TWO PROBLEMS WITH BACKPROPAGATION AND OTHER STEEPEST-DESCENT LEARNING PROCEDURES FOR NETWORKS

Richard S. Sutton *
GTE Laboratories Incorporated
Waltham, MA 02254

ABSTRACT

This article contributes to the theory of network learning procedures by identifying and analyzing two problems with the backpropagation procedure of Rumelhart, Hinton, and Williams (1985) that may slow its learning. Both problems are due to backpropagation's being a gradient- or steepest-descent method in the weight space of the network. The first problem is that steepest descent is a particularly poor descent procedure for surfaces containing *ravines*—places which curve more sharply in some directions than others—and such ravines are common and pronounced in performance surfaces arising from networks. The second problem is that steepest descent results in a high level of interference between learning with different patterns, because those units that have so far been found most useful are also those most likely to be changed to handle new patterns. The same problems probably also arise with the Boltzmann machine learning procedure (Ackley, Hinton and Sejnowski, 1985) and with reinforcement learning procedures (Barto and Anderson, 1985), as these are also steepest-descent procedures. Finally, some directions in which to look for improvements to backpropagation based on alternative descent procedures are briefly considered.

Recent years have seen the development of the first effective learning procedures for connectionist networks that have interior or "hidden" units not directly associated with input or output: the Boltzmann machine learning procedure (Ackley, Hinton and Sejnowski, 1985), the backpropagation learning procedure (Rumelhart, Hinton and Williams, 1985), and the A_{R-P} reinforcement learning procedure (Barto and Anderson, 1985; Williams, 1986). The theory behind these new learning procedures is that of gradient or steepest descent in the space of "weights"—the modifiable memory parameters weighting the efficacy of each connection of the network. At each time step, a step is taken in weight space in the direction in which performance improves most rapidly. Letting $J(w)$ denote the performance measure to be minimized, where w denotes the vector of weights, the steepest-descent strategy can be written

$$\Delta w = -\rho \nabla J(w), \tag{1}$$

where Δw is the change in the weight vector, ρ is a positive learning-rate parameter, and

* I have received help and ideas contributing to this article from a large number of people. I wish to particularly acknowledge Steve Epstein, Andy Barto, John Aspinall, Martha Steenstrup, Ron Williams, Glenn Iba, and Oliver Selfridge.

the gradient $\nabla J(w)$ is the vector of first partial derivatives

$$\nabla J(w) = \left(\frac{\partial J(w)}{\partial w_1}, \cdots, \frac{\partial J(w)}{\partial w_N} \right),$$

where N is the number of weights. Most single-unit and single-layer learning procedures are also steepest-descent procedures, including the perceptron (Rosenblatt, 1962) and the Widrow-Hoff rule (Widrow and Hoff, 1960).

Steepest-descent procedures make their largest changes to those weights where the first partial derivatives are greatest.* This may seem a good strategy, but there are at least two reasons for doing exactly the opposite. First, a small derivative may indicate a shallow, wide, gently-curving part of the surface, where large steps need to be made, whereas a large derivative may indicate a very steep and sharply curving part, where the step size must be reduced to prevent instability. Second, the derivative can only be large for weights and units to the extent that they affect performance. It follows that those with large derivatives will be those that already play a role in the behavior of the network, for example, those that have already formed features of use to the rest of the network. When the network has need to adapt and create new features for a new situation, it should do so while minimizing interference with the existing useful features, meaning that these large-derivative weights should be changed least.

The rest of this paper elaborates on these two problems as they arise in Rumelhart et al.'s backpropagation learning procedure. We concentrate on backpropagation because most results obtained so far suggest that it is significantly more efficient than the other network learning procedures (Anderson, in prep.; Hinton, personal communication), and its capabilities have been impressively demonstrated (e.g., Sejnowski and Rosenburg, 1986). The backpropagation procedure also illustrates the problems we wish to point out particularly clearly. As steepest-descent methods, the other network learning procedures are also subject to the same problems to various degrees.

Rumelhart et al.'s backpropagation procedure is the application of steepest descent to acyclic networks receiving signed multi-dimensional errors as their teaching signals (see Figure 1). In acyclic or feedforward networks, information flows in one direction only: if there is a connection from unit A to unit B, then there can be no connection or series of connections from B back to A. At each time step, a set of input units take on the values of an input pattern, activity is propagated through interior or "hidden" units to a set of output units, and then the net is told what each output unit's activity should have been. The acyclic interconnection means that the propagation of activity can occur in one sweep, updating the activity of each unit only after the activity of all its inputs have already been updated.

---

* Here, and throughout, by greatest we mean greatest in the unsigned sense, that is, greatest in absolute value.

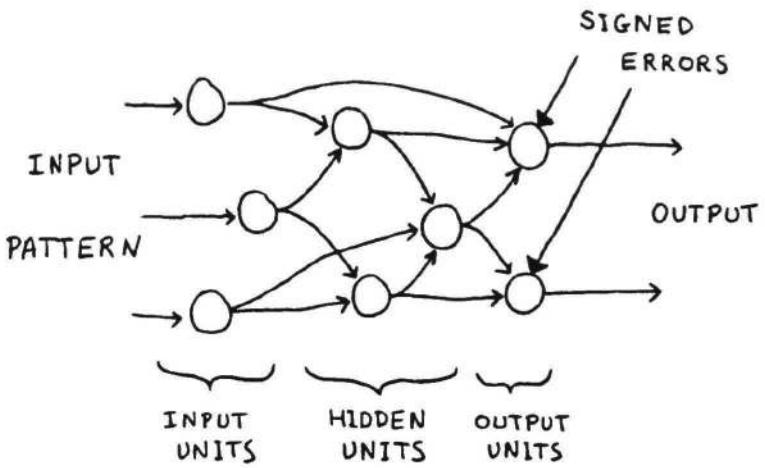

Figure 1. An acyclic network with hidden units and signed-error teaching signals.

In backpropagation, a squared-error performance measure, summed over input patterns, is typically used for $J(w)$. In order for its gradient with respect to the weights to exist, each unit's output activity must be a continuously-differentiable function of its input activity. Typically, each input to a unit is modulated by a separate weight, and the weighted sum of all input is passed through an S-shaped function from $\Re$ to $[0, 1]$. Finally, the name "backpropagation" refers to the way information is propagated in a single sweep in the backward direction, the reverse that of the propagation of activity, to exactly compute the gradient of performance on a step with respect to each weight in the network. This is then averaged or summed over steps to approximate $\nabla J(w)$. We will not need to consider the details of the gradient computation.

STEEPEST DESCENT AND RAVINES

Consider the surface whose contour map is shown in Figure 2a. In region A the surface slopes gently, whereas in region B it is steep. To find one's way from A through B to the optimum in the minimum number of steps, one would clearly want to make larger steps in A than in B; a flat, shallow surface suggests the optimum is far away, and thus that large steps be taken. In this way it is inherently a part of the idea of descent that larger steps should be taken where the gradient is *smallest*. Equation 1, however, results in the opposite, in a step size proportional to the size of the gradient. To acheive the desired step size, the learning-rate parameter ρ must be made much smaller in large-gradient regions such as B than it is in small-gradient regions such as A.

The problem is more serious when the gentle and steep slopes occur simultaneously along different dimensions, as in the surface shown in Figure 2b. Such places, in which the

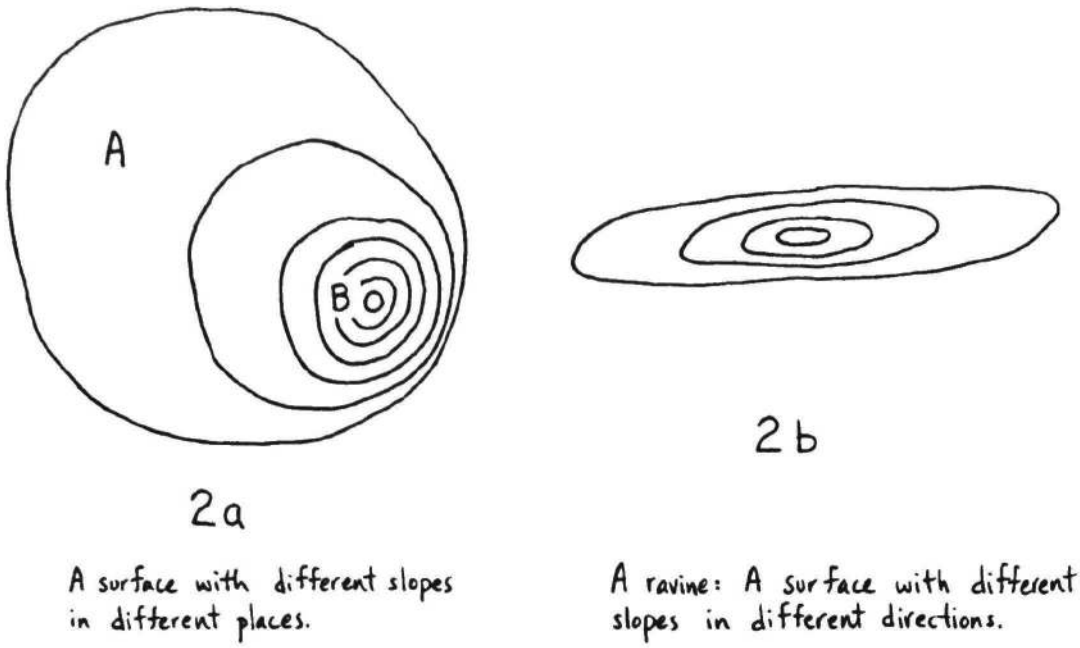

Figure 2.

surface curves much more steeply in one direction than in another, are called *ravines*. As before, we would like to take large steps where the surface is gently sloping—here, along the ravine—and small steps where the surface is steep—across the ravine. Also as before, the magnitude of the gradient is just the opposite of what is desired; it is large across and small along the ravine. Here, however, we cannot solve the problem by varying the learning rate ρ over time. In effect, we need the learning rate to be different in different directions. Since this would alter the direction of the step, it is precisely what is ruled out by steepest-descent procedures, which by definition step directly in the direction of the gradient (i.e., perpendicular to the contour lines; see Figure 2b). If the learning rate is the same in all directions, then it will have to be small enough to prevent instability in any direction, and this means that it will have to be much smaller than optimal in almost all directions, and learning will be very slow and inefficient.

Figure 3 illustrates how ravines arise naturally from the structure of networks. The output unit O receives input from three hidden units A, B, and C, across connections with weights of $w_A = 0.1$, $w_B = 1.0$, and $w_C = 10.0$. The output unit O can effect performance directly, the other three units only through affecting O. Other things being equal, then, changes in C's (input) weights will have ten times the effect on performance as changes in B's weights, and 100 times the effect of changing A's weights. The corresponding ravines will curve approximately 10 and 100 times faster along the dimensions of C's weights than they will along the dimensions of B's and A's weights. Thus, simple variations in the magnitude of weights produce many deep ravines, even more so in deeper and more layered networks. As the signals produced by some units are found to be useful

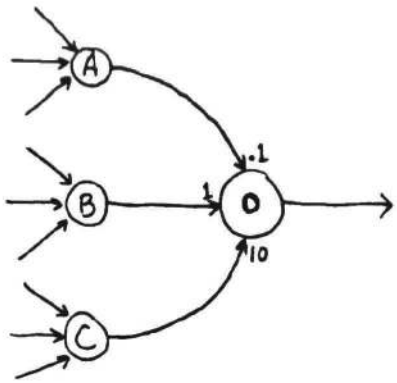

Figure 3. A network fragment. Numbers indicate weight values.

while those produced by others are found not to be, such weight variations undoubtedly will occur.

A possible saving grace is that ravines created in this way always have their principal axes parallel to the principal dimensions of the space (those corresponding to the weights). In two dimensions, for example, such ravines are either horizontal or vertical. Diagonally-oriented ravines occur only in cases of strong interaction between the changes made at different weights. For example, a 45° ravine would mean performance remained good as long as the two weights went up and down together, but worsened quickly if they stepped in different directions. Such interactions certainly can occur in the problems we would like networks to solve, but are not naturally created by network structures the way the weight-axis-aligned ravines are. This observation is important in looking for solutions to the ravine problem; as long as the ravines are aligned with the weight axes, it may be possible to alter or eliminate their effect simply by having different learning rates for each weight, so that larger steps are made along some dimensions, perhaps those of small derivative, than along others, perhaps those of large derivative. Some such possibilities are briefly discussed in a later section.

The ravine problem is appreciated by those who have been designing network learning procedures (e.g., see Derthick, 1984). For example, it is thought that the reason the "momentum" technique* (Rumelhart et al., 1985) improves performance is that it ameliorates the ravine problem. What has not been appreciated, however, is that *steepest* descent

---

* In this modification to the backpropagation algorithm, the weights are changed partly according to the current gradient and partly according to recent past gradients, giving weight motions "momentum". Strictly speaking, this is a departure from steepest descent, but the problems identified here should still be present. The momentum technique increases the rate of learning, but it is still thought to be much too slow (Hinton, personal communication).

is only one of many descent procedures, and one which is known to be inefficient in the presence of ravines (e.g., Tsypkin, 1971; Duda and Hart, 1973; Gill et al., 1981). The prominence of deep ravines in surfaces generated by networks suggests looking beyond steepest-descent procedures.

CROSS-PATTERN INTERFERENCE

Our second problem with steepest-descent network learning procedures has to do with how they handle interference among the various patterns presented to the network. If the network develops a nice set of features for classifying one set of patterns correctly, and then we ask it in addition to classify new patterns, we would like it to do so with minimal interference with the features crafted to classify the first set. If new features are developed to classify the new patterns, they should preferentially be formed from as-yet-unused units rather than by making those already in use serve double-duty.

Unfortunately, the steepest-descent procedure again produces the opposite of the desired behavior. In figure 3, output unit O has learned to listen most strongly to unit C; apparently C has formed a feature of use in solving the problem. Now suppose new patterns are presented, and new features are needed. As discussed previously, C's weights will have much larger derivatives than B's and A's, and so under steepest descent they will change much more dramatically. Alternatively, once C was found to no longer be useful, its incoming weights could have been left unchanged, while its outgoing weight onto O was reduced. Then, if ever the feature provided by C was again needed, its effect could quickly be resurrected rather than its function painstakingly recreated. But steepest descent does not do this. Steepest-descent procedures preferentially change existing, already-useful features rather than make new ones from unused units.

The desired logic here is that of generate and test: Responding to current gradients is the generation process; it is supposed to create any needed new features. The test of the feature provided by a unit is whether it plays a useful role in the network, which will be correlated with its weights having large derivatives. In generate and test we make changes, generate a variety of alternatives, until we find something that passes the test, which we then keep and insulate in some way from further changes. In steepest descent, on the other hand, we make greater and greater changes to a unit's weights the more it is found to be useful and given control over network output. Units with no effect and zero derivatives could experiment arbitrarily without degrading performance. Under steepest descent, however, they will not participate at all in the attempt to find good new features for new situations.

ALTERNATIVE DESCENT PROCEDURES

This article does not propose any specific alternatives or improvements to steepest descent and backpropagation. Here, however, we mention several possible directions in

which to look, and report our experiences with them so far.

The alternative to steepest descent is to still descend, but not directly in the direction of the gradient. A convenient way to think of this is as a distortion of the surface: By stretching the surface perpendicular to ravines, the eliptical contours of a ravine can be converted into circular ones, upon which steepest descent is very effective. Such a distortion of the space is equivalent to distorting individual steps analogously, lengthening them along ravines, shortening them across.

In general, such a distortion involves multiplying the gradient times an $N \times N$ matrix, where N is the number of weights. We will consider this impractical in that it calls for every weight to communicate with every other weight. Such communication is unnecessary if we assume, as discussed earlier, that all ravines are oriented parallel to the weight axes. In this case inter-weight communication is unnecessary; instead of a full matrix multiplication, each weight need only have an individual step-size or learning-rate scale factor. Below we briefly consider three different strategies for setting individual scale factors for each weight.

Squared-error performance measures often result in quadratic or approximately-quadratic surfaces. For such surfaces the direction and distance of the optimum can be accurately estimated from the local first and second partial derivatives. The classic descent procedure that does this is *Newton's method.** It is a matrix method, using the inverse of the *Hessian* matrix D of second partial derivatives:

$$\Delta w = -\rho D^{-1} \nabla J.$$

This method normalizes the first derivative according to how fast it itself is changing; if that rate of change is constant, and $\rho = 1$, then the method brings the weight vector exactly to where $\nabla J = 0$, i.e., to the optimum, in one step. One way to approximate this using only a single scale factor per weight is simply to assume all non-diagonal terms of D are 0, yielding

$$\Delta w_i = -\rho \frac{\partial J}{\partial w_i} \Big/ \frac{\partial^2 J}{\partial w_i^2}.$$

We note in passing that the second partial derivative $\frac{\partial^2 J}{\partial w_i^2}$ can be computed by a backpropagation process entirely analogous to that proposed by Rumelhart et al. for computing for the first derivative.

Newton's method is an entirely analytic method—based on an exactly computed second derivative matrix. Another possibility is to measure empirically the extent to which each weight undergoes changes, and adjust each weight's scale factor so that all weights change by the same amount. This would prevent units with small outgoing connections from

* Newton's method was originally devised to find the zeros of a function. As a descent procedure it is used to find the zeros of the derivative of a function, and thereby the function's extrema.

making only tiny changes and thus being wasted, and would prevent highly useful units from undergoing excessively large changes because of their large derivatives.

Finally, other empirical methods can be taken from the literature on acceleration of convergence of stochastic approximation methods (e.g., Kesten, 1958; see Fu, 1968; Tsypkin, 1971, p. 59). For example, Kesten's method is based on changes in the sign of the individual steps, in this case of the Δw_i; if the sign of the step keeps changing, oscillation is suggested, and the method reduces the step size. Similarly, repeated steps of the same sign suggest that the step size should be increased, but little work has been done pursuing this half of the idea. One particularly interesting possibility is that of using the steepest-descent concept at a second "meta" level to derive procedures for altering each weight's learning rate (as in Barto and Sutton, 1981, Appendix C).

CONCLUSION

Modern network learning procedures such as backpropagation are a significant adavance over previous connectionist learning techniques. This work is particularly exciting because the learning procedures can be directly related to their basis in the theories of stochastic approximation and gradient descent. The intent in this article has been to encourage the widening of the scope of this advance. Gradient or steepest descent is one of the simplest descent procedures, but it is neither the only nor the best one. It is unduly slow in the presence of ravines, which appear ubiquitous in the network domain, and it encourages the destruction of previously useful features upon task switches. Much is already known about the problems of steepest descent and various alternatives to it in a general setting. Just as steepest-descent theory has been successfully carried over to the network domain to produce backpropagation and the other new network learning algorithms, perhaps this other knowledge can be carried over to significantly improve the speed with which they learn.

REFERENCES

Ackley, D.H., Hinton, G.H., & Sejnowski, T.J. (1985) A learning algorithm for Boltzmann machines. *Cognitive Science* 9, 147–169.

Anderson, C.W. (in preparation) Learning new terms in connectionist systems. University of Massachusetts Ph.D. Dissertation.

Barto, A.G. & Anderson, C.W. (1985) Structural learning in connectionist systems. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, 43–53.

Barto, A.G. & Sutton, R.S. (1981) Goal seeking components for adaptive intelligence: An initial assessment. Air Force Wright Aeronautical Laboratories/Avionics Laboratory Technical Report AFWAL-TR-81-1070, Wright-Patterson AFB, Ohio.

Derthick, M. (1984) Variations on the Boltzmann machine learning algorithm. CMU Tech Report CMU-CS-84-120.

Duda, R.O. & Hart, P.E. (1973) *Pattern Classification and Scene Analysis.* New York: Wiley.

Fu, K.S. (1968) *Sequential Methods in Pattern Recognition and Machine Learning.* New York: Academic Press.

Gill P.E., Murray W., & Wright, M.H. (1981) *Practical Optimization.* New York: Academic Press.

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

Rosenblatt, F. (1962) *Principles of Neurodynamics.* New York: Spartan Books.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1985) Learning internal representations by error propagation. Institute for Cognitive Science Technical Report 8506, UCSD, La Jolla, CA 92093.

Sejnowski, T.E. & Rosenburg, C.R. (1986) NETtalk: A parallel network that learns to read aloud. Johns Hopkins University Electrical Engineering and Computer Science Technical Report JHU/EECS-86/01.

Tsypkin, Y.Z. (1971) *Adaptation and Learning in Automatic Systems.* New York: Academic Press.

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

Williams, R.J. (1986) Reinforcement learning in connectionist networks: A mathematical analysis, Institute for Cognitive Science Technical Report 8605, UCSD, La Jolla, CA 92093.