Training Feedforward Neural Networks using Orthogonal Iteration of the Hessian Eigenvectors

Andrew Hunter

Department of Computing and Engineering Technology

University of Sunderland, St. Peter's Campus, Sunderland, Tyne and Wear, England.

Andrew.Hunter@sunderland.ac.uk

Introduction

Training algorithms for Multilayer Perceptrons optimize the set of W weights and biases, $\mathbf{w}$, so as to minimize an error function, E, applied to a set of N training patterns. The well-known back propagation algorithm combines an efficient method of estimating the gradient of the error function in weight space, $\Delta E=\mathbf{g}$, with a simple gradient descent procedure to adjust the weights, $\Delta \mathbf{w} = -\eta \mathbf{g}$. More efficient algorithms maintain the gradient estimation procedure, but replace the update step with a faster non-linear optimization strategy [1].

Efficient non-linear optimization algorithms are based upon second order approximation [2]. When sufficiently close to a minimum the error surface is approximately quadratic, the shape being determined by the Hessian matrix. Bishop [1] presents a detailed discussion of the properties and significance of the Hessian matrix. In principle, if sufficiently close to a minimum it is possible to move directly to the minimum using the Newton step, $-\mathbf{H}^{-1}\mathbf{g}$.

In practice, the Newton step is not used as $\mathbf{H}^{-1}$ is very expensive to evaluate; in addition, when not sufficiently close to a minimum, the Newton step may cause a disastrously poor step to be taken. Second order algorithms either build up an approximation to $\mathbf{H}^{-1}$, or construct a search strategy that implicitly exploits its structure without evaluating it; they also either take precautions to prevent steps that lead to a deterioration in error, or explicitly reject such steps.

In applying non-linear optimization algorithms to neural networks, a key consideration is the high-dimensional nature of the search space. Neural networks with thousands of weights are not uncommon. Some algorithms have $O(W^2)$ or $O(W^3)$ memory or execution times, and are hence impracticable in such cases. It is desirable to identify algorithms that have limited memory requirements, particularly algorithms where one may trade memory usage against convergence speed.

The paper describes a new training algorithm that has scalable memory requirements, which may range from $O(W)$ to $O(W^2)$, although in practice the useful range is limited to lower complexity levels. The algorithm is based upon a novel iterative estimation of the principal eigen-subspace of the Hessian, together with a quadratic step estimation procedure.

It is shown that the new algorithm has convergence time comparable to conjugate gradient descent, and may be preferable if early stopping is used as it converges more quickly during the initial phases.

Section 2 overviews the principles of second order training algorithms. Section 3 introduces the new algorithm. Second 4 discusses some experiments to confirm the algorithm's performance; section 5 concludes the paper.

Overview of Second Order training concepts

Second order training algorithms are based upon a local quadratic approximation of the error surface [2]. Given a quadratic error function, the error surface has hyper-ellipsoid contours of equal error. The axes are aligned with the eigenvectors of the Hessian, $\mathbf{e}_j$, with the length of each axis inversely proportional to the corresponding eigenvalue, λ_j. Gradient descent is unacceptably slow since the gradient vector, $-\mathbf{g}$, tends to point across the hyper-ellipsoid in the direction of axes with large eigenvalues, and convergence speed is limited by the condition number of the Hessian, λ_1 / λ_W. In contrast to the gradient vector, the Newton direction, $-\mathbf{Hg}$, points directly to the minimum.

Quasi-Newton methods [3] explicitly build up an approximation to the inverse Hessian and line search in the direction of the estimated Newton step. Quasi-Newton algorithms are very effective, but have $O(W^2)$ memory requirements. This suggests using approximations that require less storage. For example, the Limited Memory Quasi-Newton algorithm uses only $O(W)$ storage but maintains a relatively poor approximation [1]. Saito and Ryohei [4] recently suggested a modification that allows the amount of memory used in the approximation to be scaled arbitrarily.

Conjugate gradient methods conduct a series of line searches along "non-interfering" directions that are constructed to exploit the Hessian structure without explicitly storing it; they consequently have $O(W)$ storage requirements. However, they tend to be somewhat slower than Quasi-Newton.

Moller [5] suggested a modified conjugate gradient descent algorithm that exploits an interesting fact: the line search is used to calculate the step length, for which there is an analytic formula involving the Hessian. The Hessian enters this formula only in the form $\mathbf{H}\mathbf{d}_j$. An efficient modification of the Back Propagation algorithm, the $\Re\{.\}$ operator technique [6], calculates the product of the Hessian and any vector in $O(W)$ operations, without having to explicitly store or evaluate $\mathbf{H}$. Moller uses this to generate a step size in a single operation, avoiding the line search.

The EQUAL algorithm

The algorithm described in this paper uses quadratic estimation in a very direct way. To introduce it, it is helpful to discuss Fahlmann's Quick Propagation [7] algorithm.

Consider optimization of a quadratic function in one dimension, w. Evaluate the gradient g_0 at point w_0, then move a small distance Δw_0 (typically using the gradient descent formulation, $-h g_0$) and evaluate a second gradient, g_1, at $w_1=w_0+\Delta w_0$. By linear interpolation of the gradients, the minimum is found at $w_1+\Delta w_1$, $Dw_1=\Delta w_0.g_1/(g_0-g_1)$.

This formula can be modified to produce an iterative update step for minimization of a non-linear function, as follows: $\Delta w_i=\Delta w_{i-1}.g_i(g_{i-1}-g_i)$. Once sufficiently close to the minimum, this formula converges extremely quickly, although some additional checks are required to prevent numerical problems on a poorly behaved curve.

In quick propagation, the formula is applied separately to each weight in the neural network, amounting to an assumption that the weights are independent (i.e. that the principal axes of the hyper-ellipsoids of the error surface are aligned with the weights). This suggests modifying the algorithm to operate along the eigenvectors, rather than along the weights. The eigenvectors could in principle be calculated using standard techniques such as Householder reduction and the QR algorithm [8]; however, this requires $O(W^3)$ operations and $O(W^2)$ storage.

The approach taken in this paper is to iteratively estimate the leading subset of the eigenvectors. The size of this subset is user-configurable, and can reflect available memory. The quick propagation formula is applied along the axes of this estimated eigen-subspace, and a separate step is made in the orthogonal subspace.

The estimated eigen-subspace is calculated using the Orthogonal Iteration technique [8]. An initial estimate is formed using the original gradient, $\mathbf{g}_0$, as the first eigenestimate, $\mathbf{e}_0$, with subsequent eigenestimates $\mathbf{e}_1 \ldots \mathbf{e}_v$ formed using the standard Gram-Schmidt orthogonalization procedure with the standard unit vectors, $\mathbf{u}_j$. On subsequent iterations, the estimated eigenvectors are multiplied by the Hessian, then re-orthogonalized using Gram-Schmidt. The $\mathbf{H}\mathbf{e}_j$ are calculated using $\Re\{.\}$ operator technique [1,2,6].

Orthogonal iteration isolates the eigenvectors corresponding to the largest eigenvalues: that is, the eigenvectors aligned across the narrow part of the hyper-ellipsoid. These are precisely the directions that most limit the search step size, and so isolating even one can significantly improve convergence speed. At first the approximations will be relatively poor, and as the Hessian changes on a non-linear error surface, they may take time to settle down. However, an arbitrary rotation of the original axes does not cause a deterioration in quick propagation's performance, so we can expect the algorithm at least to match that level of performance, even before the eigenestimates are stabilized.

As with all second-order algorithms, poor steps may be generated. A simple model-trust procedure is therefore applied: the step generated by the algorithm is accepted only if it causes a reduction in the error; otherwise, a standard gradient descent step (with a low learning rate) is substituted. The algorithm (call EQUAL, for Eigenvector-based QUAdratic Learning) is described in detail in figure 2. The Gram-Schmidt orthogonalization procedure is described in any introductory linear algebra text (e.g. [9]).

Terms

$\Delta\mathbf{w}_{(t)}$	The weight update vector at time t.
$\Delta\mathbf{w}_{(t)E}$, $\Delta\mathbf{w}_{(t-1)E\perp}$	Weight update in/orthogonal to eigen-subspace
$\Delta\mathbf{w}_{(t)E}[\mathbf{e}_i]$	Component of $\Delta\mathbf{w}_{(t)E}$ along vector $\mathbf{e}_i$
η	The learning rate to kick off search, =0.01
$\mathbf{g}_{(t)}$	The gradient of the error function at time t
$\mathbf{E}=\{\mathbf{e}_j\}$	The set of V "eigenestimate" vectors
$\mathbf{u}_j$	A standard unit vector (1 in j'th component; 0 in others)
$\langle\mathbf{v}_i,\mathbf{v}_j\rangle$	The dot product of two vectors
GSN($\mathbf{E}$)	Apply the Gram-Schmidt procedure to $\mathbf{E}$, then normalize all $\mathbf{e}_j$

First iteration

$$\Delta\mathbf{w}_{(0)} = -h\mathbf{g}_{(0)}$$
$$\mathbf{e}_1 = -\mathbf{g}_{(0)} \qquad \mathbf{e}_j = \mathbf{u}_j, \forall 1 < j \le V$$
$$\mathbf{E} = \mathrm{GSN}(\mathbf{E})$$

All other iterations

$$\mathbf{E} = \mathrm{GSN}(\mathbf{EH}) \quad \text{calculate } \mathbf{EH} \text{ using } \Re\{.\}, \text{ each } \mathbf{e}_j$$
$$\Delta\mathbf{w}_{(t)E}[\mathbf{e}_j] = \mathrm{QP}(<\Delta\mathbf{w}_{(t-1)E},\mathbf{e}_j>,<\mathbf{g}_{(t-1)E},\mathbf{e}_j>,<\mathbf{g}_{(t)E},\mathbf{e}_j>) \qquad \Delta\mathbf{w}_{(t)E} = \sum_j \Delta\mathbf{w}_{(t)E}[\mathbf{e}_j]$$
$$\Delta\mathbf{w}_{(t-1)E\perp} = \Delta\mathbf{w}_{(t-1)} - \sum_j <\Delta\mathbf{w}_{(t-1)},\mathbf{e}_j>\mathbf{e}_j$$
$$\mathbf{g}_{(t-1)E\perp} = \mathbf{g}_{(t-1)E} - \sum_j <\mathbf{g}_{(t-1)},\mathbf{e}_j>\mathbf{e}_j$$
$$\mathbf{g}_{(t)E\perp} = \mathbf{g}_{(t)E} - \sum_j <\mathbf{g}_{(t)},\mathbf{e}_j>\mathbf{e}_j$$
$$\Delta\mathbf{w}_{(t)E\perp}[\mathbf{e}_j] = \mathrm{QP}(<\Delta\mathbf{w}_{(t-1)E},\mathbf{u}_j>,<\mathbf{g}_{(t-1)E},\mathbf{u}_j>,<\mathbf{g}_{(t)E},\mathbf{u}_j>) \qquad \Delta\mathbf{w}_{(t)E\perp} = \sum_j \Delta\mathbf{w}_{(t)E\perp}[\mathbf{e}_j]$$
$$\Delta\mathbf{w}_{(t-1)E\perp} = \Delta\mathbf{w}_{(t-1)} - \sum_j <\Delta\mathbf{w}_{(t-1)},\mathbf{e}_j>\mathbf{e}_j$$
$$\Delta\mathbf{w}_{(t)E\perp} = \Delta\mathbf{w}_{(t)E\perp} - \sum_j <\Delta\mathbf{w}_{(t)E\perp},\mathbf{e}_j>\mathbf{e}_j$$
$$\Delta\mathbf{w}_{(t)} = \Delta\mathbf{w}_{(t)E\perp} + \Delta\mathbf{w}_{(t)E}$$
$$\Delta\mathbf{w}_{(t)} = -h\mathbf{g} \text{ if } \mathrm{E}(\mathbf{w}+\Delta\mathbf{w}_{(t)}) > \mathrm{E}(\mathbf{w})$$

Quick-propagation step

$$\mathrm{QP}(w_{t-1}, g_{t-1}, g_t) = \begin{cases} 0 & |g_{(t)}| < e \\ -hg_{(t)} & |g_{(t-1)}| < e \text{ or } |\Delta w_{t-1}| < e \\ a\Delta w_{(t-1)} & g_{(t)}\cdot g_{(t-1)} > 0 \text{ and } |g_{(t)}| > \dfrac{a}{a+1}|g_{(t-1)}| \\ \Delta w_{(t-1)}\dfrac{g_{(t)}}{g_{(t-1)} - g_{(t)}} & \text{otherwise} \end{cases}$$

Figure 1: The EQUAL Algorithm

Experiments

The algorithm has been tested on two networks. The first is for Fischer's classic Iris data set, with Sepal Length omitted. The network has three inputs, two hidden units and three outputs: a total of 17 weights (biases included); there are 150 cases. The second is for Sigillito's Ionosphere problem [10]. It has 33 inputs, 15 hidden units and one output: a total of 477 weights; there are 351 cases. Logistic activation functions and sum-squared error function were used: all inputs were normalized into the range [0,1]. All cases were used for training with both data sets and no cross-verification was performed, as we are purely interested in optimization speed, not prevention of over-fitting.

The standard quick propagation and conjugate gradient descent algorithms were used a benchmarks. Versions of EQUAL were run with the dimension of the eigen-subspace set to 1, 5 and 17 (in the case of the first problem, the last of these implies a full eigen-decomposition of the search space). Each algorithm was executed twenty times, and the results shown are the mean across these runs. As any of the algorithms may converge to significantly inferior solutions on some occasions, up to two test runs were omitted from each average. Although insufficient runs were conducted to reach a reliable conclusion on the issue, conjugate gradient decent and $EQUAL_{17}$ seem to be more prone to hit local minima than the simpler algorithms.

The computational requirements of the algorithms are best expressed using the number of propagations, expressed below as *n* props. Executing a network on each patterns costs one prop, and evaluating the gradient requires two (one forward and one backward). A single prop has $O(WN)$ cost.

An iteration of quick propagation requires 2 props. An iteration of conjugate gradient descent has variable requirements, depending on the length of the line search: the average is 12 props per iteration.

An iteration of EQUAL requires $2V$ props (the gradient calculation ca be combined with the first $\Re\{.\}$ operator application). There is also an overhead in the application of Gram-Schmidt. This has $O(WV^2)$ costs, and hence becomes significant unless $V^2 << N$. In both the test problems, which have a relatively small number of cases, this implies that Gram-Schmidt contributes marginally to the requirements of $EQUAL_{17}$, and can be ignored for $EQUAL_5$ and $EQUAL_1$. The figures used are 2 props/iteration, 10 props/iteration and 37 props/iteration for $EQUAL_1$, $EQUAL_5$ and $EQUAL_{17}$ respectively.

Figure 2 shows the performance of the algorithms on the Iris test set. Initial convergence speed is faster with the simpler algorithms, with $EQUAL_1$ being particularly effective. Terminal convergence is superior in the conjugate gradient descent and $EQUAL_{17}$ algorithms, with quick propagation proving noticeably inferior to the others. In the mid-range, the $EQUAL_1$ and $EQUAL_5$ algorithms develop a noticeable lead on the others.

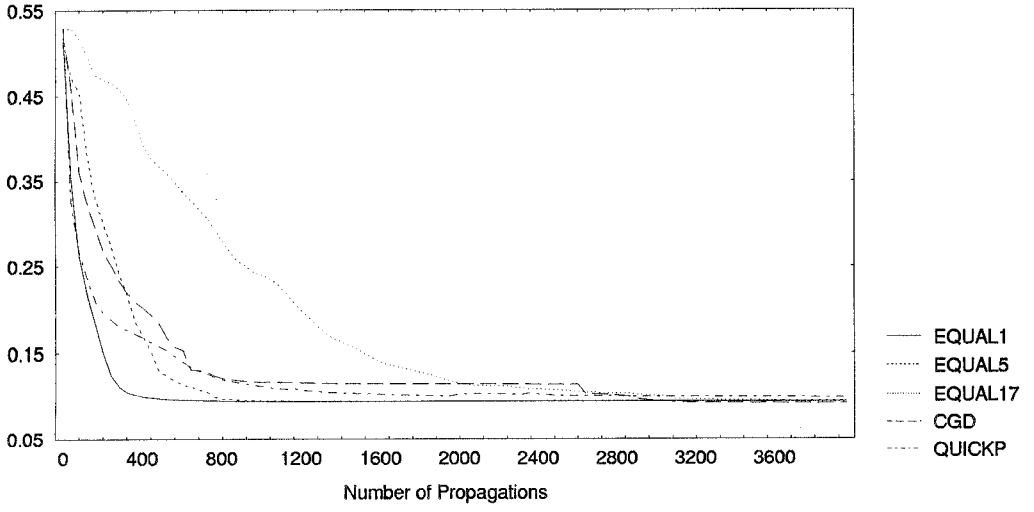

Figure 2: Algorithm performance on Iris data set

$EQUAL_1$	$EQUAL_5$	$EQUAL_{17}$	CGD	QuickProp
0.0943	0.0947	0.0933	0.0923	0.0990

Table 1: Average final error after 4000 props, Iris problem

The results indicate that the performance of $EQUAL_1$ and $EQUAL_5$ is superior to that of quick propagation. It is also comparable with that of conjugate gradient descent, although somewhat inferior during terminal convergence. However, we note that it is common practice to halt training algorithms before full minimization on the training set occurs, as this commonly leads to over-fitting, and EQUAL's fast early and mid-range convergence coincides with the stage in training when halting is likely. In practice, EQUAL's convergence speed may be better than that of conjugate gradient descent. The results on the Ionosphere data set were comparable.

Conclusion

The paper describes EQUAL, a novel second order training algorithm for feedforward neural networks. The algorithm iteratively builds up an estimate of the leading eigen-subspace of the Hessian matrix, by applying the $\Re\{.\}$ operator method to perform the orthogonal iteration algorithm. A simple quadratic estimation procedure is then applied along the axes of the eigen-subspace. The standard quick propagation procedure is applied to the portion of the gradient orthogonal to this subspace, and then added to the subspace delta. By exploiting the independence of the quadratic function along eigenvectors, the algorithm accelerates convergence in comparison to quick propagation.

In comparison with the conjugate gradient descent algorithm EQUAL has superior performance in the early and middle stages of convergence, but inferior performance during terminal convergence. However, we not that the latter stage is irrelevant in small- to medium-sized data sets, where excessive optimization on the training set merely invites over-learning.

The algorithm is noteworthy in being scalable in terms of memory requirements - the dimension of the eigen-subspace can be arbitrarily selected. Increasing the dimensionality reduces initial convergence, while improving terminal convergence, which invites selection of a good "trade-off" dimension. However, the algorithm has an unfortunate drawback in that the Gram-Schmidt orthogonalization procedure needs to be performed on each iteration, and the computational efficiency of this scales with the square of the dimension. Hence, the procedure is efficient only for a fairly small subspace.

EQUAL presents a very novel approach to optimization, which may be developed further to produce alternative improved algorithms. For example, although quadratic estimation is natural in the eigen-subspace, it may be possible to exploit a different technique in the orthogonal subspace. A particularly compelling possibility is to use Quasi-Newton in the orthogonal subspace, as this algorithm benefits greatly from even very modest reductions in dimensionality (each removed dimension halves the space requirements), and EQUAL is particularly efficient if a small number of dimensions are used. This will be the subject of further research.

Finally, we note that the eigenestimates stabilize over time. Consequently, the algorithm could be modified to search for a single eigenestimate per epoch, progressively adding more as they stabilize. This variation would require only 2 props per iteration, and might combine $EQUAL_1$'s fast initial convergence with the superior terminal convergence of the higher order versions.

References

[1] Bishop, C. (1995). *Neural Networks for Pattern Recognition.* Oxford: University Press.

[2] Shepherd, A.J. (1997). *Second-Order Methods for Neural Networks.* New York, Springer.

[3] Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992*). Numerical Recipes in C: The Art of Scientific Computing (Second ed.).* Cambridge University Press.

[4] Saito, K. and Ryohei, N. (1996). Partial BFGS Update and Efficient Step-Length Calculation for Three-Layer Neural Networks, *Neuron Computation 9 (1),* 123-141.

Moller, M (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks 6 (4),* 525-533.

[6] Perlmutter, B.A. (1994). Fast exact multiplication by the Hessian. *Neural Computation 6(1),* 147-160.

[7] Fahlmann, S.E. (1988). Faster-learning variations on back-propagation: an empirical study. *In D. Touretsky, G.E. Hinton and T.J. Sejnowski (Eds.), Proceddings of the 1988 Connectionist Models Summer School,* 38-51. San Mateo, CA: Morgan-Kaufmann.

[8] Golub, G.H. and Van Loan, C.F. (1989). *Matrix Computations.* John Hopkins University Press, Baltimore.

[9] Anton, H. and Rorres, C. (1994). *Elementary Linear Algebra, 7th Edition,* Wiley, New York.

[10]. Sigillito, V.G., Wing, S.P., Hutton, L.V. and Baker, K.B. (1989). Classification of radar returns from the ionosphere using neural networks. *John Hopkins APL Technical Digest,* 10, 262-266.