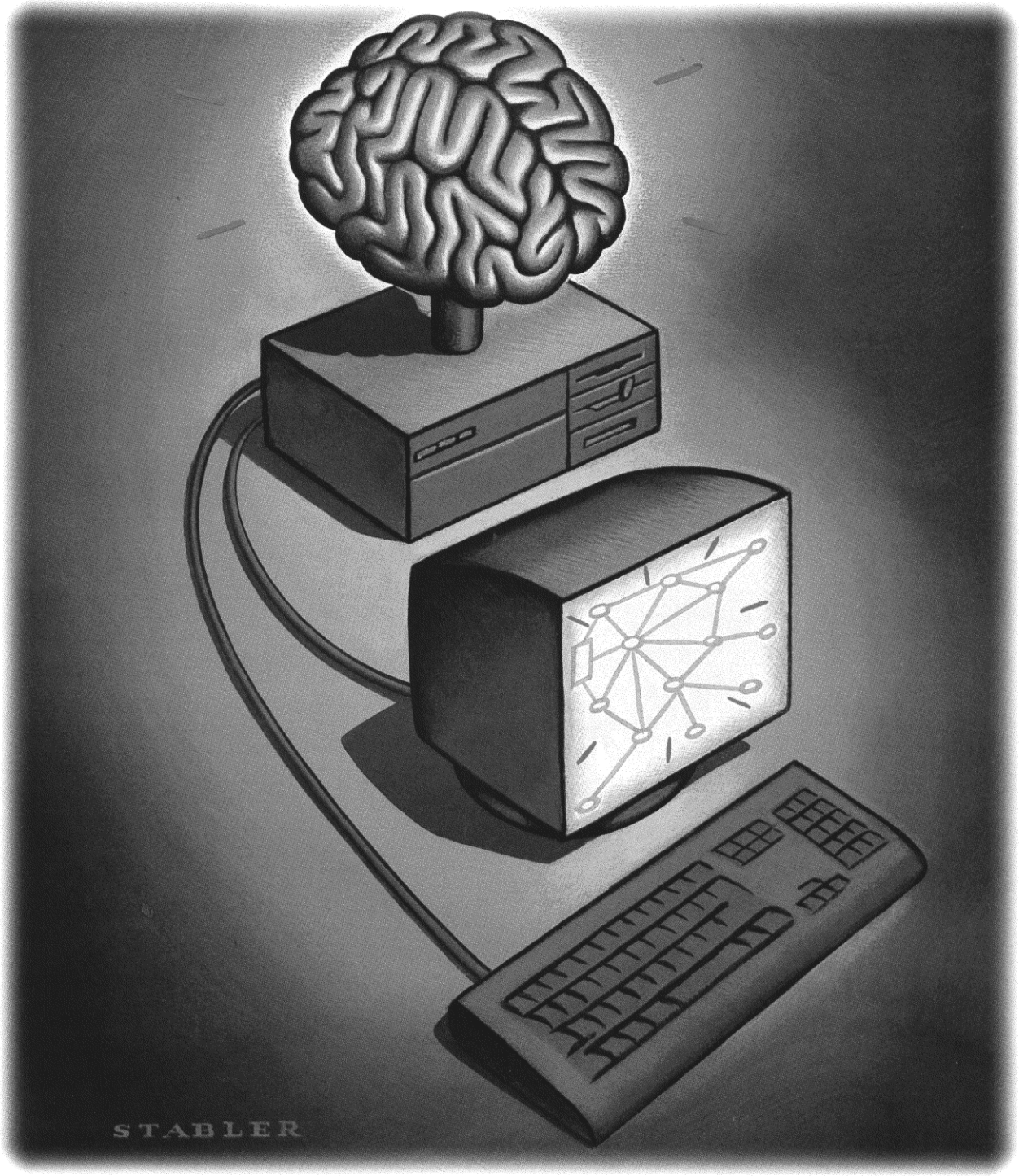
STABLER

The Basic Ideas in Neural Networks

DAVID E. RUMELHART ■ BERNARD WIDROW ■ MICHAEL A. LEHR

Interest in the study of neural networks has grown remarkably in the last several years. This effort has been characterized in a variety of ways: as the study of brain-style computation, connectionist architectures, parallel distributed-processing systems, neuromorphic computation, artificial neural systems. The common theme to these efforts has been an interest in looking at the brain as a model of a parallel computational device very different from that of a traditional serial computer.

The strategy has been to develop simplified mathematical models of brain-like systems and then to study these models to understand how various computational problems can be solved by such devices. The work has attracted scientists from a number of disciplines: neuroscientists who are interested in making models of the neural circuitry found in specific areas of the brains of various animals; physicists who see analogies between the dynamical behavior of brain-like systems and the kinds of nonlinear dynamical systems familiar in physics; computer engineers who are interested in fabricating brain-like computers; workers in artificial intelligence (AI) who are interested in building machines with the intelligence of biological organisms; engineers interested in solving practical problems; psychologists who are interested in the mechanisms of human information processing; mathematicians who are interested in the mathematics of such neural network systems; philosophers who are interested in how such systems change our view of the nature of mind and its relationship to brain; and many others. The wealth of talent and the breadth of interest have made the area a magnet for bright young students.

Although the details of the proposals vary, the most common models take the neuron as the basic processing unit. Each such processing unit is characterized by an activity level (representing the state of polarization of a neuron), an output value (representing the firing rate of the neuron), a set of input connections, (representing synapses on the cell and its dendrite), a bias value (representing an internal resting level of the neuron), and a set of output connections (representing a neuron's axonal projections). Each of these aspects of the unit are represented mathematically by real numbers. Thus, each connection has an associated weight (synaptic strength) which determines the effect of the incoming input on the activation level of the unit. The weights may be positive (excitatory) or negative (inhibitory). Frequently, the input lines are assumed to sum linearly yielding an activation value for unit i at time t, given by

$$\eta_i(t) = \sum_j w_{ij} x_j(t) + \beta_i,$$

where w_{ij} is the strength of the connection from $unit_j$ to $unit_i$, β_i is the unit's bias value; and x_j is the output value of unit j.

Note that the effect of a particular unit's output on the activity of another unit is jointly determined by its output level and the strength (and sign) of its connection to that unit. If the sign is negative, it lowers the activation; if the sign is positive it raises the activation. The magnitude of the output and the strength of the connection determine the amount of the effect. The output of such a unit is normally a nonlinear function of its activation value. A typical choice of such a function is the sigmoid. The logistic,

$$y_i(t) = \frac{1}{1 + e^{\frac{-\eta_i(t)}{T}}},$$

illustrated in Figure 1, will be employed in the examples illustrated

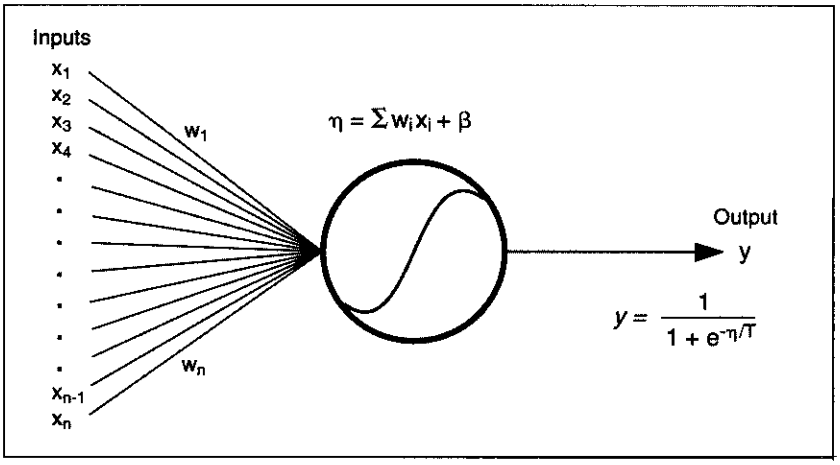

FIGURE 1. A simple sigmoidal output function

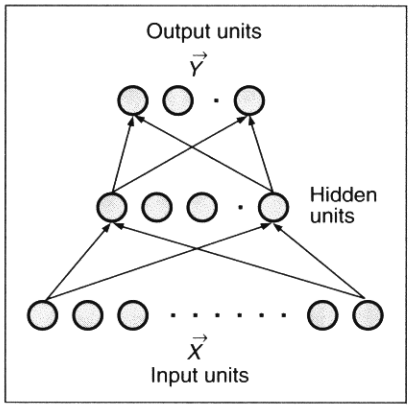

FIGURE 2. A simple three-layer network. The key to the effectiveness of the multilayer network is that the hidden units learn to rerepresent the input variables in a task-dependent way.

later. The parameter of the logistic T, yields functions of differing slopes. As T approaches zero the logistic becomes a simple logical threshold function which takes on the value of 1 if the activity level is positive and zero otherwise.

A brain-style computational device consists of a large network of such units, richly connected to one another. In real brains there are tens of billions of such units and tens of trillions of such connections. Such a network is a general computing device. The function it computes is determined by the pattern of connections. Thus, the configuration of connections is the analog of a program. The goal is to understand the kinds of algorithms that are naturally implemented by such networks.

Although there has been a good deal of activity recently, the study of brain-style computation has its roots over 50 years ago in the work of McCulloch and Pitts [7] and slightly later in Hebb's famous *Organization of Behavior* [4]. The early work in artificial intelligence was torn between those who believed that intelligent systems could best be built on computers modeled after brains [9, 13, 17], and those like Minsky and Papert [7] who believed that intelligence was fundamentally symbol processing of the kind readily modeled on the von Neumann computer. For a variety of reasons, the symbol-processing approach became the dominant theme in AI. The reasons for this were both positive and negative. On the one hand, the stored-program digital computer became the standard of the computer industry. Such computers were easy to design and easy to program. The symbol-processing/logic-based approach to AI is well suited for such an architecture. On the other hand, the fundamentally parallel neural network systems, such as Rosenblatt's perceptron system, were not well suited to implementation on serial computers. Moreover, the perceptron turned out to be rather more limited than first expected [8], and this discouraged both scientists and funding agencies. Although work continued throughout the 1970s by a number of workers including Amari, Anderson, Arbib, Fukushima, Grossberg, Kohonen, Widrow, and others, and although a number of important results were obtained during this period, the work received relatively little attention.

The 1980s showed a rebirth in interest. There seem to be at least five reasons for this. Three of the reasons are essentially pragmatic and two theoretical. First, on the more pragmatic side:

1. Today's computers are much faster than those of the 1950s and 1960s. It is thus possible to use conventional computers to simulate and experiment with much larger and more interesting networks than ever before.
2. Everyone believes that the future for faster computers must be in parallel computation. Unfortunately, there is no generally accepted paradigm for parallel computation. It is generally easier to build parallel computers than to find algorithms that are efficient for them. There is a hope that algorithms which prove efficient and effective on brain-style computers may prove a useful general paradigm for parallel computation.
3. The basic empirical tools of neuroscience are expanding, and we are learning more and more about how the neuron functions and how neurons communicate with one another. But little is known about how to go from this information about specific neurons to a theoretical account of how large networks of such neurons might function. It is hoped that the theoretical tools developed in the study of neural network computational systems will allow for the modeling of real neural networks.

In addition to the preceding three reasons, there have been two theoretical results which have been developed well enough to be appreciated.

1. The first of these results is due to Hopfield [6] and provides the mathematical foundation for understanding the dynamics of an important class of networks. In particular, Hopfield pointed out that recurrent networks with symmetric weights have a point-attractor dynamics, making their behavior relatively simple to understand and analyze. This obser-

*The **problem** of learning in neural networks is simply the problem of finding a set of connection strengths which allow the network to carry out the desired computation.*

■ ■ ■

vation has been extended and applied by Hinton and Sejnowski [5], Cohen and Grossberg [1], Smolensky [14], and a number of others to provide us with a useful mathematical understanding of how networks such as these might be configured to solve important optimization problems.

2. The second result is an extension of the work of Rosenblatt and Widrow and Hoff, to deal with learning in complex, multilayer networks and thereby provide an answer to one of the most severe criticisms of the original perceptron work. In this case, it was observed that by selecting differentiable, nonlinear functions (such as the sigmoid described earlier) it was possible to use the gradient search methods of Widrow and Hoff for nonlinear and multilayer networks. This provided a technique by which multilayer perceptron-like devices could be reliably trained. This procedure, known as the backpropagation learning algorithm, has had a major impact on the field and is the primary method employed in most of the applications we will discuss [11, 16]

Here we focus on the learning results, since they have had the greatest influence on applications.

Learning by Example

The problem of learning in neural networks is simply the problem of finding a set of connection strengths which allow the network to carry out the desired computation. In this section we focus on backpropagation, currently the most popular form of learning system and the one on which virtually all of the applications are based. The usual network architecture is illustrated in Figure 2.

There is a set of input units which are connected, through a set of so-called *hidden units*, to a set of output units. In the general case, there may be any number and configuration of hidden units and connections among the units. Generally, the hidden units are configured as a set of hidden-unit layers—most often there is a single layer of hidden units, but in some applications it is convenient to have two or more layers of hidden units. (For simplicity, we will restrict discussion here to the case of *feedforward* networks in which the activity of a given unit cannot influence, even indirectly, its own inputs.) The network is provided with a set of example input/output pairs (a training set) and is to modify its connections in order to approximate the function from which the input/output pairs have been drawn. The networks are then tested for ability to generalize.

The error correction learning procedure is simple enough in conception. The procedure is as follows: During training an input is put into the network and flows through the network generating a set of values on the output units. Then, the actual output is compared with the desired target, and a match is computed. If the output and target match, no change is made to the net. However, if the output differs from the target a change must be made to some of the connections. The problem is to determine which connections in the entire network were at fault for the error—this is called the *credit assignment* (or perhaps better, the blame assignment) problem. Although the solution to this problem for the case of networks without hidden layers has been known for some time, this is, in general, a difficult problem, and the lack of a satisfactory solution was a major factor in the earlier loss of interest in neural network systems. The 1980s has led to the development of a rather simple, yet powerful, solution to this problem. The basic idea is to define a measure of the overall performance of the system and then to find a way to optimize that performance. In this case, we can define the performance of the system as

$$E = \sum_{p,i} (t_{ip} - y_{ip})^2,$$

where i indexes the output units; p indexes the I/O pairs to be learned; t_{ip} indicates the target for a particular output unit on a particular pattern; y_{ip} indicates the actual output for that unit on that pattern; and E is the total error of the system. The goal, then, is to minimize this function. It turns out, if the output functions are differentiable, that this problem has a simple solution—namely, we can assign a particular unit blame in proportion to the degree to which changes in that unit's activity lead to changes in the error. That is, we change the weights of the system in proportion to the derivative of the error with respect to the weights. The change in w_{ij} is thus proportional to

$$\frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial w_{ij}}.$$

This simple procedure works remarkably well on a wide variety of problems. The problem of learning is thus reduced to the problem of parameter estimation.

A key advantage of neural network systems is that these simple, yet powerful learning procedures can be defined, allowing the systems to adapt to their environments. Work on the learning aspect of these neurally inspired models is what first led to an interest in them [9], and it was the conjecture *that learning procedures for complex networks could never be developed* that contributed to the loss of interest [8]. Although the *perceptron convergence procedure* and its variants had been around for some time, these learning procedures were limited to simple one-layer networks involving only *input* and *output* units. There

The **backpropagation learning procedure** *has become the single most popular method to train networks. The procedure has been used to train networks in problem domains including character recognition, speech recognition, sonar detection, and many more.*

■ ■ ■

were no *hidden units* in these cases and no *internal representation*. The coding provided by the external world had to suffice. Nevertheless, these networks have proved useful in a wide variety of applications (see [18]). Perhaps the essential character of such networks is that they map similar input patterns to similar output patterns. This characteristic is what allows these networks to make reasonable generalizations and perform reasonably on patterns that have never before been presented. The similarity of patterns in a connectionist system is determined by their overlap. The overlap in such networks is determined outside the learning system itself *by whatever produces the patterns*.

The constraint that similar input patterns lead to similar outputs can lead to an inability of the system to learn certain mappings from input to output. Whenever the representation provided by the outside world is such that the similarity structure of the input and output patterns is very different, a network without internal representations (i.e., a network without hidden units) will be unable to perform the necessary mappings.

In a multilayer network, the information coming to the input units is *recoded* into an internal representation, and the outputs are generated by the internal representation rather than by the original pattern. If we have enough connections from the input units to a large enough set of hidden units, we can always find a representation that will perform any mapping from input to output through these hidden units.

The existence of multilayer networks illustrates the potential power of hidden units and internal representations. The problem, as noted by Minsky and Papert [8], is that whereas there is a very simple guaranteed learning rule for all problems that can be solved without hidden units, namely, the perceptron convergence procedure (or the variation due originally to Widrow and Hoff [17]), there has been no equally powerful rule for learning in multilayer networks. We are thus not assured of optimal solutions—local minima are always a possibility. Nevertheless, the backpropagation procedure is sufficiently robust that local minima rarely turn out to be serious limitations.

Although the learning results do not *guarantee* that we can find a solution for all solvable problems, our analyses and simulation results have shown that as a practical matter, the backward-error propagation scheme leads to solutions in virtually every case.

Generalization

The backpropagation learning procedure sketched earlier has become, perhaps, the single most popular method to train networks. The procedure has been used to train networks in problem domains including character recognition, speech recognition, sonar detection, mapping from spelling to sound, motor control, analysis of molecular structure, diagnosis of eye diseases, prediction of chaotic functions, playing backgammon, the parsing of simple sentences, and many many more areas of application (see [18]). Perhaps the major point of these examples is the enormous range of problems to which the backpropagation learning procedure can usefully be applied. In spite of the rather impressive breadth of topics, and the success of some of these reapplications, there are a number of serious open problems. The theoretical issues of primary concern fall into four main areas:

1. The learning problem—can the network learn how to solve the problem at hand?
2. The architecture problem—are there useful architectures, beyond the standard three-layer network employed in most of these areas, which are appropriate for certain areas of application?
3. The scaling problem—how can we cut down on the substantial training time that seems to be involved for the more difficult and interesting problem application areas?
4. The generalization problem—how can we be certain that the network trained on a subset of the example set will generalize correctly to the entire set of exemplars?

The original efforts were focused on the first of these problems. The primary applications of our learning algorithms were to see if a network could learn some complex nonlinear function. Thus we focused on such problems as parity, exclusive-or, and other similar analytically defined problems. We found that with a sufficiently large network we could learn essentially any function. The initial worries about the role of local minima and similar problems turned out to be much less serious than we originally thought. However, we have come to understand that the "generalization" problem is much more serious than we might have thought. This, of course, is just the mirror image of the learning problem. The more general our learning procedure, the less constraints we have on the way the network actually solves the problem and therefore the less certain we can be about the network's ability to properly generalize to new cases. In the statistics literature this is known as the "overfitting" problem.

Models of many parameters can fit essentially any function in many different ways. Our problem is to fit the function in such a way that it maximizes its ability to generalize to an as yet unseen collection of data. There have been essentially two strategies in the connectionist literature to deal with this problem.

The first strategy is a version of "Occam's Razor"—i.e., the notion that the simplest hypothesis consistent with the data is the one that should be chosen. In the world of connectionist networks this involves the view that the simplest network consistent with the data should be chosen. There are a number of measures of simplicity in a network. We, for example, have suggested that the following variables covary with simplicity: number of weights, number of units, number of symmetries among the weights, number of bits per weight, and so forth. It is possible to define cost functions which lead to minimal-complexity networks as measured by any or all of these measurements. Generally, we find that minimal networks offer better generalization performance than more complex networks [15].

The second basic scheme for network training and, in fact, the most commonly used scheme is a version of cross-validation. In this scheme, the data are divided into three parts. One part is used for training; one part is used to evaluate the generalization performance and is set aside for a final test; and one part of the data is used for cross-validation.

The procedure is as follows: following each training epoch, the performance of the network is evaluated on the validation set. As long as the network continues to improve on the validation, set training is continued. If over-fitting is occurring, the network will at some point begin to show poorer performance on the validation data. At that point we stop training and select the weights which give optimal performance on the validation set for testing against the "test set," and the performance on this set is used as a measure of the quality of the generalization. This method is reasonably powerful and simple and often leads to good results. The results are nearly as good for this method as for the more complex method described earlier, and the training time is generally much less [2].

Hints for Successful Applications

Although some authors have suggested that neural networks are simple black boxes that can be applied without much consideration of the details of the problem, most successful applications require great care in approaching the problem at hand. Following are a number of considerations that have proved useful in some areas of application.

1. Be certain to have enough data to constrain your model sufficiently for the problem at hand.
2. Carefully design appropriate input data. This will often require theory-based data reduction of the number of input variables. This was important in the work of Rumelhart [10] on cursive handwriting. In this case a coupled oscillator model of handwriting was used followed by the parameters of this model, rather than the underlying temporal data. This allowed a five-fold reduction in the size of the input space.
3. Build known symmetries (often through weight linking) into your network wherever possible. This allows a substantial reduction in the number of weights in the network and allows the network to learn without having each region of the network see each input pattern.
4. Build a probabilistic model of the task. Make use of "forward models" to map from a representation of the input that you want to discover to a target set that is easy to construct. This method was also used in the cursive handwriting work. Here the problem was that we did not want to have to tell the network exactly where each character in the word was. Rather we wanted to simply tell the network which characters were in the word. The original network tried to predict the location of each character in the word, but we attached a second network that took the first network's guesses as to the location of the character and computed the probability that the character was "anywhere" in the word. This was a fixed network and was used to compute these probabilities. Thus while the targets could be simple information about which characters were in the word, the network could determine where each character was. The details of this are given in Rumelhart [10].
5. Use the network to solve problems it is good at, but feel free to combine the network with other statistical methods. Making certain you can offer a clear probabilistic/Bayesian interpretation of the behavior will help in interfacing the network with other statistical methods. It is very useful to have the network provide output values which are reasonably interpreted as probabilities. These probabilities can then be used to determine confidence levels and to combine with other sources of evidence. See Curry and Rumelhart [2] for a useful example. ■

References

1. Cohen, M.N. and Grossberg, S. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. Man Cybernet. 13* (1983).
2. Curry, B. and Rumelhart, D.E. MSNET: A neural network that classifies mass spectra. HPL Tech. Rep. 90–161. MPL.
3. Grossberg, S. Adaptive pattern classification and universal recoding: Part I: Parallel development and coding of neural feature detectors. *Bio. Cybernet. 23* (1976), 121–134.
4. Hebb, D.O. *The Organization of Behavior.* Wiley, New York, 1949.
5. Hinton, G.E. and Sejnowski, T. Learning and relearning in Boltzmann Machines. In *Parallel Distributed Processing: Exploration in the Microstructure of Cognition.* Vol. 1: *Foundations,* D.E. Rumelhart, J.L. McClelland, and PDP Res. Grp. MIT Press/Bradford Books, Cambridge, Mass., 1986.
6. Hopfield, J.J. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences, USA.* Vol. 79. National Academy of Sciences, Washington, D.C., 1982.
7. McCulloch, W.S. and Pitts, W. A logical calculus of the ideas imminent in nervous activity. *Bull. Math. Biophys. 5,* 115–133.
8. Minsky, M. and Papert, S. *Perceptrons.* MIT Press, Cambridge, Mass., 1969.
9. Rosenblatt, F. *Principles of Neurodynamics.* Spartan, New York, 1962.
10. Rumelhart, D.E. *Theory to practice: A*

case study—Recognizing cursive handwriting. In *Proceedings of the Third NEC Research Symposium*. SIAM, 1993.

11. Rumelhart, D.E., Hinton, G.E., and Williams, R.J. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1 *Foundations*, D.E. Rumelhart, J.L. McClelland, and PDP Res. Grp. MIT Press/Bradford Books, Cambridge, Mass., 1986.
12. Rumelhart, D.E., McClelland, J.L., and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1. *Foundations*. MIT Press/Bradford Books, Cambridge, Mass., 1986.
13. Selfridge, O.G. Pattern recognition in modern computers. In *Proceedings of the Western Joint Computer Conference*. ACM, New York, 1955.
14. Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1. *Foundations*, D.E. Rumelhart, J.L. McClelland, and the PDP Res. Grp. MIT Press/Bradford Books, Cambridge, Mass., 1986.
15. Weigend, A.S., Rumelhart, D.E., and Hubberman, B. Generalization by weight-elimination with applications to forecasting. In *Advances in Neural Information Processing*. Vol. 3, R.P. Lippman, J. Mody, and D.S. Touretsky. Morgan Kaufman, San Mateo, Calif., 1991, pp. 875–882.
16. Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard Univ., Cambridge, Mass.
17. Widrow, B. and Hoff, M.E. Adaptive switching circuits. In *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record. Part 4*, Inst. Radio Eng., 1960, pp. 96–104.
18. Widrow, B., Rumelhart, D.E., and Lehr, M.A. *Commun ACM* 37, 4 (March 1994) (this issue).

About the Authors:

DAVID E. RUMELHART is professor of psychology at Stanford University. His research has focused on how people learn complex skills such as reading, and how that knowledge is represented in the mind. He is coauthor of the well-known 2-volume set of connectionist texts, *Parallel Distributed Processing*.

BERNARD WIDROW is professor of electrical engineering at Stanford University. He does research and teaching in the fields of digital signal processing, adaptive signal processing, adaptive control systems, pattern recognition and neural networks. He is coinventor of the LMS algorithm and the neural element ADALINE and various MADALINE networks.

MICHAEL LEHR is a doctoral candidate in electrical engineering at Stanford University. His research involves the application of second-order training techniques to large neural networks.

Authors' Present Addresses: David Rumelhart can be reached at Stanford University Department of Psychology, Bldg. 420 Room 414, Stanford, CA 94305-2130. Bernard Widrow and Michael Lehr can be reached at Stanford University Department of Electrical Engineering, Durand Bldg., Stanford, CA 94305-4055.