S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with sec ond-order methods. In D. Touretzky, G. Hinton, and T Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 29–37, San Mateo, 1989. Morgan Kaufman. also published as techreport CRG-TR-88-5, Computer Science Dept, University of Toronto.

Improving the Convergence of Back-Propagation Learning with Second Order Methods

Sue Becker & Yann le Cun
Department of Computer Science, University of Toronto

Technical Report CRG-TR-88-5
Sept 1988

Requests for copies of this technical report should be addressed to:

The CRG technical report secretary
Department of Computer Science
University of Toronto
10 Kings College Road
Toronto M5S 1A4
CANADA

This research was supported by a grant to the Information Technology Research Center from the government of Ontario, and by grants to Geoffrey Hinton from E. I. Du Pont de Nemours & Company and from the National Science and Engineering Research Council of Canada.

This paper will appear in D. S. Touretzky, G. E. Hinton and T. J. Sejnowski (Eds.) *Proceedings of the 1988 Connectionist Summer School*, Morgan Kauffmann: Los Angeles, 1988.

Improving The Convergence of Back-Propagation Learning With Second Order Methods

Sue Becker, Yann le Cun
Department of Computer Science, University of Toronto
Toronto, Ontario, M5S 1A4. CANADA.

Abstract

Back-propagation has proven to be a robust algorithm for difficult connectionist learning problems. However, as with many gradient based optimization methods, it converges slowly. We describe an extension of the back-propagation algorithm which uses a simple approximation to the second derivative terms. This method is shown to reduce the required number of iterations to learn a random classification problem, with only a small increase in the complexity of each iteration.

The back-propagation learning algorithm for multilayer connectionist networks performs a gradient descent search in weight space for a minimum of some cost function C (which is frequently the mean squared error between actual and desired outputs). A general drawback of gradient-based numerical optimization methods is their slow convergence. In connectionist learning problems in particular, one typically starts a long way from the solution, and spends most of the time oscillating around "ravines" in weight space, because the gradient is sharp in some directions but shallow in others. Consequently, the learning parameters tend to be selected in an ad-hoc manner, according to the particular problem and the current performance of the network. Many numerical optimization methods, e.g. Newton's method, use the second derivative in addition to the gradient to determine the next step direction and step size, and converge quadratically when close to a solution of a convex function.

We explore the application of Newton's and other optimization methods towards improving the convergence of back-propagation. We then describe a computationally efficient, locally computable algorithm for incorporating approximate curvature information in back-propagation learning. Finally, we present results of simulations on a random four-way classification problem where this method is shown to learn somewhat faster than standard back-propagation.

1 Acceleration Methods

A variation of the back-propagation algorithm adds a "momentum" term to the weight update formula [Rumelhart *et al.*, 1986]. This acceleration method combines successive gradients by adding a fixed proportion of the previous weight change to the current one (ε is a learning constant and α, the momentum rate, is between 0 and 1):

$$\Delta w_{ij}(t) = -\varepsilon \frac{\partial C}{\partial w_{ij}(t)} + \alpha \Delta w_{ij}(t-1)$$

This tends to accelerate descent in steady downhill directions, while having a more "stabilizing" effect in directions which are oscillating in sign. This method is reported to accelerate learning once some stable descent direction has been found [Plaut *et al.*, 1986],[1] and has become a standard component of the back-propagation algorithm.

The method of conjugate gradients (e.g. [Gill *et al.*, 1981]) is related to the method of gradient descent with momentum, to the extent that both compute a linear combination of successive gradients in determining the next search direction. To minimize a quadratic function of n variables, the conjugate gradient method generates a series of k mutually conjugate search directions p_j, where p_0 is the steepest descent direction, and p_k is a linear combination of the steepest descent direction and the p_j, $j = 1 \ldots k-1$. At each step the function is minimized

[1] In practice, this takes a few iterations through the training set.

along one of these conjugate directions. For a general nonlinear function, each step involves computing a conjugate direction followed by a line search to get an approximate minimum in this direction. The method of conjugate gradients is a leading contender for large non-sparse optimization problems, since it uses only gradient information and does not require the storage of a matrix (unlike Newton's method, described in the next section). However, like the method of steepest descent, it converges only linearly in n, hence it may be prohibitively slow for large problems.

2 The Newton Method

Whereas the acceleration methods described above use only gradient and function values in performing minimization, Newton and quasi-Newton methods (e.g. [Dennis and Schnabel, 1983]) directly incorporate the Hessian matrix (of second derivatives of the error function) or an approximation to it. The Newton method is based on a quadratic model $\hat{C}$ of the objective function $C(W)$ (in our case W is a weight vector) which takes the first three terms in the Taylor expansion of C about the current point W_c:

$$\hat{C}_c(W_c+S) = C(W_c)+\nabla C(W_c)^T S+\frac{1}{2}S^T\nabla^2 C(W_c)S$$

To minimize this quadric, we solve for the step S where $\nabla\hat{C}_c(W_c + S) = 0$. The solution is $S = -\nabla^2 C(W_c)^{-1}\nabla C(W_c)$. Since the quadratic model is typically not exact, we cannot directly compute the solution in this manner in one step. Hence, Newton's algorithm iteratively computes the step S, adds it to the current point W_c to get the next approximate minimum, and repeats this process until it is sufficiently close to a solution (i.e. the gradient is sufficiently close to zero). This method works well when given a starting point within a convex region of the function, and converges quickly if the region is quadratic or nearly so.

There are several drawbacks to using the Newton method, making it unsuitable for connectionist learning, if not all difficult minimization problems. First, in order to converge it requires a good initial estimate of the solution, which is typically not available in a connectionist learning problem: since each parameter (i.e. each weight in the network) is a relatively meaningless entity by itself, it is difficult to get better than a random setting of its initial value, except for very special problems. Further, each iteration requires the computation of the Hessian matrix (of size n^2, where n is the number of weights) and also its inversion ($O(n^3)$ operations),[2] so the method is expensive in terms of both storage and computation. Finally, for a non-convex function, the method can converge (if indeed it converges at all) to a local maximum, saddle point, or minimum.

3 Quasi-Newton Methods

Quasi-Newton techniques combine Newton's method with some more globally convergent algorithm such as a line search, and overcome many of the drawbacks described above. One of the best of these techniques is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (see description in [Dennis and Schnabel, 1983]). This method has several advantages: It has good convergence properties, both in theory and in practice; it is invariant under linear transformations of the parameter space (unlike steepest descent); it computes an approximation to the ***inverse*** Hessian in only $O(n^2)$ operations, compared to $O(n^3)$ for Newton's method; and finally, the BFGS Hessian update is symmetric and positive definite, making the algorithm numerically stable (for further discussion of the properties of BFGS see [Dennis and Schnabel, 1983]).

Watrous implemented the BFGS algorithm in a connectionist learning framework, and compared it with back-propagation (BP) and two other methods - steepest descent with line search and the Davidon-Fletcher-Powell algorithm [Watrous, 1987]. For the three problems tested, BFGS converged in significantly fewer iterations than the other methods. Additionally, Watrous has reported good results using this method on speech recognition applications [Watrous *et al.*, 1987; Watrous, 1988]. Thus the BFGS method in practice is error tolerant, yields good solutions, and converges in a small number of iterations.

However, each iteration of BFGS requires $O(n^2)$ operations, compared to $O(n)$ for back-propagation. Hence, the computational advantage of BFGS over BP probably holds only for "small to moderate sized problems" [Watrous, 1987]. The largest problem reported by Watrous in the comparative study described above was a multiplexor task for a network

[2] Instead of inverting the Hessian, one can somewhat more efficiently solve the system of linear equations $\nabla^2 C(W_c)S = -\nabla C(W_c)$.

having 33 weights; in this case the number of operations was of the same order of magnitude for BFGS and BP. Furthermore, the BFGS algorithm lacks the desirable property of BP of having locally computable weight update terms.

4 A Computationally Feasible Approximation To Newton's Method

Second order methods may converge faster than gradient methods by taking into account additional information about the objective function. However, the Hessian matrix of the error function in a connectionist minimization problem is typically too large to compute and store, and too expensive to invert. [le Cun, 1987] has proposed an efficient means of computing approximate curvature information, in an extension of the back-propagation algorithm, using a simple diagonal approximation to the Hessian. Like BP, this algorithm also requires only $O(n)$ operations. The terms that must be computed by each unit depend only on that unit's outgoing weights and on the second derivative terms computed by units in the layer above; hence the diagonal Hessian terms can be computed and stored locally, using a procedure similar to the back-propagation method of obtaining the gradients. First we compute the second derivatives with respect to the total input to each unit a_i, as follows (where $f(a)$ is the nonlinear activation function computed by each unit, and w_{ki} is the weight from unit i to unit k):

$$\frac{\partial^2 C}{\partial a_i{}^2} = f^{'}(a_i)^2 \sum_k w_{ki}{}^2 \frac{\partial^2 C}{\partial a_k{}^2} - f^{''}(a_i) \sum_k w_{ki}(-\frac{\partial C}{\partial a_k})$$

From this, we can compute the second derivatives with respect to weights, again ignoring off-diagonal terms, as follows:

$$\frac{\partial^2 C}{\partial w_{ij}^2} = \frac{\partial^2 C}{\partial a_i{}^2} f(a_j)^2$$

Since this approximate Hessian has only diagonal elements, it becomes trivial to invert. The diagonal method also has the advantage of being able to deal with negative curvature (Newton's method requires that the Hessian be positive definite, and BFGS forces it to be so); we can simply reverse the step direction for weights having negative second derivative terms.

Instead of the usual back-propagation weight update rule $\Delta w_{ij} = -\epsilon \frac{\partial C}{\partial w_{ij}}$, we can now take the "pseudo-Newton step":

$$\Delta w_{ij} = -\frac{\frac{\partial C}{\partial w_{ij}}}{\left|\frac{\partial^2 C}{\partial w_{ij}^2}\right|}$$

By taking the absolute value of the Hessian term in the weight update formula, we are simply reversing the sign of the Newton step for directions having negative diagonal terms. To deal with regions of very small curvature, such as inflection points and plateaus, we can modify the above weight update rule slightly to add a small value μ to the second derivative terms:

$$\Delta w_{ij} = -\frac{\frac{\partial C}{\partial w_{ij}}}{\left|\frac{\partial^2 C}{\partial w_{ij}^2}\right| + \mu}$$

This heuristic is commonly used in numerical optimization methods to improve the conditioning of the Hessian.

Whereas the full Newton method performs both a scaling and rotation of the steepest descent step, the pseudo-Newton method described above has the effect of merely scaling the descent step in each direction. If the directions of maximal and minimal curvature are aligned with the axes in weight space, our method should capture most of the curvature information.

5 Numerical Analysis of the Diagonal Approximation

The applicability of the Newton-like method described above depends on how well the diagonal Hessian approximation models the true Hessian. If the ravines of the error surface are usually parallel to the axes in weight space, the diagonal terms of the Hessian should capture a sufficient proportion of the information about the curvature of the error function.

To determine how close our diagonal approximation is to the full Hessian, we compared matrix norms and eigenvalues for three small problems: a tiny problem for a network having only three units (one of these being a hidden unit) and one input pattern with no thresholds, a 4:2:4 encoder with four input patterns, and an 8:4:8 encoder with eight input patterns. For the tiny problem, the input unit is always on, and the desired output is 1 (we used a nonlinear activation function symmetric about the origin, having the values ±1 well within its range; it is described in the next section). Since this problem has only two weights, we can easily sweep through the entire weight-space to get complete information on the characteristics of the Hessian. In the encoder problems, the input vector has one bit on and the rest off, and the desired output pattern is identical to the input. The 4:2:4 and 8:4:8 encoders have 22 and 76 weights respectively (including thresholds). All simulations described in this and the next section were run using the SN neural network simulation package [le Cun and Bottou, 1988].

For our numerical experiments, the full and diagonal Hessians were approximated using a finite-difference method, that involved forward-differencing on the (analytic) gradient. This involved an initial pass through the input cases to compute the gradient, followed by n more passes varying one weight at a time by Δw_j to compute the n rows of the Hessian.[3] After this, the Hessian was made symmetric by replacing it by $\frac{1}{2}(H^T + H)$. The step size Δw_j for each row of the Hessian H_j was determined according to the recommendation of [Dennis and Schnabel, 1983] as follows:

$$\Delta w_j = \sqrt{\eta}\, max(w_j, \text{TypicalWeight})\, sign(w_j)$$

where η is the computed machine epsilon (the largest number such that within machine precision $1.0 + \eta = 1.0$), and TypicalWeight is set to 0.3. This method is precise to within about half the digits of precision of the gradient [Dennis and Schnabel, 1983]. Although the method is not terribly efficient, and is not recommended for large scale optimization problems, it is reasonably fast for small networks, and suitable for the present analyses.

The approximate Hessians, computed with the forward-differencing method described above, were sampled for various weight settings for the three test problems, and the L_1 norms of the Hessians were computed at each sample point. For the tiny network, the Hessians were sampled across the weight space for weights in the interval [−1.6, 1.6] at increments of 0.2. For the larger networks, samples were taken at random weight settings, as well as at solution points found by standard back-propagation learning. Learning began from an initial random weight setting, with weights in [−0.2, 0.2], and $\epsilon = 0.3$. The 4:2:4 net was run for 500 learning iterations using online BP, and the 8:4:8 for 1000; in both cases this was well beyond the point of 100% correct classification. (Note that these "solution weights" would not be considered exact solutions by the usual numerical minimization criteria that the gradient be zero and the Hessian positive.)

Table 1: Norms of Hessians, Diagonal Hessians, And Gradients For Three Networks

Net	*Weights*	$\|H\|$	$\|D\|$	$\frac{\|D\|}{\|H\|}$	$\|\nabla C\|$
tiny	Random	1.87	1.17	0.63	1.01
	Solution	0.77	0.43	0.56	0.098
4:2:4	Random	14.7	5.35	0.36	2.87
	Solution	24.6	4.76	0.19	0.042
8:4:8	Random	99.7	10.88	0.11	8.69
	Solution	164.8	16.07	0.10	0.048

The Hessians for the two larger problems exhibited certain characteristic structure. Their largest terms were predominantly in the diagonals, although there were many off-diagonal terms of the same order of magnitude. These matrices had small off-diagonal sub-blocks of zeroes, but were by no means sparse. The sub-blocks of zeros were located in regions where one would expect pairs of weights to be independent. For example, for two units i and k in the same layer having incoming weights w_{ij} and w_{kl} respectively, we find that the Hessian components $\frac{\partial^2 C}{\partial w_{ij} \partial w_{kl}}$ are very close to zero.

The norm of the diagonal Hessian relative to that of the full Hessian indicates the extent to which the latter matrix is diagonally dominant. These (L_1) norms are shown for the three networks in Table 1. All values shown were averaged over 20 replications of random weight settings, and 20 replications of

[3] Instead, one can more efficiently forward difference on the gradient w.r.t. the total input to a unit $\frac{\partial C}{\partial a_i}$, varying the state of the unit by a small amount, to obtain $\frac{\partial^2 C}{\partial a_i^2}$, and from this directly compute $\frac{\partial^2 C}{\partial w_{ji}^2}$. This involves only m passes (where m is the number of units), instead of n passes (where n is the number of weights).

solution weights.[4] The norm of the gradient is also given in each case, to indicate the proximity to a minimum.

For the tiny network, for the majority of points in weight space, the norm of the diagonal was found to be close in value to the norm of the full Hessian. For the larger encoder networks, as can be seen by the ratio of the norms of the diagonal to the full Hessians in Table 1, the diagonal terms account for a smaller proportion of the Hessian. As the weights approach a solution, it seems that the off-diagonal terms become even larger; this is especially true for the 8:4:8 network.

An analysis of the eigenvalues of the Hessian can yield much information about the applicability of second order methods, and in particular, of the diagonal approximation. For example, the spread between the largest and smallest eigenvalues indicates the eccentricity of the error surface, or how badly conditioned the problem is. Hence, the clustering of eigenvalues of our Hessian approximation, compared with those of the full Hessian, can tell us how similar these matrices are, and to what extent the principal curvatures of the error surface are captured in the diagonal terms. We show histograms of the eigenvalues of the diagonal versus the full Hessian for random weights (Figure 1), and weights near a solution (Figure 2). The EISPACK numerical software package [Smith *et al.*, 1974] was used to compute these eigenvalues. Since the clustering patterns in the eigenvalues are similar for both the small and large encoder networks, we only show those for the 8:4:8 encoder.

The eigenvalues of the full Hessian are most densely clustered in the same regions as the diagonals. However, there are in addition a small number of very large eigenvalues for the full Hessian, and also many negative eigenvalues. The negative eigenvalues are large in magnitude for random weights, and as would be expected, these negative values get dramatically small for weights near the solution. In contrast, the eigenvalues of the diagonal Hessian (which are simply the diagonal Hessian terms themselves) are always positive, and there are no extremely large diagonal terms. Many of the eigenvalues for both matrices are very near zero, indicating that the solution is highly degenerate.

Our analyses of the clustering of the eigenvalues of the Hessian, together with our observation that there are many off-diagonal zero blocks in the Hessian, suggest that the diagonal approximation should capture much of the curvature information. However, there are some large eigenvalues in the true Hessian not accounted for by the diagonal approximation. There may be regions in the error surface where small changes in some weights result in very large changes in the gradient, as evidenced by the large range in eigenvalues found for the encoder problems. If the diagonal Hessian terms are relatively small in such regions, a descent step computed strictly from these terms and the gradient may be catastrophic. Thus, care must be taken in applying this method. In the next section, we present some experimental results of a comparison between our Newton-like algorithm and back-propagation on a classification problem.

6 Learning Experiments With The Pseudo-Newton Algorithm

For our experiments, we created a problem which would be difficult for back-propagation to solve, and where hopefully the use of second order information would improve the learning speed. 64 patterns consisting of 32 bits randomly set to ±1 (with equal probability) were generated. Each pattern was then randomly assigned to one of four classes. Our networks thus have 32 input units and 4 output units.

We used the symmetric activation function $f(x) = 1.7159 \tanh(\frac{2}{3}x)$; the scale factor was chosen such that $f(1) = 1$ and $f(-1) = -1$. The cost function was the usual mean squared error between actual and desired outputs. Desired outputs were $+1$ and -1. The relative learning rate was set for each layer of the network to be $\frac{\varepsilon}{\sqrt{i}}$ where i is the fan-in or number of inputs to each unit in that layer.[5] The learning parameter ε was selected to be near optimal for each algorithm we tested. It was gradually increased until a smooth learning curve was observed, such that any further increase would cause the error to oscillate during learning. Once a near optimal epsilon was found for a particular algorithm, it was fixed at that value, for all subsequent

[4] For the tiny network, 20 solution points were selected from the statistics obtained across weight space based on the magnitude of the gradient.

[5] We compared three different rules for setting the learning rate: dividing by $\sqrt{i}$, dividing by i, and dividing by 1, and found that the first rule worked best.

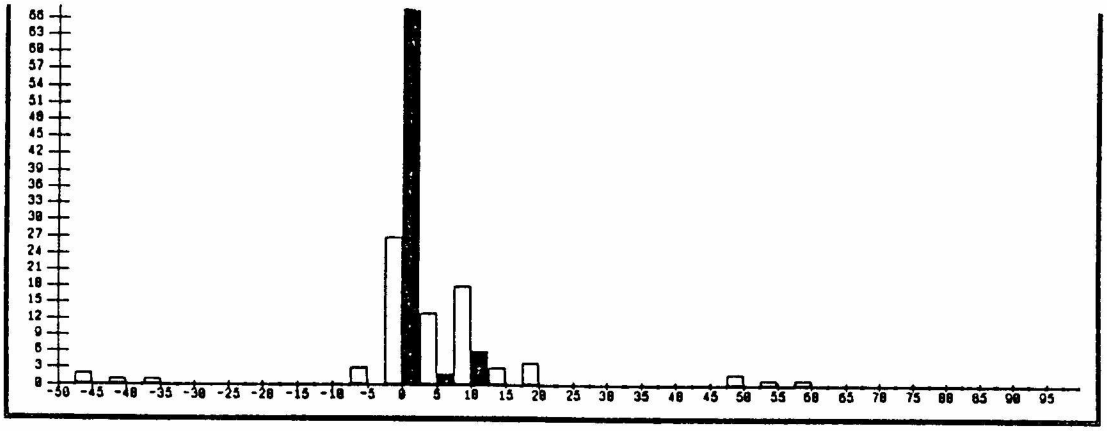

Figure 1: Eigenvalue histogram for the full and diagonal hessians of an 8:4:8 encoder network at random point in weight space. White bars are for the full Hessian, and grey bars are for the diagonal terms.

experiments. When momentum was used, its initial value was zero for the first two passes through the training set, and 0.8 from then on.[6]

We compared our algorithm with back-propagation, using both the "online" version of BP, in which the weights are adjusted after each pattern presentation, and "batch" BP, where the gradient is accumulated over the whole training set, and then weights are updated. The addition of momentum significantly accelerated learning for batch BP and Newton, but had little effect for online BP. The best performance was obtained with $\varepsilon = \frac{0.1}{\sqrt{t}}$ for online BP, $\varepsilon = \frac{0.04}{\sqrt{t}}$ for batch BP, and $\varepsilon = \frac{1.0}{\sqrt{t}}$ for Newton. For the Newton weight update rule, the parameter μ was set to 1; hence for any weight having a second derivative term less than 1 the weight change would be maximal, while for a weight with a larger second derivative term the step size would be reduced in proportion to the magnitude of that term.

Table 2 shows the mean trials to reach 100% correctness on the random classification problem for the three algorithms for optimally tuned learning parameters. These learning trials were repeated from 100 different random initial weight settings. The Newton algorithm performed significantly better than batch BP (by Student's t-test, two-tailed, $t = 19.2$, $p < .005$), which performed better than online BP ($t = 15.1$, $p < .005$). Figure 3 shows learning curves for Newton and batch BP; the mean and standard deviation of the error over 100 repetitions of learning (from 100 different initial random weight settings) are plotted over learning trials. The curves shown are for 20 passes through the learning set of 64 patterns. We omitted the learning curve for online BP from the figure since it overlapped that of batch BP to such an extent that it was difficult to tell the two curves apart.

Table 2: Mean Trials to Reach 100% Correctness On A Random Classification Problem

Algorithm	*Mean*	*Standard Deviation*
Online Back-propagation	1571.20	383.93
Batch Back-propagation	949.12	148.10
Pseudo-Newton	603.52	101.00

[6] In practice, this means of applying momentum has been found to work well on a variety of problems, for the following reason: during the first few learning iterations the error tends to drop very sharply and the gradient changes quickly in different directions, so momentum can adversely affect learning; after these first few iterations, a stable descent direction is found and momentum begins to have an accelerating effect on the learning rate.

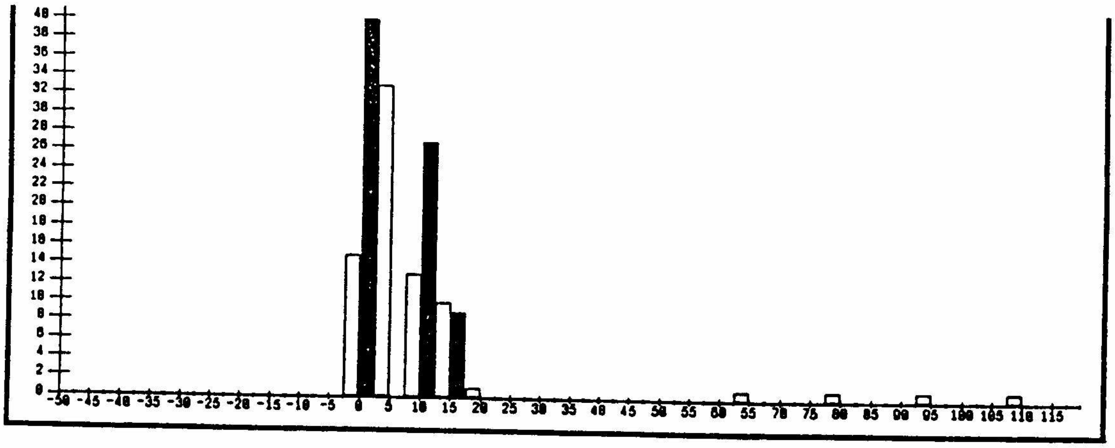

Figure 2: Eigenvalue histogram for the full and diagonal hessians of an 8:4:8 encoder network at a solution point. White bars are for the full Hessian, and grey bars are for the diagonal terms.

7 Discussion

The results presented above show that with momentum and optimally tuned learning parameters our Newton-like algorithm performs significantly better than back-propagation. The number of iterations required to learn the task to 100% correctness is reduced by a factor of about 1.5 compared to batch BP, and about 2.5 compared to online BP. The small increase in the complexity of our algorithm over BP (only in the backward pass of each iteration) is more than offset by this reduction in learning time.[7]

Under certain conditions, our algorithm fails to converge: if the initial weights are set to be very large or very small values. This is probably indicative of the fact that there are regions in weight space where the gradient is very steep and where the curvature is very shallow; hence the algorithm tends to compute steps that are too big. The standard solution to this problem would be to perform a line search in the Newton direction until a true descent step is found. However, a line search would add to the complexity of our algorithm, and require that we sacrifice the property of local computation. Other alternatives would be to either begin with a small learning rate for the first few iterations, or simply use gradient descent at first, and then switch to our second-order method to accelerate learning.

Another point to note is that without momentum, our algorithm performed about the same as back-propagation; this suggests that the method was underestimating the curvature at times, and therefore oscillating. With the addition of momentum, these oscillations would tend to be averaged out, as appears to be the case in practice with standard BP.

It is interesting that online and batch BP both performed about the same on the random association task. This particular task is rather special in that it is very low in redundancy. We have found in practice that on more "natural" pattern recognition problems which tend to have higher redundancy, online BP seems to outperform the batch version. This is not surprising: since batch BP accumulates the gradient terms for all cases in the learning set before it updates the weights, a training case that appears repeatedly in the same set contributes an identical term to the gradient each time. In contrast, online BP takes a step after each case presentation, so when presented with a case that it has seen already, it is at a different point in weight space (presumably closer to the solution), hence this presentation is not equivalent to the last one.

Finally, we note that the values of the learning parameters μ and ε are critical in getting reasonable behaviour with our Newton-like algorithm. If the value of μ is very small, our algorithm behaves more like the pure Newton method (given our Hessian approximation), and suffers the same pitfalls - we may take very large steps in regions of near-zero curvature, such as ravines and inflection points. On the

[7] On a digit recognition task, we found that our algorithm consumed approximately 30% more CPU time than BP.

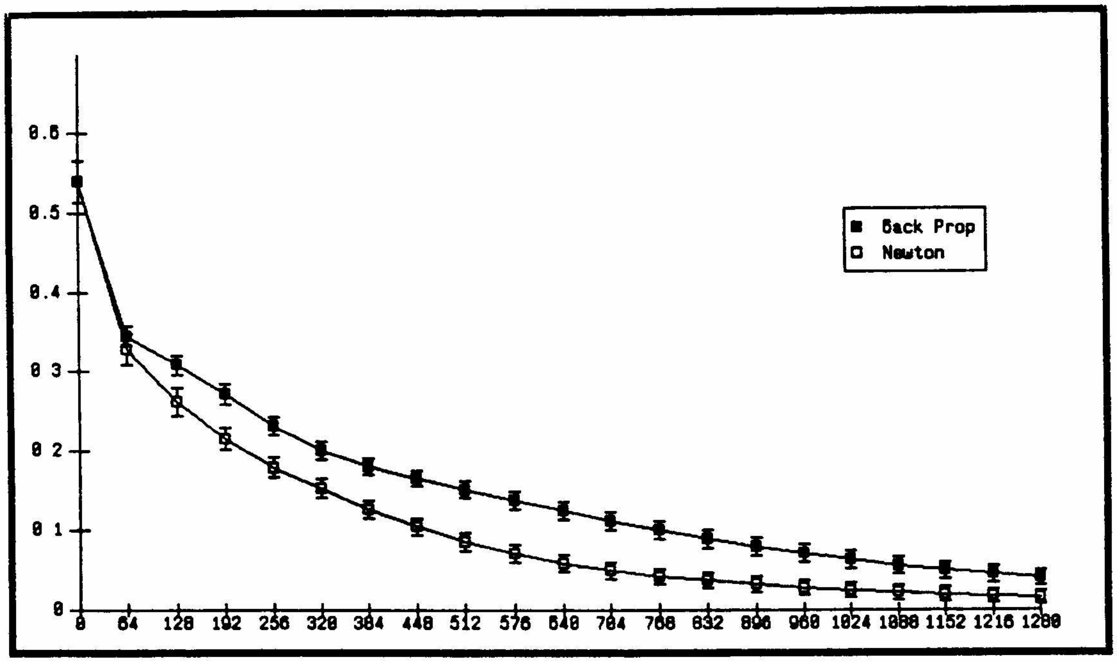

Figure 3: Mean error and standard deviation for 100 repetitions of 1280 pattern presentations with Batch Back Propagation versus Pseudo-Newton learning.

other hand, if μ is very large, our algorithm behaves like pure steepest descent, and will not take advantage of the curvature information. If ϵ is too large, we may see oscillations in the learning curve, or we may fail to converge at all, as is also true of back-propagation. We would predict that our method should be somewhat less sensitive to variations in ϵ than back-propagation, since we are using extra information about the curvature to scale the step size. More experiments are needed to determine whether this is the case on a variety of problems.

8 Conclusions

In general, second derivative methods show promise for speeding up the convergence of connectionist learning algorithms. The results of [Watrous, 1987] mentioned earlier indicate that the BFGS method, which uses a positive-definite approximation to the inverse Hessian, is in fact able to solve typical connectionist problems in relatively few iterations. However, since each iteration is computationally costly as compared to taking a Newton or steepest descent step, this method may only be advantageous on relatively small problems.

We have described an efficient means of estimating curvature, using an extension of the back-propagation algorithm which has been proposed by [le Cun, 1987]. Our analysis of the norms and eigenvalues of the Hessian for some simple problems suggest that while much of the curvature information is captured by the diagonal approximation, the true Hessian may have some large eigenvalues due to the off-diagonal terms. Thus our method may at times take a step which is much too large, and oscillate.

Our experiments on a random classification problem show that our pseudo-Newton method learns somewhat faster than both online and batch back-propagation, when all three methods have optimally tuned learning parameters and use.momentum. The reduction in the number of learning iterations by a factor of 1.5 to 2.5 more than offsets the small increase in complexity of our algorithm. The use of momentum seems to help the pseudo-Newton method in cases where it underestimates the curvature and would otherwise tend to oscillate. The algorithm fails to converge under some extreme conditions, such as when the initial weights are very large or very small, or the learning rate is too large. To get a more globally convergent algorithm, it may be desirable to use a hybrid technique, combining the Newton approximation with gradient descent as in standard BP.

Acknowledgements

The authors wish to acknowledge the contribution of Philip Sharp in providing us with many insights in the area of numerical optimization. We also thank Geoffrey Hinton and Mike Mozer for their useful discussion and comments. This work was supported in part by a grant from the government of Ontario to the Information Technology Research Center.

References

[Dennis and Schnabel, 1983] J. E., Jr. Dennis and R. B. Schnabel. *Numerical Methods For Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[Gill *et al.*, 1981] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization.* Academic Press, 1981.

[le Cun and Bottou, 1988] Y. le Cun and Léon-Yves Bottou. SN: A simulator for connectionist models. SN Manual, March 1988.

[le Cun, 1987] Y. le Cun. *Modèles Connexionnistes de l'Apprentissage.* PhD thesis, Université Pierre et Marie Curie, Paris, France, 1987.

[Plaut *et al.*, 1986] D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back-propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, Pittsburgh PA 15213, June 1986.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, volume I. Bradford Books, Cambridge, MA, 1986.

[Smith *et al.*, 1974] B.T. Smith, J.M. Boyle, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler. *Matrix Eigensystem Routines - EISPACK Guide.* Springer-Verlag, 1974.

[Watrous *et al.*, 1987] R. Watrous, L. Shastri, and A Waibel. Learned phonetic discrimination using connectionist networks. In *European Conference on Speech Technology*, pages 377–380, September 1987. Edinburgh.

[Watrous, 1987] R. Watrous. Learning algorithms for connectionist networks : Applied gradient methods of non-linear optimization. Technical Report MS-CIS-87-51, University of Pennsylvania, 1987.

[Watrous, 1988] R. Watrous. Connectionist speech recognition using the temporal flow model. In *IEEE Workshop On Speech Recognition*, 1988.