracted from the previous described classes of backpropagation accelerations. We describe two strategies of designing such Hybrid Algorithms, both use the same first order method, the Manhatten method Rprop, to detect local minimum regions in specific weight directions. The point in which the proposed algorithms differ is the strategy used to minimize the valleys found by Rprop. Note that for both hybrid algorithms proposed no additional parameters have been introduced.

5.1 QRprop

The first algorithm we propose is obtained by adaptively switching between Rprop and local onedimensional secant steps. Since these secant steps are also extensively used in Quickprop we called the algorithm QRprop.
A first version of QRprop was described in [PR94], it was significantly improved meanwhile. The idea is still the same: the Rprop-strategy is used if two subsequent error function gradient components $\nabla E(w_{lij})$ have the same sign which guarantees a fast approaches to minimum regions. If the sign of the gradient $\nabla E(w_{lij})$ changes, we know that we have overjumped a local minimum in this specific weight direction. Then neither weights nor learning rates are changed, which is done in the next step (in contrary to the old version of QRprop) using a constrained secant approximation. This situation is illustrated in Figure 1 below. The pseudocode of the **improved version of QRprop** follows:

Step 1: Update the individual learning rates

IF $(\nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) = 0)$ THEN
  IF $(\nabla E(w_{lij}^{(k)}) \neq \nabla E(w_{lij}^{(k-2)}))$ THEN
    $quad_{lij} = MAX(d,\ MIN(1/u,\ |\nabla E(w_{lij}^{(k)})/(\nabla E(w_{lij}^{(k)}) - \nabla E(w_{lij}^{(k-2)}))|))$
  ELSE
    $quad_{lij} = 1/u$
  ENDIF
ENDIF

$$\gamma_{lij}^{(k)} = \begin{cases} MIN(u \cdot \gamma_{lij}^{(k-1)},\ \gamma_{max}) & \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) > 0 \\ \gamma_{lij}^{(k-1)} & \text{if} \quad \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) < 0 \\ MAX(quad_{lij} \cdot \gamma_{lij}^{(k-1)},\ \gamma_{min}) & \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) = 0 \end{cases}$$

Step 2: is identical to **Step 2** of Rprop.

The parameters d, u are those also used in RProp. The *lower limit* of the secant step is chosen to be d since experiments show that reducing the learning rates by a factor much smaller than d $(:= 0.5)$ causes some troubles, especially if the secant approximation is done in regions which do not fit the quadratic model to well. The *upper limit* $1/u$ has been chosen to combine two oppositional deliberations. On one hand we have to reduce the learning rate, since we are in a critical local minimum region and finer steps have to be performed. On the other hand the approximation may not be too good, especially if it is done for $\nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-2)}) > 0$, so the algorithm must have the chance to 'recover' as quick as possible. By choosing the upper limit $1/u < 1$ we guarantee that the learning rate is reduced but can also be increased to the old amount in just one iteration.

5.2 DERprop

The idea of the DERprop algorithms is similar to that of QRprop. The main difference is that second order information, which is included into the Manhattan method Rprop, is not approximated by secant steps but directly computed by evaluating the diagonal terms of $\nabla^2 E(W)$, hence the name of the algorithm which stands for *Diagonal Estimation Rprop.* Using the diagonal terms of the hessian matrix has the advantage that quadratic local minimum regions are not just recognized after they have been jumped over but eventually earlier, as illustrated in Figure 1.

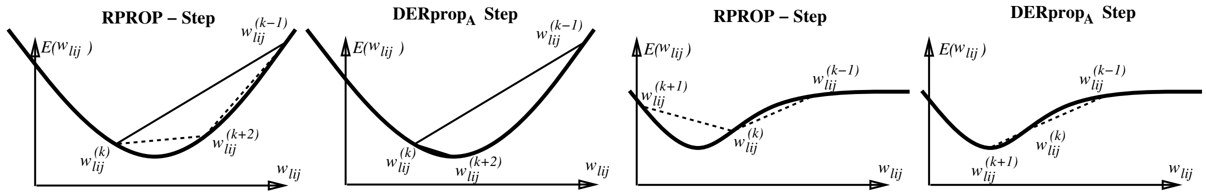

Figure 1: Comparison of Rprop- and a second order steps.

DERprop can thus be divided into two 'subalgorithms'. In the first one (DERprop$_A$), second order information is only used to minimize local minima as in QRprop and in the second one (DERprop$_B$), second order information is only used to avoid to overshoot local minima. We have again to constrain the second order steps. For DERprop$_B$ it should not be larger then an Rprop step to avoid too large steps and oscillations and it should not be smaller than the previous step so that, in case of a suboptimal step, the learning rate is at least not worsened. For DERprop$_A$ the same constraints for the second order steps as for QRprop are applied. The pseudocode of Step 1 of **DERprop**$_A$ and **DERprop**$_B$ becomes

Step 1 (DERprop$_A$): Update the individual learning rates

IF $(\nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) = 0)$ THEN
  IF $(\nabla E(w_{lij}^{(k)}) \neq \nabla E(w_{lij}^{(k-2)}))$ THEN
    $quad_{lij} = MAX(d,\ MIN(1/u,\ |\nabla E(w_{lij}^{(k)})/(\nabla E(w_{lij}^{(k)}) - \nabla E(w_{lij}^{(k-2)}))|))$
  ELSE
    $quad_{lij} = 1/u$
  ENDIF
ENDIF

$$\gamma_{lij}^{(k)} = \begin{cases} MIN(u \cdot \gamma_{lij}^{(k-1)},\ \gamma_{max}) & \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) > 0 \\ \gamma_{lij}^{(k-1)} & \text{if} \quad \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) < 0 \\ MAX(quad_{lij} \cdot \gamma_{lij}^{(k-1)},\ \gamma_{min}) & \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) = 0 \end{cases}$$

Step 1 (DERprop$_B$): Update the individual learning rates

IF $(\nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) = 0)$ THEN
  IF $(\nabla^2 E(w_{lij}^{(k)}) > 0$ THEN
    $quad_{lij} = MAX(\gamma_{lij}^{(k-1)},\ MIN(u \cdot \gamma_{lij}^{(k-1)},\ |\nabla E(w_{lij}^{(k)})/\nabla^2 E(w_{lij}^{(k)})|))$

ELSE

$quad_{lij} = u \cdot \gamma_{lij}^{(k-1)}$

ENDIF

ENDIF

$$\gamma_{lij}^{(k)} = \begin{cases} MAX(MIN(quad_{lij},\ \gamma_{max}),\ \gamma_{min}) & \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) > 0 \\ MAX(d \cdot \gamma_{lij}^{(k-1)},\ \gamma_{min}) & \text{if} \quad \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) < 0 \\ \gamma_{lij}^{(k-1)} & \nabla E(w_{lij}^{(k)}) \cdot \nabla E(w_{lij}^{(k-1)}) = 0 \end{cases}$$

For both algorithms **Step 2:** is identical to **Step 2** of Rprop and Qrprop, as are the parameters d and u. The entire DERprop algorithm is simply the combination of DERprop$_A$ and DERprop$_B$.

6 Numerical Simulations

To have the ability to benchmark our algorithms against large and complex problems we implemented them on a special SIMD neurocomputer, Adaptive Solutions' CNAPS with 256 *processor nodes*. As benchmarks we used the *Carnegie-Mellon* ones, which can (as well as further descriptions and references) be achieved via anonymous ftp from `ftp://ftp.cs.cmu.edu/afs/cs/project/connect/bench/*`.
The first benchmark was NETtalk. We used a set of 200 words (1513 examples), cleared of contradictions, which had to be learned by a 182-80-57 network. The second task was the prediction of the secondary structure of certain globular proteins. A NETtalk like approach (140-80-3 network) was used to learn 10 proteins (1146 examples). The third task was to discriminate between sonar signals bounced off a mine and those bounced off a roughly cylindrical rock. The data set consists of 104 examples, which had to be learned by a 60-24-2 network. The last problem was a small speech recognition problem were recorded examples of english vowels had to be discriminated. The set consists of 528 (= 11 Vowels $\times$ 48 speakers) examples, which had to be learned by a 10-88-11 network.
Standard backpropagation as well as three fast backpropagation-variations were implemented on the CNAPS computer and compared to our hybrid algorithms. For each problem and algorithm 50 trials starting from random weights $w_{lij} \in (-0.3, 0.3)$ were made. Training was stopped if the maximum individual error was smaller than 5% or no solution was reached after 5000 iterations for the vowels and 1000 iterations for the other problems. The sonar signals and vowels input data was transformed to lie in $(-1, 1)$ and so was the hidden layers output for all problems. In table 6 the results are given.

	Sonar Signals			**Vowels**			**NETtalk**			**Proteins**		
Algorithm	Conv	**Aver**	StDr	Conv	**Aver**	StDr	Conv	**Aver**	StDr	Conv	**Aver**	StDr
Batch BP	100%	**322.1**	77.0	0%	**0.0**	0.0	100%	**262.8**	16.6	90%	**366.0**	116.5
Quickprop	100%	**416.1**	32.0	90%	**3491.5**	458.2	100%	**489.7**	28.4	100%	**539.1**	21.8
Dyn. Adapt.	96%	**245.3**	69.0	84%	**2031.6**	810.7	48%	**281.3**	33.5	62%	**379.2**	159.8
Rprop	100%	**82.0**	8.8	96%	**1532.9**	679.0	100%	**108.7**	7.4	98%	**139.2**	41.1
QRprop	100%	**80.9**	7.4	98%	**1153.3**	555.8	100%	**107.0**	5.5	94%	**121.5**	29.6
DERProp A	100%	**80.4**	7.8	94%	**1301.3**	543.5	100%	**100.5**	5.4	100%	**118.0**	7.6
DERProp B	100%	**81.1**	9.0	94%	**1351.0**	754.8	100%	**106.1**	6.4	100%	**109.9**	8.4
DERProp	100%	**80.6**	7.4	98%	**1561.0**	908.6	100%	**99.3**	5.7	100.0	**123.9**	60.6

Table 1: Results for the runtime comparison

The **standard backpropagation algorithm** was run in batch mode. We had a learning rate $\gamma = 0.01$ and a momentum rate $\alpha = 0.9$ for the sonar signals, $\gamma = 0.05$ and $\alpha = 0.3$ for the vowels, $\gamma = 0.06$ and $\alpha = 0.3$ for NETtalk and $\gamma = 0.03$ and $\alpha = 0.3$ for the proteins. The **Dynamic Adaption algorithm** [PR93] was used adapting both learning and momentum rates. The gradient was not normalized to avoid the use of nonlocal information, therefore we had to constrain the learning and momentum rates. We had $\gamma = 0.05$ and $\alpha = 0.1$ as initial learning and momentum rates which were constrained with $\gamma_{max} = 0.2$, $\gamma_{min} = 0.005$ and $\alpha_{max} = 0.9$, $\alpha_{min} = 0.001$. We used a modification factor $\zeta = 1.3$. Choosing bigger ones led to problems including weight oscillations and divergency. We also used **Quickprop** [Fah88] to which we had to apply two small changes. The secant factor with which the momentum term is adapted was constrained to an absolute value 0.5. Originally the same learning rate γ is used both to bootstrap the algorithm as well as to accelerate it in downhill directions. We decided to have two different learning rates $\gamma_{ini} = 0.05$ and $\gamma_{add} = 0.001$, since the rather large initial learning rates needed to start the algorithm caused the weights to oscillate when used as accelerators. **Rprop, Qrprop** and **DERprop** were all implemented as described above. For all algorithms and all problems we had $\gamma_{ini} = 0.01$ as the initial learning rate and $\gamma_{max} = 0.1$ and $\gamma_{min} = 0.0003$ as its constraints. Choosing larger values for γ_{max} caused the fraction of divergent trials to increase rapidly, the value of γ_{min} is motivated by the machine precision. Besides that, we had $u = 1.2$ and $d = 0.5$.

7 Conclusion and Future Work

We propose in this paper QRprop and DERprop, two hybrid learning algorithms which adaptively include second order information into an adaptive step method. They combine the fast global convergence of Rprop with the good local properties of second order methods. As can be seen by the numerical simulations QRprop is up to 33% and the DERprop algorithms are up to 27% faster than Rprop, and all 'Xprop' algorithms are magnitudes faster than the other algorithms. It should be noted that for the vowels QRprop is much faster than the (theoretically) more accurate DERprop algorithms. Also DERprop$_A$ and DERprop$_B$ alone are often faster then their combination DERprop, so there seems to be a 'negative synergetic' effect.

The algorithms introduced are hybrid algorithms using a *local* strategy i.e. the adaptiveinclusion of second order information takes place for each weight independently. It could be object of further research to develop *global* strategies were more sophisticated second order methods like cg- or BFGS algorithms are included.

QRprop and DERprop are batch algorithms which means that the whole training set is presented before the weights are updated. For very large and redundant data sets this a waste of computational time, so effort should be spent to develop stochastic or on-line versions of hybrid algorithms.

References

[Ada92] Adaptive Solutions Inc., Beaverton OR. *CNAPS - Manual*, 1992.

[BlC88] S. Becker and Y. le Cun. Improving the convergence of backpropagation learning with second order methods. In *Proceedings of the '88 Connectionist Models Summer School*, pages 29–37, 1988.

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

[PR93] M. Pfister and R. Rojas. Speeding - up Backpropagation – A Comparison of orthogonal Techniques. In *Proceedings of the IJCNN '93 Nagoya, Japan*, pages 517–523.

[PR94] M. Pfister and R. Rojas. Qrprop – a hybrid learning algorithm which adaptively includes second order information. In *Proceedings of the 4. Dortmund Fuzzy Days*, pages 55–62, 1994.

[RB94] M. Riedmiller and H. Braun. RPROP – Description and Implementation Details. Technical Report. Universität Karlsruhe, 1994.

[Roj93a] R. Rojas. *Theorie der Neuronalen Netze – Eine systematische Einführung.* Springer, 1993.

[Roj93b] R. Rojas. Second Order Backpropagation – Efficient Computation of the Hessian Matrix for Neural Networks. Technical Report TR-93-057. ICSI, 1993.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In: D. E. Rumelhart and J. McClelland, eds., *Par. Dist. Processing.* MIT Press, 1986.

[SJW92] W. Schiffmann, M. Joost and R. Werner. Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons. Technical Report. Universität Koblenz, Institut für Physik, 1992.