LEARNING ALGORITHMS FOR CONNECTIONIST NETWORKS: APPLIED GRADIENT METHODS OF NONLINEAR OPTIMIZATION

Raymond L. Watrous

MS-CIS-88-62
LINC LAB 124

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

(Revised July 1988)

Acknowledgements: This research was supported in part by grants DARPA/ONR-NOOO14-85-K-0807, DARPA-NOOO14-85-K-0018, NSF-CER grant MCS-8219196 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization

Raymond L. Watrous
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA
Siemens Research and Technology Laboratories
Princeton, NJ

May 5,1987
Revised July 22, 1988

Abstract

The problem of learning using connectionist networks, in which network connection strengths are modified systematically so that the response of the network increasingly approximates the desired response can be structured as an optimization problem. The widely used back propagation method of connectionist learning [19, 21, 18] is set in the context of nonlinear optimization. In this framework, the issues of stability, convergence and parallelism are considered. As a form of gradient descent with fixed step size, back propagation is known to be unstable, which is illustrated using Rosenbrock's function. This is contrasted with stable methods which involve a line search in the gradient direction. The convergence criterion for connectionist problems involving binary functions is discussed relative to the behavior of gradient descent in the vicinity of local minima. A minimax criterion is compared with the least squares criterion.

The contribution of the momentum term [19, 18] to more rapid convergence is interpreted relative to the geometry of the weight space. It is shown that in plateau regions of relatively constant gradient, the momentum term acts to increase the step size by a factor of $\frac{1}{1-\mu}$, where μ is the momentum term. In valley regions with steep sides, the momentum constant acts to focus the search direction toward the local minimum by averaging oscillations in the gradient.

The Davidon-Fletcher-Powell [8] and Broyden-Fletcher-Goldfarb-Shanno [7] methods are considered in light of computational complexity (time and space), convergence properties, and suitability to parallel machines. These algorithms approximate the second derivative of the objective function iteratively. This additional information about the shape of the weight space allows for dramatically faster convergence. The performance of these algorithms is compared with the steepest descent and back propagation algorithms for several sample connectionist problems, including exclusive-OR and a multiplexor problem.

It is concluded that for moderate sized problems to be solved on sequential machines, the use of higher-order techniques is mandated by their excellent convergence properties.

1 Introduction

The ability to extract structure from data is an important prerequisite for artificial intelligence, and especially for artificial perception. Here, it is often uncertain which characteristics of the object of perception, as mediated by the sensing mechanism, serve to define the corresponding psychological categories of perception. Thus, the relationship between stimulus and percept is difficult to specify.

This is the case, for example, in speech recognition, where the acoustic correlates of phonetic categories have been difficult to determine, since they are multi-dimensional, context-dependent and speaker dependent.

A direct approach to this problem is to observe parameters of the stimulus and encode regularities in rules. For example, many of the methods for speech recognition are derived from visual observations of the speech waveform represented as spectral coefficients over time. Consequently, a process which is inherently time-dependent is viewed spatially as a static pattern. This difference in presentation may have negative consequences. Consequently, the learning approach provides a means for extraction of significant signal characteristics which may be less obvious visually, but inherent nevertheless in the acoustic signal [5].

The human speech processing system is extremely sensitive to small variations along certain dimensions of the acoustic signal, and indifferent to variations along others. Thus, attempts to formalize in rules the regularities of the signal must be flexible with regard to variations. In other approaches, an estimate of the dimensions and correlations of the acoustic space as related to the perceptual space is computed. Where this dimensional analysis is inadequate, a small variation in the input signal leads to an error in clas-

sification; this is, in fact, often characteristic of current speech recognition systems.

The strength of the connectionist approach seems to be that the multiple sources of knowledge can be appropriately integrated, and that statistical covariance of multiple cues can be estimated accurately from small data samples [9].

Learning in connectionist networks is not only important, it is in general difficult. The networks are often complex, involving many nodes and even more interconnections. This results in a learning problem of high dimensionality. In addition, the computational characteristics of the network nodes are typically nonlinear. This nonlinearity constrains the choice of learning algorithm which will solve the problem.

Such learning algorithms exist and have been used very successfully in solving a wide variety of connectionist learning tasks [19, 11, 10]. These algorithms obtain problem solutions often after thousands of iterations, which may take minutes or hours of processing. Consequently, the question of computational efficiency and complexity is important. Are there more powerful methods of learning which will find correct solutions in significantly shorter time?

The problem of learning using connectionist networks, in which the network connection strengths are modified systematically so that the response of the network increasingly approximates the desired response, can be structured as an optimization problem.

In the past thirty years, the subject of numerical techniques for nonlinear optimization has been extensively researched and is highly developed [12, 15, 7]. The properties of stability and convergence rate for many optimization algorithms have been reported and investigated computationally.

The goal of this paper is to set the widely used back propagation method of connectionist learning [19, 21, 18] in the context of nonlinear optimization. In this framework, the issues of stability, convergence and parallelism are considered. In addition, the contribution of the momentum term to more rapid convergence is interpreted relative to the geometry of the weight space.

Further, several more powerful optimization methods are considered in light of computational complexity (time and space), convergence properties, and suitability to parallel machines. It is concluded that for moderate sized problems on sequential machines, the use of higher order techniques is mandated by their excellent convergence properties.

2 Nonlinear Optimization

The problem of optimization has been widely studied in many disciplines. Indeed, as Tsypkin notes [22, page 5].

> One can say without exaggeration that the problem of optimality is a central problem of science, engineering and even everyday life.

It surfaces in control theory, economics, operations research, chemical engineering, VLSI design and artificial intelligence. The mathematical theory is highly developed, and well-founded on proofs of convergence rates and stability for many powerful algorithms. The availability of digital computers since 1950 has fostered the enrichment of the field with numerical algorithms which can be applied to very large problems, with thousands of variables. These well-researched techniques are most appropriate for use by researchers in connectionist learning, which often involves nonlinear optimization of large networks.

2.1 Overview

In order to discuss intelligently any method of optimization, it is necessary first to give a careful definition of the optimization problem, and introduce some formal notation. It is also necessary to locate, within the vast landscape of general optimization algorithms, the set of methods under consideration. The following overview serves that purpose.

2.1.1 Optimality Criteria

To cast a particular problem as an optimization problem requires, among other things, a definition of optimality. The value or effectiveness of a given solution is measured in terms of this optimality criterion, which is variously called the objective function, performance index, or cost function.

This measure can be defined [22] as an expected value

$$f(x) = \int_C Q(x,c)p(c)dc \tag{1}$$

where $Q(x,c)$ is the cost functional of the solution space vector x and the input vector c, from the space of input vectors C. The statistical distribution of c is given by $p(c)$.

For outputs which are deterministic (non-stochastic) functions of the inputs, which are sets or sampled processes, the performance index can be expressed [22] as:

$$f(x) = \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N} Q(x, c[n]) \tag{2}$$

For connectionist learning problems, the function Q can be made explicit in terms of the difference between the actual and desired responses of the network to the input set of items.

In general, Q is not a linear function of x or $c[n]$, and the optimization problem is consequently nonlinear. This is typically the case with connectionist networks, for two reasons. First, the input-output characteristics of neurological systems, which are paradigmatic for connectionist networks, are decidedly nonlinear. Second, it appears that it is exactly the nonlinear characteristics of connectionist networks which permit robust solutions to interesting and difficult problems [9, 14].

2.1.2 Constraints

Another consideration in the formulation of an optimization problem is the expression of constraints on the possible solutions. These constraints can be of various sources and forms, such as physical laws, limits on resources of time and money, feasible policies, etc. In the case of connectionist networks, relevant constraints might arise from a goal to maintain some plausibility for the network as a biological model, in which case bounded connection weights corresponding to finite synaptic strengths or limits on time constants corresponding to neural properties of temporal integration and diffusion should be considered. On the other hand, a goal of implementing connectionist networks in hardware might generate constraints due to limited data precision, unit interconnection strategies, and limits on local processing power.

Although the issue of constraints is important for optimization problems, it would complicate a basic picture comparing types of algorithms. Also, for many practical connectionist problems, unconstrained optimization leads to solutions within the (inexplicit) constraints. For these reasons, the issue of constraints in optimization will not be considered further.

2.1.3 Algorithms

There is a vast domain of optimization algorithms from which to choose. Some discriminating features are useful for classifying the choices. The

distinction between gradient and search methods is basic. Search methods, such as the simplex search [16], perform function evaluations at various point in the variable space, the selection of which is motivated by results at previous points. Consequently, the search methods can avoid small local minima in the search for the global minimum. However, the number of points which must be evaluated in a search method can be intractable for large problems, little use is made of the shape of the error surface. Gradient methods are appropriate where the error surface is smooth and the gradient is computable.

Another major division of methods is between stochastic and deterministic methods. The stochastic methods have the property of arbitrarily increasing the probability of locating the minimum at the cost of greatly increased steps in the search process. For this reason, the deterministic methods have been selected for study here. Stochastic methods can be used in conjunction with deterministic methods, to evaluate the stability of the region of the minimum.

In concluding this overview, then, the focus of this paper is on *iterative deterministic gradient* methods for solutions to nonlinear optimization problems.

2.2 Steepest Descent

There is a common and fundamental structure to the methods of iterative solutions to optimization problems, which consists of a cycle of two subproblems [13, Chapter 7]. First, from a chosen starting point, a desirable direction of search is computed. Then, the function is minimized along the direction of search. This cycle is repeated until some stopping criteria are met.

The most obvious choice for a search direction is defined by the gradient vector,

$$g = \nabla f(x) = \frac{\partial f}{\partial x_i}$$

The search is conducted in the opposite direction of the gradient, and is called steepest descent.

The convergence characteristics of steepest descent are well-known. For a quadratic function,

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

the rate of convergence can be shown to depend on the ratio of the largest and smallest eigenvalues of Q [13, pages 215-219]. This can be appreciated from a graphical point of view as the eccentricity of the error surface. The more skewed the curvatures of the space, the more the successive line searches oscillate across the ideal path toward the minimum. If, by a process of scaling, the eigenvalues are made approximately equal, the error surface approaches spherical, and the gradient from any point is directly toward the minimum.

The method for computing the optimal search direction is what serves to categorize iterative optimization algorithms. They differ in the method of direction selection, and employ some method of line search. The line search algorithm, however, can affect the overall performance significantly [7], and should be considered carefully.

2.2.1 Line Search

The line search is an optimization subproblem which is generally a standard component in a more complex optimization algorithm. The problem is to find the minimum value of the objective function along a particular direction of search.

$$\min_{\alpha}(f(x + \alpha\sigma))$$

where α is a scalar, and σ is the search direction vector.

There are three aspects of the line search, bracketing, sectioning, and interpolation [7]. A bracket maintains the permissible range for α, which is reduced in a controlled way as the search progresses. The bracket is used to protect against undesirable results of an interpolation algorithm for which assumptions about the objective function are not met; for example, extrapolation from a point in a function which is not convex in the interval could lead to values of α outside the interval, and result in a failure to converge.

Sectioning is a method for subdividing the range of α in order to reduce the interval in which the minimum must lie. It is generally assumed that the objective function is unimodal in the region of interest, and that the process of sectioning can be carried out to isolate the neighborhood of that minimum. The most efficient method, as measured by the rate of reduction of the interval of uncertainty, can be shown to be the Fibonacci search [13].

In this case, the interval reduction rate is

$$\frac{F_{N-1}}{F_N}$$

where N is the number of measurements to be made. As N becomes large, this ratio becomes:

$$\lim_{N\to\infty} \frac{F_{N-1}}{F_N} = \frac{2}{1+\sqrt{5}} \simeq 0.618$$

which is defines the Golden Section search method.

More powerful methods can be employed in the case of smooth functions, especially if the gradient is included in the line search. These methods include Newton's method, which requires the gradient and its first derivative, and the method of false position, which approximates the gradient derivative as a difference of two gradients [13]. These methods require that the starting point be sufficiently close to the actual minimum, in order to guarantee convergence. They are, therefore, less desirable for a general solution.

The methods of interpolation (and extrapolation) are based on low-order polynomial approximations to the objective function. The coefficients of the polynomial can be computed from a few points, and the corresponding minimum estimated directly. The quadratic interpolation method uses three data points and has an order of convergence of 1.3 [13]. The cubic method requires two data points and their associated gradient values, and has a convergence order of 2 [13].

In general, very few points are maintained in search methods. Most objective functions are not simple, and the accumulation of information about their precise behavior is apparently not expected to greatly increase the efficiency of search.

The initial estimate of α also bears consideration [7, page 28]. In practice, it has been found that

$$\alpha = \min(1, -2\frac{\Delta f}{g \cdot \sigma})$$

works well, where

$$\Delta f = \max(f_{k+1} - f_k, 10\epsilon)$$

The line search involves design decisions in the tradeoff between accuracy and computational cost, especially since high accuracy along search directions at points far from the global minimum may be unnecessary. Consequently, several criteria have been developed for terminating line searches

prior to absolute convergence. These tests include the percentage test, Armijo's Rule, the Goldstein Test, and the Wolfe test [13, pages 211-214]. These methods contain parameters which can be set to vary the accuracy of the line search, and also contain bracketing conditions which limit the acceptable values for α.

2.2.2 Accelerated Methods

As discussed above, the method of steepest descent is affected by the contour of the error-surface in N-space, and its convergence behavior is determined by the ratio of the largest and smallest eigenvalues of the objective function. Consequently, steepest descent makes increasingly slow progress the closer it gets to the solution.

It was observed that the successive gradient directions were nearly orthogonal and that alternate minima along the gradient directions defined a set of vectors which pointed directly to the global minimum. This observation lead to *accelerated gradient methods.* One early such method involved a weighted averaging of successive minima [24], while other methods combined successive gradient vectors in order to generate a search direction more consistently in the direction of the solution. This lead, in turn, to the method of parallel tangents (PARTAN) which generates and maintains parallel search vectors [13].

The importance of the accelerated gradient methods is that they represent an attempt to take into consideration the curvature of the objective surface without computation of the second derivative. More powerful techniques which address the same goal more rigorously are the Newton and quasi-Newton methods.

2.3 Quasi-Newton Methods

The gradient methods of optimization represent the use of a limited amount of knowledge about the objective function in computing its minimum. It is intuitively clear that the rate of convergence to the minimum should be proportional to the amount of knowledge about the objective function used in the optimization process. That is to say that second and higher order derivatives provide information about the error surface which can be exploited to achieve more rapid convergence. It is also clear that this increased information has an associated computational cost, so that the goal becomes most rapid convergence for a given computational effort.

The second derivative, which gives information about the curvature of the error surface, is the basis for a set of powerful optimization algorithms. The objective function is approximated by Taylor series expansion as

$$f(x+\delta) \approx f(x) + g^T\delta + \frac{1}{2}\delta^T G\delta$$

Thus, the objective function is approximated by a quadratic function, for which the minimum can be computed directly as

$$x = x_0 - G^{-1}g$$

which is Newton's method.

This method involves the computation of the $N \times N$ second derivative matrix and its inversion in order to solve for the minimum x. Since the computation of the second derivative may be impossible, or computationally expensive, and since the matrix inversion is $O(N^3)$, the so-called quasi-Newton methods were developed in which the inverse matrix is approximated iteratively. This avoids the direct computation of the second derivative, and the computational complexity is reduced by a factor of $O(N)$ [7].

The basic quasi-Newton algorithm consists of the following steps:

1. set a search direction $\sigma = -Hg$
2. minimize f along σ
3. update H

The initial inverse matrix H is selected to be symmetric positive definite [1], usually I. The matrix update procedures are designed to maintain the positive definiteness of H, and thus the descent property[2].

The heart of the algorithm is the update strategy for the approximate inverse Hessian. The general approach is to consider various update families which meet the quasi-Newton condition:

$$H_{i+1}\gamma_i = \delta_i$$

where $\gamma_i = g_{i+1} - g_i$ and $\delta_i = x_{i+1} - x_i$. Thus, the Hessian maps a change in gradient to a change in position. The families of possible update relations are indexed by the rank of the update formula. This will become clear in the following sections.

[1] A positive definite matrix has only positive eigenvalues. For the Hessian this implies positive curvature in the error surface, which is a necessary condition for a local minimum.

[2] The descent property, $g^T\sigma < 0$, guarantees that the search direction is downhill. This is ensured for $\sigma = -Hg$ if H is positive definite.

2.3.1 Rank One Correction

Consider the simple update relation

$$H_{i+1} = H_i + E_i$$

where

$$E_i = azz^T$$

The symmetric rank one[3] matrix E_i is constrained by the quasi-Newton condition as follows:

$$(H_i + E_i)\gamma_i = \delta i$$

Thus,

$$azz^T\gamma_i = \delta i - H_i\gamma_i$$

From the fact that $z^T\gamma_i$ is a scalar (dot product) it is clear that

$$z = \frac{1}{az^T\gamma_i}(\delta i - H_i\gamma_i)$$

Then choose $az^T\gamma_i = 1$, which implies that $z = \delta i - H_i\gamma_i$. Then,

$$H_{i+1} = H_i + \frac{zz^T}{z\gamma_i}$$

It can be proved that the rank one method converges on a quadratic function for N linearly independent vectors δ_i in at most $N+1$ iterations such that $H_{N+1} = G^{-1}$ [7, 13]. Which is merely to say that the inverse matrix can be found by N iterations, and the minimum computed in one step, given G^{-1}. The problem, however, is that the rank one method does not guarantee that positive definiteness is preserved. In addition, the denominator $z\gamma_i$ can become quite small, leading to numerical difficulties [13, page 265]. These problems are addressed by the rank two update formulae.

2.3.2 Rank Two Correction

A more flexible update formula for the inverse matrix approximation would be of the form:

$$H_{i+1} = H_i + azz^T + byy^T$$

[3]Rank is the maximum number of linearly independent rows, or columns. zz^T is obviously of rank one.

This is a rank two update formula, which has more degrees of freedom than the rank one method.

The first solution to this problem was advanced by Davidon [4] and improved by Fletcher and Powell [8], and is called the Davidon-Fletcher-Powell (DFP) method.

In brief, the quasi-Newton condition must again be met, and with obvious choices for $z = \delta$ and $y = H\gamma$, the constraints $az^T\gamma = 1$ and $by^T\gamma = -1$ arise, yielding as a complete solution:

$$H_{DFP} = H + \frac{\delta\delta^T}{\delta^T\gamma} - \frac{H\gamma\gamma^T H}{\gamma^T H\gamma} \tag{3}$$

Another solution to the rank two update equation was developed, which is the dual of the DFP algorithm, called the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [3, 6].

$$H_{BFGS} = H + \left(1 + \frac{\gamma^T H\gamma}{\delta^T\gamma}\right)\frac{\delta\delta^T}{\delta^T\gamma} - \frac{\delta\gamma^T H + H\gamma\delta^T}{\delta^T\gamma} \tag{4}$$

This algorithm is the dual of the DFP in the sense that B, the approximate Hessian matrix, and H are interchanged, as are γ and δ.

It was soon noted that these methods form a family, which can be generalized as

$$H_\phi = (1 - \phi)H_{DFP} + \phi H_{BFGS}$$

Since this update formula maintains positive definiteness for $\phi \geq 0$, this restriction is usually employed [13].

It has been shown that the Broyden family update procedures maintain positive definiteness, and converges in at most N iterations for quadratic functions [8]. However, subsequent application of the DFP method lead to occasional problems of instability, in which the matrix became singular [2]. The difficulty is due to scaling of the variable x, and the solution involves rescaling x and reinitializing H. This is apparently true of all members of the Broyden class.

It has been shown [13, page 271] that all members of the Broyden family generate the identical update sequences, and thus reach convergence in the same number of steps, provided an exact line search is used. In the case of inexact line searches, the BFGS algorithm has consistently given results superior to the DFP algorithm [7]. As a consequence, the BFGS algorithm is generally preferred over DFP.

3 Back Propagation Algorithm

The back propagation (BP), or error propagation algorithm for learning in connectionist networks was reported by Rumelhart, Hinton and Williams in 1985 [20, 19]. They developed a network learning algorithm called the generalized delta rule, which is a supervised learning procedure for connectionist networks with deterministic unit functions. The name error- or back propagation derives from the definition of an error value for each unit, which is recursively computed from the error values of units to which it is connected. The error at the output units is defined as the difference between the actual and desired values.

This algorithm provides a solution to the so-called fault-assignment problem, which for multi-layer networks had presented an obstacle to research in the use of connectionist networks. The solution of the weight modification problem contributed to breaking the effect of Minsky and Papert's [14] pessimistic report on connectionist network research. The realization that practical connectionist solutions using multi-layer networks to interesting problems could be achieved without a formal guarantee of convergence to a global optimum spurred a burst of papers on multi-layer network problems [21, 18, 5, 23].

The generalized delta rule is derived as a minimization of the squared-error between the actual and desired values of the output units. The algorithm is derived from the relation

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

where w_{ij} is the weight of the link connecting unit i to unit j, E is the squared-error, and η is the learning constant. Obviously, in light of the above discussion of nonlinear optimization, this algorithm is simply gradient descent with a fixed step size. Contrary to frequent claims, however, it is *not* steepest descent, since it lacks a line search along the negative gradient.

As Rumelhart, et. al. indicate, similar learning algorithms were proposed by Parker [17] and Le Cun [19]. It is extraordinary that such a simple and well-known technique should have been overlooked for so long.

The generalized delta rule, however, goes beyond simple optimization in that the algorithm permits changes to the network link weights based only on local information. The gradient computation is decomposed in such a way that individual units make weight changes based on the error values of their immediate neighbors. This suggests a direct mapping to a paral-

lel architecture in which individual units are separate processing elements. Furthermore, this suggests possibly a feasible biological model for learning, in that locality of information is used to improve performance in response to repeated examples. The issue of locality, computation and biological modeling is considered further below.

3.1 Learning Constant

In practice, the back propagation algorithm depends significantly on the value of the learning constant. Clearly, the optimum value for η depends on the problem being solved. For error surfaces with broad local minima, the gradient will be small even far from the solution, and thus a larger value of η will result in more rapid convergence. However, for problems with steep narrow minima, a small value of η must be chosen to avoid overshooting the solution. This can be illustrated using Rosenbrock's function:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

This function has a steep narrow curving valley [7], with a minimum at $(1, 1)$. Starting at $(-1.5, -1.0)$, for example, for values of $\eta \geq 0.00169$ the back propagation algorithm will not only fail to converge, the function will increase rapidly to infinity.

This makes clear the fact that η must be chosen experimentally for each problem or problem type. Moreover, it is desirable to monitor the progress of the optimization, so that η can be increased at appropriate points to speed up convergence.

3.2 Convergence Criteria

A great number of the problems which have been investigated using the back propagation algorithm are simulated Boolean functions [21, 19, 18, 5]. In these cases, the problems of interest are mappings between, or at least onto, Boolean vectors which are chosen to encode the desired output.

The Boolean functions are typically approximated using real numbers: 0.1 for 0, 0.9 for 1, for example. Now, it becomes clear that for N output units and M tokens, the optimization goal is to have the actual error for each unit/token less than, say 0.3. In order to achieve this error, the units and tokens must be treated independently, and thus the necessary squared-error convergence value should be $(0.3)^{(M+N)}$. Clearly, this number vanishes rapidly for large M and N. In practice, the output units are

treated separately, and optimization is terminated when all the units are behaving correctly. This could be formalized by a different error metric, which makes this explicit:

$$E = \max_{j,n}(targ_j(n) - output_j(n))$$

This defines a minimax procedure for Boolean function realization. It is offered as a conjecture that the success of the back propagation algorithm on these Boolean problems is due to the use of the minimax convergence criterion rather than the required least-squares criterion, since the minimax criterion identifies convergence long before least-squares would, and thus avoids the interminable iterations of gradient methods near the solution.

3.3 Momentum

The momentum term [19] is a modification to the back propagation algorithm designed to alleviate the problem of oscillation for a given choice of η [19]. The momentum term is defined by:

$$\Delta w_{ij}(n+1) = \mu \Delta w_{ij}(n) - \eta \frac{\partial E}{\partial w_{ij}(n)}$$

The momentum term is clearly a form of the convergence acceleration techniques for gradient methods discussed above.

$$w_{ij}(n+1) = (1+\mu) w_{ij}(n) - \mu w_{ij}(n-1) - \eta \frac{\partial E}{\partial w_{ij}(n)}$$

The effect of the momentum term is to magnify η by a factor $1/(1-\mu)$ for plateau regions of weight space where the gradient is essentially constant[4]. The effect of the momentum term for narrow steep regions of weight space is to focus the movement in a downhill direction by averaging out the components of the gradient which alternate in sign.

4 Computational Complexity

The computational complexity of nonlinear optimization algorithms is an important consideration in choosing a tool for research in connectionist network learning. From a practical point of view, the important parameter is

[4]This is derived precisely in Appendix A

the number of floating point operations required to reach a solution. This depends on the convergence rate of the optimization algorithm as well as the computational complexity per iteration of the algorithm. That is, the combination of algorithmic effectiveness and computational complexity must be considered in evaluating algorithms.

A measure of the combined time complexity and optimization efficiency is the approximate total number of operations to reach a solution. Let

$$\mathcal{F} = \sum_{i=1} C_i \times R_i$$

where $\mathcal{F}$ is the cost of solution for a task using a particular algorithm, which is composed of various components, such as gradient evaluations, function evaluations and search direction updates, each of which has complexity C_i. Each component is executed R_i times in solving the problem in question. The best algorithm is that for which $\mathcal{F}$ is minimal.

The complexity of the components of the algorithms reviewed in this report are developed in Appendix B. The results of applying these methods to several example problems are reported here to give some experience with the effectiveness of the different algorithms.

4.1 Example Problems

Several experiments were conducted to compare the convergence properties of the back propagation (BP), steepest descent (SD), DFP and BFGS algorithms. The convergence was measured as a function of floating point operations in order to provide a basis for choosing an algorithm for further use.

4.1.1 Rosenbrock's Function

The first problem chosen was Rosenbrock's function, which is defined above. This is a numerical problem which requires minimization over two parameters, which is used as a standard test problem for optimization algorithm, since the surface has a minimum which lies at the base of steep and curving walls.

The complexity values for the Rosenbrock function evaluation and gradient evaluation were computed directly from the function and gradient equations. These values were (4M+4A) and (5M+3A) respectively. The complexity values per iteration were computed using $N = 2$. Table 1 gives the complexity constants for the different algorithms.

Algorithm	C_{iter}	C_{func}	C_{grad}
BP	4	8	8
SD	8	12	12
DFP/BFGS	53	12	12

Table 1: Complexity Constants for Rosenbrock's Function

Algorithm	R_{iter}	R_{func}	R_{grad}	$\mathcal{F}$
BP	6543	6544	6543	130860
SD	1563	1742	1727	54132
DFP	24	68	64	2856
BFGS	19	42	39	1979

Table 2: Solution Statistics for Rosenbrock's Function

The four optimization algorithms were tested on this problem for a single initial condition[5]. A value of 0.001 was used as a termination criterion. The learning constant for the BP algorithm was 10^{-3}; no momentum was used.

The results of the experiment are given in Table 2, where the number of iterations, function evaluations and gradient evaluations listed for each optimization algorithm.

The convergence behavior of these algorithms is seen in Figure 1. The graphs show the convergence as a function of the total number of operations at each iteration.

The graph makes clear the fact that the first order algorithms make excruciatingly slow progress as they get closer to the minimum. This is precisely where the second order methods show dramatic convergence, since the Hessian provides excellent orientation to the search vector.

4.1.2 XOR Problem

The XOR problem was also tested. This is a very simple problem which requires the ability to compute mutual exclusion between two bits. It is a standard connectionist network demonstration problem because the input patterns are not linearly separable, and thus require a multilayer network.

A network consisting of two input units, one hidden unit, and one output unit, as well as threshold unit (K=2,O=1) was used. There were seven links (N=7). The training set consists of four examples (T=4). Using these pa-

[5]The initial starting point for this test was $(-1.2, 1.00)$.

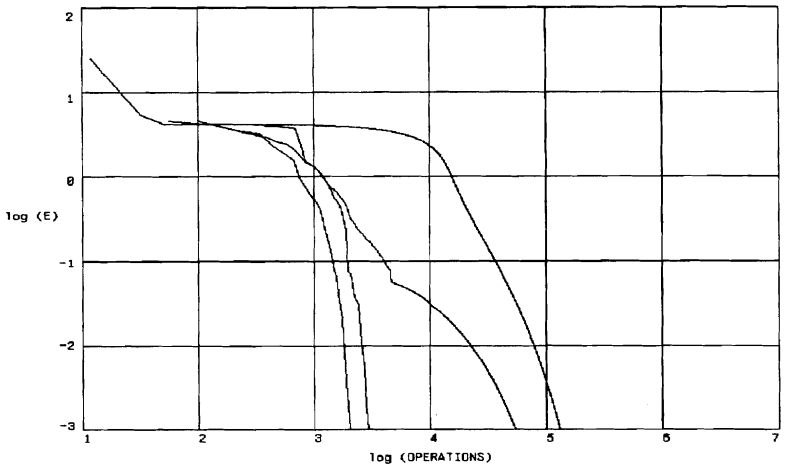

Figure 1: Progress of Optimization for Rosenbrock's Function

Algorithm	C_{iter}	C_{func}	C_{grad}
BP	14	92	132
SD	28	106	146
DFP/BFGS	413	106	146

Table 3: Complexity Constants for XOR Network

rameters, the algorithm component complexity constants can be computed as shown in Table 3.

This problem was repeated 5 times for each algorithm, using randomly initialized weight values. The same 5 initial networks were used for each algorithm. The target values for 0 and 1 were 0.1 and 0.9, respectively. A termination criterion of 0.001 was used.

The average number of iterations, function evaluations and gradient evaluations was computed for each algorithm. These results are summarized in Table 4. The BP algorithm was run only on one network with a learning constant of 0.15 and no momentum. The optimization was terminated after 10,000 iterations, at which point the algorithm had reached an objective function value of 0.04279. Thus, after 2,338,000 operations, no solutions was found. A second run was made with a momentum value of 0.9, which is listed in the table[6]. The DFP algorithm reached the termination value of

[6]Note that C_{iter} is now 28 instead of 14 because of the momentum term in the update

Algorithm	R_{iter}	R_{func}	R_{grad}	$\mathcal{F}$
BP	3102	3103	3102	781796
SD	154	460	460	120232
DFP	56	176	171	66750
BFGS	22	59	54	23224

Table 4: XOR Problem at 0.001 criterion

Algorithm	R_{iter}	R_{func}	R_{grad}	$\mathcal{F}$
BP	3063	3064	3063	771968
SD	139	406	406	106204
DFP	77	278	268	100397
BFGS	17	50	46	19037

Table 5: XOR Results at 0.003 criterion

0.001 in only one of the five cases; the counts for this single case are given in Table 4.

Because of the problems with the DFP algorithm reaching an objective function value of 0.001, a second experiment was conducted in which the termination condition was relaxed to 0.003. This represents a maximum squared error of 0.024, or a maximum error of 0.15, which still guarantees correct response. The average number of iterations, function evaluations and gradient evaluations are summarized in Table 5. The BP algorithm was run with a learning constant of 0.15 and momentum of 0.9. The DFP algorithm failed to reach the termination value of 0.003 in one of the five cases; this case is excluded from the average values.

The convergence behavior of the different methods for this problem is shown in Figures 2 through 5. It can be seen that the DFP algorithm is strongly dependent on initial condition, as is steepest descent. BP with momentum gives more consistent results, while BFGS gives most consistent and most rapid convergence.

4.1.3 Multiplexor Problem

Finally, a small multiplexor problem was also tested [1]. The problem consists of a two-bit select field, a four-bit data field, and a one-bit output field. The output bit is to correspond to the selected bit of the input data field.

---

equation.

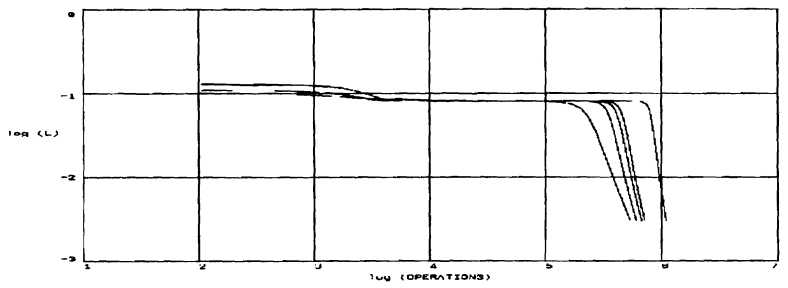

Figure 2: Solutions of XOR using BP with momentum

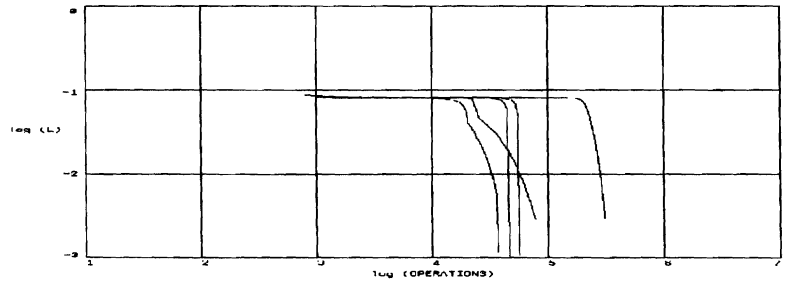

Figure 3: Solutions of XOR using Steepest Descent

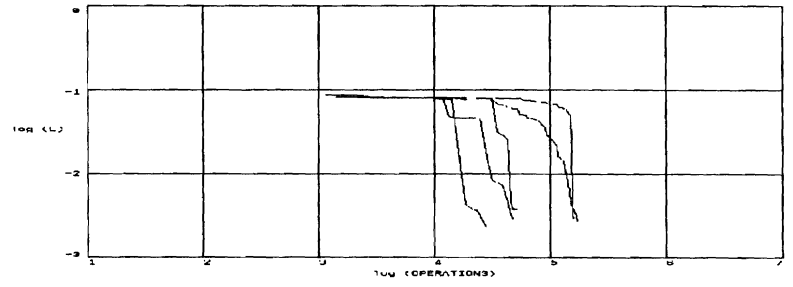

Figure 4: Solutions of XOR using DFP

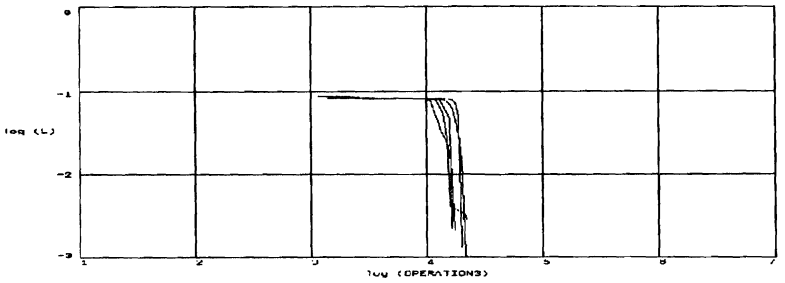

Figure 5: Solutions of XOR using BFGS

Algorithm	C_{iter}	C_{func}	C_{grad}
BP	156	6144	10688
SD	156	6222	10766
DFP/BFGS	10413	6222	10766

Table 6: Complexity Constants for Multiplexor Network

This problem was solved using a connectionist network with one output unit, four hidden units and six input units, with a total of 39 links (O=1,K=5, N=39). The network was trained on all possible input data patterns (T=64). The complexity constants for this network are shown in Table 6.

Five randomly initialized networks were optimized until a termination criterion of 0.003 mean squared error was met, or a maximum iteration count, or failure of the line search algorithm. The average number of iterations, function evaluations and gradient evaluations was then computed over the test runs which met the 0.003 squared error goal. These results are summarized in Table 7. The average number of operations to reach this level of convergence was computed from the constants in Table 6.

Algorithm	R_{iter}	R_{func}	R_{grad}	$\mathcal{F}$
BP	7645	7646	7645	129879404
SD	729	1777	1777	30301400
DFP				
BFGS	181	367	351	7947093

Table 7: Multiplexor Results at 0.003 criterion

The BP algorithm was run with a learning constant of 0.15 and a momentum constant of 0.9. The algorithm failed to reach the 0.003 criterion in only one case, when the maximum iteration count of 10,000 was reached, with a mean squared error value of 0.004. The table thus reflects the average of four runs. The steepest descent algorithm met the 0.003 criterion in all five cases.

The DFP algorithm failed in every case to reach convergence. The algorithm was apparently thrashing, as evidenced by the huge value of α, and the large ratio of function/gradient evaluations per line search (25). This suggests that the iterative approximation of the inverse second derivative matrix is inaccurate and misleading at points far from local solutions. It has been suggested that the approximation matrix be reinitialized every N iterations to address this problem, but this solution was not investigated in this experiment.

The BFGS algorithm failed to reach convergence in two cases, halting at mean squared error values of 0.008 and 0.014.

One might conclude from this experiment that the second-order methods offer computationally efficient convergence rates at the cost of an increased risk of getting stuck in local minima which are not acceptable as problem solutions.

4.1.4 Speech Recognition Problem

Finally, one acoustic phonetic discrimination task is selected as more representative of realistic problems. This task involves the discrimination of 5 consonant phonemes in the context of 3 vowels in simple consonant-vowel syllables.

The network consists of 3 layers, with 16 input units, 16 hidden units in the first layer, 12 hidden units in the second layer and 5 output units (K=33, O=5). The number of links is 1054. For this experiment, 10 repetitions of each of 15 utterances were used, which had an average sample length of 60. This results in a sample training set size of 9000. Under these conditions, the component complexities are as shown in Table 8.

Note that the number of training samples is large relative to the number of network connections. Thus, while the complexity of the second-order methods is generally measured by the N^2 operations per iteration, the function and gradient evaluations begin to dominate in all algorithms, as $T \gg N$. Consequently, the convergence power of the quasi-Newton methods ensures that the computational complexity of networks solutions is likely to remain

Algorithm	C_{iter}	C_{func}	C_{grad}
BP	2108	19998000	38583000
SD	4216	20000108	38585108
DFP/BFGS	7235183	20000108	38585108

Table 8: Complexity Constants for Acoustic Phonetic Network

several orders of magnitude less than those obtained by BP methods, even for large problems.

5 Locality and Parallelism

Locality, or the constraint that each unit be able to make appropriate weight changes on the basis of only local information, has been used to restrict the use of optimization algorithms on computational and philosophical grounds. The degree to which the use of local information should constrain the choice of optimization algorithms is examined in the following sections.

5.1 Locality and Biological Modeling

As was noted above, the BP algorithm provides a decomposition of the gradient computation such that weights changes can be made based on local information. This has been taken to suggest an architecture for parallel processing, as well as a plausible model for biological learning.

In light of the availability and convergence rates of higher-order nonlinear optimization algorithms, such as the quasi-Newton methods, it becomes imperative to evaluate the restriction of locality from both the perspective of biological modeling and parallel processing.

The question of whether a connectionist network algorithm ought to meet some criterion of biological plausibility appears to demand an affirmative answer. It turns out, however, that this requirement is difficult to maintain. It will be argued, therefore, that failure to meet supposed biological plausibility constraints should not rule out the use of powerful optimization techniques.

If faithfulness to a biological model is adopted as a design goal, then it must be decided from the start which biological process is intended to be modeled. There are processes of development, adaptation, learning and design. Each of these processes has its own mechanisms, initial conditions and time scales.

The process of design refers to the optimization of basic architectures which endow an organism with neurosensory processing capabilities and potentialities which, in turn, are the substrate of development, adaptation and learning. This process is understood to consist of either the quasi-random generation of genetic variations which interact with the environment under the law of survival, or the personal expression of the infinite intelligence of the Creator. Attempting to model the evolutionary process requires that the process be well-focussed, and accelerated by many orders of magnitude. On the creation model, those of us with finite minds and finite lifespans may wish to adopt a pragmatic approach to solving design issues, in order to live to see the results. Thus, it is quite possible to view connectionist networks solely as a computational paradigm, without regard for the biological process to which the networks are analogous. In this case, the choice of optimization algorithms is unrestricted.

In the second place, it is not clear that biological plausibility can be well defined, given the current state of ignorance about biological learning processes. It would therefore seem premature to restrict the use of optimization algorithms based on incomplete understanding of the constraints.

In the third place, the biological plausibility of the BP algorithm is rather meager. The requirement that error values back-propagate along symmetric links and serve to increase or decrease synaptic strengths seems improbable. Furthermore, it is becoming well known that learning occurs through rapid synaptic formation, and not (merely) by synaptic strength modification. In addition, it is quite clear that there are global effects in learning, which are probably chemically mediated. Thus, although BP may be plausible for some type of learning process, it fails to account for other types.

Consequently, the issue of biological plausibility, although interesting, ought to be reserved for the time when this requirement can be more accurately specified, and clearly reserved for problems in which biological modeling is part of the scientific objective.

5.2 Locality and Parallelism

In order to validate the use of locality as a principle for restricting on computational grounds the choice of optimization algorithm, two conditions ought to be met. It should be the case that methods which are local can be easily mapped to parallel architectures, whereas nonlocal methods cannot be easily mapped, and that execution times for local methods scale well with the number of processors, whereas the nonlocal methods do not.

It is clear that the BP algorithm can be mapped onto a parallel architecture with N processors, where N is the number of units. The quasi-Newton methods more naturally map onto architectures of W processors, where W is the number of links. Although time does not permit a detailed examination of the parallelization of each algorithm, including interconnection issues, it is clear that the mapping of an order W process onto N processors may not be straightforward, or result in $O(N)$ speedup. Since $N \ll W$, it is more likely to have N than W processors. However, for large problems, the number of units may exceed the number of processors. In this case, the (marginal) advantage of the BP algorithm as far as parallelism may disappear completely.

6 Conclusion

We conclude that there are iterative gradient techniques that are far superior to back propagation as tools for solving nonlinear optimization problems. We also conclude that the principle of locality should not be used to restrict the use of optimization algorithms, either for biological modeling or parallel processing. We finally conclude that the BFGS algorithm is one of the most powerful and appropriate methods for solving connectionist learning problems.

A Back-Propagation Momentum Term

The form of acceleration method for gradient-descent proposed for the back-propagation algorithm [19, 18] is defined by:

$$\Delta w_{n+1} = \mu \Delta w_n + \eta \sigma_n \tag{5}$$

where σ_n is the search vector in weight space (generally $-g_n$), η is the learning constant, and μ is the momentum constant.

This is a first-order difference equation in Δw, which in the general form

$$x_{n+1} = a_n x_n + b_n$$

has as its solution

$$x_{n+1} = \prod_{i=1}^{n} a_i x_1 + \sum_{j=1}^{n} \prod_{k=j+1}^{n} a_k b_j$$

The momentum equation, then, can be written as:

$$\Delta w_{n+1} = \mu^n \Delta w_1 + \eta \sum_{j=1}^{n} \mu^{n-j} \sigma_j$$

since $a_n = \mu$ and $b_n = \eta\sigma_n$.

Since $\Delta w_{n+1} = w_{n+1} - w_n$, this becomes:

$$w_{n+1} = w_n + \mu^n \Delta w_1 + \eta \sum_{j=1}^{n} \mu^{n-j} \sigma_j$$

Also, since $\Delta w_1 = w_1 - w_0 = \eta\sigma_0$, this can be reduced to

$$w_{n+1} = w_n + \eta \sum_{j=0}^{n} \mu^{n-j} \sigma_j$$

This, in turn, is a first-order difference equation in w, which can be solved in the same way, with $a_n = 1$ and $b_n = \eta \sum_{j=0}^{n} \mu^{n-j} \sigma_j$. Thus,

$$w_{n+1} = w_0 + \sum_{k=0}^{n} \eta \sum_{j=0}^{k} \mu^{k-j} \sigma_j$$

By reordering the summation terms, this becomes:

$$w_{n+1} = w_0 + \eta \sum_{k=0}^{n} \mu^k \sum_{j=0}^{n-k} \sigma_j \tag{6}$$

Now, assume that the gradient is essentially constant in a certain region of weight space. Thus,

$$\sigma_n \approx \sigma_{n-1} = \sigma$$

Under these conditions,

$$w_{n+1} = w_0 + \eta\sigma \sum_{k=0}^{n} (n - k + 1)\mu^k$$

since

$$\sum_{j=0}^{n-k} \sigma = \sigma \sum_{j=0}^{n-k} (1) = \sigma(n - k + 1)$$

The finite sum can be expressed as a difference of infinite sums:

$$\sum_{k=0}^{n}(n-k+1)\mu^k = \sum_{k=0}^{\infty}(n-k+1)\mu^k - \sum_{k=n+1}^{\infty}(n-k+1)\mu^k$$

Simplified by factoring μ^{n+1} and some algebra:

$$w_{n+1} = w_0 + \eta\sigma\left[(n+1)\sum_{k=0}^{\infty}\mu^k - (1-\mu^{n+1})\sum_{k=0}^{\infty}k\mu^k\right]$$

Using binomial series expansion, this can be rewritten as:

$$w_{n+1} = w_0 + \eta\sigma\left[(n+1)\frac{1}{1-\mu} - (1-\mu^{n+1})\frac{\mu}{(1-\mu)^2}\right]$$

Combining terms, this becomes:

$$w_{n+1} = w_0 + \eta\sigma\left(\frac{n+1}{1-\mu}\right)\left[1 - \frac{(1-\mu^{n+1})}{n+1}\frac{\mu}{(1-\mu)}\right]$$

This equation makes clear the effect of the acceleration term μ on the weight update formula. If the gradient is relatively small, as in a plateau, the weight increment is small, and the number of iterations to cross the plateau can be quite large. As n get large, $\frac{1-\mu^{n+1}}{n+1}$ becomes small, and the effective acceleration factor approaches $\frac{1}{1-\mu}$.

B Complexity

The computational complexity of several optimization algorithms are discussed in this appendix. First, some notation is defined, and then the time and space complexity of the various algorithm components are developed. These results are summarized in Table 9.

B.1 Notation

We define a connectionist network for the purposes of this derivation using the following notation:

U = $\{u_1, u_2, ..., u_K\}$, units of the network, *not counting input units.*

O = number of output units, $O < K$.

W = $\{w_{ijd}\}$, a set of connection strengths from unit u_i to u_j with delay d

N = number of links in the network, $N = |W|$.

T = total number of token samples in training set.

The connectionist network is represented by $\lambda = (U, W)$. It is noted that $N \gg K$, for most networks.

B.2 Function Evaluations

The evaluation of the objective function involves a forward pass through the network and an accumulation of the squared error at the output units for each item in the training set.

The forward pass requires, at each sample point, one multiplication and addition for each link, in computing the contribution to the potential function, an evaluation of the unit output function for each unit, and a multiply and two adds for each output unit to evaluate the squared error. The unit output function requires an exponential function call, an add and a divide. Assume that the exponential function evaluation is equivalent to a multiplication operation.

Then, the complexity of the function evaluation is:

$$C_f = T * (N(M + A) + K(2M + A) + O(M + 2A))$$

which, for $M = A$ is:

$$C_f = T * (2N + 3K + 3O)$$

operations.

The space requirements are determined primarily by the weight values and unit values $N + K$; if all token activations are stored, this value is $T * (N + K)$.

B.3 Gradient Computation

The gradient computation is accomplished by computing the error at the output units, back propagating the error across all the units and links, and accumulating the error for each weight. The complexity of the gradient is developed for the complete gradient method which is valid for networks with recurrent links.

It is assumed that the forward activation pass is computed prior to the gradient call, and that the unit activation history is available. Then, at each time step, the output unit error is computed for each output unit, the unit slope is computed for each unit, and the local unit error is multiplied by the link weight for each link, and accumulated in a global error buffer. The contribution to the gradient is computed for each link by summing over time the product of the appropriate local error and unit output values.

Thus, the gradient complexity is

$$C_g = T * (2N(M + A) + K(M + A) + O(A))$$

which for $M = A$ is:

$$C_g = T * (4N + 2K + O)$$

operations. If the forward pass must be recomputed, the complexity becomes:

$$C'_g = T * (6N + 3K + O)$$

B.4 Weight Update

The weight update for the BP algorithm involves multiplying the search vector by the learning constant and updating the weight vector. For the SD algorithm, the search vector is multiplied by the minimizing scalar constant. This requires N mults and adds. Momentum would add another N(M+A) operations.

B.5 Line Search

The line search involves a series of function evaluations and gradient evaluations, several dot product evaluations for checking descent conditions and completion criteria, and some arithmetic overhead for interpolation and extrapolation.

A dot product operation which computes the magnitude of the gradient is involved in every call to the line search. This adds N(M+A) operations. Each function evaluation step involves a weight update that requires N(M+A) operations, and each gradient evaluation is followed by another dot product to check for termination conditions. Thus, the line search algorithm adds N(M+A) operations per iteration, function evaluation and gradient evaluation.

B.6 Matrix Update

The BFGS algorithm specifies an update formula for the approximate inverse Hessian matrix. The complexity of this operation will be analyzed as representative of the quasi-Newton methods.

The update procedure requires a vector difference, a matrix product, two dot products, and a accumulation step. The algorithm also requires a matrix product to modify the gradient direction by the inverse matrix. The vector difference involves a scaling factor, so N(M+2A) operations are needed. The matrix product operation requires $N * N(M + A)$ operations, and each dot product N(M+A).

The accumulation step involves 4 adds and 5 multiplies for each element of H. Since H is symmetric, this can be reduced to $\frac{N(N+1)}{2}(5M + 4A)$.

The total then becomes:

$$C_{BFGS} = (3.5M + 3A)N^2 + (5.5M + 6A)N$$

Counting $A = M$, this becomes:

$$C_{BFGS} = N(11.5 + 6.5N)$$

The space requirements for the BFGS method are dominated by the inverse Hessian matrix, which is N^2. Since this matrix is symmetric, this could be reduced to $\frac{1}{2}N(N + 1)$. This would slightly complicate the addressing scheme for computation.

The memoryless quasi-Newton methods [13] are very similar; the use of the identity matrix instead of H for updates reduces the space requirements to $O(N)$ from $O(N^2)$. The computation requirements are reduced by one matrix product and one dot product, yielding $N(9.5 + 4.5N)$ operations.

B.7 Summary

This section draws together the complexities of the algorithm components to give costs per iteration, function evaluation and gradient evaluation which are parameterized by the size of the network and training set. These results are summarized in Table 9.

The BP algorithm consists of a gradient evaluation, weight update and function evaluation per iteration. The SD algorithm consists of a line search, which contains function and gradient evaluations, and a weight update per iteration. The BFGS and DFP algorithms consist of a line search and a matrix update and search vector alignment step. The table summarizes the

Algorithm	C_{iter}	C_{func}	C_{grad}
BP	2N	(2N+3K+3O)T	(4N+2K+O)T
SD	4N	(2N+3K+3O)T + 2N	(4N+2K+O)T + 2N
BFGS	N(13.5 + 6.5N)	(2N+3K+3O)T + 2N	(4N+2K+O)T + 2N
DFP	N(13.5 + 6.5N)	(2N+3K+3O)T + 2N	(4N+2K+O)T + 2N

Table 9: Computational Complexity of Algorithm Components

computational cost per iteration, function evaluation and gradient evaluation for each of the four optimization algorithms. Recall that the line search adds 2N operations to each iteration, function and gradient call. Also recall that the momentum term adds 2N operations per iteration for the BP algorithm.

References

[1] Charles William Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, September 1986.

[2] Yonathan Bard. On a numerical instability of Davidon-like methods. *Mathematical Computation*, 22:665–666, 1968.

[3] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *J. Inst. Maths Applics*, 6:76–90,222–231, 1970.

[4] William C. Davidon. *Variable Metric Methods for Minimization.* AEC Research and Development Report ANL-5990 Rev, Argonne National Laboratories, November 1959.

[5] Jeffrey L. Elman and David Zipser. *Learning the Hidden Structure of Speech.* Technical Report ICS Report 8701, UCSD Institute for Cognitive Science, February 1987.

[6] Roger Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, August 1970.

[7] Roger Fletcher. *Practical Methods of Optimization.* Volume 1 Unconstrained Optimization, John Wiley & Sons, 1980.

[8] Roger Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6:163–168, 1963.

[9] Stephen Jose Hanson and David J. Burr. Knowledge representation in connectionist networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, July 1987. submitted.

[10] Geoffrey Hinton, Terrence Sejnowski, and David Ackley. *Boltzmann Machines: Constraint Satisfaction Networks That Learn.* Technical Report CMU-CS-84-119, Carnegie-Mellon University, 1984.

[11] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the Natural Academy of Sciences USA*, 79:2554–2558, 1982.

[12] F. A. Lootsma, editor. *Numerical Methods for Non-linear Optimization.* Academic Press, 1972.

[13] David G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, second edition, 1984.

[14] Marvin Minsky and Seymour Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[15] W. Murray, editor. *Numerical Methods for Unconstrained Optimization.* Academic Press, New York, 1972.

[16] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[17] David B. Parker. *Learning-Logic.* Center for Computational Research in Economics and Management Science TR-47, Massachusetts Institute of Technology, 1985.

[18] David C. Plaut, Steven Nowlan, and Geoffrey Hinton. *Experiments on Learning by Back Propagation.* Technical Report CMU-CS-86-126, Carnegie-Mellon University, 1986.

[19] David E. Rumelhart, Goeffrey Hinton, and Ronald Williams. Learning internal representations by error propagation. In J.L.McClelland D.E.Rumelhart and the PDP research group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of -Cognition: Volume I Foundations*, chapter 8, MIT Press, Cambridge, MA, 1986.

[20] David E. Rumelhart, Goeffrey Hinton, and Ronald Williams. *Learning Internal Representations by Error Propagation.* Institute for Cognitive Science ICS Report 8506, University of California, San Diego, September 1985.

[21] Terrence J. Sejnowski and Charles R. Rosenberg. *NETtalk: A Parallel Network that Learns to Read Aloud.* Technical Report JHU/EECS-86/01, Johns Hopkins University, 1986.

[22] Ya. Z. Tsypkin. *Adaptation and Learning in Automatic Systems.* Volume 73 of *Mathematics in Science and Engineering*, Academic Press, 1971. Translated by Z. J. Nikolic.

[23] Raymond L. Watrous and Lokendra Shastri. Learning phonetic features using connectionist networks. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 851–854, August 1987.

[24] J. H. Wegstein. Accelerating convergence of iterative processes. *Communications of the ACM*, 1(6):9–13, 1958.