International Journal of Neural Systems, Vol. 2, No. 4 (1992) 291–301

BACKPROPAGATION LEARNING FOR MULTILAYER FEED-FORWARD NEURAL NETWORKS USING THE CONJUGATE GRADIENT METHOD

E. M. Johansson, F. U. Dowla and D. M. Goodman
Lawrence Livermore National Laboratory, University of California, P.O. Box 808, L-495, Livermore, CA 94550, USA

Received 22 April 1991
Revised 9 January 1992

In many applications, the number of interconnects or weights in a neural network is so large that the learning time for the conventional backpropagation algorithm can become excessively long. Numerical optimization theory offers a rich and robust set of techniques which can be applied to neural networks to improve learning rates. In particular, the conjugate gradient method is easily adapted to the backpropagation learning problem. This paper describes the conjugate gradient method, its application to the backpropagation learning problem and presents results of numerical tests which compare conventional backpropagation, steepest descent and the conjugate gradient methods. For the parity problem, we find that the conjugate gradient method is an order of magnitude faster than conventional backpropagation with momentum.

1. Introduction

The backpropagation algorithm[1,2] is probably the most widely used supervised learning algorithm in neural network applications. For many problems, however, the number of interconnects or weights in a network can be so large that the backpropagation learning time is excessively long and use of the algorithm becomes impractical. There are several solutions to this problem. One is to reduce the size of the problem by pre-processing the data: employ some form of decimation, projection or feature extraction algorithms to reduce the dimensions of the input. Another is to use faster computers or machines with parallel architectures. A third approach, the underlying motivation for this paper, is to apply numerical optimization theory to make the backpropagation method significantly faster. Numerical optimization theory offers a rich and robust set of techniques which can be applied to neural networks in an attempt to improve learning rates. In particular, the conjugate gradient method[3–7] is easily adapted to the backpropagation learning problem.

The major advantages of the conjugate gradient method are its speed and simplicity. It is much faster than steepest descent and does not suffer from the inefficiencies and possible instabilities that arise from using a fixed step size, as in the conventional backpropagation method. Although it is slower than the second order Newton and quasi-Newton minimization methods, it is considerably less complex. The second order methods usually require that the Hessian (the matrix of second order partial derivatives of the error function) be evaluated and inverted or that an approximation to its inverse be evaluated at each iteration. This poses a problem for neural networks, which are often quite large. Consider an application in which 256×256 images are used as inputs to a multi-layer neural network. The input layer alone would have 256^2 weights. A second order method would require the evaluation (and possibly the inversion) of a 256^4 element Hessian at each iteration! Although second order methods may be faster for small and moderate sized problems, it is clear that storage and computation requirements render them not feasible for applications of this magnitude. This is compounded by the fact that 256×256 is not considered a large size for an image. For problems such as these, the method of conjugate gradients is one of the few alternatives. The conjugate gradient method is well suited for the neural network learning problem because it is fast, simple and requires little additional storage space (only the current and previous gradient and search vectors must be stored in addition to the weights).

Improving the speed of backpropagation is currently quite an important problem. Consequently, other researchers have also applied optimization techniques to this problem. In particular, the reader is referred to the

works of Becker and Cun,[8] and Kramer and Sangiovanni–Vincentelli.[9] Becker and Cun use variations of a quasi-Newton method to speed up backpropagation by approximating the second order derivatives. They report mixed results and note that the approximation might result in oscillations during convergence. Kramer and Sangiovanni–Vincentelli compare parallel implementations of backpropagation, steepest descent and conjugate gradient technique using the Polak–Ribière method. They report the Polak–Ribière method to be superior to conventional backpropagation and steepest descent for small boolean encoder problems and for the parity problem. The results of our paper support the conclusions of Kramer and Sangiovanni–Vincentelli. However, our paper considers several different conjugate gradient techniques applying a wide range of optimization parameters to each of these methods.

In this study, we have found the conjugate gradient method to be an order of magnitude faster than conventional backpropagation with momentum. Since we have found the conjugate gradient method to be quite successful and because it appears not to be widely known among neural network researchers, a major goal of this paper is to provide a detailed tutorial introduction to the conjugate gradient method. We discuss many issues which will aid the reader in implementing this algorithm for his particular application.

This paper is organized as follows: We begin in Sec. 2 with a tutorial introduction to the conjugate gradient optimization method. Section 3 discusses the application of the conjugate gradient method to the backpropagation learning problem. Finally, results of numerical tests comparing backpropagation, steepest descent and the conjugate gradient training methods are presented in Sec. 4.

2. The Conjugate Gradient Method

The learning phase of a backpropagation network can easily be viewed as a classical unconstrained nonlinear optimization problem. The solution of such a problem typically involves modifying a set of independent variables in a systematic fashion to minimize or maximize some objective function. The multilayer feed-forward neural network fits this model quite nicely: the weights are the independent variables and the normalized sum of the output errors from the training set is the objective function. The goal of training the network is to modify the weights in such a manner that the network outputs for each pattern match the desired outputs.

An integral part of applying an optimization tool such as the conjugate gradient method is an understanding of how the algorithm works, what its limitations are and when it can best be applied. We have chosen to address these issues in the form of a detailed tutorial. We begin by introducing the theory of conjugate directions and show how it can be used to solve a simple quadratic problem. Next we explain how the conjugate gradient algorithm is derived from conjugate direction theory. This is followed by a discussion of the extension to generalized non-quadratic problems. We conclude with a short discussion about line search techniques.

2.1. *Conjugate directions*

Suppose we wish to minimize the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} - \mathbf{b}^T\mathbf{x} , \tag{1}$$

where $\mathbf{x}$ is of size n, and $\mathbf{Q}$ is of size $n \times n$, symmetric and positive definite. The unique minimum of $f(\mathbf{x})$ occurs at the point of zero gradient: the solution to $\nabla f(\mathbf{x}) = \mathbf{Q}\mathbf{x} - \mathbf{b} = 0$. In other words, the minimum point, $\mathbf{x}^*$, is the solution to the linear equations

$$\mathbf{Q}\mathbf{x}^* = \mathbf{b} . \tag{2}$$

Finding the minimum of the quadratic or equivalently, the solution to the linear equations is simplified considerably if we have available a set of n non-zero vectors which are $\mathbf{Q}$-conjugate.

Definition: The vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ are said to be $\mathbf{Q}$-conjugate if

$$\mathbf{d}_i^T\mathbf{Q}\mathbf{d}_j = 0 , \qquad i \neq j . \tag{3}$$

It is easily shown that a set of non-zero $\mathbf{Q}$-conjugate vectors are linearly independent and form a basis which spans the vector space of $\mathbf{x}$; we omit the proof.

Assume that we are given a starting point $\mathbf{x}_0$ and a $\mathbf{Q}$-conjugate set $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ and that we wish to move to the solution $\mathbf{x}^*$. Since the vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ are a basis, we can write the vector representing the move from $\mathbf{x}_0$ to $\mathbf{x}^*$ as a linear combination of these vectors;

$$\mathbf{x}^* - \mathbf{x}_0 = \sum_{i=0}^{n-1} \alpha_i \mathbf{d}_i , \tag{4}$$

where $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ are scalars which must be determined. Multiplying this equation by $\mathbf{d}_j^T\mathbf{Q}$ and substituting $\mathbf{b}$ for $\mathbf{Qx}^*$ gives

$$\mathbf{d}_j^T[\mathbf{b} - \mathbf{Qx}_0] = \sum_{i=0}^{n-1} \alpha_i \mathbf{d}_j^T\mathbf{Qd}_i \,. \tag{5}$$

Here the advantage of having the $\mathbf{Q}$-conjugate vectors becomes clear. If $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ were not $\mathbf{Q}$-conjugate, then determining $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ would involve solving n linear equations in n unknowns. However, $\mathbf{Q}$-conjugacy eliminates the cross terms and gives a closed form equation for these scalars;

$$\alpha_j = \frac{\mathbf{d}_j^T[\mathbf{b} - \mathbf{Qx}_0]}{\mathbf{d}_j^T\mathbf{Qd}_j} \,. \tag{6}$$

If we define

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{j=0}^{k-1} \alpha_j \mathbf{d}_j \,, \tag{7}$$

then we can write an iterative expression for $\mathbf{x}$;

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \tag{8}$$

and we note that $\mathbf{x}_n = \mathbf{x}^*$. Thus, we can interpret the move from $\mathbf{x}_0$ to $\mathbf{x}^*$ as a series of n separate moves along n different "direction" vectors, $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$. Using (3) and (7) to show that $\mathbf{d}_k^T\mathbf{Qx}_k = \mathbf{d}_k^T\mathbf{Qx}_0$ and using (6) and the notation $\mathbf{g}_k = \nabla f(\mathbf{x}_k) = \mathbf{Qx}_k - \mathbf{b}$, we arrive at the following expression for α_k;

$$\alpha_k = -\frac{\mathbf{d}_k^T\mathbf{g}_k}{\mathbf{d}_k^T\mathbf{Qd}_k} \,. \tag{9}$$

Equations (8) and (9) define the general form of the conjugate directions algorithm. Using this algorithm, a positive definite quadratic function or set of linear equations can be solved in exactly n steps.

An important property of the algorithm is that the current gradient is orthogonal to the previous direction vectors. We begin the proof of this property by computing the difference of the gradients at successive points. Using (8), it can be shown that

$$\begin{aligned} \mathbf{g}_{k+1} - \mathbf{g}_k &= \mathbf{Q}[\mathbf{x}_{k+1} - \mathbf{x}_k] \\ &= \alpha_k \mathbf{Qd}_k \,. \end{aligned} \tag{10}$$

Taking the inner product with $\mathbf{d}_k$ and using (9) gives

$$\begin{aligned} \mathbf{g}_{k+1}^T\mathbf{d}_k &= \mathbf{d}_k^T\mathbf{g}_k + \alpha_k \mathbf{d}_k^T\mathbf{Qd}_k \\ &= 0 \,. \end{aligned} \tag{11}$$

It also follows from (10) and (3) that

$$[\mathbf{g}_{k+1} - \mathbf{g}_k]^T\mathbf{d}_j = 0 \,, \qquad \text{for } j = 1, 2, \ldots, k-1 \,. \tag{12}$$

A simple inductive argument applied to the last two equations gives the desired result

$$\mathbf{g}_k^T\mathbf{d}_j = 0 \,, \qquad \text{for } j < k < n \,. \tag{13}$$

This orthogonality property is important because it shows that at each step, the $\mathbf{x}_{k+1}$ defined by (7), (8) and (9) minimizes $f(\mathbf{x})$ over the entire subspace spanned by $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_k$. Furthermore, it leads us to a fact that we shall need shortly; because $\mathbf{g}_{k+1}^T\mathbf{d}_k = 0$, α_k is the solution to

$$\min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{d}_k) \,. \tag{14}$$

The minimum of $f(\mathbf{x}_k + \alpha\mathbf{d}_k)$ occurs at the point where $\partial f(\mathbf{x}_k + \alpha\mathbf{d}_k)/\partial\alpha = 0$. Differentiating with respect to α yields

$$\frac{\partial f(\mathbf{x}_k + \alpha\mathbf{d}_k)}{\partial\alpha} = \nabla f(\mathbf{x}_k + \alpha\mathbf{d}_k)^T\mathbf{d}_k = \mathbf{g}^T\mathbf{d}_k \,. \tag{15}$$

This dot product is known as the directional derivative (the derivative of $f(\mathbf{x}_k)$ along the direction $\mathbf{d}_k$) and vanishes when $f(\mathbf{x}_k + \alpha\mathbf{d}_k)$ is minimized with respect to α (i.e. when $\alpha = \alpha_k$). We will encounter the directional derivative again during our discussion of line search techniques.

2.2. *The conjugate gradient algorithm*

The obvious question at this point is: "Where does one get the direction vectors?" There are several conjugate directions algorithms that generate these vectors as part of the iteration. By far the most important is the method of conjugate gradients. The initial direction vector is chosen as the negative gradient at the initial point: $\mathbf{d}_0 = -\mathbf{g}_0$. Successive directions are obtained from a linear combination of the current gradient and the previous direction;

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k \,. \tag{16}$$

The requirement that successive directions be **Q**-conjugate or

$$\mathbf{d}_{k+1}^T\mathbf{Q}\mathbf{d}_k = [-\mathbf{g}_{k+1} + \beta_k\mathbf{d}_k]^T\mathbf{Q}\mathbf{d}_k = 0 \tag{17}$$

gives

$$\beta_k = \frac{\mathbf{g}_{k+1}^T\mathbf{Q}\mathbf{d}_k}{\mathbf{d}_k^T\mathbf{Q}\mathbf{d}_k} . \tag{18}$$

Fortunately, using (16) and (18) to force conjugacy between successive directions suffices to make $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ a **Q**-conjugate set. We will prove this by induction. The scalar β_0 is selected to force $\mathbf{d}_1^T\mathbf{Q}\mathbf{d}_0 = 0$. Assume that for some $k < n - 2$ we have $\mathbf{d}_i^T\mathbf{Q}\mathbf{d}_j = 0$ for all $j < i \leq k$. We must then show that $\mathbf{d}_i^T\mathbf{Q}\mathbf{d}_j = 0$ for all $j < i \leq k + 1$. First we show that under our assumption the first $k + 1$ gradients form an orthogonal set. Equation (16) shows that $\mathbf{d}_p$ is a linear combination of the first $p + 1$ gradients

$$\mathbf{d}_p = -\mathbf{g}_p + \sum_{q=0}^{p-1} \gamma_{p,q}\mathbf{g}_q . \tag{19}$$

Because the first k directions are assumed to be **Q**-conjugate, it follows from (13) that for $p < m \leq k + 1$, taking the inner product of $\mathbf{g}_m$ with (19) gives

$$\mathbf{g}_m^T\mathbf{g}_p = \sum_{q=0}^{p-1} \gamma_{p,q}\mathbf{g}_m^T\mathbf{g}_q . \tag{20}$$

Now $\mathbf{g}_m^T\mathbf{g}_0 = 0$ because $\mathbf{d}_0 = -\mathbf{g}_0$ and it follows from an inductive argument applied to (20) that the first $k + 1$ gradients are orthogonal;

$$\mathbf{g}_m^T\mathbf{g}_p = 0 , \qquad \text{for all } p < m \leq k + 1 . \tag{21}$$

Recall we selected β_k to force $\mathbf{d}_{k+1}^T\mathbf{Q}\mathbf{d}_k = 0$. We can now complete the proof by showing that $\mathbf{d}_{k+1}^T\mathbf{Q}\mathbf{d}_j = 0$ for $j < k$. Taking the inner product of $\mathbf{Q}\mathbf{d}_j$ with (16) gives

$$\mathbf{d}_{k+1}^T\mathbf{Q}\mathbf{d}_j = -\mathbf{g}_{k+1}^T\mathbf{Q}\mathbf{d}_j + \beta_k\mathbf{d}_k^T\mathbf{Q}\mathbf{d}_j . \tag{22}$$

The second term on the right hand side is zero from our assumption and (10) shows that the first term is $-\mathbf{g}_{k+1}^T[\mathbf{g}_{j+1} - \mathbf{g}_j]/\alpha_k$, which is zero because the gradients are orthogonal.

The algorithm is summarized below:

1. Set $k = 0$, select the initial point $\mathbf{x}_0$. (23a)

2. $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$.
   If $\mathbf{g}_0 = \mathbf{0}$, stop, else set $\mathbf{d}_0 = -\mathbf{g}_0$. (23b)

3. $\alpha_k = -\dfrac{\mathbf{d}_k^T\mathbf{g}_k}{\mathbf{d}_k^T\mathbf{Q}\mathbf{d}_k}$. (23c)

4. $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k$. (23d)

5. $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$.
   If $\mathbf{g}_{k+1} = \mathbf{0}$, stop. (23e)

6. $\beta_k = \dfrac{\mathbf{g}_{k+1}^T\mathbf{Q}\mathbf{d}_k}{\mathbf{d}_k^T\mathbf{Q}\mathbf{d}_k}$. (23f)

7. $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k\mathbf{d}_k$. (23g)

8. If $k = n - 1$, stop, else $k \leftarrow k + 1$, go to 3. (23h)

2.3. *Extensions to non-quadratic problems*

We have shown that the conjugate gradient algorithm is indeed a conjugate directions method that minimizes a positive definite quadratic function in n steps. The algorithm can be extended to general nonlinear functions by interpreting (1) as a second order Taylor series expansion of the objective function. The justification for this is, near the solution, the Taylor series expansion shows that such functions behave approximately as quadratics. Furthermore, much computational experience has shown that the conjugate gradient algorithm outperforms steepest descent methods at points far from the solution.[14] For a quadratic, **Q** is the matrix of second partial derivatives (i.e. the Hessian) and is constant. However, for a general nonlinear function, the Hessian is a function of $\mathbf{x}_k$ and is very CPU intensive to compute. An efficient implementation of the conjugate gradient method would be possible with the elimination of the calculation of the Hessian. Note that **Q** appears only in the computation of the scalars α_k and β_k. Eliminating **Q** from the calculation of these scalars results in an algorithm which depends only on the function and gradient values at each iteration. We begin our discussion by considering how to modify the algorithm for a quadratic function in the case where **Q** is not known but function values and gradients are available.

Recall from our earlier discussion that α_k minimizes $f(\mathbf{x}_k + \alpha\mathbf{d}_k)$, i.e.

$$\alpha_k = \text{argmin}_\alpha\{f(\mathbf{x}_k + \alpha\mathbf{d}_k)\} \ . \tag{24}$$

Therefore, the closed form solution for α_k in (9) can be replaced by a numerical one-dimensional minimization or line search procedure. In addition, using (10) we can eliminate $\mathbf{Q}$ from (18), which gives

$$\beta_k = \frac{\mathbf{g}_{k+1}^T[\mathbf{g}_{k+1} - \mathbf{g}_k]}{\mathbf{d}_k^T[\mathbf{g}_{k+1} - \mathbf{g}_k]} \ . \tag{25}$$

This is the Hestenes–Stiefel formula for β_k.[3] The orthogonality relationships (13) and (20) provide two alternate expressions for β_k; the Polak–Ribière formula[5]

$$\beta_k = \frac{\mathbf{g}_{k+1}^T[\mathbf{g}_{k+1} - \mathbf{g}_k]}{\mathbf{g}_k^T\mathbf{g}_k} \tag{26}$$

and the Fletcher-Reeves formula[4]

$$\beta_k = \frac{\mathbf{g}_{k+1}^T\mathbf{g}_{k+1}}{\mathbf{g}_k^T\mathbf{g}_k} \ . \tag{27}$$

We now have a conjugate gradient algorithm which does not require an explicit knowledge of $\mathbf{Q}$, only function and gradient values at each iteration.

With a few slight modifications, the algorithm can be applied to general nonlinear functions. The termination criterion in (23e) is not a good one, since the numerical computation of the gradient will most likely never be identically equal to zero. In practice, the algorithm is terminated if the function is less than a pre-set maximum allowable value or if the maximum number of iterations have been reached. Furthermore, for non-quadratic problems, the algorithm will not successfully converge in exactly n steps (from a numerical viewpoint, this is true of the quadratic case also). As the algorithm progresses, the $\mathbf{Q}$-conjugacy of the direction vectors deteriorates. Hence, it is common practice to reinitialize the direction vector to the negative gradient at every nth iteration and continue until the algorithm converges or the maximum number of iterations is reached. The reinitialization is known as a "restart" and effectively restarts the entire algorithm at the current point. This is a relatively simple method of performing a restart. However, other more sophisticated methods exist.[10,11]

2.4. *Line search*

The purpose of the line search is to minimize $f(\mathbf{x} + \alpha\mathbf{d})$ with respect to α. That is, given fixed vectors $\mathbf{x}$ and $\mathbf{d}$, we must vary α such that $f(\mathbf{x} + \alpha\mathbf{d})$ is minimized. As α varies, $\mathbf{x} + \alpha\mathbf{d}$ forms a line in the n dimensional vector space of $\mathbf{x}$, hence the name line search. Typically, a line search involves bracketing or straddling the minimum (ensuring that two points on either side of the minimum are known), then using function and gradient information to fit a second or third order curve to the data and estimating the minimum point (quadratic or cubic approximation). The curve fit approximation is iterated until the proper termination criteria are reached.

The accuracy of the line search plays a critical role in the performance of the conjugate gradient algorithm. Determining a very accurate approximation to the minimum is usually inefficient because it requires many function and gradient evaluations. Consequently, it is desirable to have criteria that terminate the line search when a "reasonably" accurate estimate has been obtained.[6] In the case of inaccurate line search, analysis of the quadratic problem suggests that the Hestenes–Stiefel formula for β_k may be preferable. Without exact values for α_k, the direction vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ will not be a $\mathbf{Q}$-conjugate set, even in the quadratic case but the Hestenes–Stiefel formula at least forces $\mathbf{Q}$-conjugacy between successive directions as in (17). For the general nonlinear problem, one indication that the algorithm is getting stuck is that very small steps are being taken, orthogonality between successive gradients is lost and $\mathbf{g}_{k+1} \approx \mathbf{g}_k$. In this case, the Polak–Ribière formula has an advantage because it yields a β_k that is nearly zero and forces the algorithm to take what is approximately a steepest descent step. The best formula for β_k is highly problem-dependent; we provide comparisons for all three formulas in Sec. 4.

Shanno has shown that use of an inaccurate line search with the Polak–Ribière or Fletcher–Reeves methods may yield a $\mathbf{d}_{k+1}$ which is not necessarily a descent direction, and numerical instability may result.[13] He derives a method for calculating $\mathbf{d}_{k+1}$ which guarantees a descent direction;

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} - \left[\left(1 + \frac{\mathbf{y}_k^T\mathbf{y}_k}{\mathbf{p}_k^T\mathbf{y}_k}\right)\frac{\mathbf{p}_k^T\mathbf{g}_k^T}{\mathbf{p}_k^T\mathbf{y}_k} - \frac{\mathbf{y}_k^T\mathbf{g}_k}{\mathbf{p}_k^T\mathbf{y}_k}\right]\mathbf{p}_k + \frac{\mathbf{p}_k^T\mathbf{g}_k}{\mathbf{p}_k^T\mathbf{y}_k}\mathbf{y}_k \ , \tag{28}$$

where $\mathbf{p}_k = \alpha_k\mathbf{d}_k$ and $\mathbf{y}_k = (\mathbf{g}_{k+1} - \mathbf{g}_k)$. Shanno's method is included in the comparisons presented in Sec. 4.

3. Conjugate Gradient Backpropagation

To apply the conjugate gradient method to backpropagation, we first view backpropagation learning as an unconstrained non-linear optimization problem. Thus, we require a vector of independent variables, $\mathbf{x}$, and an objective function, f, defined on the vector space of $\mathbf{x}$, which is to be minimized. We must be able to evaluate $f(\mathbf{x})$ and its gradient $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$ at any point $\mathbf{x}$. Recall that the goal of training the network is to modify the weights in such a manner that the network outputs for each pattern match the desired outputs. Therefore, the weights will be the vector of independent variables. Although each weight is associated with a layer in the network and a particular neuron within that layer, for the purposes of the conjugate gradient minimization they are considered to be a single one-dimensional vector. The weights are ordered in the vector by layer and then by neuron and number within the neuron. The normalized sum of the output errors over the training set will be the objective function.

More formally, we define

$$f(\mathbf{x}) = \frac{1}{P}\sum_{p} E_p \,, \tag{29}$$

where P is the number of patterns in the training set and E_p is the output error for each pattern p. E_p is defined as

$$E_p = \frac{1}{2}\sum_{j} (t_{pj} - o_{pj}(\mathbf{x}))^2 \,, \tag{30}$$

where $o_{pj}(\mathbf{x})$ and t_{pj} are the actual and desired outputs of the jth output neuron for pth pattern, respectively. We have denoted o_{pj} as a function of $\mathbf{x}$ to indicate the dependency of the network outputs on the weight vector $\mathbf{x}$. Thus, the objective function is

$$f(\mathbf{x}) = \frac{1}{2P}\sum_{p}\sum_{j} (t_{pj} - o_{pj}(\mathbf{x}))^2 \,. \tag{31}$$

A single function evaluation requires that the entire training set be passed through the network, the errors be calculated for each pattern and the results summed and normalized. As the size of the weight vector and number of patterns in the training set increase, the cost of computing $f(\mathbf{x})$ also increases.

To compute the gradient of the objective function, $\nabla f(\mathbf{x})$, we differentiate (29) with respect to $\mathbf{x}$, which gives

$$\mathbf{g}(\mathbf{x}) = \frac{1}{P}\sum_{p} \nabla E_p(\mathbf{x}) \,. \tag{32}$$

The derivation of $\nabla E_p(\mathbf{x})$ is nearly identical to the derivation of $(\partial E_p/\partial w_{ji})$ in Rumelhart[1] and is summarized briefly here. The vector $\nabla E_p(\mathbf{x})$ is composed of elements $(\partial E_p/\partial \mathbf{x}_{ji})$, where $\mathbf{x}_{ji}$ is the weight element from the ith to the jth neuron. We now write the derivative as a product of two parts

$$\frac{\partial E_p}{\partial \mathbf{x}_{ji}} = \frac{\partial E_p}{\partial net_{pj}}\frac{\partial net_{pj}}{\partial \mathbf{x}_{ji}} \,, \tag{33}$$

where $(\partial E_p/\partial net_{pj})$ represents the change in E_p due to a change in the input of the jth neuron and $(\partial net_{pj}/\partial \mathbf{x}_{ji})$ represents the change in the input of the jth neuron due to a change in the weight $\mathbf{x}_{ji}$. The input to the jth neuron, net_{pj}, is defined as

$$net_{pj} = \sum_{i} \mathbf{x}_{ji} o_{pi}(\mathbf{x}) \,, \tag{34}$$

where $o_{pi}(\mathbf{x})$ is the output of the ith neuron ($o_{pi}(\mathbf{x})$ is the ith input when i is an input neuron). Using (34), it can be shown that

$$\frac{\partial net_{pj}}{\partial \mathbf{x}_{ji}} = o_{pi}(\mathbf{x}) \,. \tag{35}$$

We now define

$$\delta_{pj} = \frac{\partial E_p}{\partial net_{pj}} \,. \tag{36}$$

Note that this differs from the δ_{pj} of Rumelhart by a minus sign. Thus,

$$\frac{\partial E_p}{\partial \mathbf{x}_{ji}} = \delta_{pj} o_{pi}(\mathbf{x}) \,. \tag{37}$$

Using this definition of δ_{pj} and following the same arguments as Rumelhart, we arrive at the following relationships. For output neurons,

$$\delta_{pj} = -(t_{pj} - o_{pj}(\mathbf{x}))s_j'(net_{pj}) \,, \tag{38}$$

where $s_j'(net_{pj})$ is the derivative of the semi-linear activation or squashing function and for hidden layer neurons

$$\delta_{pj} = s_j'(net_{pj})\sum_{k} \delta_{pk}\mathbf{x}_{kj} \,. \tag{39}$$

Therefore, evaluating the gradient requires forward propagating each pattern in the training set through the network to generate the neuron outputs and then propagating the values of δ_{pj} backwards through the network. The final gradient is computed by summing $\nabla E_p(\mathbf{x})$ over all the patterns. Since the conjugate gradient method requires evaluating both the function and gradient values, the calculations should be performed together to maximize efficiency.

4. Comparison with Conventional Backpropagation

This section describes the implementation of the conjugate gradient and conventional backpropagation algorithms, the basis for comparing the two, the details of the comparison test and concludes with the results of the test.

4.1. *Conjugate gradient backpropagation implementation*

The following conjugate gradient algorithm was implemented:

1. A starting point, $\mathbf{x}_0$, is selected by initializing the weights randomly, uniformly distributed between -0.5 and $+0.5$. The gradient, $\mathbf{g}_0$, is computed at this point (as described above) and an initial direction vector, $\mathbf{d}_0 = -\mathbf{g}_0$ is selected.
2. The constant α_k which minimizes $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k)$ is computed by line search. The weight vector is updated to the new point: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.
3. The termination criteria are evaluated at the new point. If the error at this point is acceptable or if the maximum number of iterations have been reached, the algorithm terminates.
4. A new direction vector, $\mathbf{d}_{k+1}$ is computed. If $k + 1$ is an integral multiple of n, then $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1}$. Otherwise, $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$, where β_k is computed according to the method selected by the user: Fletcher–Reeves, Polak–Ribière or Hestenes–Stiefel.
5. k is replaced by $k + 1$ and the algorithm continues in Step 2.

The line search implemented is an adaptation of the line search in Shanno's CONMIN conjugate gradient code[12] and is summarized below:

1. The directional derivative for $\alpha = 0$ is computed.
2. An initial guess for α is made.
3. The point $f(\mathbf{x} + \alpha \mathbf{d})$ is evaluated and the directional derivative computed.
4. If the minimum is bracketed, the algorithm proceeds, else α is increased or decreased as required and Step 3 is repeated.
5. A cubic approximation to the minimum is generated and adjusted if necessary to ensure that it falls sufficiently within the bracketed interval. The new point and the directional derivative are evaluated.
6. If the new point meets the termination criteria (discussed below), the algorithm terminates, else the two points which are closest to the minimum and bracket it are kept and Step 5 is repeated.

The following criteria are used to judge if the line search should terminate:

$$|\mathbf{g}(\mathbf{x} + \alpha \mathbf{d})^T \mathbf{d}| \leq -\eta \mathbf{g}(\mathbf{x})^T \mathbf{d} , \qquad 0 \leq \eta < 1 \quad (40)$$

and

$$f(\mathbf{x}) - f(\mathbf{x} + \alpha \mathbf{d}) \geq -\mu \alpha \mathbf{g}(\mathbf{x})^T \mathbf{d} , \quad 0 < \mu \leq 1/2 . \quad (41)$$

The first requires that there be a sufficient decrease in the value of the directional derivative, the second that there be a suitable reduction in the value of f.[6] We implemented $\mu = 10^{-4}$ and left η to be specified by the user.

Due to the extreme nonlinear nature of the neural network error surface, it is necessary to implement limits on the number of function and gradient evaluations used in the initial search for an α which brackets the minimum and in the cubic approximation. This prevents the line search from getting "stuck" near extreme non-linearities in the surface. The limits are specified by the user at run time. A flag is set when the line search fails and a restart is attempted upon return to the main program (providing this was not already a restart iteration).

4.2. *Conventional backpropagation implementation*

The backpropagation algorithm we implemented is commonly referred to as "off-line" backpropagation. That is, the weights are updated after all of the patterns in the training set have been passed through the network. A momentum term is used in addition to the standard negative gradient learning rate term. The weight update equation is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \mathbf{g}_k - \alpha \mathbf{g}_{k-1} , \quad (42)$$

Table 1. Number of function evaluations required to achieve a final error $\leq 10^{-6}$ on the parity problem using the conventional backpropagation method. Optimizations which did not converge are noted with an asterisk (*).

Architecture	Learning Rate/Momentum						
	0.1/0.1	0.1/0.9	0.3/0.3	0.5/0.5	0.7/0.7	0.9/0.1	0.9/0.9
3-3-1	16572	3317	5526	3316	2370	3316	1843
4-4-1	50000*	50000*	50000*	50000*	50000*	50000*	50000*
5-5-1	50000*	50000*	50000*	50000*	50000*	28468	25087
3-3-3-1	28451	5593	9499	5496	3918	5677	3078
4-4-4-1	50000*	32039	30115	18358	13462	18345	50000*
5-5-5-1	50000*	50000*	50000*	50000*	50000*	50000*	50000*

where the learning rate, η, and the momentum rate, α, are specified by the user. The function and gradient values are computed as described previously.

4.3. *Comparison metric*

To compare conjugate gradient backpropagation with conventional backpropagation we require some kind of metric with which to judge algorithm performance. The number of iterations or cycles is not a valid metric, since in backpropagation the training set is passed through the network once per iteration, whereas in the conjugate gradient method the training set can be passed through several times in a single iteration. The majority of CPU time required to train a backpropagation network is spent computing the error function and its gradient. Computing the error function requires forward propagating the entire training set through the network. Computing the gradient requires backpropagating the outputs from each pattern in the training set (computed during the forward propagation) through the network.

Conventional backpropagation requires one forward propagation and one backpropagation per iteration. The conjugate gradient method will have several forward and backpropagations at each iteration. The number of forward and back propagations may or may not be the same, depending on the type of line search used. Some line searches use both the function value and the gradient at each step, some use only the function value and some use a combination of the two. Since most of the CPU time is spent computing the error function and gradient, the logical metric for comparison is the total number of function calculations *and* the total number of gradient calculations required to train a network. For the conjugate gradient algorithm we implemented, the number of function calculations and gradient calculations are the same. Therefore, the comparison metric is simply the number of function evaluations required to train the network.

4.4. *The benchmark test*

The parity problem was chosen as the benchmark test for comparing the conjugate gradient backpropagation and conventional backpropagation methods. It was chosen because it is simple, well known in the literature and presents quite a difficult problem for neural networks to solve. Both one and two hidden layer networks were tested on three, four and five bit parity problems (i.e. 3-3-1, 4-4-1, 5-5-1, 3-3-3-1, 4-4-4-1 and 5-5-5-1 architectures). A full training set was used for each test (2^n patterns), with a stopping criterion of 10^{-6} for the normalized system error (the average of the pattern errors). The number of function evaluations were limited to 50000; the test terminated if this limit was reached. The weights were initialized to random numbers uniformly distributed between -0.5 and $+0.5$. Several combinations of learning rate and momentum rate were used to test the conventional backpropagation method, while each of the different conjugate gradient methods was tested using various values of the line search convergence parameter.

5. Results

The test results for the conventional backpropagation method are shown in Table 7. The test results for the conjugate gradient backpropagation method and steepest descent with line search (included for comparison purposes) are shown in Tables 1–7. The four and five bit parity problems proved to be the most difficult of the tests, with the single hidden layer problems being more difficult than the two hidden layer problems. This is evidenced by the higher number of convergence failures on the single hidden layer problems for both the conventional and conjugate gradient backpropagation methods. Of the conjugate gradient backpropagation methods, the Polak–Ribière and Hestenes–Stiefel formulas gave the best results overall. Although Shanno's method was somewhat

Table 2. Number of function evaluations required to achieve a final error $\leq 10^{-6}$ on the parity problem using the Fletcher–Reeves conjugate gradient backpropagation method. Optimizations which did not converge are noted with an asterisk (*).

Architecture	Convergence Parameter η 0.01	0.1	0.5	0.9
3-3-1	656	331	388	264
4-4-1	8555*	50000*	3465	1617
5-5-1	2641*	3745*	2145	3921
3-3-3-1	710	473	452	372
4-4-4-1	737	2238	1791	1544
5-5-5-1	50000*	1561	3421	4224

Table 3. Number of function evaluations required to achieve a final error $\leq 10^{-6}$ on the parity problem using the Polak–Ribière conjugate gradient backpropagation method. Optimizations which did not converge are noted with an asterisk (*).

Architecture	Convergence Parameter η 0.01	0.1	0.5	0.9
3-3-1	195	116	128	123
4-4-1	461	7068	12928	5256*
5-5-1	1966	9244	21633	6633*
3-3-3-1	275	244	200	200
4-4-4-1	415	401	430	1834
5-5-5-1	1385	1343	1849	1643

Table 4. Number of function evaluations required to achieve a final error $\leq 10^{-6}$ on the parity problem using the Hestenes–Stiefel conjugate gradient backpropagation method. Optimizations which did not converge are noted with an asterisk (*).

Architecture	Convergence Parameter η 0.01	0.1	0.5	0.9
3-3-1	148	150	121	147
4-4-1	2289	45510*	8731	306
5-5-1	3249	50000*	12342	10752
3-3-3-1	335	523	239	243
4-4-4-1	521	50000*	468	429
5-5-5-1	1314	1661	1165	50000*

Table 5. Number of function evaluations required to achieve a final error $\leq 10^{-6}$ on the parity problem using the Shanno conjugate gradient backpropagation method. Optimizations which did not converge are noted with an asterisk (*).

Architecture	Convergence Parameter η 0.01	0.1	0.5	0.9
3-3-1	512	191	249	396
4-4-1	2079	2651	3247	2482
5-5-1	3372	750	11728	2251
3-3-3-1	306	275	308	368
4-4-4-1	568	792	763	801
5-5-5-1	1890	4348	1985	7254

Table 6. Number of function evaluations required to achieve a final error $\leq 10^{-6}$ on the parity problem using the steepest descent with line search backpropagation method. Optimizations which did not converge are noted with an asterisk (*).

Architecture	Convergence Parameter η 0.01	0.1	0.5	0.9
3-3-1	978	1834	616	624
4-4-1	50000*	5505	17913	18340
5-5-1	50000*	50000*	14289	11894
3-3-3-1	3357	2542	1178	1148
4-4-4-1	4487	4588	3321	1840
5-5-5-1	50000*	50000*	11853	9500

Table 7. Comparison of best test results for conjugate gradient, steepest descent and conventional backpropagation. Abbreviations: FR: Fletcher–Reeves, PR: Polak–Ribière, HS: Hestenes–Stiefel, SH: Shanno, SD: steepest descent with line search, BP: conventional backpropagation.

Architecture	Method	Function Evaluations	Speed-up over Backpropagation
3-3-1	FR	264	6.98
3-3-1	PR	116	15.88
3-3-1	HS	121	15.23
3-3-1	SH	191	9.64
3-3-1	SD	616	2.99
3-3-1	BP	1843	N/A
4-4-1	FR	1617	N/A
4-4-1	PR	461	N/A
4-4-1	HS	306	N/A
4-4-1	SH	2079	N/A
4-4-1	SD	5505	N/A
4-4-1	BP	50000	N/A
5-5-1	FR	2145	11.70
5-5-1	PR	1966	12.76
5-5-1	HS	3249	7.72
5-5-1	SH	750	33.45
5-5-1	SD	11894	2.11
5-5-1	BP	25087	N/A
3-3-3-1	FR	372	8.27
3-3-3-1	PR	200	15.39
3-3-3-1	HS	239	12.88
3-3-3-1	SH	275	11.19
3-3-3-1	SD	1148	2.68
3-3-3-1	BP	3078	N/A
4-4-4-1	FR	737	18.27
4-4-4-1	PR	401	33.57
4-4-4-1	HS	429	31.38
4-4-4-1	SH	568	23.70
4-4-4-1	SD	1840	7.32
4-4-4-1	BP	13462	N/A
5-5-5-1	FR	1561	N/A
5-5-5-1	PR	1343	N/A
5-5-5-1	HS	1165	N/A
5-5-5-1	SH	1890	N/A
5-5-5-1	SD	9500	N/A
5-5-5-1	BP	50000	N/A

slower than Polak–Ribière and Hestenes–Stiefel, it never failed to converge, as did all the others on occasion. It is interesting to note that the method of steepest descent with line search is in general faster than conventional backpropagation.

The choice of conjugate gradient method and line search parameter which gives the best results appears to be highly problem dependent. There is no single parameter value or conjugate gradient method which gives the best results overall or favors a particular architecture or layer structure. Indeed, our experience using conjugate gradient techniques has shown that the user must experiment to determine the optimum combination of conjugate gradient method and line search termination criteria for a particular application. However, for most of the applications we have encountered, the Hestenes–Stiefel or Polak–Ribière methods used with a line search termination parameter of $\eta = 0.1$ are quite satisfactory.

The best results for each problem are summarized in Table 7, which also includes the relative speed-up of each method over conventional backpropagation. We see that for each problem, the conjugate gradient methods are an order of magnitude faster than conventional backpropagation. Although the statistical accuracy of these results could be improved by averaging the tests over a number of trials, our intent here has been to show that the conjugate gradient method can easily be used as a neural network learning algorithm and to demonstrate the approximate level of speedup of which it is capable. Furthermore, the relative successes of optimization algorithms are highly problem dependent and one difficulty with comparing algorithms is that it is never clear how to best choose an ensemble of representative problems over which to average. However, our experience using the conjugate gradient method to solve neural network as well as other problems indicates that these results are typical.

6. Conclusions

In this paper, we have successfully applied the conjugate gradient minimization method to the neural network learning problem. We have tested conjugate gradient backpropagation and conventional backpropagation using three, four and five bit parity problems on neural network architectures with both one and two hidden layers. In all test cases, the conjugate gradient methods are an order of magnitude faster than conventional backpropagation. Moreover, the conjugate gradient method's combination of unique direction vectors and line search minimization results in an algorithm which is not susceptible to the possible instabilities and oscillatory behavior associated with the use of a fixed step size as in the conventional backpropagation method. Furthermore, our experience applying neural networks to a wide variety of problems[15,16] has shown that conjugate gradient backpropagation is clearly superior in overall performance to conventional backpropagation.

7. Acknowledgements

We would like to thank the LLNL Engineering Research Program and Bob Glass of the LLNL Chemistry Department for their support of this work. We would also like to thank the members of the Artificial Neural Network Interest Group at LLNL for their input and comments on this work and Sean K. Lehman for his software development work.

This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.

References

1. 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, Vol. 1: Foundations*, eds. D. E. Rumelhart and J. L. McClelland (MIT Press, 1986).
2. Y. Pao, *Adaptive Pattern Recognition and Neural Networks* (Addison-Wesley, 1989).
3. M. R. Hestenes and E. L. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, **49**(6), 409–436 (1952).
4. R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *Computer Journal* **7**, 149–154 (1964).
5. E. Polak and G. Ribière, "Note sur la convergence de methods de directions conjures," *Revue Francaise Information Recherche Operationnelle* **16**, 35–43 (1969).
6. P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization* (Academic Press, 1981).
7. D. G. Luenberger, *Linear and Nonlinear Programming* (Addison-Wesley, 1984).
8. S. Becker and Y. L. Cun, "Improving the convergence of back-propagation learning with second order methods," *Proc. 1988 Connectionist Models Summer School*, ed. Morgan Kaufman (1989), pp. 29–37.
9. A. H. Kramer and A. Sangiovanni–Vincentelli, "Efficient parallel learning algorithms for neural networks," in *Advances in Neural Information Processing Systems I*, ed. D. S. Touretzky, Morgan Kaufman (1989), pp. 40–48.

10. E. M. L. Beale, "A derivation of conjugate gradients," in *Numerical Methods for Non-linear Optimization*, ed. F. A. Lootsma (1972), pp. 39–43.
11. M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Mathematical Programming* **12**, 241–254 (1977).
12. D. F. Shanno and K. Phua, "Remark on algorithm 500," *ACM Transactions on Mathematical Software* **6**, 618–622 (1980).
13. D. F. Shanno, "Conjugate gradient methods with inexact line searches," *Mathematics of Operations Research*, **3** 244–256 (1978).
14. D. M. Goodman, T. W. Lawrence, J. P. Fitch, and E. M. Johansson, "Bispectral-based optimization algorithms for speckle imaging," in *Digital Image Synthesis and Inverse Optics*, eds. A. F. Gmitro, P. S. Idell, I. J. LaHaie, *Proc. SPIE* **1351**, (1990) 546–550.
15. F. U. Dowla, S. R. Taylor and R. W. Anderson, "Seismic discrimination with artificial neural networks," *Bulletin of the Seismological Soc. of America*, **80**(5), 1346–1373 (1990).
16. J. P. Fitch, S. K. Lehman, F. U. Dowla, S. Y. Lu, E. M. Johansson and D. M. Goodman, "Ship wake detection procedure using conjugate gradient trained artificial neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, **29**(5), 718–726 (1991).