Neural Networks, Vol. 1, pp. 295–307, 1988
Printed in the USA. All rights reserved.

0893-6080/88 $3.00 + .00

ORIGINAL CONTRIBUTION

Increased Rates of Convergence Through Learning Rate Adaptation

ROBERT A. JACOBS

University of Massachusetts

(*Received November* 1987; *revised and accepted March* 1988)

Abstract—*While there exist many techniques for finding the parameters that minimize an error function, only those methods that solely perform local computations are used in connectionist networks. The most popular learning algorithm for connectionist networks is the back-propagation procedure, which can be used to update the weights by the method of steepest descent. In this paper, we examine steepest descent and analyze why it can be slow to converge. We then propose four heuristics for achieving faster rates of convergence while adhering to the locality constraint. These heuristics suggest that every weight of a network should be given its own learning rate and that these rates should be allowed to vary over time. Additionally, the heuristics suggest how the learning rates should be adjusted. Two implementations of these heuristics, namely momentum and an algorithm called the delta-bar-delta rule, are studied and simulation results are presented.*

1. INTRODUCTION

From the optimization literature, we know that there exist many iterative techniques for finding the parameters that minimize a function. For example, steepest descent, Newton's Method, recursive least squares, and conjugate-gradient methods are all procedures for solving such a problem. For purposes that will shortly be apparent, we can categorize these procedures according to how they adjust their estimates of the parameter values that minimize a function. Those methods that modify each estimate of a parameter value based solely on information about that parameter perform local computations and are classified as local techniques. All other methods are classified as non-local techniques.

Despite the fact that non-local techniques frequently possess good convergence properties, these procedures are not used to update the weights of connectionist networks. This is because the connectionist paradigm seeks to discover the information processing capabilities of networks that rely on local computations. The desire to use only local techniques is referred to as the locality constraint. This restriction is justified on several grounds. First, networks that perform local computations are frequently viewed as metaphores for biological neural networks. Second, such networks are thought to be easier to implement in parallel architectures.

In this paper, we investigate several connectionist learning algorithms. Our study is restricted to methods that adhere to the locality constraint. Recent years have seen the development of reasonably successful learning algorithms for multilayer networks. The most popular such algorithm, called the back-propagation procedure (Rumelhart, Hinton, and Williams, 1985), can be used to update weights by the method of steepest descent. Despite its effectiveness, many researchers still find this algorithm's rate of convergence too slow for the technique to be used in many practical situations. Therefore, investigations of new, and hopefully faster, algorithms continue to be pursued. The main theoretical contribution of this paper consists of four heuristics for achieving faster rates of convergence than steepest descent techniques. These heuristics suggest that every weight of a network should be given its own learning rate and that these learning rates should be allowed to vary over time. Additionally, the heuristics suggest how the learning rates should be adjusted.

In Section 2, we describe the method of steepest descent and detail why it can be slow to converge. Section 3 surveys previous research. Section 4 lists the four heuristics. In Section 5, we study two implementations

The author acknowledges and appreciates the extensive contributions of Rich Sutton and Andy Barto. This research was supported by the Air Force Office of Scientific Research, Bolling AFB, through grant AFOSR-87-0030 and by GTE Laboratories Incorporated.

Requests for reprints should be sent to Robert A. Jacobs, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

of these heuristics, namely momentum and the delta-delta learning rule. In Section 6, we introduce an alternative method of implementing these heuristics, called the delta-bar-delta learning rule. Section 7 presents simulation results comparing the steepest descent, momentum, and delta-bar-delta procedures.

2. STEEPEST DESCENT

As mentioned above, the method of steepest descent is an iterative procedure for obtaining the values of parameters that minimize a function. As applied in learning rules for connectionist networks, the function's parameters are the weights of a network and its value is an error measure. Geometrically, the function specifies an error surface defined over weight space. At each iteration of the steepest descent procedure, the values of the weights are modified in the direction in which the error function decreases most rapidly. This direction is given by the gradient of the error surface at the current point in weight space. The magnitude of the modification is a constant proportion of the magnitude of the gradient. Specifically, each weight is adjusted by a constant proportion of the partial derivative of the error with respect to the weight. This proportion is commonly referred to as the learning rate or step size. The procedure can be written

$$w(t+1) = w(t) - \epsilon \nabla J(t), \tag{1}$$

where $w(t)$ is the vector of weights at time t, ϵ is the learning rate parameter, J is the error function to be minimized, $J(t)$ is the error value at time t, $\nabla J(t)$ is the vector of first partial derivatives

$$\nabla J(t) = \left[\frac{\partial J(t)}{\partial w_1(t)}, \ldots, \frac{\partial J(t)}{\partial w_N(t)} \right]^T,$$

and N is the number of weights. Since the gradient vector points in the direction of maximum increasing error, in order to minimize the error it is necessary to multiply the gradient vector by negative one.

While steepest descent can be an efficient method for obtaining the weight values that minimize an error measure, error surfaces frequently possess properties that make this procedure slow to converge. There are at least two reasons for this slow rate of convergence. These reasons involve the magnitude of the components of the gradient vector and the direction of the gradient vector.

The magnitude of a partial derivative of the error with respect to a weight may be such that modifying a weight by a constant proportion of that derivative will yield a minor reduction in the error measure. This occurs in two situations. Where the error surface is fairly flat along a weight dimension, the derivative of the weight is small in magnitude. Thus, the value of the weight is adjusted by a small amount and many steps are required to achieve a significant reduction in error. Alternatively, where the error surface is highly curved along a weight dimension, the derivative of the weight is large in magnitude. Thus, the value of the weight is adjusted by a large amount and the value may overshoot the minimum of the error surface along that weight dimension.

A second reason for the slow rate of convergence of a steepest descent algorithm is that the direction of the negative gradient vector may not point directly towards the minimum of the error surface. This is illustrated in Figure 1 for a two-dimensional weight space. The error surface is drawn topographically using contours to represent regions of equal error. The minimum of the error surface is represented by the black dot in the center of the contours. The current weight vector is given by $w(t)$. Since at the current point in weight space the error surface is steeper along the w_2 dimension than the w_1 dimension, the derivative of w_2 is greater than the derivative of w_1. In general, for the direction of the negative gradient vector to equal the direction of the minimum of the error surface for all points in weight space, the contours that represent regions of equal error must be circular.

A refinement of these points can be seen by looking at the relevant results from adaptive filter theory. This material is included because it provides an analytic framework within which we can study the reasons for the slow rate of convergence of the steepest descent procedure. The following analysis can be found in many textbooks (e.g., see Haykin, 1986; Honig & Messerschmitt, 1984; Widrow & Stearns, 1985).

The system to be analyzed is a single processing element. Let $w(t)$ be the weight vector of the element at time t, $x(t)$ be the input vector at time t, and $d(t)$ be

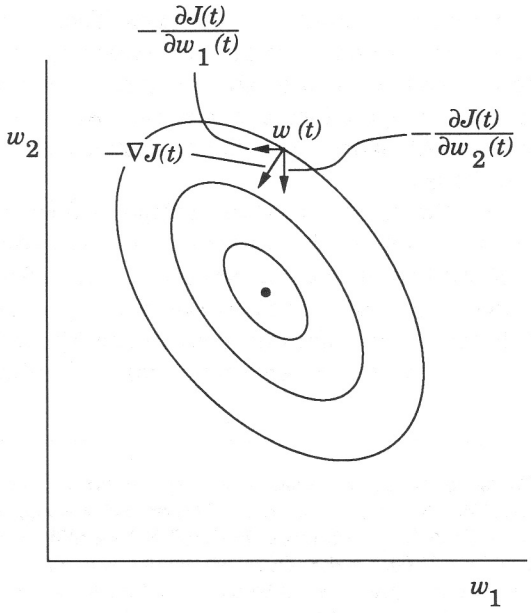

FIGURE 1. Error surface over two-dimensional weight space.

the desired response of the system at time t. Let p denote the expectation $E[x(t)d(t)]$, which is the cross-correlation vector between the input vector and the desired response. Let R denote the expectation $E[x(t)x^T(t)]$, which is the auto-correlation matrix of the input vector. As is commonly the case, the performance measure to be minimized is the mean-squared error. The value of this performance measure at time t can be written

$$J(t) = (d(t) - w^T(t)x(t))^2.$$

It is a fact (which will not be derived here) that this value can be re-written

$$J(t) = E[d(t)d(t)] - w^T(t)p - p^Tw(t) + w^T(t)Rw(t).$$

This performance measure specifies a quadratic error surface defined over weight space. The orientation and shape of the error surface is a function of R. In addition, the error surface possesses a unique minimum.

By examining how the eigenvalues of R determine the shape of the error surface, we illustrate two reasons why steepest descent search can be slow. The matrix R can be written

$$R = Q\Lambda Q^{-1},$$

where Λ is a diagonal matrix whose diagonal components are the eigenvalues of R, and Q is a matrix whose columns are the orthogonal set of eigenvectors associated with the eigenvalues. Denote the N eigenvalues of R as $\lambda_1, \ldots, \lambda_N$. The largest eigenvalue is labelled λ_{max} and the smallest eigenvalue is labeled λ_{min}. The principal axes of the error surface are in the direction of the eigenvectors of R. The mean-squared error increases most rapidly in the direction of the eigenvector corresponding to the eigenvalue λ_{max} and most slowly in the direction of the eigenvector corresponding to the eigenvalue λ_{min}. Define the eigenvalues spread to be $\frac{\lambda_{max}}{\lambda_{min}}$. This spread has a minimum value of one and can be arbitrarily large.

Recall that one reason why a steepest descent procedure may be slow to converge is that the negative gradient vector may not point towards the minimum of the error surface. When the eigenvalues spread equals one, contours of equal error are circular. In this case, the negative gradient is in the direction of the minimum error. When the eigenvalues spread is larger than one, contours of equal error are elliptical. In this case, the direction of the negative gradient is not necessarily equal to the direction of the minimum error.

A second reason why a steepest descent procedure may be slow to converge is that the learning rate may not be appropriate for all portions of the error surface. The component of the gradient vector is smaller in the direction of the eigenvector corresponding to λ_{min} (the major axis of the ellipse) than in the direction of the eigenvector corresponding to λ_{max} (the minor axis of the ellipse). Thus, a value for the learning rate that yields moderate steps in weight space along the major axis of the ellipse may result in large steps in weight space along the minor axis. In this event, the error measure may show only a slight reduction at each time step because in the direction of the minor axis, the weight vector may oscillate across the minimum of the error surface instead of approaching this minimum. Similarly, a value for the learning rate that yields moderate steps in weight space along the minor axis of the ellipse may result in small steps in weight space along the major axis of the ellipse. This will also result in only a slight reduction of the error at each time step. In practice, the value of the learning rate parameter is usually set so that successive steps in weight space do not overshoot the minimum of the error surface. Therefore, it is frequently the case that the value of the learning rate is limited by the value of λ_{max}, and only small steps in weight space are made in the direction of the major axis of the ellipse.

3. PREVIOUS RESEARCH

After considering the reasons for the possible slow rate of convergence of a steepest descent procedure, we can appreciate why researchers continue to study a variety of techniques for finding the parameters that minimize the value of an error function. In this section, we briefly review several of these techniques. For a more complete survey of learning algorithms for connectionist networks, see Anderson (1986).

This paper is largely a follow-up of techniques developed by Kesten (1958), Saridis (1970), Sutton (1986), and Barto and Sutton (1981). Kesten proposes that if consecutive changes of a weight (i.e., $\Delta w_i(t-1)$ and $\Delta w_i(t)$) possess opposite signs, then the weight value is oscillating. Hence, the learning rate for that weight should be decremented. Saridis uses this basic notion to both increase and decrease learning rates. Thus, if consecutive derivatives of a weight possess the same sign, then the learning rate for the weight should be incremented. If consecutive derivatives of a weight possess opposite signs, then the learning rate for the weight should be decremented. In this scheme, the sum of all learning rates for all weights is maintained at a constant. Thus, weights are effectively competing for this fixed quantity. Sutton (1986) discusses why these techniques are particularly pertinent to connectionist networks and makes several alternative suggestions for increasing rates of convergence. In Barto and Sutton (1981), Sutton introduces a learning rule which is very similar to the delta-delta rule discussed below. The logic behind the Kesten, Saridis, and Sutton proposals are explained at length later in this paper. First, we briefly survey alternative ideas.

Some researchers advocate utilizing traditional techniques from the function optimization literature

to train connectionist networks. In order to adhere to the locality constraint, Scalettar and Zee (in press) have employed a variation of Newton's Method. This algorithm, which is a modification of a recommendation by Sutton (1986), can be written

$$\Delta w_i(t) = -\epsilon \frac{\partial J(t)/\partial w_i(t)}{|\partial^2 J(t)/\partial w_i^2(t)|}.$$

Extensive results of the performance of this algorithm are not reported. Parker (1986) has developed an algorithm, called the second-order LMS rule, that uses both first and second derivatives in the update of weights. Parker maintains that this rule shows "nearly optimal performance."

Derthick (1984) has made several recommendations for learning procedures. For example, he suggests that each weight be updated by a fixed amount depending only on the sign of the weight's derivative. Alternatively, he proposes that at each weight update, the current point in weight space always be moved by a constant distance in the gradient direction.

Another approach towards developing learning algorithms with fast rates of convergence is to notice which inputs are most predictive of the correct output. The principal behind this approach is that faster rates of convergence can be achieved if the search for good weight values is confined to the most relevant weight dimensions. For example, a technique proposed by Hampson and Volper (1986) iteratively approximates several conditional probabilities involving the inputs and outputs and uses these approximations to update the weights. Similarly, Littlestone (1987) has a scheme that detects irrelevant inputs. Mjolsness (1987) suggests a method for networks with symmetric weights that shows qualities resembling the approaches that detect irrelevant features and the approaches that are variations of Newton's Method.

4. HEURISTICS FOR INCREASING THE RATE OF CONVERGENCE

Based on the above discussion and following the recommendations of Kesten, Saridis, and Sutton, four heuristics can be proposed that provide guidelines for how to achieve faster rates of convergence than steepest descent techniques.

First, every parameter of the performance measure to be minimized should have its own individual learning rate. As mentioned above, the step size appropriate for any one parameter dimension is not necessarily appropriate for other parameter dimensions.

Second, every learning rate should be allowed to vary over time. It is common for error surfaces to possess different properties along different regions of a single parameter dimension. In order to take appropriate step sizes as the parameter varies over its possible values, the learning rate will also need to vary.

Third, when the derivative of a parameter possesses the same sign for several consecutive time steps, the learning rate for that parameter should be increased. When the sign of the derivative behaves in this manner, it is frequently the case that the error surface at the current point in parameter space along that parameter dimension possesses a small curvature, and therefore, continues to slope in the same direction for some significant distance. By increasing the learning rate for this parameter, the number of time steps required for the value of this parameter to traverse this distance can be reduced.

Fourth, when the sign of the derivative of a parameter alternates for several consecutive time steps, the learning rate for that parameter should be decreased. When the sign of the derivative behaves in this manner, it is frequently the case that the error surface at the current point in parameter space along that parameter dimension possesses a high curvature, and therefore, the slope of this area of the error surface may quickly change sign. In order to prevent the value of the parameter from oscillating, this value should be adjusted by a smaller amount.

Note that by providing different learning rates for each parameter dimension, the current point in weight space is not modified in the direction of the negative gradient. Thus, such a system is not performing steepest descent search. Instead, parameters are updated based on both the partial derivatives of the error with respect to the parameters and on an estimate of the curvatures of the error surface at the current point in parameter space along each parameter dimension.

While the above heuristics provide guidelines for how faster rates of convergence can generally be achieved, certain caveats must be included in order to provide a balanced presentation. Most importantly, there exist error surfaces where use of these heuristics may not result in the desired effects. For example, consider an error surface defined over a two-dimensional weight space where the surface possesses a valley that is at a 45 degree angle to both weight axes. In the vicinity of this valley, the surface possesses a high curvature along both weight dimensions. Thus, when the current point in weight space is in the valley, these heuristics would cause the learning rate for each weight to be decreased. Ideally, in order to decrease the error measure the fastest, the learning rates for both weights should be increased.

The failure of the heuristics in this circumstance can be attributed to the fact that they satisfy the locality constraint. As a further example of how this constraint limits the utility of the heuristics, consider the quadratic error surface described in the discussion on adaptive filter theory. Ideally, in this situation, we would like heuristics that govern the step size in each eigendirec-

tion of the matrix R. Unfortunately, the heuristics can only be used to control the step size in each weight dimension.

Despite their limitations, we believe that the remainder of this paper demonstrates the merits of the heuristics.

5. TWO EXISTING IMPLEMENTATIONS OF THE HEURISTICS

There exist several possible implementations of the heuristics described above. Two that are reviewed here are momentum and the delta-delta learning rule.

5.1. Momentum

Momentum implements the heuristics through the addition of a new term to the weight update equation. At time step t, each weight $w(t)$ of a network is updated according to the following rule:

$$\Delta w(t) = -(1-\alpha)\epsilon \frac{\partial J(t)}{\partial w(t)} + \alpha \Delta w(t-1) \qquad (2)$$

$$= -(1-\alpha)\epsilon \sum_{i=0}^{t} \alpha^i \frac{\partial J(t-i)}{\partial w(t-i)},$$

where α is the momentum factor that determines the relative contribution of the current and past partial derivatives to the current weight change. This contribution is the exponentially weighted sum of the weight's current and past partial derivatives where α is the base and the time from the current time step is the exponent. Note that without the use of momentum (when α is set to zero), the update rule performs steepest descent.

Momentum is considered an implementation of the heuristics for the following reasons. When consecutive derivatives of a weight possess the same sign, the exponentially weighted sum grows large in magnitude and the weight is adjusted by a large amount. Similarly, when consecutive derivatives of a weight possess opposite signs, this sum becomes small in magnitude and the weight is adjusted by a small amount.

In order to analyze the advantages of momentum, we examined its effect on the optimization of a one dimensional quadratic error surface. The error surface is defined by the following function:

$$J(t) = \tfrac{1}{2} k w^2(t),$$

where k is the curvature of the surface. This specifies a quadratic error surface whose minimum is at $w = 0$. Four experiments were performed corresponding to four values for momentum. These values were 0.0, 0.25, 0.5, and 0.9. Within each experiment, several simulations were executed. In order to test the performance of a particular value of momentum over a wide range of curvatures, the curvature of the error surface was varied logarithmically between 0.00001 and 2.0 across simulations. For each experiment, a value for the learning rate was selected that resulted in the best performance over the entire range of curvatures. For the four experiments, the values of the learning rate were 1.0, 1.67, 3.0, and 19.0 respectively. At the start of each simulation, w was initialized to 1000. Solution of the problem is reached at the first of ten consecutive time steps during which the absolute value of w is less than 10.

The results are displayed in Figure 2. The x-axis of the graph logarithmically represents the curvature of the error surface. The y-axis logarithmically represents the number of steps until solution. Two important properties should be noted. First, the width of the range of curvatures over which performance is best for the given values of the learning rate and momentum dramatically increases with increases in momentum. Second, there exists a range of curvatures over which momentum decreases performance of the learning algorithm. This latter result suggests that there exists room for improvement in the development of techniques to achieve faster rates of convergence.

There are at least two limitations to momentum's effectiveness. First, there exists an upper bound on how large an adjustment momentum can make to a weight. For example, if all derivatives of a weight over time are assumed to be equal to one, then the exponentially weighted sum of the weight's current and past derivatives converges to $\dfrac{1}{1-\alpha}$ as time goes to infinity. In this event, the most a weight can be modified by is the value of the learning rate. A second limitation of momentum is that this exponentially weighted sum may have a sign opposite to the sign of the weight's current derivative. Thus, momentum can cause the weight to be adjusted up the slope of the error surface along the weight dimension, instead of down the slope as is desired.

5.2. Delta-Delta Learning Rule

The delta-delta learning rule consists of both a weight update rule and a learning rate update rule. The weight update rule is similar to the steepest descent algorithm with the exception that each weight possesses its own learning rate parameter. This weight update rule can be written

$$w(t+1) = w(t) - \epsilon(t+1) \frac{\partial J(t)}{\partial w(t)}, \qquad (3)$$

where $w(t)$ is the value of a single weight at time t and $\epsilon(t)$ is the learning rate value corresponding to $w(t)$ at time t.

We now derive a learning rate update rule that performs steepest descent on an error surface defined over learning rate parameter space. This rule and its derivation are a slight variation of the rule and derivation

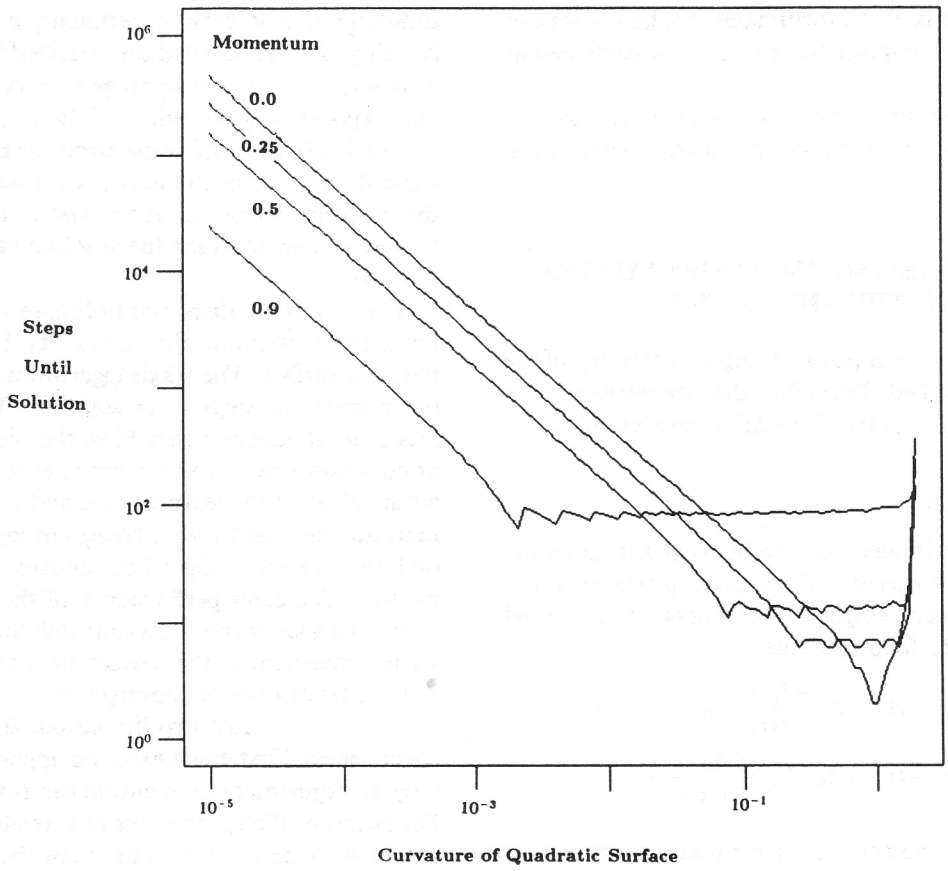

FIGURE 2. Effect of momentum on number of steps required to descend quadratic surfaces.

introduced by Sutton (Barto & Sutton, 1981). For simplicity, we consider the case of a single linear element whose weight update rule is based on the LMS rule (Widrow & Hoff, 1960). First, we define the input and output properties of the element. Let $w(t)$ be the element's weight vector at time t and $x(t)$ be the input vector at time t. The element's output at time t, $y(t)$, is defined as

$$y(t) = w^T(t)x(t).$$

Let $\epsilon(t)$ be a diagonal matrix of learning rate values at time t and $d(t)$ be the desired response at time t. In accordance with the LMS update rule, the weight update algorithm for this element is

$$w(t+1) = w(t) + \epsilon(t+1)[d(t) - y(t)]x(t).$$

Now we define the error function to be minimized by the learning rate update rule. Note that this error function is different from the error function minimized by the weight update rule. To distinguish these two functions, we denote the error function minimized by the weight update rule as J and the error function minimized by the learning rate update rule as G. As is commonly the case, let the value of G at time t be defined as follows:

$$G(t) = \tfrac{1}{2}(d(t) - y(t))^2.$$

After some algebraic manipulation,

$$\begin{aligned} G(t) &= \tfrac{1}{2}(d(t) - w^T(t)x(t))^2 \\ &= \tfrac{1}{2}(d(t) - (w(t-1) + \epsilon(t)[d(t-1) \\ &\quad - w^T(t-1)x(t-1)]x(t-1))^T x(t))^2. \end{aligned}$$

Finally, we differentiate this error function with respect to each learning rate. Let ϵ_i be the learning rate for the ith weight, w_i. This value is also the (i, i)th component of the diagonal matrix ϵ. Let I_i be a square matrix with all zero components except the (i, i)th component which is set equal to one. Then,

$$\begin{aligned} \frac{\partial G(t)}{\partial \epsilon_i(t)} &= (d(t) - y(t))((-d(t-1)I_i x(t-1))^T x(t) \\ &\quad + ((w^T(t-1)x(t-1))I_i x(t-1))^T x(t)) \\ &= -(d(t) - y(t))(((d(t-1) \\ &\quad - (w^T(t-1)x(t-1)))I_i x(t-1))^T x(t)) \\ &= -((d(t) - y(t))I_i x(t))^T((d(t-1) \\ &\quad - y(t-1))I_i x(t-1)) \\ &= -\frac{\partial J(t)}{\partial w_i(t)} \frac{\partial J(t-1)}{\partial w_i(t-1)}. \end{aligned}$$

Thus, a learning rate update rule that performs steepest descent on an error surface defined over parameter space is

$$\Delta\epsilon_i(t) = \gamma \frac{\partial J(t)}{\partial w_i(t)} \frac{\partial J(t-1)}{\partial w_i(t-1)},$$

where γ is the step size parameter. Most important for our purposes is that this algorithm for updating the learning rates implements the heuristics listed above. When the sign of the derivative of a weight is the same on consecutive time steps, the algorithm increases the learning rate for that weight. When the sign of the derivative of a weight alternates on consecutive time steps, the algorithm decreases the learning rate for that weight.

Unfortunately, the delta-delta rule is of only limited practical use. When the error surface at the current point in weight space along a weight dimension possesses a shallow slope and a small curvature, the derivatives of the weight at two consecutive time steps will likely have the same sign and have a small magnitude. Thus, their product is a very small positive number. In order to significantly increase the learning rate for this weight dimension, it is necessary to set γ sufficiently high. However, the sign of the derivative is likely to remain constant for many consecutive time steps and the learning rate may grow very large. In this event, the weight may be adjusted by an extremely large amount, often well beyond the minimum of the error surface along that weight dimension. Frequently, a large value for γ results in weights with huge magnitudes whose derivatives are equal to zero. In addition, problems may also arise when the error surface at the current point in weight space along a weight dimension possesses a high curvature. The derivatives of the weight at two consecutive time steps will likely have opposite signs and have a large magnitude. Thus, their product is a large negative number. If γ is set to a large value, it is frequently the case that the learning rate decreases until it is a negative number. In this event, the weight is adjusted up the slope of the error surface along the weight dimension instead of down it. To alleviate these problems, it is necessary to set γ to a small positive value. However, this results in only small changes to the learning rates and only a slight increase in the rate of convergence over normal steepest descent. For these reasons, the delta-delta rule was not used in the simulations reported in this paper. Instead, we introduce a related algorithm called the delta-bar-delta learning rule.

6. DELTA-BAR-DELTA LEARNING RULE

The delta-bar-delta learning rule, which was developed with the assistance of Rich Sutton, also consists of a weight update rule and a learning rate update rule. The weight update rule is the same as the delta-delta weight update rule and is given by Equation (3). We now introduce the delta-bar-delta learning rate update rule. Let $w(t)$ denote the value of a single weight at time t and $\epsilon(t)$ denote the learning rate value corresponding to $w(t)$ at time t. The learning rate update rule is defined as follows:

$$\Delta\epsilon(t) = \begin{cases} \kappa & \text{if } \bar{\delta}(t-1)\delta(t) > 0 \\ -\phi\epsilon(t) & \text{if } \bar{\delta}(t-1)\delta(t) < 0 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where

$$\delta(t) = \frac{\partial J(t)}{\partial w(t)}$$

and

$$\bar{\delta}(t) = (1-\theta)\delta(t) + \theta\bar{\delta}(t-1).$$

In these equations, $\delta(t)$ is the partial derivative of the error with respect to w at time t and $\bar{\delta}(t)$ is an exponential average of the current and past derivatives with θ as the base and time as the exponent. Note how the delta-bar-delta rule uses a technique similar to that of the delta-delta rule to implement the heuristics listed above. According to the delta-bar-delta algorithm, if the current derivative of a weight and the exponential average of the weight's previous derivatives possess the same sign, then the learning rate for that weight is incremented by a constant, κ. If the current derivative of a weight and the exponential average of the weight's previous derivatives possess opposite signs, then the learning rate for that weight is decremented by a proportion, ϕ, of its current value.

The delta-bar-delta rule increments learning rates linearly, but decrements them exponentially. Incrementing linearly prevents the learning rates from becoming too large too fast. Decrementing exponentially ensures that the rates are always positive and allows them to be decreased rapidly.

6.1. Examples of the Benefits of the Delta-Bar-Delta Rule

Recall that in Section 2, we listed several reasons why steepest descent procedures may be slow to converge. In this section, for each of these reasons, we illustrate why the delta-bar-delta rule may show a faster rate of convergence than the method of steepest descent. These illustrations consist of graphs of error surfaces. All error graphs plot data from a single training period of a connectionist network. First we discuss the nature of the training period. Next, we discuss the data presented in the graphs.

The mapping that the network is being trained to perform is a binary-to-local task. This problem and the network architecture are discussed below. For our current purposes, the task and architecture are not important because the problems posed by the shapes of

error surfaces discussed in this section are believed to be common to a wide variety of tasks and architectures. During the training period, each time step is called an epoch and is defined to be a single sweep through all training patterns. At the end of each epoch, the weights of the network are updated. At each update, we compute how to modify the current point in weight space by two methods. One method is the method of steepest descent and the second method is by the delta-bar-delta algorithm. Note that in the case of the delta-bar-delta algorithm, the learning rate for each weight is allowed to vary. However, at the start of the training period, these parameters are initialized to the value of the learning rate that is used by the steepest descent rule. The partial derivatives are computed using the back-propagation procedure. After calculating how to modify the weights by the two methods, the current point in weight space is actually updated by the delta-bar-delta algorithm.

At the end of each epoch, a graph is produced. Above each graph is listed several relevant pieces of information. The length of the steepest descent update vector is the length of the vector of weight modifications if the modifications are determined by the method of steepest descent. Similarly, the length of the delta-bar-delta update vector is the length of the vector of weight modifications if the modifications are determined by the delta-bar-delta method. The difference between the angle of the two update vectors is given in radians. On each graph, the error surface is shown in two directions in weight space. One direction is the gradient direction from the current point in weight space. The second direction is the direction that the current point in weight space would be moved if the weights are adjusted using the delta-bar-delta rule. The surface in each direction is generated by changing the weights in the network by a weight change factor times the normalized vector representing that direction and plotting the error produced by the network with the modified weight values. The x-axis is the weight change factor and represents the distance in weight space that the current weight vector is adjusted by. The y-axis is the error measure. This measure is the sum of squared errors which is defined as the squared discrepancy between a desired output value and a given output value summed over all output units of the network summed over all training patterns. The current point in weight space is identified by a weight change factor of zero. The tick mark on

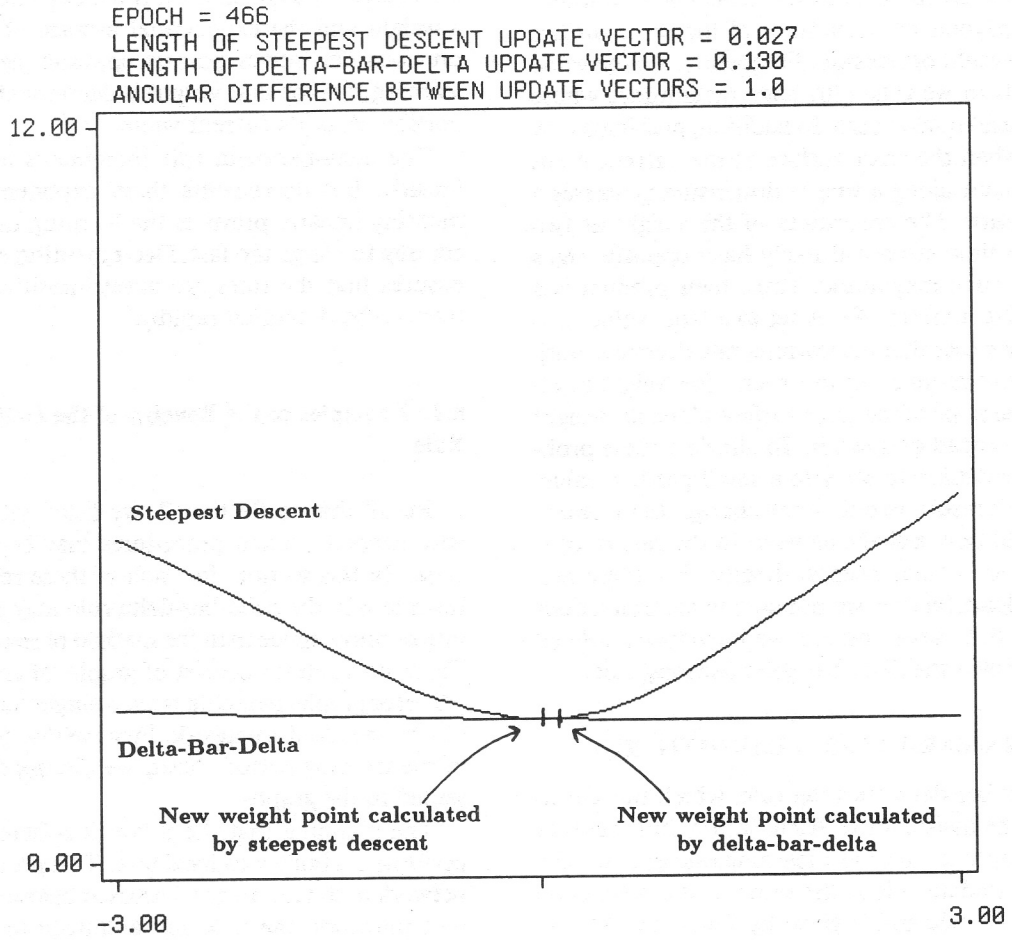

FIGURE 3. Performance of algorithms at region of error surface with small curvature.

the plot of the error surface in the gradient direction indicates the new position in weight space if the current point is modified by the steepest descent procedure. Similarly, the tick mark on the plot of the error surface in the direction of the delta-bar-delta update vector indicates the new position in weight space if the current point is modified by the delta-bar-delta procedure.

Steepest descent procedures tend to progress slowly where the error surface is fairly flat along a weight dimension. An analogous situation is illustrated in Figure 3. Note how the error surface at the current point in weight space possesses a fairly gentle slope. The length of the gradient vector is relatively small. One reason why the delta-bar-delta rule may increase the rate of convergence of the network is that the delta-bar-delta update vector is significantly longer than the steepest descent update vector. Hence, use of the delta-bar-delta rule causes the current point in weight space to traverse a greater portion of the error surface than the use of steepest descent.

In addition, Figure 3 illustrates a case where modifying the current point in weight space in the direction of the delta-bar-delta update vector is better than modifying this point in the gradient direction. The error surface curves less and is generally lower in the direction of the delta-bar-delta update vector than in the gradient direction. An intuitive interpretation is that the gradient vector is roughly perpendicular to a trough in the error surface while the delta-bar-delta update vector is closer to being parallel to this trough.

A second environment where steepest descent procedures may show slow progress is where the error surface is highly curved along a weight dimension. An analogous situation is illustrated in Figure 4. Note how the error surface at the current point in weight space possesses a high curvature. The length of the gradient vector is relatively large. The delta-bar-delta rule is beneficial in this circumstance because the length of the delta-bar-delta update vector is less than the length of the steepest descent update vector. Therefore, use of the delta-bar-delta method causes the current point in weight space to be modified by a smaller amount than use of steepest descent, thereby curtailing movement of this point across the curvature and promoting movement of the point down into the curvature. Once again, modifying the current point in weight space in the direction of the delta-bar-delta update vector is better than modifying this point in the gradient direction.

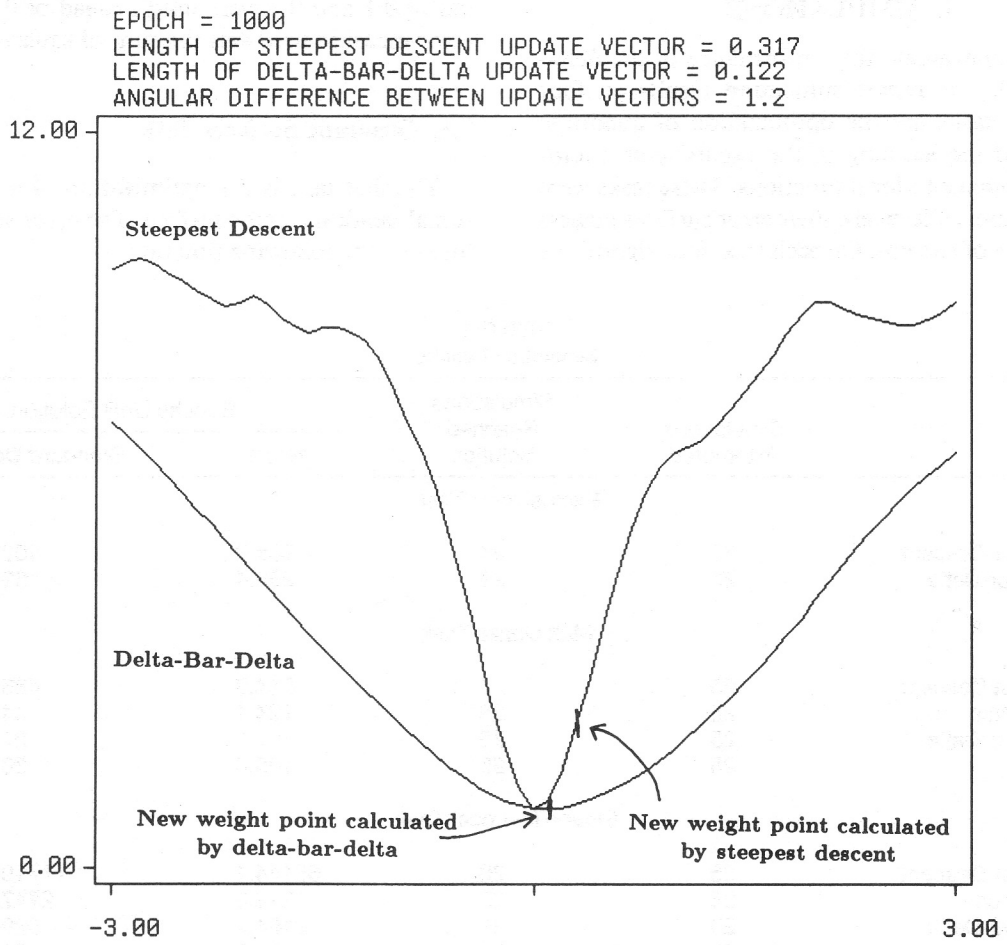

FIGURE 4. Performance of algorithms at region of error surface with large curvature.

TABLE 1
Parameter Values

	Initial ϵ	α	κ	ϕ	θ
Quadratic Surfaces Task					
Steepest Descent	1.0				
Momentum	19.0	0.9			
Delta-Bar-Delta	1.0		5.0	0.1	0.7
Hybrid	10.0	0.9	3.75	0.1	0.7
Exclusive-or Task					
Steepest Descent	4.0				
Delta-Bar-Delta	0.8		0.095	0.1	0.7
Multiplexer Task					
Steepest Descent	1.0				
Momentum	2.5	0.9			
Delta-Bar-Delta	0.8		0.085	0.15	0.7
Hybrid	2.25	0.9	0.045	0.01	0.7
Binary-To-Local Task					
Steepest Descent	0.1				
Momentum	0.75	0.9			
Delta-Bar-Delta	0.8		0.035	0.333	0.7
Hybrid	0.75	0.9	0.075	0.2	0.7

7. SIMULATIONS

In order to evaluate the performance of the delta-bar-delta rule, we report simulation results on four tasks. These tasks are the optimization of quadratic surfaces, and the learning of the exclusive-or, multiplexer, and binary-to-local functions. These tasks were chosen because, collectively, their error surfaces possess a wide variety of terrains. On each task, four algorithms were simulated. These algorithms are the steepest descent procedure (Equation 1), momentum (Equation 2), delta-bar-delta (weight update by Equation 3 and learning rate update by Equation 4), and a hybrid algorithm that combines the momentum and delta-bar-delta procedures. The hybrid algorithm uses the momentum weight update rule (Equation 2) with the exception that each weight possesses its own variable learning rate parameter. The learning rate parameters are modified by the delta-bar-delta learning rate update rule (Equation 4). For each algorithm, parameter values were selected that resulted in good performance. These values are shown in Table 1. The results of all four algorithms on the last three tasks are shown in Table 2.

Note that the last three tasks require the use of multilayer networks. For all algorithms, the back-propagation procedure was used to calculate the partial derivative of the error with respect to each weight. At the start of each simulation, the weights of the network were initialized to random values between -0.5 and 0.5. During training of the network, each time step is called an epoch and is defined to be a single sweep through all training patterns. At the end of each epoch, the weights of the network were updated. The inputs to all networks were binary values. However, target values of 0.1 and 0.9 were used instead of 0 and 1. The error measure used was the sum of squared errors.

7.1. Quadratic Surfaces Task

The first task is the optimization of a one dimensional quadratic error surface. The error surface is defined by the following function:

TABLE 2
Simulation Results

	Simulations Attempted	Simulations Reached Solution	Epochs Until Solution	
			Mean	Standard Deviation
Exclusive-or Task				
Steepest Descent	25	24	538.9	402.7
Delta-Bar-Delta	25	23	250.4	59.5
Multiplexer Task				
Steepest Descent	25	25	344.3	485.2
Momentum	25	25	124.1	44.4
Delta-Bar-Delta	25	25	137.1	27.2
Hybrid	25	25	105.6	20.3
Binary-To-Local Task				
Steepest Descent	25	20	58144.1	17719.9
Momentum	25	25	7212.3	2747.4
Delta-Bar-Delta	25	25	2154.2	929.9
Hybrid	25	25	871.5	236.3

$$J(t) = \tfrac{1}{2}kw^2(t),$$

where k is the curvature of the surface. For each weight update algorithm, several simulations were performed. In order to test each algorithm over a wide range of curvatures, the curvature of the error surface was varied logarithmically between 0.0001 and 2.0 across simulations. At the start of each simulation, w was initialized to 1000. Solution of the problem is reached at the first of ten consecutive time steps during which the absolute value of w is below 10.

The results of the simulations are displayed in Figure 5. The x-axis logarithmically represents the curvature of the error surface. The y-axis logarithmically represents the number of steps until solution. Overall, the delta-bar-delta algorithm performs the best.

7.2. Exclusive-Or Task

The second task is the exclusive-or problem. The network architecture consisted of two input units which were connected to two hidden units which were connected to a single output unit. The task is considered solved when the sum of squared errors for each epoch averaged over the previous fifty epochs is below 0.04. Note that when selecting parameter values, a non-zero value for α did not result in a faster rate of convergence than α set equal to zero in either the momentum or hybrid algorithms. When α is set equal to zero, momentum reduces to steepest descent and the hybrid algorithm reduces to delta-bar-delta. Thus, the momentum and hybrid algorithms' results on this task are not reported.

For each algorithm, twenty-five simulations were attempted. Of those attempted, some may fail to reach the solution criteria because they converge to local minima. The statistics on the number of epochs required to reach solution only includes those simulations that achieve the solution criteria. Again, the delta-bar-delta algorithm shows the best performance.

7.3. Multiplexer Task

For the multiplexer task, the network architecture consisted of six input units which were connected to six hidden units which were connected to a single output unit. The input to the network is four data lines and two address lines. The address lines indicate one of the data lines using a binary code. The desired output of the system is the input to the data line indicated by the address lines. Solution of the task occurs when the

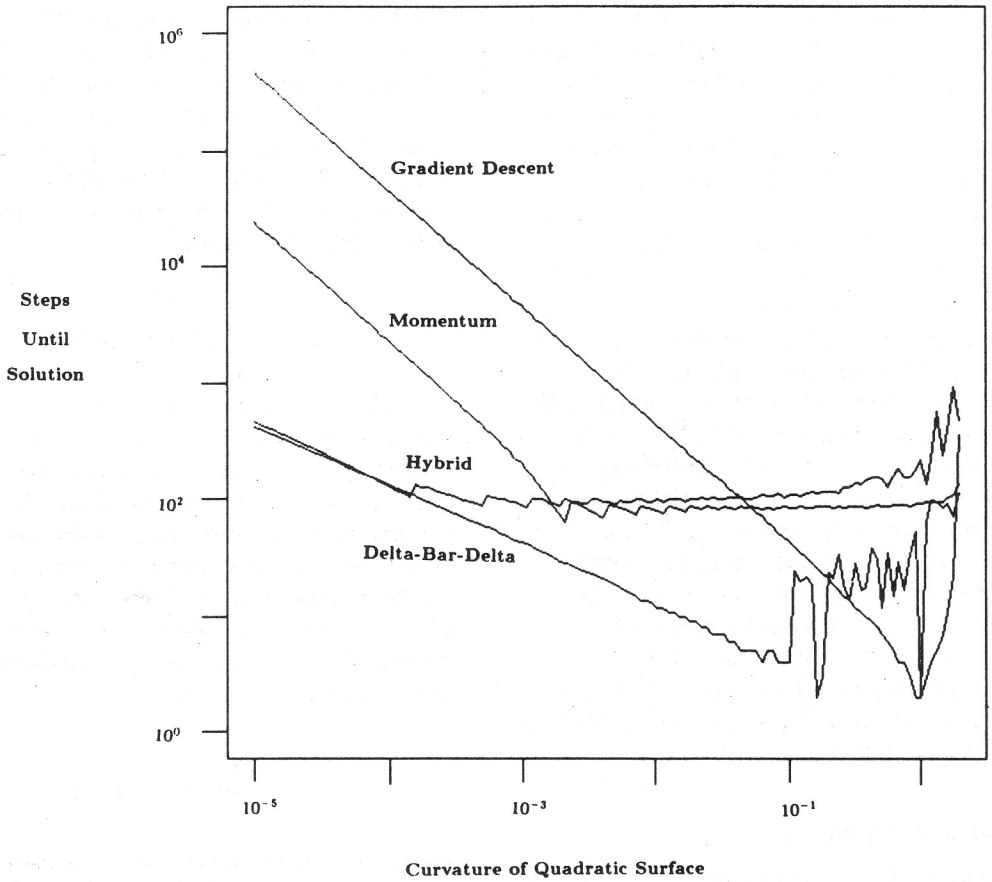

FIGURE 5. Performance of algorithms on quadratic surfaces.

TABLE 3
Results Using Same Parameter Values On All Tasks

	Simulations Attempted	Simulations Reached Solution	Epochs Until Solution	
			Mean	Standard Deviation
Exclusive-or Task				
Steepest Descent	25	24	16859.8	10866.3
Momentum	25	25	2056.3	1051.2
Delta-Bar-Delta	25	22	447.3	258.2
Hybrid	25	23	345.1	49.3
Multiplexer Task				
Steepest Descent	25	25	1244.0	399.2
Momentum	25	25	214.8	50.2
Delta-Bar-Delta	25	25	191.0	43.9
Hybrid	25	25	168.4	25.1

sum of squared errors for each epoch averaged over the previous fifty epochs is below 0.64. On this task, the hybrid algorithm shows the fastest rate of convergence. All algorithms perform better than steepest descent.

7.4. Binary-to-Local Task

The network architecture for the binary-to-local task consisted of three input units which were connected to a single hidden unit which was connected to eight hidden units which were connected to eight output units. The three components of the input vector encode in binary a value between zero and seven. The target output vector has all its components set to 0 except the component indicated by the input vector which is set to 1. For example, if the input vector is $[011]^T$ (binary for the value three), then the target output vector is $[00100000]^T$. Thus, the desired mapping translates a binary representation of a number into a local representation. Note that the single hidden unit that receives the activation of the input units must learn to output eight different activation values corresponding to the eight different input patterns in order for the net to perform the desired mapping. This network was chosen because it is believed that such an architecture produces an error surface with sharp ravines. Thus, it provides an interesting test-bed for the learning algorithms. Solution of the task occurs when the sum of squared errors for each epoch averaged over the previous fifty epochs is below 0.64. Again, the hybrid algorithm shows the fastest rate of convergence. Delta-bar-delta performs better then momentum and all algorithms perform better than steepest descent.

7.5. Robustness of Algorithms

Ideally, we would like a learning algorithm using the same parameter values to exhibit a fast rate of convergence in a variety of tasks and architectures. Such robust performance would alleviate the need to conduct a search for good parameter values each time we change tasks or architectures. Here, we report simulation results of the four algorithms on the exclusive-or and multiplexer tasks using the same parameter values that were found to result in good performance on the binary-to-local task.

The results of the simulations are listed in Table 3. The hybrid algorithm shows the fastest rates of convergence. The delta-bar-delta rule is faster than momentum. All three are faster than the steepest descent procedure. These results suggest that, with the same parameter settings, the learning rate update rule of the delta-bar-delta algorithm is able to adapt the learning rates of a network to appropriate values for a variety of tasks and architectures.

8. CONCLUSIONS

The above simulation results provide support for the four heuristics for how to achieve faster rates of convergence than steepest descent algorithms. We have examined two implementations of these heuristics, namely the momentum and delta-bar-delta procedures. In addition, we considered a hybrid algorithm that combines these two techniques. All of these methods generally show a faster rate of convergence than steepest descent. Clearly there are other implementations of the heuristics worth studying.

REFERENCES

Anderson, C. W. (1986). *Learning and problem solving with multilayer connectionist systems.* Ph.D. thesis, University of Massachusetts.

Barto, A. G., & Sutton, R. S. (1981). Goal seeking components for

adaptive intelligence: An initial assessment (Air Force Wright Aeronautical Laboratories/Avionics Laboratory Tech. Rep. AFWAL-TR-81-1070). Ohio: Wright-Patterson AFB.

Derthick, M. (1984). *Variations on the Boltzmann machine learning algorithm* (Tech. Rep. CMU-CS-84-120). Pittsburgh, PA: Carnegie-Mellon University.

Hampson, S. E., & Volper, D. J. (1986). Linear function neurons: Structure and training. *Biological Cybernetics,* **53,** 203–217.

Haykin, S. (1986). *Adaptive filter theory.* Englewood Cliffs, NJ: Prentice-Hall.

Honig, M. L., & Messerschmitt, D. G. (1984). *Adaptive filters: Structures, algorithms, and applications.* Boston: Kluwer Academic Publishers.

Kesten, H. (1958). Accelerated stochastic approximation. *Annals of Mathematical Statistics,* **29,** 41–59.

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Proceedings of the 28th IEEE Conference on Foundations of Computer Science.*

Mjolsness, E. (1987). *Control of attention in neural networks.* (Research Rep. YALEU/DCS/RR-545). Cambridge, MA: Yale University.

Parker, D. B. (1986). A comparison of algorithms for neuron-like cells. *Neural Networks for Computing, AIP Conference Proceedings,* **151,** 327–332.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Institute for Cognitive Science Report 8506). San Diego: University of California.

Saridis, G. N. (1970). Learning applied to successive approximation algorithms. *IEEE Transactions on Systems Science and Cybernetics,* **SSC-6,** 97–103.

Scalettar, R., & Zee, A. (in press). A feed-forward memory with decay. *Cognitive Science.*

Sutton, R. S. (1986). Two problems with backpropagation and other steepest-descent learning procedures for networks. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society,* 823–831.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. *1960 WESCON Convention Record Part IV,* 96–104.

Widrow, B., & Stearns, S. D. (1985). *Adaptive signal processing.* Englewood Cliffs, NJ: Prentice-Hall.