Adaptive Neural Networks and their Applications

Bernard Widrow and Michael A. Lehr
Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, California 94305-4055

Fundamental developments in feedforward artificial neural networks from the past 30 years are reviewed. The central theme of this article is a description of the history, origination, operating characteristics, and basic theory of several supervised neural network training algorithms including the Perceptron rule, the LMS algorithm, three Madaline rules, and the backpropagation technique. These methods were developed independently, but with the perspective of history they can all be related to each other. The concept which underlies these algorithms is the "minimal disturbance principle," which suggests that during training it is advisable to inject new information into a network in a manner which disturbs stored information to the smallest extent possible. In the utilization of present-day rule-based expert systems, decision rules must always be known for the application of interest. Sometimes there are no rules, however. The rules are either not explicit or they simply do not exist. For such applications, trainable expert systems might be usable. Rather than working with decision rules, an adaptive expert system might observe the decisions made by a human expert. Looking over the expert's shoulders, an adaptive system can learn to make similar decisions to those of the human. Trainable expert systems have been used in the laboratory for real-time control of a "broom-balancing system."

I. INTRODUCTION

More than 30 years have passed since the development of two of the earliest and most important rules for training adaptive elements. The Perceptron rule and the LMS algorithm were both first published in 1960. In the years following these discoveries, many new techniques have been developed in the field of neural networks, and the discipline has grown rapidly. One early development was Steinbuch's Learning Matrix,[1] a pattern recognition machine based on linear discriminant functions. In the same time frame, Widrow and his students devised Madaline Rule I (MRI), the earliest popular learning rule for neural networks with multiple adaptive elements.[2] Other early work included the "mode-seeking" technique of Stark, Okajima, and Whipple.[3] This was probably the first example of competitive learning in the literature, though it could be argued that earlier work by Rosenblatt on "spontaneous learning"[4,5]

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 8, 453–507 (1993)
 CCC 0884-8173/93/040453-55

deserves this distinction. Further pioneering work on competitive learning and self-organization was performed in the 1970s by von der Malsburg[6] and Grossberg.[7] Fukushima explored related ideas with his biologically inspired Cognitron and Neocognitron models.[8,9]

In the mid-1960s, Widrow devised a reinforcement learning algorithm called "punish/reward" or "bootstrapping."[10,11] This can be used to solve problems when uncertainty about the error signal causes supervised training methods to be impractical. A related reinforcement learning approach was later explored in a classic paper by Barto, Sutton, and Anderson on the "credit assignment" problem.[12] Barto *et al.*'s technique is also somewhat reminiscent of Albus's adaptive CMAC, a distributed table-lookup system based on models of human memory.[13,14] Yet another approach related to bootstrapping is the associative reward–penalty algorithm of Barto and Anandan.[15] This method solves associative reinforcement learning tasks, providing a link between pattern classification and stochastic learning automata.

In the 1970s Grossberg developed his Adaptive Resonance Theory (ART), a number of novel hypotheses about underlying principles which govern biological neural systems.[16] These ideas served as the basis for later work by Carpenter and Grossberg involving three classes of ART architectures: ART 1,[17] ART 2,[18] and ART 3.[19] These are self-organizing neural implementations of pattern clustering algorithms. Another important theory on self-organizing systems was pioneered by Kohonen with his work on feature maps.[20,21]

In the early 1980s, Hopfield and others introduced outer product rules as well as equivalent approaches based on the early work of Hebb[22] for training a class of recurrent (signal feedback) networks now called Hopfield models.[23,24] More recently, Kosko extended some of the ideas of Hopfield and Grossberg to develop his adaptive Bidirectional Associative Memory (BAM),[25] a network model employing differential as well as Hebbian and competitive learning laws. Another model utilizing a differential learning mechanism is Harry Klopf's Drive Reinforcement Theory,[26] an extension of Hebbian learning which explains Pavlovian classical conditioning. Other significant models from the past decade include probabilistic ones such as Hinton, Sejnowski, and Ackley's Boltzmann Machine,[27,28] which to oversimplify, is a Hopfield model that settles into solutions by a simulated annealing process governed by Boltzmann statistics. The Boltzmann Machine is trained by a clever two-phase Hebbian-based technique.

While these developments were taking place, adaptive systems research at Stanford traveled an independent path. After devising their Madaline I rule, Widrow and his students developed uses for the Adaline and Madaline. Early applications included, among others, speech and pattern recognition,[29] weather forecasting,[30] and adaptive controls.[31] Work then switched to adaptive filtering and adaptive signal processing[32] after attempts to develop learning rules for networks with multiple adaptive layers were unsuccessful. Adaptive signal processing proved to be a fruitful avenue for research with applications involving adaptive antennas,[33] adaptive inverse controls,[34] adaptive noise cancelling,[35] and seismic signal processing.[32] Outstanding work by R. W. Lucky and

others at Bell Laboratories led to major commercial applications of adaptive filters and the LMS algorithm to adaptive equalization in high speed modems[36,37] and to adaptive echo cancellers for long distance telephone and satellite circuits.[38] After 20 years of research in adaptive signal processing, the work in Widrow's laboratory has once again returned to neural networks.

The first major extension of the feedforward neural network beyond Madaline I took place in 1971 when Werbos developed a backpropagation training algorithm which, in 1974, he first published in his doctoral dissertation.[39]* Unfortunately, Werbos's work remained almost unknown in the scientific community. In 1982, Parker rediscovered the technique[41] and in 1985, published a report on it at MIT.[42] Not long after Parker published his findings, Rumelhart, Hinton, and Williams[43,44] also rediscovered the technique and, largely as a result of the clear framework within which they presented their ideas, they finally succeeded in making it widely known.

The elements used by Rumelhart *et al.* in the backpropagation network differ from those used in the earlier Madaline architectures. The adaptive elements in the original Madaline structure used hard-limiting quantizers (signums), while the elements in the backpropagation network use only differentiable nonlinearities, or "sigmoid" functions.† In digital implementations, the hard-limiting quantizer is more easily computed than any of the differentiable nonlinearities used in backpropagation networks. In 1987, Widrow and Winter looked back at the original Madaline I algorithm with the goal of developing a new technique that could adapt multiple layers of adaptive elements which use the simpler hard-limiting quantizers. The result was Madaline Rule II.[45]

David Andes of U.S. Naval Weapons Center of China Lake, CA, modified Madaline II in 1988 by replacing the hard-limiting quantizers in the Adaline with sigmoid functions, thereby inventing Madaline Rule III (MRIII). Widrow and his students were first to recognize that this rule is mathematically equivalent to backpropagation.

The outline above gives only a partial view of the discipline, and many landmark discoveries have not been mentioned. Needless to say, the field of neural networks is quickly becoming a vast one, and in one short survey we could not hope to cover the entire subject in any detail. Consequently, many significant developments, including some of those mentioned above, will not be discussed in this article. The algorithms described will be limited primarily to

*We should note, however, that in the field of variational calculus the idea of error backpropagation through nonlinear systems existed centuries before Werbos first thought to apply this concept to neural networks. In the past 25 years, these methods have been used widely in the field of optimal control. as discussed by Le Cun.[40]

†The term *sigmoid* is usually used in reference to monotonically increasing "S-shaped" functions, such as the hyperbolic tangent. In this article, however, we generally use the term to denote any smooth nonlinear functions at the output of a linear adaptive element. In other articles, these nonlinearities go by a variety of names, such as "squashing functions," "activation functions," "transfer characteristics," or "threshold functions."

those developed in our laboratory at Stanford, and to related techniques developed elsewhere, the most important of which is the backpropagation algorithm.

The section headings indicate the range and coverage of the article:

Information about the neural network paradigms not discussed in this article can be obtained from a number of other sources, such as the concise survey by Richard Lippmann,[46] and the collection of classics by Anderson and Rosenfeld.[47] Much of the early work in the field from the 1960s is carefully reviewed in Nilsson's monograph.[48] A good view of some of the more recent results is presented in Rumelhart and McClelland's popular two-volume set.[49] An article by Moore[50] presents a clear discussion about ART 1 and some of Grossberg's terminology. Another resource is the DARPA Study report[51] which gives a very comprehensive and readable "snapshot" of the field in 1988.

II. FUNDAMENTAL CONCEPTS

Today we can build computers and other machines which perform a variety of well-defined tasks with celerity and reliability unmatched by humans. No human can invert matrices or solve systems of differential equations at speeds which rival modern workstations. Nonetheless, there are still many problems which have yet to be solved to our satisfaction by any man-made machine, but are easily disentangled by the perceptual or cognitive powers of humans, and often lower mammals, or even fish and insects. No computer vision system can rival the human ability to recognize visual images formed by objects of all shapes and orientations under a wide range of conditions. Humans effortlessly recognize objects in diverse environments and lighting conditions, even when obscured by dirt, or occluded by other objects. Likewise, the performance of current speech recognition technology pales when compared to the performance of the human adult who easily recognizes words spoken by different people, at different rates, pitches, and volumes, even in the presence of distortion or background noise.

The problems solved more effectively by the brain than by the digital computer typically have two characteristics: they are generally ill-defined, and they usually require an enormous amount of processing. Recognizing the character of an object from its image on television, for instance, involves resolving ambiguities associated with distortion and lighting. It also involves filling in

information about a three-dimensional scene which is missing from the two-dimensional image on the screen. There are an infinite number of three-dimensional scenes which can be projected into a two-dimensional image. Nonetheless, the brain deals well with this ambiguity, and using learned cues usually has little difficulty correctly determining the role played by the missing dimension.

As anyone who has performed even simple filtering operations on images is aware, processing high resolution images requires a great deal of computation. Our brains accomplish this by utilizing massive parallelism, with millions and even billions of neurons in parts of the brain working together to solve complicated problems. Because solid state operational amplifiers and logic gates can compute many orders of magnitude faster than current estimates of the computational speed of neurons in the brain, we may soon be able to build relatively inexpensive machines with the ability to process as much information as the human brain. This enormous processing power will do little to help us solve problems, however, unless we can utilize it effectively. For instance, coordinating many thousands of processors which must efficiently cooperate to solve a problem is not a simple task. If each processor must be programmed separately, and if all contingencies associated with various ambiguities must be designed into the software, even a relatively simple problem can quickly become unmanageable. The slow progress over the past 25 years or so in machine vision and other areas of artificial intelligence is testament to the difficulties associated with solving ambiguous and computationally intensive problems on von Neumann computers and related architectures.

Thus, there is some reason to consider attacking certain problems by designing naturally parallel computers which process information and learn by principles borrowed from the nervous systems of biological creatures. This does not necessarily mean we should attempt to copy the brain part for part. Although the bird served to inspire development of the airplane, birds do not have propellers, and airplanes do not operate by flapping feathered wings. The primary parallel between biological nervous systems and artificial neural networks is that each typically consists of a large number of simple elements that learn and are able to collectively solve complicated and ambiguous problems.

Today, most artificial neural network research and application is accomplished by simulating networks on serial computers. Speed limitations keep such networks relatively small, but even with small networks some surprisingly difficult problems have been tackled. Networks with fewer than 150 neural elements have been used successfully in vehicular control simulations,[52] speech generation,[53,54], and undersea mine detection.[51] Small networks have also been used successfully in airport explosive detection,[55] expert systems,[56,57] and scores of other applications. Furthermore, efforts to develop parallel neural network hardware are being met with some success, and such hardware should be available in the future for attacking more difficult problems like speech recognition.[58,59]

Whether implemented in parallel hardware or simulated on a computer, all neural networks consist of a collection of simple elements that work together to solve problems. A basic building block of nearly all artificial neural networks, and most other adaptive systems, is the adaptive linear combiner.

A. The Adaptive Linear Combiner

The adaptive linear combiner is diagrammed in Figure 1. Its output is a linear combination of its inputs. In a digital implementation, this element receives at time k an input signal vector or input pattern vector $\mathbf{X}_k = [x_0, x_{1_k}, x_{2_k}, \ldots x_{n_k}]^T$, and a desired response d_k, a special input used to effect learning. The components of the input vector are weighted by a set of coefficients, the weight vector $\mathbf{W}_k = [w_{0_k}, w_{1_k}, w_{2_k}, \ldots w_{n_k}]^T$. The sum of the weighted inputs is then computed, producing a linear output, the inner product $s_k = \mathbf{X}_k^T\mathbf{W}_k$. The components of $\mathbf{X}_k$ may be either continuous analog values or binary values. The weights are essentially continuously variable, and can take on negative as well as positive values.

During the training process, input patterns and corresponding desired responses are presented to the linear combiner. An adaptation algorithm automatically adjusts the weights so that the output responses to the input patterns will be as close as possible to their respective desired responses. In signal processing applications, the most popular method for adapting the weights is the simple LMS (least mean square) algorithm,[60,61] often called the Widrow–Hoff Delta Rule.[44] This algorithm minimizes the sum of squares of the linear errors over the training set. The linear error ε_k is defined to be the difference between the desired response d_k and the linear output s_k, during presentation k. Having this error signal is necessary for adapting the weights. When the adaptive linear combiner is embedded in a multi-element neural network, however, an error signal is often not directly available for each individual linear combiner and more complicated procedures must be devised for adapting the weight vectors. These procedures are the main focus of this article.

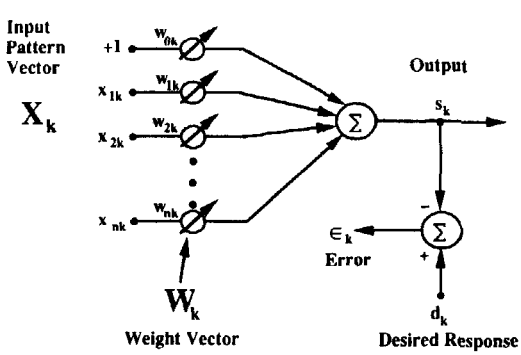

Figure 1. Adaptive linear combiner.

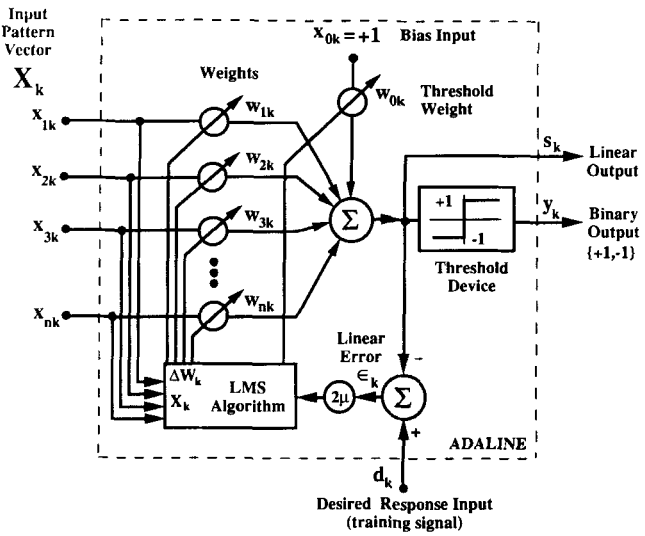

Figure 2. An adaptive linear element (Adaline).

B. A Linear Classifier—The Single Threshold Element

The basic building block used in many neural networks is the "adaptive linear element," or Adaline[60,*] shown in Figure 2.

This is an adaptive threshold logic element. It consists of an adaptive linear combiner cascaded with a hard-limiting quantizer which is used to produce a binary ± 1 output, $y_k = \text{sgn}(s_k)$. The bias weight w_{0_k} which is connected to a constant input, $x_0 = +1$, effectively controls the threshold level of the quantizer.

In single-element neural networks, an adaptive algorithm (such as the LMS algorithm, or the Perceptron rule) is often used to adjust the weights of the Adaline so that it responds correctly to as many patterns as possible in a training set which has binary desired responses. Once the weights are adjusted, the responses of the trained element can be tested by applying various input patterns. If the Adaline responds correctly with high probability to input patterns that were not included in the training set, it is said that generalization has taken place. Learning and generalization are among the most useful attributes of Adalines and neural networks.

*In the neural network literature, such elements are often referred to as "adaptive neurons." However, in a conversation between David Hubel of Harvard Medical School and Bernard Widrow, Dr. Hubel pointed out that the Adaline differs from the biological neuron since it contains not only the neural cell body, but also the input synapses and a mechanism for training them.

1. *Linear Separability*

With n binary inputs and one binary output, a single Adaline of the type shown in Fig. 2 is capable of implementing certain logic functions. There are 2^n possible input patterns. A general logic implementation would be capable of classifying each pattern as either $+1$ or -1, in accord with the desired response. Thus, there are 2^{2^n} possible logic functions connecting n inputs to a single binary output. A single Adaline is capable of realizing only a small subset of these functions, known as the linearly separable logic functions or threshold logic functions.[62] These are the set of logic functions that can be obtained with all possible weight variations.

The linear classifier is limited in the number of distinct patterns it can learn correctly. The Adaline's pattern capacity is limited roughly to twice the number of adaptive weights in the classifier.[63,64]. To achieve higher pattern capacities, or to solve problems which are not linearly separable, nonlinear classifiers must be used.

C. Nonlinear Classifiers

Many nonlinear classifiers are simple extensions of the Adaline. Two of the most common are described here. The first is a fixed preprocessing network connected to a single adaptive element, and the other is the multi-element feedforward neural network. The pattern capacities of both structures can be approximated by the number of weights in the classifier divided by the number of output nodes.

1. *Polynomial Discriminant Functions*

Nonlinear functions of the inputs applied to the single Adaline can yield nonlinear decision boundaries. Useful nonlinearities include the polynomial functions. Consider the system illustrated in Figure 3 which contains only

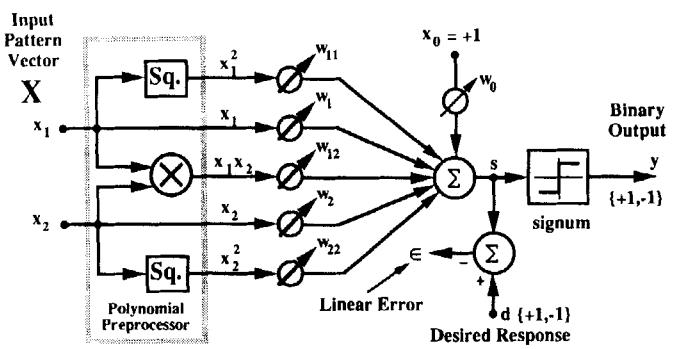

Figure 3. An Adaline with inputs mapped through nonlinearities.

linear and quadratic input functions. The critical thresholding condition for this system is

$$s = u_0 + x_1 w_1 + x_1^2 w_{11} + x_1 x_2 w_{12} + x_2^2 w_{22} + x_2 w_2 = 0 \qquad (1)$$

With proper choice of the weights, the separating boundary in pattern space can be established as shown, for example, in Figure 4. This represents a solution for the Exclusive NOR, a function which is not linearly separable. Of course, all of the linearly separable functions are also realizable. The use of such nonlinearities can be generalized for more inputs than two and for higher degree polynomial functions of the inputs. Some of the first work in this area was done by D. F. Specht[65–67] at Stanford in the 1960s when he successfully applied polynomial discriminants to the classification and analysis of electrocardiographic signals. Work on this topic has also been done by Barron[68–70] and by A. G. Ivankhnenko[71] in the Soviet Union.

The polynomial approach offers great simplicity and beauty. Through it one can realize a wide variety of adaptive nonlinear discriminant functions by adapting only a single Adaline element. Several methods have been developed for training the polynomial discriminant function. Specht developed a very efficient noniterative (i.e., single pass through the training set) training procedure, the Polynomial Discriminant Method (PDM), which allows the polynomial discriminant function to implement a nonparametric classifier based on the Bayes decision rule. Other methods for training the system include iterative error correction rules such as the Perceptron and α-LMS rules, and iterative gradient descent procedures such as the μ-LMS and SER (also called RLS) algorithms.[32] Gradient descent with a single adaptive element is typically much faster than with a layered neural network. Furthermore, as we shall see, when the single Adaline is trained by a gradient descent procedure, it will converge to a unique global solution.

After the polynomial discriminant function has been trained by a gradient descent procedure, the weights of the Adaline will represent an approximation to the coefficients in a multi-dimensional Taylor series expansion of the desired

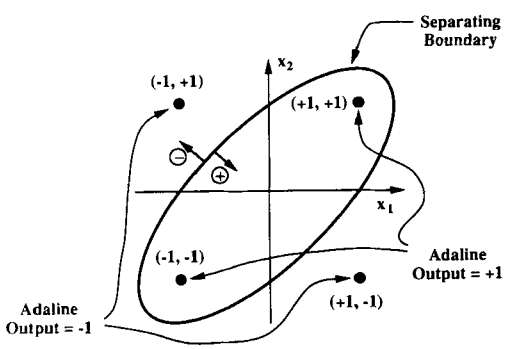

Figure 4. An elliptical separating boundary for realizing a function which is not linearly separable.

response function. Likewise, if appropriate trigonometric terms are used in place of the polynomial preprocessor, the Adaline's weight solution will approximate the terms in the (truncated) multi-dimensional Fourier series decomposition of a periodic version of the desired response function. The choice of preprocessing functions determines how well a network will generalize for patterns outside the training set. Determining "good" functions remains a focus of current research.[72,73] Experience seems to indicate that unless the nonlinearities are chosen with care to suit the problem at hand, often better generalization can be obtained from networks with more than one adaptive layer. In fact, one can view multi-layer networks as single-layer networks with trainable preprocessors which are essentially self-optimizing.

2. *Madaline I*

One of the earliest trainable layered neural networks with multiple adaptive elements was the Madaline I structure of Widrow and Hoff.[2,74] Mathematical analyses of Madaline I were developed in the Ph.D. theses of Ridgway,[75] Hoff,[74] and Glanz.[76] In the early 1960s, a 1000-weight Madaline I was built out of hardware[77] and used in pattern recognition research. The weights in this machine were memistors, electrically variable resistors developed by Widrow and Hoff which are adjusted by electroplating a resistive link.[78]

Madaline I was configured in the following way. Retinal inputs were connected to a layer of adaptive Adaline elements, the outputs of which were connected to a fixed logic device that generated the system output. Methods for adapting such systems were developed at that time. An example of this kind of network is shown in Figure 5. Two Adalines are connected to an AND logic device to provide an output.

With weights suitably chosen, the separating boundary in pattern space for the system of Fig. 5 would be as shown in Figure 6. This separating boundary also implements the Exclusive NOR function.

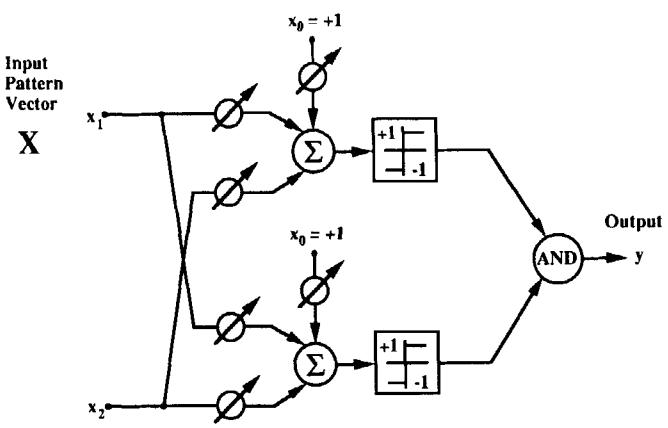

Figure 5. A two-Adaline form of Madaline.

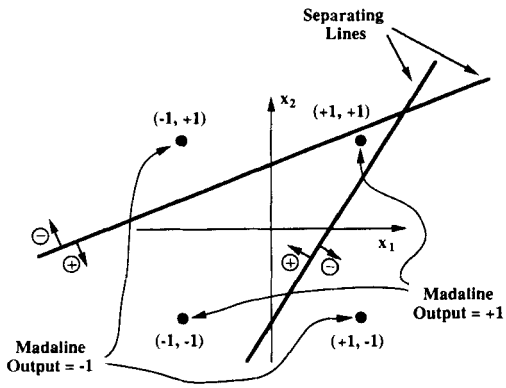

Figure 6. Separating lines for Madaline of Figure 5.

Madalines were constructed with many more inputs, with many more Adaline elements in the first layer, and with various fixed logic devices such as AND, OR, and Majority vote-taker elements in the second layer. Those three functions, illustrated in Figure 7, are all threshold logic functions. The given weight values will implement these three functions, but the weight choices are not unique.

3. *Feedforward Networks*

The Madalines of the 1960s had adaptive first layers and fixed threshold functions in the second (output) layers.[48,75] The feedforward neural networks of today often have many layers, and usually all layers are adaptive. The backpropagation networks of Rumelhart *et al.*[49] are perhaps the best-known examples of multi-layer networks. A fully connected three-layer* feedforward adaptive network is illustrated in Figure 8. In a fully connected layered network, each Adaline receives inputs from every output in the preceding layer.

During training, the response of each output element in the network is compared with a corresponding desired response. Error signals associated with the output elements are readily computed, so adaptation of the output layer is straightforward. The fundamental difficulty associated with adapting a layered network lies in obtaining "error signals" for hidden layer Adalines, that is, for Adalines in layers other than the output layer. The backpropagation and Madaline III algorithms contain methods for establishing these error signals.

A network's capacity is of little utility unless it is accompanied by useful generalizations to patterns not presented during training. In fact, if generalization is not needed, we can simply store the associations in a look-up table, and

*In Rumelhart *et al.*'s terminology, this would be called a 4-layer network, following Rosenblatt's convention of counting layers of signals, including the input layer. For our purposes, we find it more useful to count only layers of computing elements. We do not count as a layer the set of input terminal points.

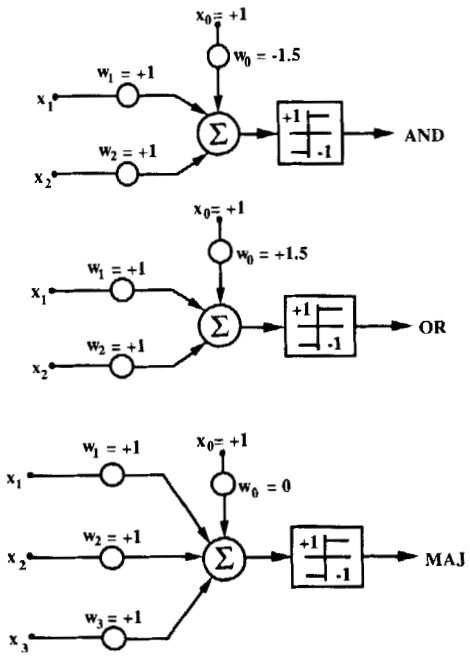

Figure 7. Fixed-weight Adaline implementations of AND, OR, and MAJ logic functions.

will have little need for a neural network. The relationship between generalization and pattern capacity represents a fundamental tradeoff in neural network applications: reduced capacity translates to improved generalization. The fact that the Adaline is unable to realize all functions is in a sense a strength rather than the fatal flaw envisioned by some critics of neural networks[79] because it helps limit the capacity of the device and thereby improves its ability to generalize.

For good generalization, the training set should contain a number of patterns at least several times larger than the network's capacity. This can be

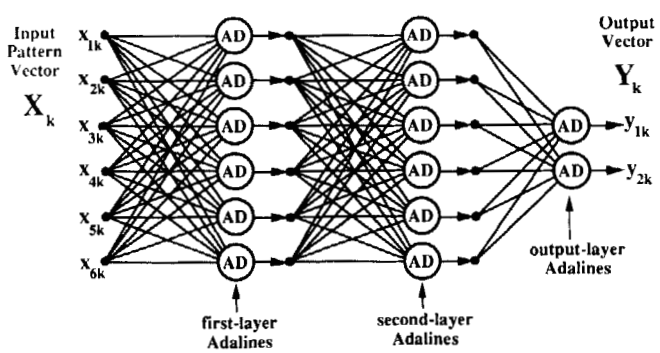

Figure 8. A three-layer adaptive neural network.

understood intuitively by noting that if the number of degrees of freedom in a network (i.e., the number of weights) is larger than the number of constraints associated with the desired response function (i.e., the product of the number of patterns and the number of outputs), the training procedure will be unable to completely constrain the weights in the network. A detailed analysis of generalization performance of signum networks as a function of training set size is described in Ref. 80.

There is no reason why a feedforward network must have the layered structure of Figure 8. In Werbos's development of the backpropagation algorithm,[39] in fact, the Adalines are ordered and each receives signals directly from each input component and from the output of each preceding Adaline. Many other variations of the feedforward network are possible. An interesting area of current research involves a generalized backpropagation method which can be used to train "high order" or "signal-pi" networks that incorporate a polynomial preprocessor for each Adaline.[49,81]

4. *A Nonlinear Classifier Application*

Neural networks have been used successfully in a wide range of applications. To gain some insight about how neural networks are trained and what they can be used to compute, it is instructive to consider Sejnowski and Rosenberg's 1986 NETtalk demonstration.[53,54] With the exception of work on the traveling salesman problem with Hopfield networks,[82] this was the first neural network application since the 1960s to draw widespread attention. NETtalk is a two-layer feedforward sigmoid network with 80 Adalines in the first layer and 26 Adalines in the second layer. The network is trained to convert text into phonetically correct speech, a task well suited to neural implementation. The pronunciation of most words follows general rules based upon spelling and word context, but there are many exceptions and special cases. Rather than programming a system to respond properly to each case, the network can learn the general rules and special cases by example.

One of the most remarkable characteristics of NETtalk is that it learns to pronounce words in stages suggestive of the learning process in children. When the output of NETtalk is connected to a voice synthesizer, the system makes babbling noises during the early stages of the training process. As the network learns, it next conquers the general rules, and like a child, tends to make a lot of errors by using these rules even when not appropriate. As the training continues, however, the network eventually abstracts the exceptions and special cases and is able to produce intelligible speech with few errors.

The operation of NETtalk is surprisingly simple. Its input is a vector of seven characters (including spaces) from a transcript of text, and its output is phonetic information corresponding to the pronunciation of the center (fourth) character in the seven-character input field. The other six characters provide context which helps determine the desired phoneme. To read text, the seven-character window is scanned across a document in computer memory and the network generates a sequence of phonetic symbols which can be used to con-

trol a speech synthesizer. Each of the seven characters at the network's input is a 29-component binary vector, with each component representing a different alphabetic character or punctuation mark. A "one" is placed in the component associated with the represented character while all other components are set to zero.

The system's 26 outputs correspond to 23 articulatory features and 3 additional features which encode stress and syllable boundaries. When training the network, the desired response vector has zeros in all components except those which correspond to the phonetic features associated with the center character in the input field. In one experiment, Sejnowski and Rosenberg had the system scan a 1024-word transcript of phonetically transcribed continuous speech. With the presentation of each seven-character window, the system's weights were trained by the backpropagation algorithm in response to the network's output error. After roughly 50 presentations of the entire training set, the network was able to produce accurate speech from data the network had not been exposed to during training.

Backpropagation is not the only technique that might be used to train NETtalk. In other experiments, the slower Boltzmann learning method was used, and, in fact, Madaline Rule III could be used as well. Likewise, if the sigmoid network was replaced by a similar signum network, Madaline Rule II would also work, although more first-layer Adalines would likely be needed for comparable performance.

The remainder of this article develops and compares various adaptive algorithms for training Adalines and artificial neural networks to solve classification problems such as NETtalk. These same algorithms can be used to train networks for other problems such as those involving nonlinear control,[52] system identification,[52,83] signal processing,[32] or decision making.[57]

D. Adaptation—The Minimal Disturbance Principle

The iterative algorithms described in this article are all designed in accord with a single underlying principle. These techniques—the two LMS algorithms, Mays's rules, and the Perceptron procedure for training a single Adaline, the MRI rule for training the simple Madaline, as well as MRII, MRIII, and backpropagation techniques for training multi-layer Madalines—all rely upon the principle of minimal disturbance: *Adapt to reduce the output error for the current training pattern, with minimal disturbance to responses already learned.* Unless this principle is practiced, it is difficult to simultaneously store the required pattern responses. The minimal disturbance principle is intuitive. It was the motivating idea that led to the discovery of the LMS algorithm and the Madaline rules. In fact, the LMS algorithm had existed for several months as an error reduction rule before it was discovered that the algorithm uses an instantaneous gradient to follow the path of steepest descent and minimize the mean-square-error of the training set. It was then given the name "LMS" (Least Mean Square) algorithm.

III. ERROR CORRECTION RULES—SINGLE THRESHOLD ELEMENT

As adaptive algorithms evolved, principally two kinds of on-line rules have come to exist. One kind, *error correction rules*, alter the weights of a network to correct a certain proportion of the error in the output response to the present input pattern. The other kind, *gradient rules*, alter the weights of a network during each pattern presentation by gradient descent with the objective of reducing mean-square-error, averaged over all training patterns. Both types of rules invoke similar training procedures. Because they are based upon different objectives, however, they can have significantly different learning characteristics.

Error correction rules, of necessity, often tend to be *ad hoc*. They are most often used when training objectives are not easily quantified, or when a problem does not lend itself to tractable analysis. A common application, for instance, concerns training neural networks that contain discontinuous functions. An exception is the α-LMS algorithm, an error correction rule which has proven to be an extremely useful technique for finding solutions to well-defined and tractable linear problems.

We begin with error correction rules applied initially to single Adaline elements, and then to networks of Adalines.

A. Linear Rules

Linear error correction rules alter the weights of the adaptive threshold element with each pattern presentation to make an error correction which is proportional to the error itself. The one linear rule, α-LMS, is described next.

1. *The α-LMS Algorithm*

The α-LMS algorithm or Widrow–Hoff delta rule applied to the adaptation of a single Adaline (Fig. 2) embodies the *minimal disturbance principle*. The weight update equation for the original form of the algorithm can be written as

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha \frac{\varepsilon_k \mathbf{X}_k}{|\mathbf{X}_k|^2}. \tag{2}$$

The time index or adaptation cycle number is k. $\mathbf{W}_{k+1}$ is the next value of the weight vector, $\mathbf{W}_k$ is the present value of the weight vector, and $\mathbf{X}_k$ is the present input pattern vector. The present linear error ε_k is defined to be the difference between the desired response d_k and the linear output $s_k = \mathbf{W}_k^T \mathbf{X}_k$ before adaptation:

$$\varepsilon_k \triangleq d_k - \mathbf{W}_k^T \mathbf{X}_k. \tag{3}$$

Changing the weights yields a corresponding change in the error:

$$\Delta\varepsilon_k = \Delta(d_k - \mathbf{W}_k^T \mathbf{X}_k) = -\mathbf{X}_k^T \Delta\mathbf{W}_k. \tag{4}$$

In accordance with the α-LMS rule of Eq. (2), the weight change is as follows:

$$\Delta \mathbf{W}_k = \mathbf{W}_{k+1} - \mathbf{W}_k = \alpha \frac{\varepsilon_k \mathbf{X}_k}{|\mathbf{X}_k|^2}. \tag{5}$$

Combining Eqs. (4) and (5), we obtain

$$\Delta \varepsilon_k = -\alpha \frac{\varepsilon_k \mathbf{X}_k^T \mathbf{X}_k}{|\mathbf{X}_k|^2} = -\alpha \varepsilon_k. \tag{6}$$

Therefore, the error is reduced by a factor of α as the weights are changed while holding the input pattern fixed. Presenting a new input pattern starts the next adaptation cycle. The next error is then reduced by a factor of α, and the process continues. The initial weight vector is usually chosen to be zero and is adapted until convergence. In nonstationary environments, the weights are generally adapted continually.

The choice of α controls stability and speed of convergence.[32] For input pattern vectors independent over time, stability is ensured for most practical purposes if

$$0 < \alpha < 2. \tag{7}$$

Making α greater than 1 generally does not make sense, since the error would be over corrected. Total error correction comes with $\alpha = 1$. A practical range for α is

$$0.1 < \alpha < 1.0. \tag{8}$$

This algorithm is self-normalizing in the sense that the choice of α does not depend on the magnitude of the input signals. The weight update is collinear with the input pattern and of a magnitude inversely proportional to $|\mathbf{X}_k|^2$. With binary ± 1 inputs, $|\mathbf{X}_k|^2$ is equal to the number of weights and does not vary from pattern to pattern. If the binary inputs are the usual 1 and 0, no adaptation occurs for weights with 0 inputs, while with ± 1 inputs, all weights are adapted each cycle and convergence tends to be faster. For this reason, the symmetric inputs $+1$ and -1 are generally preferred.

The α-LMS algorithm corrects error and if all input pattern vectors are of equal length, it minimizes mean-square-error.[32] The algorithm is best known for this property.

B. Nonlinear Rules

The α-LMS algorithm is a linear rule which makes error corrections that are proportional to the error. It is known[84] that in some cases this linear rule may fail to separate training patterns that are linearly separable. Where this creates difficulties, nonlinear rules may be used. In the following paragraphs, we describe early nonlinear rules which were devised by Rosenblatt[5,85] and Mays.[84] These nonlinear rules also make weight vector changes collinear with the input pattern vector (the direction which causes minimal disturbance), changes which are based on the linear error but are not directly proportional to it.

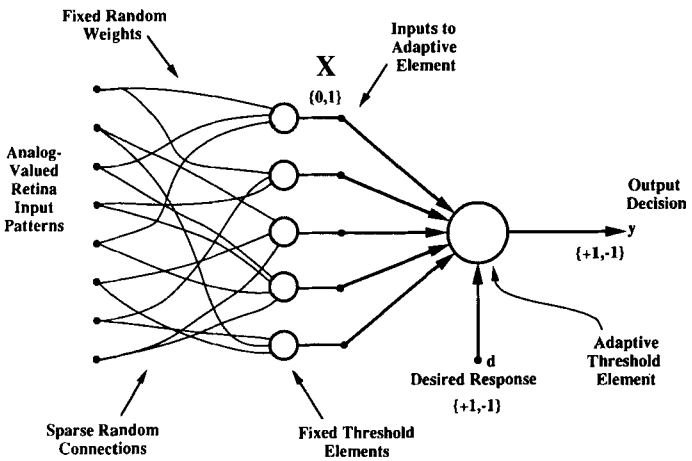

Figure 9. Rosenblatt's α-Perceptron.

1. The Perceptron Learning Rule

The Rosenblatt α-Perceptron,[5,85] diagrammed in Figure 9, processed input patterns with a first layer of sparse, randomly connected, fixed-logic devices. The outputs of the fixed first layer fed a second layer which consisted of a single adaptive linear threshold element. Other than the convention that its input signals were {1,0} binary, and that no bias weight was included, this element is equivalent to the Adaline element. The learning rule for the α-Perceptron is very similar to LMS, but its behavior is in fact quite different.

It is interesting to note that Rosenblatt's Perceptron learning rule was first presented in 1960,[85] and Widrow and Hoff's LMS rule was first presented the same year, a few months later.[61] These rules were developed independently in 1959.

The adaptive threshold element of the α-Perceptron is shown in Figure 10. Adapting with the Perceptron rule makes use of the "quantizer error" $\tilde{\tilde{\varepsilon}}_k$, defined to be the difference between the desired response and the output of the quantizer

$$\tilde{\tilde{\varepsilon}}_k \triangleq d_k - y_k. \tag{9}$$

The Perceptron rule, sometimes called the Perceptron Convergence Procedure, does not adapt the weights if the output decision y_k is correct, i.e., if $\tilde{\tilde{\varepsilon}}_k = 0$. If the output decision disagrees with the binary desired response d_k, however, adaptation is effected by adding the input vector to the weight vector when the error $\tilde{\tilde{\varepsilon}}_k$ is positive, or subtracting the input vector from the weight vector when the error $\tilde{\tilde{\varepsilon}}_k$ is negative. Note that the quantizer error $\tilde{\tilde{\varepsilon}}_k$ is always equal to either $+2$, -2, or 0. Thus, half the product of the input vector and the quantizer error $\tilde{\tilde{\varepsilon}}_k$ is added to the weight vector. The Perceptron rule is identical to the α-LMS algorithm, except that with the Perceptron rule, half of the

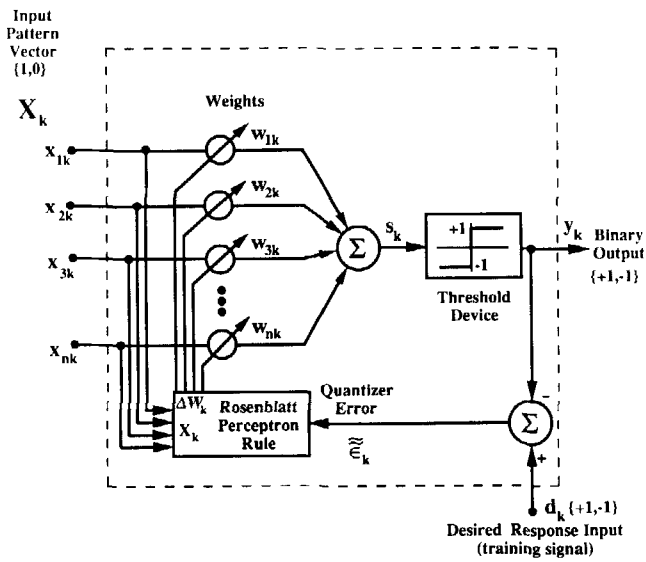

Figure 10. The adaptive threshold element of the Perceptron.

quantizer error, $\tilde{\tilde{\varepsilon}}_k/2$, is used in place of the normalized linear error $\varepsilon_k/|\mathbf{X}_k|^2$ of the α-LMS rule. The Perceptron rule is nonlinear in contrast to the LMS rule which is linear (compare Figs. 2 and 10). Nonetheless, the Perceptron rule can be written in a form which is very similar to the α-LMS rule of Eq. (2):

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha \frac{\tilde{\tilde{\varepsilon}}_k}{2} \mathbf{X}_k. \tag{10}$$

Rosenblatt normally set α to one. In contrast to α-LMS, the choice of α does not affect the stability of the Perceptron algorithm, and it affects convergence time only if the initial weight vector is nonzero. Also, while α-LMS can be used with either analog or binary desired responses, Rosenblatt's rule can be used only with binary desired responses.

The Perceptron rule stops adapting when the training patterns are correctly separated. There is no restraining force controlling the magnitude of the weights, however. The direction of the weight vector, not its magnitude, determines the decision function. The Perceptron rule has been proven to be capable of separating any linearly separable set of training patterns.[5,48,84,86] If the training patterns are not linearly separable, the Perceptron algorithm goes on forever, and often does not yield a low-error solution, even if one exists. In most cases, if the training set is not separable, the weight vector tends to gravitate toward zero* so that even if α is very small, each adaptation can dramatically affect the switching function implemented by the Perceptron.

*This results because the length of the weight vector decreases with each adaptation that does not cause the linear output s_k to change sign and assume a magnitude greater than that before adaptation. Although there are exceptions, for most problems

This behavior is very different from that of the α-LMS algorithm. Continued use of α-LMS does not lead to an unreasonable weight solution if the pattern set is not linearly separable. Nor, however, is this algorithm guaranteed to separate any linearly separable pattern set. α-LMS typically comes close to achieving such separation, but its objective is different, i.e., error reduction at the linear output of the adaptive element.

Rosenblatt also introduced variants of the fixed-increment rule that we have discussed thus far. A popular one was the absolute-correction version of the Perceptron rule.† This rule is identical to that stated in Eq. (10) except that increment size α is chosen with each presentation to be the smallest integer which corrects the output error in one presentation. If the training set is separable, this variant has all the characteristics of the fixed-increment version with α set to 1, except that it usually reaches a solution in fewer presentations.

2. *Mays's Algorithms*

In his Ph.D. thesis,[84] C. H. Mays described an "increment adaptation" rule‡ and a "modified relaxation adaptation" rule. The fixed-increment version of the Perceptron rule is a special case of the increment adaptation rule.

Increment adaptation, in its general form, involves the use of a "dead zone" for the linear output s_k equal to $\pm\gamma$ about zero. All desired responses are ± 1 (refer to Fig. 10). If the linear output s_k falls outside the dead zone ($|s_k| \geq \gamma$), adaptation follows a normalized variant of the fixed-increment Perceptron rule (with $\alpha/|\mathbf{X}_k|^2$ used in place of α). If the linear output falls within the dead zone, whether or not the output response y_k is correct, the weights are adapted by the normalized variant of the Perceptron rule as though the output response y_k had been incorrect. The weight update rule for Mays's increment adaptation algorithm can be written mathematically as

$$\mathbf{W}_{k+1} = \begin{cases} \mathbf{W}_k + \alpha \tilde{\tilde{\varepsilon}}_k \dfrac{\mathbf{X}_k}{2|\mathbf{X}_k|^2} & \text{if } |s_k| \geq \gamma \\ \mathbf{W}_k + \alpha d_k \dfrac{\mathbf{X}_k}{|\mathbf{X}_k|^2} & \text{if } |s_k| < \gamma \end{cases}, \tag{11}$$

where $\tilde{\tilde{\varepsilon}}_k$ is the quantizer error of Eq. (9).

With the dead zone $\gamma = 0$, Mays's increment adaptation reduces to a normalized version of the Perceptron rule (10). Mays proved that if the training patterns are linearly separable, increment adaptation will always converge and separate the patterns in a finite number of steps. He also showed that use of the

---

this situation occurs only rarely if the weight vector is much longer than the weight increment vector.

†The terms *fixed-increment* and *absolute correction* are due to Nilsson.[48] Rosenblatt referred to methods of these types, respectively, as *quantized* and *nonquantized learning rules*.

‡The increment adaptation rule was proposed by others before Mays, though from a different perspective.[86]

dead zone reduces sensitivity to weight errors. If the training set is not linearly separable, Mays's increment adaptation rule typically performs much better than the Perceptron rule because a sufficiently large dead zone tends to cause the weight vector to adapt away from zero when any reasonably good solution exists. In such cases, the weight vector may sometimes appear to meander rather aimlessly, but it will typically remain in a region associated with relatively low average error.

The increment adaptation rule changes the weights with increments that generally are not proportional to the linear error, ε_k. The other Mays rule, modified relaxation, is closer to α-LMS in its use of the linear error ε_k (refer to Fig. 2). The desired response and the quantizer output levels are binary ± 1. If the quantizer output y_k is wrong or if the linear output s_k falls within the dead zone $\pm\gamma$, adaptation follows α-LMS to reduce the linear error. If the quantizer output y_k is correct and the linear output s_k falls outside the dead zone, the weights are not adapted. The weight update rule for this algorithm can be written as

$$\mathbf{W}_{k+1} = \begin{cases} \mathbf{W}_k & \text{if } \tilde{\tilde{\varepsilon}}_k = 0 \text{ and } |s_k| \geq \gamma \\ \mathbf{W}_k + \alpha\varepsilon_k \dfrac{\mathbf{X}_k}{|\mathbf{X}_k|^2} & \text{otherwise} \end{cases}, \tag{12}$$

where, $\tilde{\tilde{\varepsilon}}_k$ is the quantizer error of Eq. (9).

If the dead zone γ is set to ∞, this algorithm reduces to the α-LMS algorithm (2). Mays showed that, for dead zone $0 < \gamma < 1$, and learning rate $0 < \alpha < 2$, this algorithm will converge and separate any linearly separable input set in a finite number of steps. If the training set is not linearly separable, this algorithm performs much like Mays's increment adaptation rule.

Mays's two algorithms achieve similar pattern separation results. The choice of α does not affect stability, although it does affect convergence time. The two rules differ in their convergence properties but there is no consensus on which is the better algorithm. Algorithms like these can be quite useful, and we feel that there are many more to be invented and analyzed.

The α-LMS algorithm, the Perceptron procedure, and Mays's algorithms can all be used for adapting the single Adaline element or they can be incorporated into procedures for adapting networks of such elements. Multi-layer network adaption procedures which use some of these algorithms will be discussed below.

IV. ERROR CORRECTION RULES—MULTI-ELEMENT NETWORKS

The algorithms discussed next are the Widrow–Hoff Madaline rule from the early 1960s, now called Madaline Rule I (MRI), and Madaline Rule II, (MRII) developed by Widrow and Winter in 1987.

A. Madaline Rule I

The MRI rule allows the adaptation of a first layer of hard-limited (signum) Adaline elements whose outputs provide inputs to a second layer, consisting of

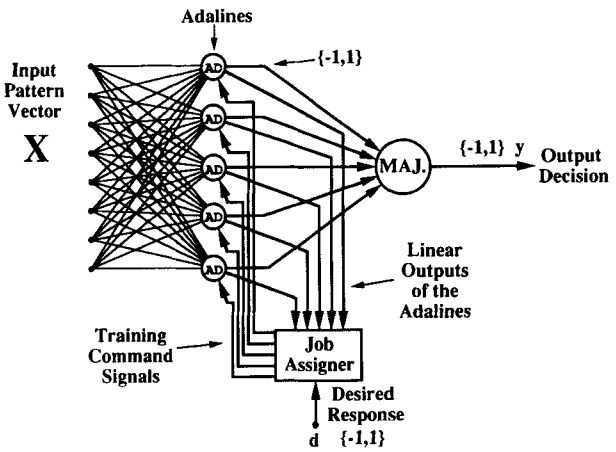

Figure 11. A five-Adaline example of the Madaline I architecture.

a single fixed threshold logic element which may be, for example, the OR gate, AND gate, or Majority Vote Taker discussed previously. The weights of the Adalines are initially set to small random values.

Figure 11 shows a Madaline I architecture with five fully connected first-layer Adalines. The second layer is a Majority element (MAJ). Because the second-layer logic element is fixed and known, it is possible to determine which first-layer Adalines can be adapted to correct an output error. The Adalines in the first layer assist each other in solving problems by automatic load-sharing.

One procedure for training the network in Fig. 11 follows. A pattern is presented, and if the output response of the Majority element matches the desired response, no adaptation takes place. However, if, for instance, the desired response is $+1$ and three of the five Adalines read -1 for a given input pattern, one of the latter three must be adapted to the $+1$ state. The element that is adapted by MRI is the one whose linear output s_k is closest to zero—i.e., the one whose analog response is closest to the desired response. If more of the Adalines were originally in the -1 state, enough of them are adapted to the $+1$ state to make the majority decision equal $+1$. The elements adapted are those whose linear outputs are closest to zero. A similar procedure is followed when the desired response is -1. When adapting a given element, the weight vector can be moved in the LMS direction far enough to reverse the Adaline's output (absolute correction or "fast" learning), or it can be adapted by the small increment determined by the α-LMS algorithm (statistical or "slow" learning). The one desired response, d_k, is used for all Adalines that are adapted. The procedure can also be modified to allow one of Mays's rules to be used. In that event, for the case we have considered (Majority output element), adaptations take place if at least half of the Adalines either have outputs which differ from the desired response or have analog outputs which are in the dead zone. By setting the dead zone of Mays's increment adaptation rule to zero, the weights can also be adapted by Rosenblatt's Perceptron rule.

Differences in initial conditions and the results of subsequent adaptation cause the various elements to take "responsibility" for certain parts of the training problem. The basic principle of load sharing is summarized thus: *Assign responsibility to the Adaline or Adalines that can most easily assume it.*

In Figure 11, the "job assigner," a purely mechanized process, assigns responsibility during training by transferring the appropriate adapt commands and desired response signals to the selected Adalines. The job assigner utilizes linear-output information. Load sharing is important, since it results in the various adaptive elements developing individual weight vectors. If all the weight vectors were the same, there would be no point in having more than one element in the first layer.

When training the Madaline, the pattern presentation sequence should be random. Experimenting with this, Ridgway[75] found that cyclic presentation of the patterns could lead to cycles of adaptation. These cycles would cause the weights of the entire Madaline to cycle, preventing convergence.

The MRI rule obeys the "minimal disturbance principle" in the following sense. No more Adaline elements are adapted than necessary to correct the output decision and any dead-zone constraint. The elements whose linear outputs are nearest to zero are adapted because they require the smallest weight changes to reverse their output responses. Furthermore, whenever an Adaline is adapted, the weights are changed in the direction of its input vector, providing the requisite error correction with minimal weight change.

B. Madaline Rule II

The MRI rule was recently extended to allow the adaptation of multi-layer binary networks by Winter and Widrow with the introduction of Madaline Rule II (MRII).[45,87,88] A typical two-layer MRII network is shown in Figure 12. The weights in both layers are adaptive.

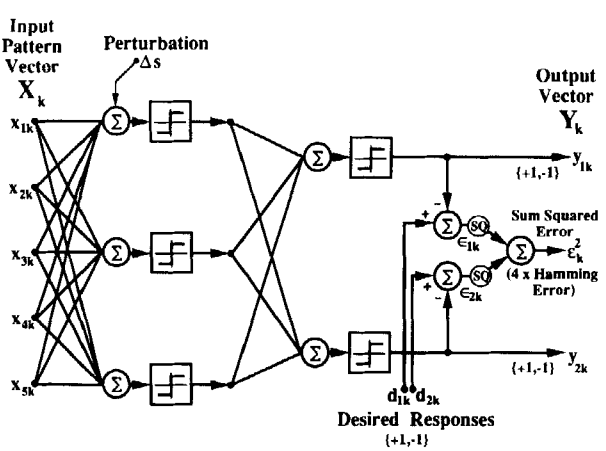

Figure 12. Typical two-layer Madaline II architecture.

Training with the MRII rule is similar to training with the MRI algorithm. The weights are initially set to small random values. Training patterns are presented in a random sequence. If the network produces an error during a training presentation, we begin by adapting first-layer Adalines. By the "minimal disturbance principle," we select the first-layer Adaline with the smallest linear output magnitude and perform a "trial adaptation" by inverting its binary output. This can be done without adaptation by adding a perturbation Δs of suitable amplitude and polarity to the Adaline's sum (refer to Fig. 12). If the output Hamming error is reduced by this bit inversion, i.e., if the number of output errors is reduced, the perturbation Δs is removed and the weights of the selected Adaline element are changed by α-LMS in a direction collinear with the corresponding input vector—the direction which reinforces the bit reversal with minimal disturbance to the weights. Conversely, if the trial adaptation does not improve the network response, no weight adaptation is performed.

After finishing with the first element, we perturb and update other Adalines in the first layer which have "sufficiently small" linear-output magnitudes. Further error reductions can be achieved, if desired, by reversing pairs, triples, etc., up to some predetermined limit. After exhausting possibilities with the first layer, we move on to the next layer and proceed in a like manner. When the final layer is reached, each of the output elements is adapted by α-LMS. At this point, a new training pattern is selected at random and the procedure is repeated. The goal is to reduce Hamming error with each presentation, thereby hopefully minimizing the average Hamming error over the training set. Like MRI, the procedure can be modified so that adaptations follow an absolute correction rule or one of Mays's rules rather than α-LMS. Like MRI, MRII can "hang-up" on local optima.

V. STEEPEST-DESCENT RULES—SINGLE THRESHOLD ELEMENT

Thus far, we have described a variety of adaptation rules that act to reduce a given proportion of the error with the presentation of each training pattern. Often, the objective of adaptation is to reduce error averaged in some way over the training set. The most common error function is mean-square-error (MSE), although in some situations other error criteria may be more appropriate.[89–91] The most popular approaches to mean-square-error reduction in both single-element and multi-element networks are based upon the method of steepest descent. More sophisticated gradient approaches such as quasi-Newton[32,92–94] and conjugate gradient[94,95] techniques often have better convergence properties, but the conditions under which the additional complexity is warranted are not generally known. The discussion that follows is restricted to minimization of MSE by the method of steepest descent.[96,97] More sophisticated learning procedures usually require many of the same computations used in the basic steepest-descent procedure.

Adaptation of a network by steepest-descent starts with an arbitrary initial value $\mathbf{W}_0$ for the system's weight vector. The gradient of the mean-square-error function is measured and the weight vector is altered in the direction corre-

sponding to the negative of the measured gradient. This procedure is repeated, causing the MSE to be successively reduced on average and causing the weight vector to approach a locally optimal value.

The method of steepest descent can be described by the relation

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\nabla_k), \tag{13}$$

where μ is a parameter that controls stability and rate of convergence, and ∇_k is the value of the gradient at a point on the MSE surface corresponding to $\mathbf{W} = \mathbf{W}_k$.

To begin, we derive rules for steepest-descent minimization of the MSE associated with a single Adaline element. These rules are then generalized to apply to full-blown neural networks. Like error correction rules, the most practical and efficient steepest-descent rules typically work with one pattern at a time. They minimize mean-square-error, approximately, averaged over the entire set of training patterns.

A. Linear Rules

Steepest-descent rules for the single threshold element are said to be linear if weight changes are proportional to the linear error, the difference between the desired response d_k and the linear output of the element, s_k.

1. Mean-Square-Error Surface of the Linear Combiner

In this section we demonstrate that the MSE surface of the linear combiner of Figure 1 is a quadratic function of the weights, and is thus easily traversed by gradient descent.

Let the input pattern $\mathbf{X}_k$ and the associated desired response d_k be drawn from a statistically stationary population. During adaptation, the weight vector varies so that even with stationary inputs, the output s_k and error ε_k will generally be nonstationary. Care must be taken in defining the mean-square-error since it is time-varying. The only possibility is an ensemble average, defined below.

At the kth iteration, let the weight vector be $\mathbf{W}_k$. Squaring and expanding Eq. (3) yields

$$\varepsilon_k^2 = (d_k - \mathbf{X}_k^T\mathbf{W}_k)^2 \tag{14}$$

$$= d_k^2 - 2d_k\mathbf{X}_k^T\mathbf{W}_k + \mathbf{W}_k^T\mathbf{X}_k\mathbf{X}_k^T\mathbf{W}_k. \tag{15}$$

Now assume an ensemble of identical adaptive linear combiners, each having the same weight vector $\mathbf{W}_k$ at the kth iteration. Let each combiner have individual inputs $\mathbf{X}_k$ and d_k derived from stationary ergodic ensembles. Each combiner will produce an individual error ε_k represented by Eq. (15). Averaging Eq. (15) over the ensemble yields

$$E[\varepsilon_k^2]_{\mathbf{W}=\mathbf{W}_k} = E[d_k^2] - 2E[d_k\mathbf{X}_k^T]\,\mathbf{W}_k + \mathbf{W}_k^T E[\mathbf{X}_k\mathbf{X}_k^T]\mathbf{W}_k. \tag{16}$$

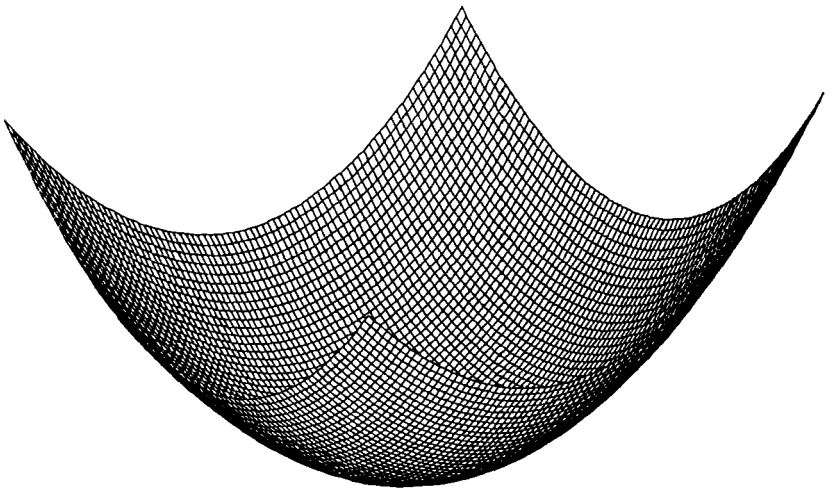

Figure 13. Typical mean-square-error surface of a linear combiner.

Defining the vector **P** as the cross correlation between the desired response (a scalar) and the **X**-vector* then yields

$$\mathbf{P}^T \triangleq E[d_k \mathbf{X}_k^T] = E[d_k, d_k x_{1k}, \ldots d_k x_{nk}]^T. \tag{17}$$

The input correlation matrix **R** is defined in terms of the ensemble average

$$\begin{aligned} \mathbf{R} &\triangleq E[\mathbf{X}_k \mathbf{X}_k^T] \\ &= E\begin{bmatrix} 1 & x_{1k} & \ldots & x_{nk} \\ x_{1k} & x_{1k}x_{1k} & \ldots & x_{1k}x_{nk} \\ \vdots & \vdots & & \vdots \\ x_{nk} & x_{nk}x_{1k} & \ldots & x_{nk}x_{nk} \end{bmatrix}. \end{aligned} \tag{18}$$

This matrix is real, symmetric, and positive definite, or in rare cases, positive semi-definite. The mean-square-error ξ_k can thus be expressed as

$$\begin{aligned} \xi_k &\triangleq E[\varepsilon_k^2]_{\mathbf{W}=\mathbf{W}_k} \\ &= E[d_k^2] - 2\mathbf{P}^T\mathbf{W}_k + \mathbf{W}_k^T\mathbf{R}\mathbf{W}_k. \end{aligned} \tag{19}$$

Note that the mean-square-error is a quadratic function of the weights. It is a convex hyperparaboloidal surface, a function that never goes negative. Figure 13 shows a typical mean-square-error surface for a linear combiner with two weights. The position of a point on the grid in this figure represents the value of the Adaline's two weights. The height of the surface at each point represents the mean-square-error over the training set when the Adaline's weights are

*We assume here that **X** includes a bias component $x_{0k} = +1$.

fixed at the values associated with the grid point. Adjusting the weights involves descending along this surface toward the unique minimum point (''the bottom of the bowl'') by the method of steepest descent.

The gradient ∇_k of the mean-square-error function with $\mathbf{W} = \mathbf{W}_k$ is obtained by differentiating Eq. (19):

$$\nabla_k \triangleq \left\{ \begin{array}{c} \dfrac{\partial E[\varepsilon_k^2]}{\partial w_{0k}} \\ \vdots \\ \dfrac{\partial E[\varepsilon_k^2]}{\partial w_{nk}} \end{array} \right\}_{\mathbf{W}=\mathbf{W}_k} = -2\mathbf{P} + 2\mathbf{R}\mathbf{W}_k. \tag{20}$$

This is a linear function of the weights. The optimal weight vector $\mathbf{W}^*$, generally called the Wiener weight vector, is obtained from Eq. (20) by setting the gradient to zero:

$$\mathbf{W}^* = \mathbf{R}^{-1}\mathbf{P}. \tag{21}$$

This is a matrix form of the Wiener–Hopf equation.[98–100] In the next section we examine μ-LMS, an algorithm which enables us to obtain an accurate estimate of $\mathbf{W}^*$ without first computing $\mathbf{R}^{-1}$ and $\mathbf{P}$.

2. *The μ-LMS Algorithm*

The μ-LMS algorithm works by performing approximate steepest descent on the mean-square-error surface in weight space. Because it is a quadratic function of the weights, this surface is convex and has a unique (global) minimum.* An instantaneous gradient based upon the square of the instantaneous linear error is

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k} = \left\{ \begin{array}{c} \dfrac{\partial \varepsilon_k^2}{\partial w_{0k}} \\ \vdots \\ \dfrac{\varepsilon_k^2}{\partial w_{nk}} \end{array} \right\}. \tag{22}$$

LMS works by using this crude gradient estimate in place of the true gradient ∇_k of Eq. (20). Making this replacement into Eq. (13) yields

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k - \mu \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k}. \tag{23}$$

The instantaneous gradient is used because it is readily available from a single data sample. The true gradient is generally difficult to obtain. Computing it

*Unless the autocorrelation matrix of the pattern vector set has m zero eigenvalues, in which case the minimum MSE solution will be an m dimensional subspace in weight space.[32]

would involve averaging the instantaneous gradients associated with all patterns in the training set. This is usually impractical and almost always inefficient.

Performing the differentiation in Eq. (23) and replacing the linear error by definition (3) gives

$$\begin{aligned} \mathbf{W}_{k+1} &= \mathbf{W}_k - 2\mu\varepsilon_k \frac{\partial \varepsilon_k}{\partial \mathbf{W}_k} \\ &= \mathbf{W}_k - 2\mu\varepsilon_k \frac{\partial (d_k - \mathbf{W}_k^T \mathbf{X}_k)}{\partial \mathbf{W}_k}. \end{aligned} \tag{24}$$

Noting that d_k and $\mathbf{X}_k$ are independent of $\mathbf{W}_k$, yields

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\varepsilon_k \mathbf{X}_k. \tag{25}$$

This is the μ-LMS algorithm. The learning constant μ determines stability and convergence rate. For input patterns independent over time, convergence of the mean and variance of the weight vector is ensured,[32] for most practical purposes if

$$0 < \mu < \frac{1}{trace[\mathbf{R}]}, \tag{26}$$

where $trace[\mathbf{R}] = \Sigma$(diagonal element of $\mathbf{R}$) is the average signal power of the $\mathbf{X}$-vectors, i.e., $E(\mathbf{X}^T\mathbf{X})$. With μ set within this range,* the μ-LMS algorithm converges in the mean to $\mathbf{W}^*$, the optimal Wiener solution discussed above. A proof of this can be found in Ref. 32.

In the μ-LMS algorithm, use of the instantaneous gradients is perfectly justified if the step size is small. For small μ, $\mathbf{W}$ will remain essentially constant over a relatively small number of training presentations, K. The total weight change during this period will be proportional to

$$\begin{aligned} -\sum_{l=0}^{K-1} \frac{\partial \varepsilon_{k+l}^2}{\partial \mathbf{W}_{k+l}} &\simeq -K\left(\frac{1}{K}\sum_{l=0}^{K-1} \frac{\partial \varepsilon_{k+l}^2}{\partial \mathbf{W}_k}\right) \\ &= -K \frac{\partial}{\partial \mathbf{W}_k}\left(\frac{1}{K}\sum_{l=0}^{K-1} \varepsilon_{k+l}^2\right) \\ &\simeq -K \frac{\partial \xi}{\partial \mathbf{W}_k}, \end{aligned} \tag{27}$$

where ξ denotes the mean-square-error function. Thus, on average the weights follow the true gradient. It is shown in Ref. 32 that the instantaneous gradient is an unbiased estimate of the true gradient.

*Horowitz and Senne[101] have proven that Equation (26) is not sufficient in general to ensure convergence of the weight vector's variance. For input patterns generated by a zero-mean Gaussian process independent over time, instability can occur in the worst case if μ is greater than 1/(3 trace [$\mathbf{R}$]).

3. *Comparison of μ-LMS and α-LMS*

We have now presented two forms of the LMS algorithm, μ-LMS (25) above, and α-LMS (2) in Section III–A. They are very similar algorithms, both using the LMS instantaneous gradient. α-LMS is self-normalizing, with the parameter α determining the fraction of the instantaneous error to be corrected with each adaptation. μ-LMS is a constant-coefficient linear algorithm which is considerably easier to analyze than α-LMS. Comparing the two, the α-LMS algorithm is like the μ-LMS algorithm with a continually variable learning constant. Although α-LMS is somewhat more difficult to implement and analyze, it has been demonstrated experimentally to be a better algorithm than μ-LMS when the eigenvalues of the input autocorrelation matrix, **R**, are highly disparate, giving faster convergence for a given level of gradient noise* propagated into the weights. It will be shown next that μ-LMS has the advantage that it will always converge in the mean to the minimum mean-square-error solution, while α-LMS may converge to a somewhat biased solution.

We began with α-LMS of Eq. (2):

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha \frac{\varepsilon_k \mathbf{X}_k}{|\mathbf{X}_k|^2} \tag{28}$$

Replacing the error with its definition (3) and rearranging terms yields

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha \frac{(d_k - \mathbf{W}_k^T \mathbf{X}_k)\mathbf{X}_k}{|\mathbf{X}_k|^2} \tag{29}$$

$$= \mathbf{W}_k + \alpha \left(\frac{d_k}{|\mathbf{X}_k|} - \mathbf{W}_k^T \frac{\mathbf{X}_k}{|\mathbf{X}_k|} \right) \frac{\mathbf{X}_k}{|\mathbf{X}_k|}. \tag{30}$$

We define a new training set of pattern vectors and desired responses, $\{\tilde{\mathbf{X}}_k, \tilde{d}_k\}$, by normalizing elements of the original training set as follows,†

$$\tilde{\mathbf{X}}_k \triangleq \frac{\mathbf{X}_k}{|\mathbf{X}_k|} \tag{31}$$

$$\tilde{d}_k \triangleq \frac{d_k}{|\mathbf{X}_k|}.$$

Eq. (30) then becomes

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha(\tilde{d}_k - \mathbf{W}_k^T \tilde{\mathbf{X}}_k)\tilde{\mathbf{X}}_k. \tag{32}$$

This is the μ-LMS rule of Eq. (25) with 2μ replaced by α. The weight adaptations chosen by the α-LMS rule are equivalent to those of the μ-LMS algorithm presented with a different training set—the normalized training set defined by (31). The solution that will be reached by the μ-LMS algorithm is the Wiener solution of this training set,

*Gradient noise is the difference between the gradient estimate and the true gradient.

†The idea of a normalized training set was suggested by Derrick Nguyen.

$$\tilde{\mathbf{W}}^* = (\tilde{\mathbf{R}})^{-1}\,\tilde{\mathbf{P}}, \tag{33}$$

where

$$\tilde{\mathbf{R}} = E[\tilde{\mathbf{X}}_k \tilde{\mathbf{X}}_k^T] \tag{34}$$

is the input correlation matrix of the normalized training set and the vector

$$\tilde{\mathbf{P}} = E[\tilde{d}_k \tilde{\mathbf{X}}_k] \tag{35}$$

is the cross correlation between the normalized input and the normalized desired response. Therefore α-LMS converges in the mean to the Wiener solution of the normalized training set. When the input vectors are binary with ± 1 components, all input vectors have the same magnitude and the two algorithms are equivalent. For nonbinary training patterns, however, the Wiener solution of the normalized training set generally is no longer equal to that of the original problem, so α-LMS converges in the mean to a somewhat biased version of the optimal least squares solution.

The idea of a normalized training set can also be used to relate the stable ranges for the learning constants α and μ in the two algorithms. The stable range for α in the α-LMS algorithm given in Eq. (7) can be computed from the corresponding range for μ given in Eq. (26) by replacing $\mathbf{R}$ and μ in Eq. (26) by $\tilde{\mathbf{R}}$ and $\alpha/2$, respectively, and then noting that $trace[\tilde{\mathbf{R}}]$ is equal to one:

$$\begin{aligned} &0 < \alpha < \frac{2}{trace[\tilde{R}]}, \quad \text{or} \\ &0 < \alpha < 2. \end{aligned} \tag{36}$$

B. Nonlinear Rules

The Adaline elements considered thus far use at their outputs either hard-limiting quantizers (signums), or no nonlinearity at all. The input–output mapping of the hard-limiting quantizer is $y_k = \text{sgn}(s_k)$. Other forms of nonlinearity have come into use in the past two decades, primarily of the sigmoid type. These nonlinearities provide saturation for decision making, yet they have differentiable input–output characteristics that facilitate adaptivity. We generalize the definition of the Adaline element to include the possible use of a sigmoid in place of the signum, and then determine suitable adaptation algorithms.

Figure 14 shows a "Sigmoid Adaline" element which incorporates a sigmoidal nonlinearity. The input–output relation of the sigmoid can be denoted by $y_k = \text{sgm}(s_k)$. A typical sigmoid function is the hyperbolic tangent:

$$y_k = \tanh(s_k) = \left(\frac{1 - e^{-2s_k}}{1 + e^{-2s_k}}\right). \tag{37}$$

We shall adapt this Adaline with the objective of minimizing the mean square of the sigmoid error $\tilde{\varepsilon}_k$, defined as

$$\tilde{\varepsilon}_k \triangleq d_k - y_k = d_k - \text{sgm}(s_k). \tag{38}$$

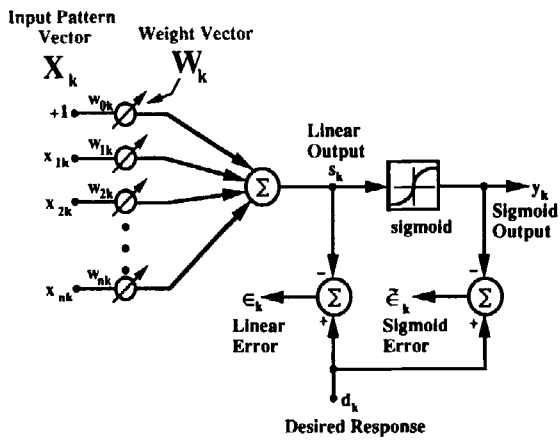

Figure 14. Adaline with sigmoidal nonlinearity.

1. Backpropagation for the Sigmoid Adaline

Our objective is to minimize $E[(\tilde{\varepsilon}_k)^2]$, averaged over the set of training patterns, by proper choice of the weight vector. To accomplish this, we shall derive a backpropagation algorithm for the Sigmoid Adaline element. An instantaneous gradient is obtained with each input vector presentation, and the method of steepest descent is used to minimize error as was done with the μ-LMS algorithm of Eq. (25).

Referring to Figure 14, the instantaneous gradient estimate obtained during presentation of the kth input vector $\mathbf{X}_k$ is given by

$$\hat{\nabla}_k = \frac{\partial(\tilde{\varepsilon}_k)^2}{\partial \mathbf{W}_k} = 2\tilde{\varepsilon}_k \frac{\partial \tilde{\varepsilon}_k}{\partial \mathbf{W}_k}. \tag{39}$$

Differentiating Eq. (38) yields

$$\frac{\partial \tilde{\varepsilon}_k}{\partial \mathbf{W}_k} = -\frac{\partial \mathrm{sgm}(s_k)}{\partial \mathbf{W}_k} = -\mathrm{sgm}'(s_k) \frac{\partial s_k}{\partial \mathbf{W}_k}. \tag{40}$$

We may note that

$$s_k = \mathbf{X}_k^T \mathbf{W}_k. \tag{41}$$

Therefore,

$$\frac{\partial s_k}{\partial \mathbf{W}_k} = \mathbf{X}_k. \tag{42}$$

Substituting into Eq. (40) gives

$$\frac{\partial \tilde{\varepsilon}_k}{\partial \mathbf{W}_k} = -\mathrm{sgm}'(s_k)\mathbf{X}_k. \tag{43}$$

Inserting this into Eq. (39) yields

$$\hat{\nabla}_k = -2\tilde{\varepsilon}_k \text{sgm}'(s_k)\mathbf{X}_k. \tag{44}$$

Using this gradient estimate with the method of steepest descent provides a means for minimizing the mean-square-error even after the summed signal s_k goes through the nonlinear sigmoid. The algorithm is

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) \tag{45}$$

$$= \mathbf{W}_k + 2\mu\tilde{\varepsilon}_k \text{sgm}'(s_k)\mathbf{X}_k. \tag{46}$$

Algorithm (46) is the backpropagation algorithm for the Sigmoid Adaline element. The backpropagation name makes more sense when the algorithm is utilized in a layered network, which will be studied below. Implementation of algorithm (46) is illustrated in Figure 15.

If the sigmoid is chosen to be the hyperbolic tangent function (37), then the derivative $\text{sgm}'(s_k)$ is given by

$$\text{sgm}'(s_k) = \frac{\partial(\tanh(s_k))}{\partial s_k} = 1 - (\tanh(s_k))^2 = 1 - y_k^2. \tag{47}$$

Accordingly Eq. (46) becomes

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\tilde{\varepsilon}_k(1 - y_k^2)\mathbf{X}_k. \tag{48}$$

2. *Madaline Rule III for the Sigmoid Adaline*

The implementation of algorithm (46), illustrated in Figure 15, requires accurate realization of the sigmoid function and its derivative function. These functions may not be realized accurately when implemented with analog hardware. Indeed, in an analog network, each Adaline will have its own individual

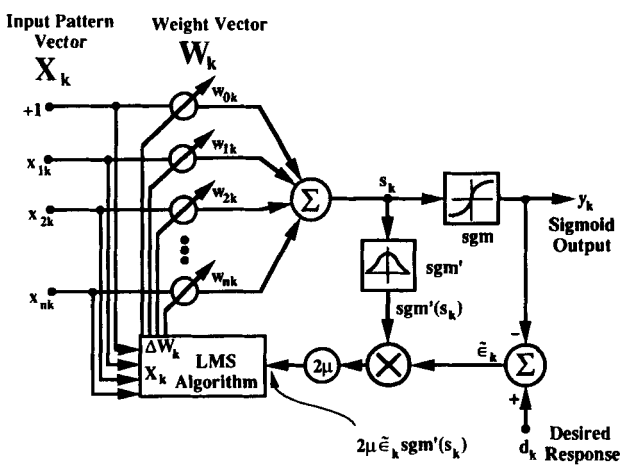

Figure 15. Implementation of backpropagation for the Sigmoid Adaline element.

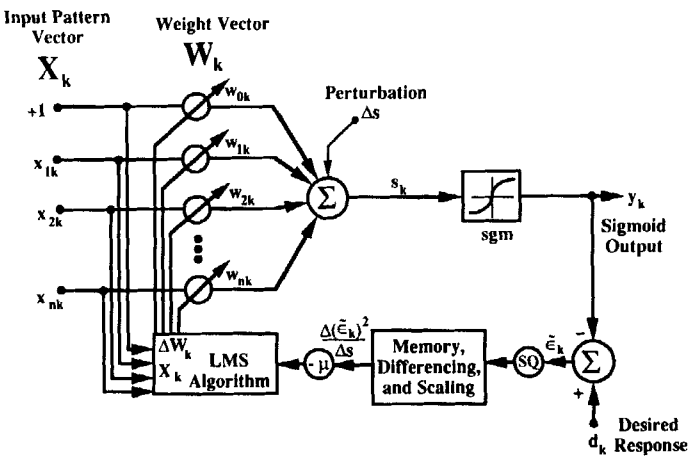

Figure 16. Implementation of the MRIII algorithm for the sigmoid Adaline element.

nonlinearities. Difficulties in adaptation have been encountered in practice with the backpropagation algorithm because of imperfections in the nonlinear functions.

To circumvent these problems, a new algorithm has been devised by David Andes for adapting networks of Sigmoid Adalines. This is the Madaline Rule III (MRIII) algorithm.

The idea of MRIII for a Sigmoid Adaline is illustrated in Figure 16. The derivative of the sigmoid function is not used here. Instead, a small perturbation signal Δs is added to the sum s_k, and the effect of this perturbation upon output y_k and error $\tilde{\varepsilon}_k$ is noted.

An instantaneous estimated gradient can be obtained as follows:

$$\hat{\nabla}_k = \frac{\partial(\tilde{\varepsilon}_k)^2}{\partial \mathbf{W}_k} = \frac{\partial(\tilde{\varepsilon}_k)^2}{\partial s_k}\frac{\partial s_k}{\partial \mathbf{W}_k} = \frac{\partial(\tilde{\varepsilon}_k)^2}{\partial s_k}\mathbf{X}_k. \tag{49}$$

Since Δs is small,

$$\hat{\nabla}_k \approx \left(\frac{\Delta(\tilde{\varepsilon}_k)^2}{\Delta s}\right)\mathbf{X}_k. \tag{50}$$

Another way to obtain an approximate instantaneous gradient by measuring the effects of the perturbation Δs can be obtained from Eq. (49).

$$\hat{\nabla}_k = \frac{\partial(\tilde{\varepsilon}_k)^2}{\partial s_k}\mathbf{X}_k = 2\tilde{\varepsilon}_k\frac{\partial \tilde{\varepsilon}_k}{\partial s_k}\mathbf{X}_k \approx 2\tilde{\varepsilon}_k\left(\frac{\Delta \tilde{\varepsilon}_k}{\Delta s}\right)\mathbf{X}_k. \tag{51}$$

Accordingly, there are two forms of the MRIII algorithm for the Sigmoid Adaline. They are based on the method of steepest descent, using the estimated instantaneous gradients:

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \mu\left(\frac{\Delta(\tilde{\varepsilon}_k)^2}{\Delta s}\right)\mathbf{X}_k \tag{52}$$

or,

$$\mathbf{W}_{k+1} = \mathbf{W}_k - 2\mu\tilde{\varepsilon}_k \left(\frac{\Delta(\tilde{\varepsilon}_k)^2}{\Delta s}\right) \mathbf{X}_k. \tag{53}$$

For small perturbations, these two forms are essentially identical. Neither one requires *a priori* knowledge of the sigmoid's derivative, and both are robust with respect to natural variations, biases, and drift in the analog hardware. Which form to use is a matter of implementational convenience. The algorithm of Eq. (52) is illustrated in Figure 16.

Regarding algorithm (53), some changes can be made to establish a point of interest. Note that, in accord with Eq. (38),

$$\tilde{\varepsilon}_k = d_k - y_k. \tag{54}$$

Adding the perturbation Δs causes a change in ε_k equal to

$$\Delta\tilde{\varepsilon}_k = -\Delta y_k. \tag{55}$$

Now, Eq. (53) may be rewritten as

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\tilde{\varepsilon}_k \left(\frac{\Delta y_k}{\Delta s}\right) \mathbf{X}_k. \tag{56}$$

Since Δs is small, the ratio of increments may be replaced by a ratio of differentials finally giving

$$\mathbf{W}_{k+1} \simeq \mathbf{W}_k + 2\mu\tilde{\varepsilon}_k \frac{\partial y_k}{\partial s_k} \mathbf{X}_k \tag{57}$$

$$= \mathbf{W}_k + 2\mu\tilde{\varepsilon}_k \text{sgm}'(s_k)\mathbf{X}_k. \tag{58}$$

This is identical to the backpropagation algorithm (46) for the Sigmoid Adaline. Thus, backpropagation and MRIII are mathematically equivalent if the perturbation Δs is small, but MRIII is robust, even with analog implementations.

VI. STEEPEST-DESCENT RULES—MULTI-ELEMENT NETWORKS

We now study rules for steepest-descent minimization of the MSE associated with entire networks of Sigmoid Adaline elements. Like their single-element counterparts, the most practical and efficient steepest-descent rules for multi-element networks typically work with one pattern presentation at a time. We will describe two steepest-descent rules for multi-element sigmoid networks: backpropagation and Madaline Rule III.

A. Backpropagation for Networks

The publication of the backpropagation technique by Rumelhart *et al.*[44] has unquestionably been the most influential development in the field of neural networks during the past decade. In retrospect, the technique seems simple. Nonetheless, largely because early neural network research dealt almost exclu-

sively with hard-limiting nonlinearities, the idea never occurred to neural network researchers throughout the 1960s.

The basic concepts of backpropagation are easily grasped. Unfortunately, these simple ideas are often obscured by relatively intricate notation, so formal derivations of the backpropagation rule are often tedious. We instead present a brief outline of the algorithm and illustrate how it works for the simple network shown in Figure 17.

The backpropagation technique is a nontrivial generalization of the single Sigmoid Adaline case of Section V–B. When applied to multi-element networks, the backpropagation technique adjusts the weights in the direction opposite the instantaneous error gradient:

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k} = \begin{Bmatrix} \frac{\partial \varepsilon_k^2}{\partial w_{1k}} \\ \vdots \\ \frac{\partial \varepsilon_k^2}{\partial w_{nk}} \end{Bmatrix}. \tag{59}$$

Now, however, $\mathbf{W}_k$ is a long n-component vector of all weights in the entire network. The instantaneous sum squared error ε_k^2 is the sum of the squares of the errors at each of the network's outputs. Thus for a network with an N_y-component output vector,

$$\varepsilon_k^2 = \sum_{i=1}^{N_y} \varepsilon_{ik}^2. \tag{60}$$

In the network example shown in Figure 17, the sum square error is given by

$$\varepsilon^2 = (d_1 - y_1)^2 + (d_2 - y_2)^2, \tag{61}$$

where we now suppress the time index k for convenience.

In its simplest form, backpropagation training begins by presenting an input pattern vector $\mathbf{X}$ to the network, sweeping forward through the system to generate an output response vector $\mathbf{Y}$, and computing the errors at each output. The next step involves sweeping the effects of the errors backward through the network to associate a "square error derivative" δ with each Adaline, computing a gradient from each δ, and finally updating the weights of each Adaline based upon the corresponding gradient. A new pattern is then presented and the process is repeated. The initial weight values are normally set to small random numbers. The algorithm will not work properly with multilayer networks if the initial weights are either zero or poorly chosen nonzero values.*

*Recently, Nguyen has discovered that a more sophisticated choice of initial weight values in hidden layers can lead to reduced problems with local optima and dramatic increases in network training speed.[102] Experimental evidence suggests that it is advisable to choose the initial weights of each hidden layer in a quasi-random manner which ensures that at each position in a layer's input space the outputs of all but a few of its Adalines will be saturated, while ensuring that each Adaline in the layer is unsaturated in some region of its input space. When this method is used, the weights in the output layer are set to small random values.

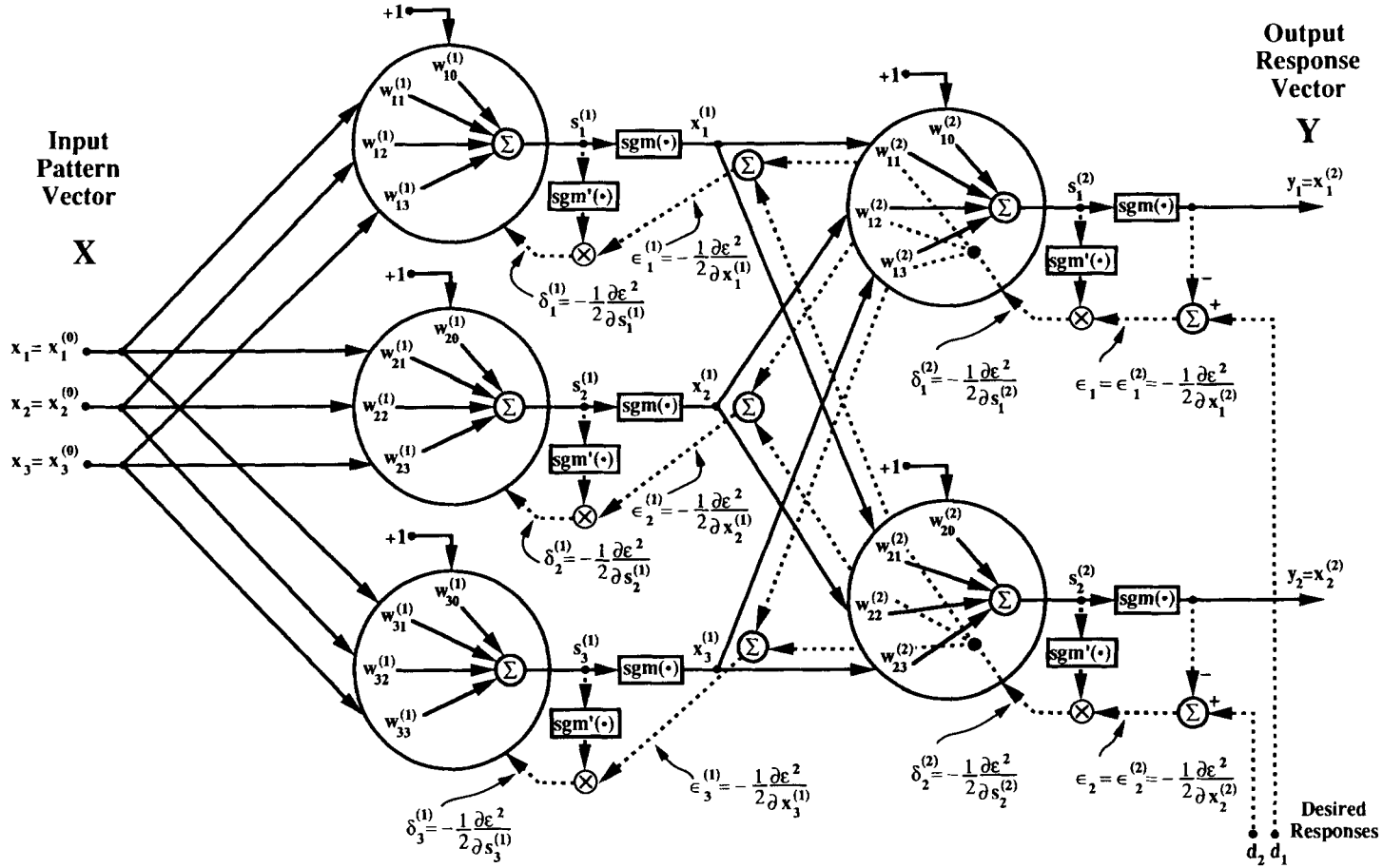

Figure 17. Example of two-layer backpropagation network architecture.

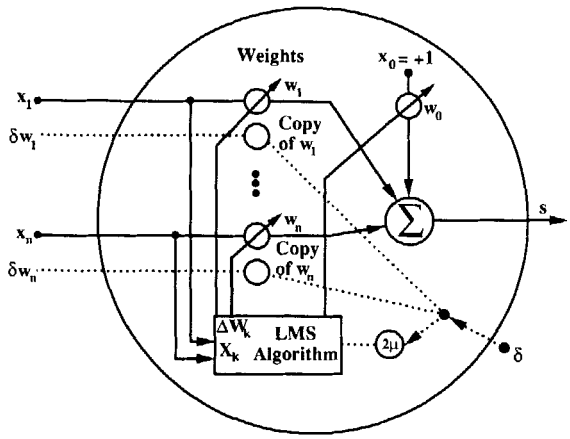

Figure 18. Detail of linear combiner and associated circuitry in backpropagation network.

Furthermore, the order in which training patterns are presented can affect stability of the algorithm. It's usually safest to present patterns in random order, if possible.

We can get some idea about what is involved in the calculations associated with the backpropagation algorithm by examining the network of Figure 17. Each of the five large circles represents a linear combiner, as well as some associated signal paths for error backpropagation, and the corresponding adaptive machinery for updating the weights. This detail is shown in Figure 18. The solid lines in these diagrams represent forward signal paths through the network, and the dotted lines represent the separate backward paths that are used in association with calculations of the square error derivatives δ. From Figure 17, we see that the calculations associated with the backward sweep are of a complexity which is roughly equal to that represented by the forward pass through the network. The backward sweep requires the same number of function calculations as the forward sweep, but fewer weight multiplications.

As stated above, after a pattern has been presented to the network, and the response error of each output has been calculated, the next step of the backpropagation algorithm involves finding the instantaneous square error derivative δ associated with each summing junction in the network. The square error derivative associated with the jth Adaline in layer l is defined as*

$$\delta_j^{(l)} \triangleq -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_j^{(l)}}. \tag{62}$$

*In Figure 17, all notation follows the convention that superscripts within parentheses indicate the layer number of the associated Adaline or input node, while subscripts identify the associated Adaline(s) within a layer.

Each of these derivatives in essence tells us how sensitive the sum square output error of the network is to changes in the linear output of the associated Adaline element.

The derivation of δ for hidden layer elements is not difficult and can be found in many publications.[49,103] For our purposes, however, the easiest way to find values of δ for these units is to follow the schematic diagram of Figure 17. For instance, we find from this picture that $\delta_1^{(2)}$, the value δ corresponding to the first Adaline in the second layer is given simply by

$$\delta_1^{(2)} = (d_1 - \text{sgm}(s_1^{(2)}))\text{sgm}'(s_1^{(2)}). \tag{63}$$

Likewise, the procedure for finding $\delta^{(l)}$, the square error derivative associated with a given Adaline in hidden layer l, involves respectively multiplying each derivative $\delta^{(l+1)}$ associated with each element in the layer immediately downstream from a given Adaline by the weight which connects it to the given Adaline. These weighted square error derivatives are then added together, producing an error term $\varepsilon^{(l)}$, which, in turn, is multiplied by $\text{sgm}'(s^{(l)})$, the derivative of the given Adaline's sigmoid function at its current operating point. If a network has more than two layers, this process of backpropagating the instantaneous square error derivatives from one layer to the immediately preceding layer is successively repeated until a square error derivative δ is computed for each Adaline in the network.

We now have a general method for finding a derivative δ for each Adaline element in the network. The next step is to use these δ's to obtain the corresponding gradients. Consider an Adaline somewhere in the network which, during presentation k, has a weight vector $\mathbf{W}_k$, an input vector $\mathbf{X}_k$, and a linear output $s_k = \mathbf{W}_k^T\mathbf{X}_k$.

The instantaneous gradient for this Adaline element is

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k}. \tag{64}$$

This can be written as

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k} = \frac{\partial \varepsilon_k^2}{\partial s_k}\frac{\partial s_k}{\partial \mathbf{W}_k}. \tag{65}$$

Note that $\mathbf{W}_k$ and $\mathbf{X}_k$ are independent so

$$\frac{\partial s_k}{\partial \mathbf{W}_k} = \frac{\partial \mathbf{W}_k^T\mathbf{X}_k}{\partial \mathbf{W}_k} = \mathbf{X}_k. \tag{66}$$

Therefore,

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial s_k}\mathbf{X}_k. \tag{67}$$

For this element,

$$\delta_k = -\frac{1}{2}\frac{\partial \varepsilon_k^2}{\partial s_k}. \tag{68}$$

Accordingly,

$$\hat{\nabla}_k = -2\delta_k \mathbf{X}_k. \tag{69}$$

Updating the weights of the Adaline element using the method of steepest descent with the instantaneous gradient is a process represented by

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k + 2\mu\delta_k \mathbf{X}_k. \tag{70}$$

Thus, after backpropagating all square error derivatives, we complete a backpropagation iteration by adding to each weight vector the corresponding input vector scaled by the associated square error derivative. Equation (70) and the means for finding δ_k comprise the general weight update rule of the backpropagation algorithm.

There is a great similarity between Eq. (70) and the μ-LMS algorithm (25), but one should view this similarity with caution. The quantity δ_k, defined as a squared error derivative, might appear to play the same role in backpropagation as that played by the error in the μ-LMS algorithm. However, δ_k is not an error. Adaptation of the given Adaline is effected to reduce the squared output error ε_k^2, not δ_k of the given Adaline or of any other Adaline in the network. The objective is not to reduce the δ_k's of the network, but to reduce ε_k^2 at the network output.

It is interesting to examine the weight updates that backpropagation imposes on the Adaline elements in the output layer. Substituting Eq. (63) into Eq. (70) reveals that the Adaline which provides output y_1 in Figure 17 is updated by the rule

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\varepsilon_1^{(2)}\mathrm{sgm}'(s_1^{(2)})\mathbf{X}_k. \tag{71}$$

This rule turns out to be identical to the single Adaline version (46) of the backpropagation rule. This is not surprising since the output Adaline is provided with both input signals and desired responses, so its training circumstance is the same as that experienced by an Adaline trained in isolation.

There are many variants of the backpropagation algorithm. Sometimes, the size of μ is reduced during training to diminish the effects of gradient noise in the weights. Another extension is the momentum technique[44] which involves including in the weight change vector $\Delta\mathbf{W}_k$ of each Adaline a term proportional to the corresponding weight change from the previous iteration. That is, Eq. (70) is replaced by a pair of equations:

$$\Delta\mathbf{W}_k = 2\mu(1 - \eta)\delta_k \mathbf{X}_k + \eta\Delta\mathbf{W}_{k-1} \tag{72}$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \Delta\mathbf{W}_k \tag{73}$$

where the momentum constant $0 \leq \eta < 1$ is in practice usually set to something around 0.8 or 0.9.

The momentum technique low-pass filters the weight updates and thereby tends to resist erratic weight changes due either to gradient noise or to high spatial frequencies in the mean-square-error surface. The factor $(1 - \eta)$ in Eq. (72) is included to give the filter a DC gain of unity so that the learning rate μ

does not need to be stepped down as the momentum constant η is increased. A momentum term can also be added to the update equations of other algorithms discussed in this article. A detailed analysis of stability issues associated with momentum updating for the μ-LMS algorithm, for instance, has been described by Shynk and Roy.[104]

In our experience, the momentum technique used alone is usually of little value. We have found, however, that is is often useful to apply the technique in situations that require relatively "clean"* gradient estimates. One case is a normalized weight update equation which makes the network's weight vector move the same Euclidean distance with each iteration. This can be accomplished by replacing Eqs. (72) and (73) with

$$\Delta_k = \delta_k \mathbf{X}_k + \eta \Delta_{k-1} \tag{74}$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \frac{\mu \Delta_k}{\sqrt{\sum_{\text{all Adalines}} |\Delta_k|^2}}. \tag{75}$$

where again $0 < \eta < 1$. The weight updates determined by Eqs. (74) and (75) can help a network find a solution when a relatively flat local region in the mean-square-error surface is encountered. The weights move by the same amount whether the surface is flat or inclined. It is reminiscent of α-LMS because the gradient term in the weight update equation is normalized by a time-varying factor. The weight update rule could be further modified by including terms from both techniques associated with Eqs. (72) through (75). Other methods for speeding up backpropagation training include Fahlman's popular quickprop method,[105] as well as the delta-bar-delta approach reported in an excellent article by Jacobs.[106,†]

One of the most promising new areas of neural network research involves backpropagation variants for training various recurrent (signal feedback) networks. Recently, backpropagation rules have been devised for training recurrent networks to learn static associations.[107,108] More interesting is the on-line technique of Williams and Zipser[109] which allows a wide class of recurrent networks to learn dynamic associations and trajectories. A more general and computationally viable variant of this technique has been advanced by Narendra and Parthasarathy.[83] These on-line methods are generalizations of a well-known steepest-descent algorithm for training linear IIR filters.[32,110]

*"Clean" gradient estimates are those with little gradient noise.

†Jacobs's article, like many other articles in the literature, assumes for analysis that the true gradients rather than instantaneous gradients are used to update the weights, i.e., that weights are changed periodically, only after all training patterns are presented. This eliminates gradient noise but can slow down training enormously if the training set is large. The delta-bar-delta procedure in Jacobs's article involves monitoring changes of the true gradients in response to weight changes. It should be possible to avoid the expense of computing the true gradients explicitly in this case by instead monitoring changes in the outputs of, say, two momentum filters with different time constants.

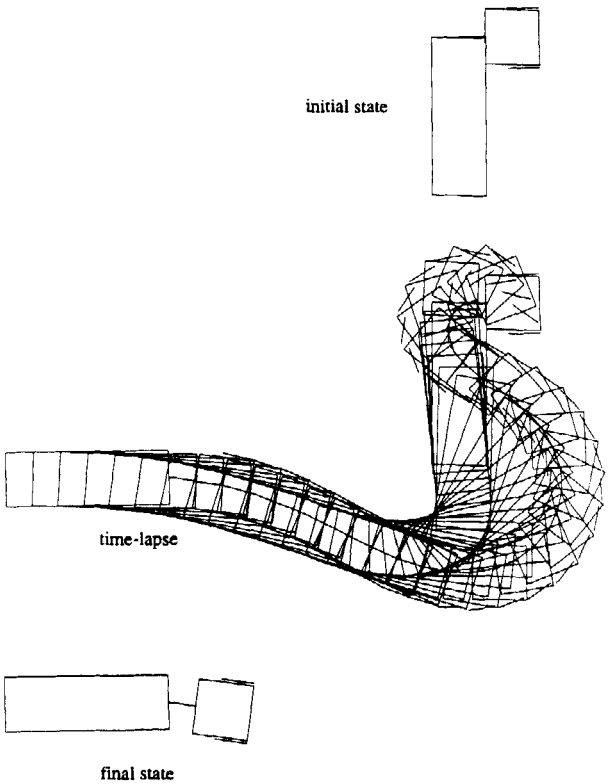

Figure 19. Example truck backup sequence.

An equivalent technique which is usually far less computationally intensive, but best suited for off-line computation also exists.[39,44,111] This approach, called "backpropagation through time," has been used by Nguyen and Widrow[52] to enable a neural network to learn without a teacher how to back up a computer-simulated trailer truck to a loading dock (Fig. 19). This is a highly nonlinear steering task and it is not yet known how to design a controller to perform it. Nevertheless, with just 6 inputs providing information about the current position of the truck, a two-layer neural network with only 26 Adalines was able to learn of its own accord to solve this problem. Once trained, the network could successfully back up the truck from any initial position and orientation in front of the loading dock.

B. Madaline Rule III for Networks

It is difficult to build neural networks with analog hardware which can be trained effectively by the popular backpropagation technique. Attempts to overcome this difficulty have led to the development of the MRIII algorithm. A commercial analog neurocomputing chip based primarily on this algorithm has

already been devised.[112] The method described in this section is a generalization of the single Adaline MRIII technique (52). The multi-element generalization of the other single element MRIII rule (53) is described in Ref. 113.

The MRIII algorithm can be readily described by referring to Figure 20. Although this figure shows a simple two-layer feedforward architecture, the procedure to be developed will work for neural networks with any number of Adaline elements in any feedforward structure. In Ref. 113, we discuss variants of the basic MRIII approach that allow steepest-descent training to be applied to more general network topologies, even those with signal feedback.

Assume that an input pattern $\mathbf{X}$ and its associated desired output responses d_1 and d_2 are presented to the network of Figure 20. At this point, we measure the sum squared output response error $\varepsilon^2 = (d_1 - y_1)^2 + (d_2 - y_2)^2 = \varepsilon_1^2 + \varepsilon_2^2$. We than add a small quantity Δs to a selected Adaline in the network, providing a perturbation to the element's linear sum. This perturbation propagates through the network, and causes a change in the sum of the squares of the errors, $\Delta(\varepsilon^2) = \Delta(\varepsilon_1^2 + \varepsilon_2^2)$. An easily measured ratio is

$$\frac{\Delta(\varepsilon^2)}{\Delta s} = \frac{\Delta(\varepsilon_1^2 + \varepsilon_2^2)}{\Delta s} \simeq \frac{\partial(\varepsilon^2)}{\partial s}. \tag{76}$$

Below we use this to obtain the instantaneous gradient of ε_k^2 with respect to the weight vector of the selected Adaline. For the kth presentation, the instantaneous gradient is

$$\hat{\nabla}_k = \frac{\partial(\varepsilon_k^2)}{\partial \mathbf{W}_k} = \frac{\partial(\varepsilon_k^2)}{\partial s_k} \frac{\partial s_k}{\partial \mathbf{W}_k} = \frac{\partial(\varepsilon_k^2)}{\partial s_k} \mathbf{X}_k. \tag{77}$$

Replacing the derivative with a ratio of differences yields

$$\hat{\nabla}_k \simeq \frac{\Delta(\varepsilon_k^2)}{\Delta s} \mathbf{X}_k. \tag{78}$$

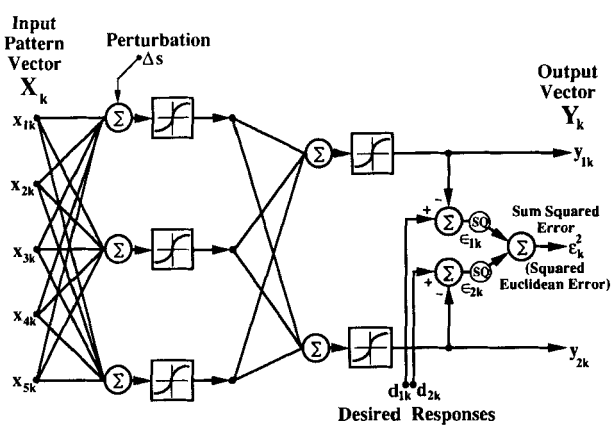

Figure 20. Example two-layer Madaline III architecture.

The idea of obtaining a derivative by perturbing the linear output of the selected Adaline element is the same as that expressed for the single element in Section V–B, except that here the error is obtained from the output of a multi-element network rather than from the output of a single element.

The gradient (78) can be used to optimize the weight vector in accord with the method of steepest descent:

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \mu \frac{\Delta(\varepsilon_k^2)}{\Delta s} \mathbf{X}_k. \tag{79}$$

Maintaining the same input pattern, one could either perturb all the elements in the network in sequence, adapting after each gradient calculation, or else the derivatives could be computed and stored to allow all Adalines to be adapted at once. These two MRIII approaches both involve the same weight update Eq. (79), and if μ is small, both lead to equivalent solutions. With large μ, experience indicates that adapting one element at a time results in convergence after fewer iterations, especially in large networks. Storing the gradients, however, has the advantage that after the initial unperturbed error is measured during a given training presentation, each gradient estimate requires only the perturbed error measurement. If adaptations take place after each error measurement, however, both perturbed and unperturbed errors must be measured for each gradient calculation. This is because each weight update changes the associated unperturbed error.

C. Comparison of MRIII and MRII

MRIII was derived from MRII replacing the signum nonlinearities with sigmoids. The similarity of these algorithms becomes evident when comparing Figure 20, representing MRIII, with Figure 12, representing MRII.

The MRII network is highly discontinuous and nonlinear. Using an instantaneous gradient to adjust the weights is not possible. The idea of adding a perturbation to the linear sum of a selected Adaline element is workable, however. If the Hamming error has been reduced by the perturbation, the Adaline is adapted to reverse its output decision. This weight change is in the LMS direction, along its **X**-vector. If adapting the Adaline would not reduce network output error, it is not adapted. This is in accord with the minimal disturbance principle. The Adalines selected for possible adaptation are those whose analog sums are closest to zero, i.e., the Adalines which can be adapted to give opposite responses with the smallest weight changes. It is useful to note that with binary ± 1 desired responses, the Hamming error is equal to one quarter the sum square error. Minimizing the output Hamming error is therefore equivalent to minimizing the output sum square error.

The MRIII algorithm works in a similar manner. All the Adalines in the MRIII network are adapted, but those whose analog sums are closest to zero will usually be adapted most strongly, because the sigmoid has its maximum slope at zero, contributing to high gradient values. As with MRII, the objective is to change the weights for the given input presentation to reduce the sum

square error at the network output. In accord with the minimal disturbance principle, the weight vectors of the Adaline elements are adapted in the LMS direction, along their **X**-vectors, and are adapted in proportion to their capabilities for reducing the sum square error (the square of the Euclidean error) at the output.

D. Comparison of MRIII with Backpropagation

In Section V–B, we argued that, for the Sigmoid Adaline element, the MRIII algorithm (53) is essentially equivalent to the backpropagation algorithm (46). The same argument can be extended to the network of Adaline elements, demonstrating that if Δs is small and adaptation is applied to all elements in the network at once, the MRIII is essentially equivalent to backpropagation. That is, to the extent that the sample derivative $\Delta\varepsilon_k^2/\Delta s$ from Eq. (79) is equal to the analytical derivative $\partial\varepsilon_k^2/\partial s_k$ from Eq. (67), the two rules follow identical instantaneous gradients, and thus perform identical weight updates.

The backpropagation algorithm requires fewer operations than MRIII to calculate gradients, since it is able to take advantage of *a priori* knowledge of the sigmoid nonlinearities and their derivative functions. Conversely, the MRIII algorithm uses no prior knowledge about the characteristics of the sigmoid functions. Rather, it acquires instantaneous gradients from perturbation measurements. Using MRIII, tolerances on the sigmoid implementations can be greatly relaxed compared to acceptable tolerances for successful backpropagation.

Steepest-descent training of multi-layer networks implemented by computer simulation or by precise parallel digital hardware is usually best carried out by backpropagation. During each training presentation, the backpropagation method requires only one forward computation through the network followed by one backward computation in order to adapt all the weights of an entire network. To accomplish the same effect with the form of MRIII that updates all weights at once, one measures the unperturbed error followed by a number of perturbed error measurements equal to the number of elements in the network. This process can represent a significant amount of computation.

If a network is to be implemented in analog hardware, however, experience has shown that MRIII offers strong advantages over backpropagation. Comparison of Figure 17 with Figure 20 demonstrates the relative simplicity of MRIII: All the apparatus for backward propagation of error-related signals is eliminated, and the weights do not need to carry signals in both directions (see Fig. 18). MRIII is a much simpler algorithm to build and to understand, and in principle it produces the same instantaneous gradient as the backpropagation algorithm. The momentum technique and most other common variants of the backpropagation algorithm can be applied to MRIII training.

VII. A NETWORK TOPOLOGY FOR PATTERN RECOGNITION

It would be useful to devise a neural network configuration that could be trained to classify an important set of training patterns as required, but have

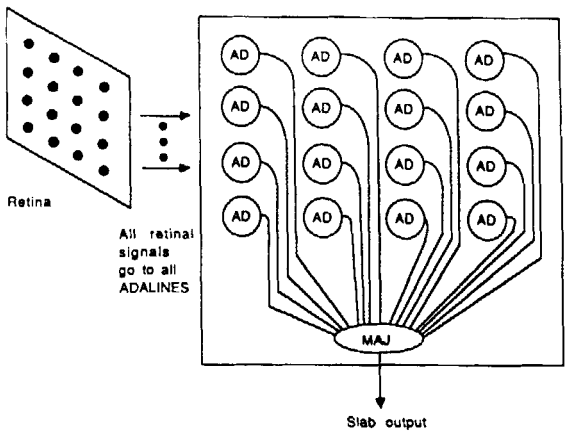

Figure 21. One slab of a left–right, up–down translation invariant network.

these network responses be invariant to translation, rotation, and scale change of the input pattern within the field of view. It should not be necessary to train the system with the specific training patterns of interest in all combinations of translation, rotation, and scale.

The first step is to show that a neural network having these properties exists. (The invariance methods that follow are extensions of results reported earlier by Widrow[2].) The next step is to obtain training algorithms to achieve the desired objectives.

A. Invariance to Up–Down, Left–Right Pattern Translation

Figure 21 shows a planar network configuration (a "slab" of neurons) that could be used to map a retinal image* into a single-bit output so that, with proper weights in the network's neurons, the response will be insensitive to left–right and/or up–down translation. The same slab structure can be replicated, with different weights, to allow the retinal pattern to be independently mapped into additional single-bit outputs, all insensitive to left–right, up–down translation.

Figure 22 illustrates the general idea. A retinal image having a given number of pixels can be mapped through an array of slabs into a different image having the same, more, or fewer pixels, depending on the number of slabs used. In any event, the mapped image is insensitive to up–down, left–right translation of the original image. The mapped image in Figure 22 is fed to a set of Adaline neurons that can be easily trained to provide output responses to the original image as required. This amounts to a "descrambling" of the preprocessor's outputs. The descrambler's output responses classify the original input

*We should note that properties of the defined artificial retinal and neural systems have no biological significance.

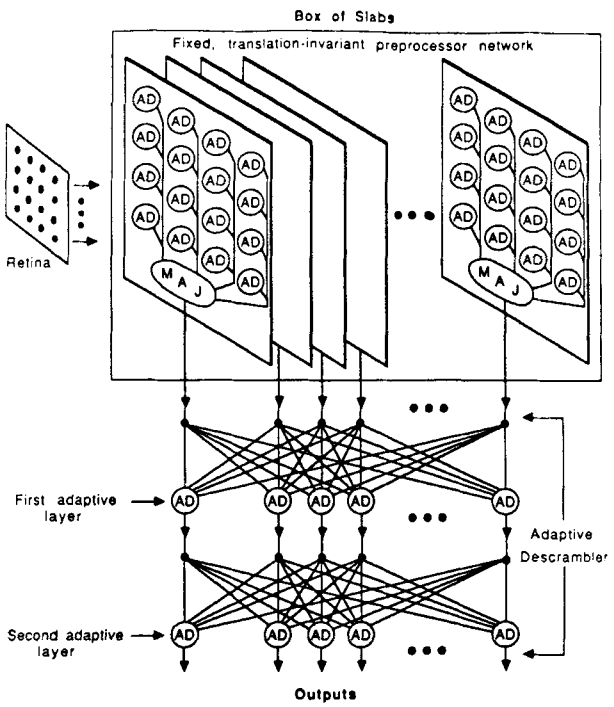

Figure 22. A translation-invariant preprocessor network and an adaptive two-layer descrambler network.

images and, at the same time, are insensitive to their left–right, up–down translations.

In the systems of Figure 21 and 22, the elements labeled "AD" are Adalines. Those labeled "MAJ" are majority vote-takers. (If the number of input lines to MAJ is even and there is a tie vote, these elements are biased to give a positive response.) The AD elements are adaptive neurons and the MAJ elements are fixed neurons, as in Figure 7.

In the system shown in Figure 21, the structuring of the weights so that the output is insensitive to left–right and up–down translation needs further explanation. Let the weights of each Adaline be arranged in a square array and the corresponding retinal pixels arrayed in a square pattern. Let the square matrix (W_1) designate the array of weights of the upper-left Adaline, and let $T_{D1}(W_1)$ be the array of weights of the next lower Adaline. The operator T_{D1} represents "translate down one," so the second set of weights is the same as the topmost set, but translated down *en masse* by one pixel. The bottom row wraps around to comprise the top row. The patterns on the retina itself wrap around on a cylinder when they undergo translation. The weights of the next lower Adaline are $T_{D2}(W_1)$, and those of the next lower Adaline are $T_{D3}(W_1)$. Returning to the upper-left Adaline, let its neighbor to the right be designated by $T_{R1}(W_1)$ with

T_{R1} being a "translate right one" operator. The pattern of weights for the entire array of Adalines in Figure 21 is

$$\begin{array}{cccc} (W_1) & T_{R1}(W_1) & T_{R2}(W_1) & T_{R3}(W_1) \\ T_{D1}(W_1) & T_{R1}T_{D1}(W_1) & T_{R2}T_{D1}(W_1) & T_{R3}T_{D1}(W_1) \\ T_{D2}(W_1) & T_{R1}T_{D2}(W_1) & T_{R2}T_{D2}(W_1) & T_{R3}T_{D2}(W_1) \\ T_{D3}(W_1) & T_{R1}T_{D3}(W_1) & T_{R2}T_{D3}(W_1) & T_{R3}T_{D3}(W_1) \end{array} \tag{80}$$

As the input pattern moves up, down, left, or right on the retina, the roles of the various Adalines interchange. Since all Adaline outputs are equally weighted by the MAJ element, translating the input pattern up–down and/or left–right on the retina has no effect on the MAJ element output.

The set of "key" weights (W_1) can be randomly chosen. Once chosen, they can be translated according to Equation (80) to fill out the array of weights for the system of Figure 21. This array of weights can be incorporated as the weights for the first slab of Adalines shown in Figure 22. The weights for the second slab would require the same translational symmetries, but be based on a different randomly chosen set of key weights W_2. The mapping function of the second slab would therefore be distinct from that of the first slab.

The translational symmetries in the weights called for in Figure 21 could be fixed and manufactured in, or they could be arrived at through training. If, when designing an application-specific pattern recognition system, one knew that translational invariance would be required, it would make sense to manufacture the appropriate symmetry into a fixed weight system, leaving only the final-output Adaline layers plastic and trainable (see Fig. 22). Such a preprocessor would definitely work, would provide very high speed response without scanning and searching for pattern location and alignment, and would be an excellent application of neural networks.

B. Invariance to Rotation

Figure 22 represents a system for preprocessing retinal patterns with a translation-invariant fixed neural network followed by a two-layer adaptive descrambler network. The system can be expanded to incorporate rotational invariance. Suppose that all input patterns can be presented in "normal" vertical orientation, approximately centered within the field of view of the retina. Suppose further that all input patterns can be presented when rotated from normal by 90, 180, and 270°. Thus, each pattern can be presented in all four rotations and in all possible left–right, up–down translations. The number of combinations would be large. The problem is to design a neural network preprocessor that is invariant to translation and to rotation by 90°.

Begin with a single slab of Adaline elements, as shown in Figure 21, producing a majority output that is insensitive to translation of the input pattern on the retina. Next, replicate this slab fourfold, and let the majority outputs feed into a single majority output element. In the first slab, (W_1) designates the upper-left Adaline's matrix of weights. [See Eq. (80) for the weight matrices of

all first-slab Adalines.] In the second slab, the upper-left Adaline's weight matrix corresponds to the first-slab weight matrix rotated 90° clockwise. This can be designated by $R_{C1}(W_1)$, and the corresponding third- and fourth-slab weight matrices can be designated by $R_{C2}(W_1)$ and $R_{C3}(W_1)$. Thus, the weight matrices of the upper-left Adalines begin with (W_1) in the first slab, and are rotated clockwise by 90° in the second slab, by 180° in the third slab, and by 270° in the fourth slab. The weight matrices of all slabs are translated right and down, in the fashion of Equation (80), starting with the Adalines in the upper left-hand corner. For example, the array of weight matrices for the second slab is

$$\begin{matrix} R_{C1}(W_1) & T_{R1}R_{C1}(W_1) & T_{R2}R_{C1}(W_1) & T_{R3}R_{C1}(W_1) \\ T_{D1}R_{C1}(W_1) & T_{R1}T_{D1}R_{C1}(W_1) & T_{R2}T_{D1}R_{C1}(W_1) & T_{R3}T_{D1}R_{C1}(W_1) \\ T_{D2}R_{C1}(W_1) & T_{R1}T_{D2}R_{C1}(W_1) & T_{R2}T_{D2}R_{C1}(W_1) & T_{R3}T_{D2}R_{C1}(W_1) \\ T_{D3}R_{C1}(W_1) & T_{R1}T_{D3}R_{C1}(W_1) & T_{R2}T_{D3}R_{C1}(W_1) & T_{R3}T_{D3}R_{C1}(W_1) \end{matrix} \tag{81}$$

Clearly, translating the pattern on the retina does not change the majority output response. Rotating the pattern 90° causes an interchange of the roles of the slabs in making their responses, but, since the output majority element weights them equally, the output response is unchanged. Insensitivity to 45° rotation can be accomplished by using more slabs; thus, a complete neural network providing invariance to rotation and translation could be constructed. Each translation-invariant slab of Figure 22 would need to be replaced by the rotation-invariant multiple slab and majority-element system described above.

C. Invariance to Scale

The same principles can be used to design invariance networks that are insensitive to scale or pattern size. By establishing a "point of expansion" on the retina so that input patterns can be expanded or contracted with respect to this point, two Adalines can be trained to give similar responses to patterns of two different sizes if the weight matrix of one expands (or contracts) about the point of expansion like the patterns themselves. The amplitude of the weights must be scaled in inverse proportion to the square of the linear dimension of the retinal pattern. By adding many more slabs, the invariance network can be built around this idea to be insensitive to pattern size as well as to translation and rotation. (Implementation would, of course, require the abundance and low cost of VLSI electronics.)

The general pattern-recognition concept we've described involves use of an invariance net followed by a trainable classifier. Figure 23 illustrates the key ideas. The invariance net can be trained or designed to produce a set of outputs that are insensitive to translation, rotation, scale change, etc., of the retinal pattern. These outputs are scrambled, but the adaptive layers can be trained to descramble them and reproduce the original patterns in "standard" position, orientation, and scale.

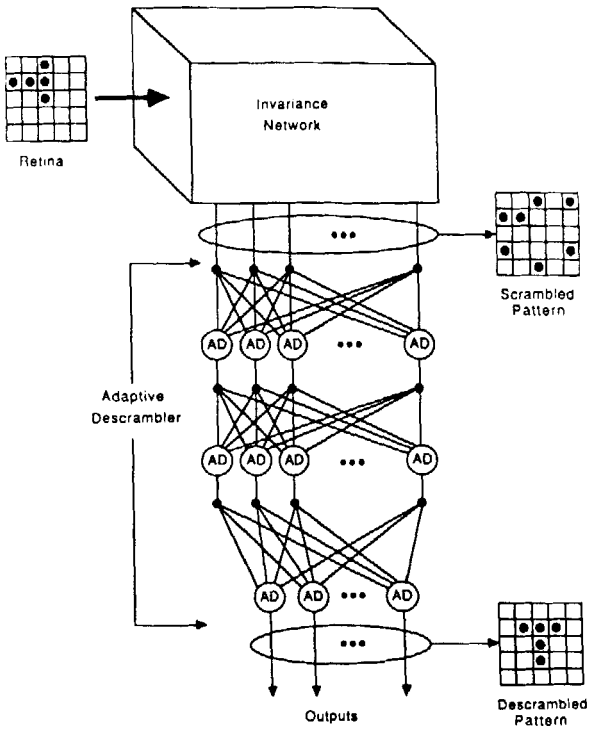

Figure 23. A neural network system for pattern recognition.

VIII. THE TRAINABLE EXPERT SYSTEM

In the utilization of present-day rule-based expert systems, decision rules must always be known for the application of interest. Sometimes, however, determining these rules may present difficulties. The rules may not be explicit or they may not exist. For such applications, trainable expert systems might be usable. Rather than working with decision rules, an adaptive expert system might observe the decisions made by a human expert. Looking over the expert's shoulders, an adaptive system can learn to make similar decisions to those of the human teacher when facing given sets of input circumstances.

A sample application for a trainable expert system is the following: Driving a car down a narrow congested street is done every day by experienced human drivers. Imagine an adaptive pattern recognition system equipped with a visual input (a retina of photo receptors or a TV camera) looking through the windshield of the car seeing the dynamic scene confronting the driver and at the same time observing the driver's responses via the steering wheel, brake pedal, and accelerator pedal. With an adequate training sample, the adaptive system should be able to make decisions very much like those of the teacher, an expert human driver.

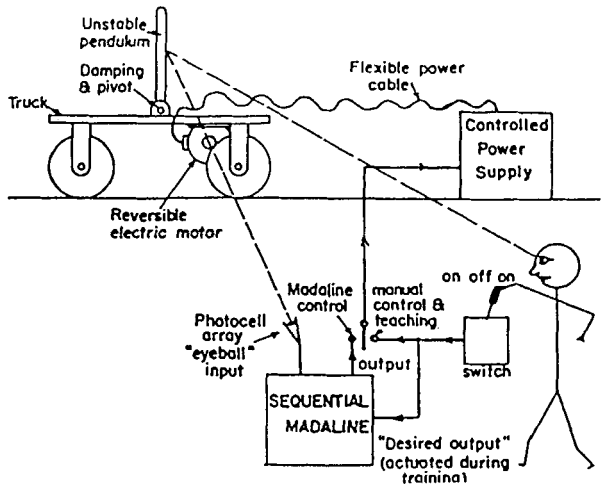

Figure 24. A trainable expert system designed to balance a broom.

Instead of developing at this time a system to learn to drive an automobile in traffic, a simpler and more quantifiable problem to begin with is the broom balancing problem. Referring to Figure 24, a trainable controller consisting of an Adaline network learns to respond like the teacher by observing the teacher's control decisions. However, the teacher has the proper state variables as its set of inputs. The Adaline net has an "eyeball" input—a photocell retina or a TV camera—with which to observe the positions and motions of the car and the pendulum. Acting on its own, the Adaline will need to obtain the equivalent state-variable information from visual observations of the scene and its time rate of change, the scene being the picture of the cart and pendulum. With an adequate training sample, the Adaline net will be able to take over the control function from the teacher and thus become a trained expert. A trainable expert system of this nature has already been simulated on a computer with good results.[114] Doing the research to develop an adaptive controller such as that used with the broom balancer, enables us to progress toward the goal of being able to design trainable expert systems for much more general control applications.

IX. SUMMARY

We have discussed and compared the LMS algorithm, the Perceptron rule, the backpropagation algorithm, and several other learning rules. Although they differ significantly from each other, they all belong to the same "family."

A distinction was drawn between error correction rules and steepest descent rules. The former includes the Perceptron rule, Mays's rules, the α-LMS algorithm, the original Madaline I rule of 1962, and the Madaline II rule. The latter includes the μ-LMS algorithm, the Madaline III rule, and the backpropa-

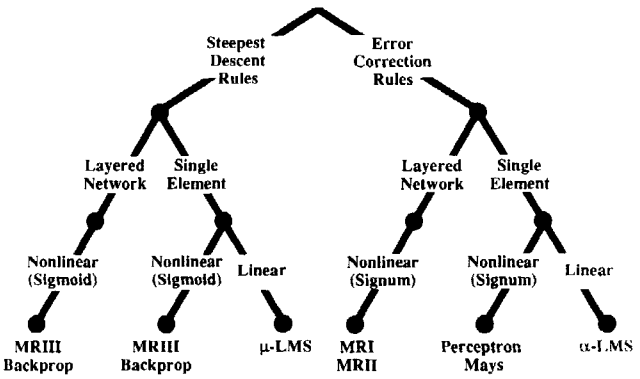

Figure 25. Learning rules.

gation algorithm. The chart in Figure 25 categorizes the learning rules that have been studied.

Although these algorithms have been presented as established learning rules, one should not gain the impression that they are perfect and frozen for all time. Variations are possible for every one of them. They should be regarded as substrates upon which to build new and better rules. There is a tremendous amount of invention waiting "in the wings." We look forward to the next 30 years of development of adaptive and learning algorithms and their applications to pattern recognition, control systems, and expert systems.

This work was sponsored by SDIO Innovative Science and Technology Office and managed by ONR under contract No. N00014-86-K-0718, by the Department of the Army Belvoir R D & E Center under contracts No. DAAK70-87-P-3134 and No. DAAK70-89-K-0001, by a grant from the Lockheed Missiles and Space Company, by NASA under contract No. NCA2-389, and by Rome Air Development Center under contract No. F30602-88-D-0025, subcontract E-21-T22-S1.

References

1. K. Steinbuch and V.A.W. Piske, "Learning matrices and their applications," *IEEE Trans. Elect. Computers* **EC-12**(5), 846–862 (1963).
2. B. Widrow, "Generalization and information storage in networks of adaline 'neurons'," In *Self-Organizing Systems 1962,* M. Yovitz, G. Jacobi, and G. Goldstein, (Eds.), Spartan Books, Washington, DC, 1962, pp. 435–461.
3. L. Stark, M. Okajima, and G.H. Whipple, "Computer pattern recognition techniques: Electrocardiographic diagnosis," *Commun. ACM,* **5,** 527–532 (1962).
4. F. Rosenblatt, "Two theorems of statistical separability in the perceptron," In *Mechanization of Thought Processes: Proceedings of a Symposium held at the National Physical Laboratory, November 1958,* Vol. 1, (HM Stationery Office, London, 1959), pp. 421–456.
5. F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms,* Spartan Books, Washington, DC, 1962.

6. C. von der Malsburg, "Self-organizing of orientation sensitive cells in the striate cortex," *Kybernetik,* **14,** 85–100 (1973).
7. S. Grossberg, "Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors," *Biol. Cybern.,* **23,** 121–134 (1976).
8. K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biol. Cybern.,* **20,** 121–136 (1975).
9. K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.,* **36**(4) 193–202 (1980).
10. B. Widrow, "Bootstrap learning in threshold logic systems," Presented at the American Automatic Control Council (Theory Committee), IFAC Meeting, London, England, June 1966.
11. B. Widrow, N.K. Gupta, and S. Maitra, "Punish/reward: Learning with a critic in adaptive threshold systems," *IEEE Trans. Syst. Man Cybern.,* **SMC-3**(5) 455–465 (1973).
12. A.G. Barto, R.S. Sutton, and C.W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst. Man Cybern.,* **SMC-13,** 834–846 (1983).
13. J.S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (cmac)," *J. Dyn. Sys. Meas. Contr.,* **97,** 220–227 (1975).
14. W.T. Miller, III, "Sensor-based control of robotic manipulators using a general learning algorithm," *IEEE J. Robotics Automa.,* **RA-3**(2), 157–165 (April 1987).
15. A. Barto and P. Anandan, "Pattern recognizing stochastic learning automata," *IEEE Trans. Syst. Man Cybern.,* **15,** 360–375 (1985).
16. S. Grossberg, "Adaptive pattern classification and universal recoding, II: Feedback, expectation, olfaction, and illusions," *Biol. Cybern.,* **23,** 187–202 (1976).
17. G.A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Comp. Vision, Graphics, and Image Processing,* **37,** 54–115 (1983).
18. G.A. Carpenter and S. Grossberg, "Art 2: Self-organization of stable category recognition codes for analog output patterns," *Appl. Optics,* **26**(23), 4919–4930 (1987).
19. G.A. Carpenter and S. Grossberg, "Art 3 hierarchical search: Chemical transmitters in self-organizing pattern recognition architectures," In *Proceedings of the International Joint Conference on Neural Networks* Lawrence Erlbaum, Washington, DC, January 1990, Vol. II, pp. 30–33.
20. T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.,* **43,** 59–69 (1982).
21. T. Kohonen, *Self-Organization and Associative Memory* 2nd ed., Springer-Verlag, New York, 1988.
22. D.O. Hebb, *The Organization of Behavior,* Wiley, New York, 1949.
23. J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci.,* **79,** 2554–2558 (1982).
24. J.J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad. Sci.,* **81,** 3088–3092 (1984).
25. B. Kosko, "Adaptive bidirectional associative memories," *Appl. Optics,* **26**(23), 4947–4960 (1987).
26. H.A. Klopf, "Drive-reinforcement learning: A real-time learning mechanism for unsupervised learning," In *Proceedings of the IEEE First International Conference on Neural Networks,* San Diego, CA, June 21–24, 1987, Vol. II, pp. 441–445.
27. G.E. Hinton, T.J. Sejnowski, and D.H. Ackley, *Boltzmann Machines: Constraint Satisfaction Networks that Learn,* Technical Report CMU-CS-84-119, Carnegie-Mellon University, Department of Computer Science, 1984.

28. G.E. Hinton and T.J. Sejnowski, "Learning and relearning in Boltzmann machines," In *Parallel Distributed Processing,* D.E. Rumelhart and J.L. McClelland, (Eds.), The MIT Press, Cambridge, MA, 1986, Vol. 1, Chap. 7.
29. L.R. Talbert, G.F. Groner, J.S. Koford, R.J. Brown, P.R. Low, and C.H. Mays, *A Real-Time Adaptive Speech-Recognition System,* Technical Report, Stanford University, 1963.
30. M.J.C. Hu, *Application of the Adaline System to Weather Forecasting,* E.E. Degree Thesis, Technical Report 6775-1, Stanford Electron. Labs., Stanford, CA, June 1964.
31. B. Widrow, "The original adaptive neural net broom-balancer," In *Proceedings of the IEEE International Symposium on Circuits and Systems,* Philadelphia, PA, May 4–7, 1987, pp. 351–357.
32. B. Widrow and S.D. Stearns, *Adaptive Signal Processing,* Prentice-Hall, Englewood Cliffs, NJ, 1985.
33. B. Windrow, P. Mantey, L. Griffiths, and B. Goode, "Adaptive antenna systems," *Proc. IEEE* **55,** 2143–2159 (1967).
34. B. Widrow, "Adaptive inverse control," In *Proceedings of the 2nd International Federation of Automatic Control Workshop,* Lund, Sweden, July 1–3, 1986, pp. 1–5.
35. B. Widrow *et al.,* "Adaptive noise cancelling: Principles and applications," *Proc. IEEE,* **63,** 1692–1716 (1975).
36. R.W. Lucky, "Automatic equalization for digital communication," *Bell Syst. Tech. J.,* **44,** 547–588 (1965).
37. R.W. Lucky *et al., Principles of Data Communication,* McGraw-Hill, New York, 1968.
38. M.M. Sondhi, "An adaptive echo canceller," *Bell Syst. Tech. J.,* **46,** 497–511 (1967).
39. P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences,* Ph.D. Thesis, Harvard University, Cambridge, MA, August 1974.
40. Y. le Cun, "A theoretical framework for back-propagation," In *Proceedings of the 1988 Connectionist Models Summer School,* D. Touretzky, G. Hinton, and T. Sejnowski (Eds.), Morgan Kaufmann, San Mateo, CA, June 17–26, 1988, pp. 21–28.
41. D. Parker, *Learning-logic,* Invention Report S81-64, File 1, Office of Technology Licensing, Stanford University, Stanford, CA, October 1982.
42. D. Parker, *Learning-logic,* Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, April 1985.
43. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning Internal Representations by Error Propagation,* ICS Report 8506, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA, September 1985.
44. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," In *Parallel Distributed Processing,* D.E. Rumelhart and J.L. McClelland (Eds.), The MIT Press, Cambridge, MA, 1986, Vol. 1, Chap. 8.
45. B. Widrow, R.G. Winter, and R. Baxter, "Learning phenomena in layered neural networks," In *Proceedings of the IEEE First International Conference on Neural Networks,* San Diego, CA, June 1987, Vol. II, pp. 411–429.
46. R.P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.* April 1987.
47. J.A. Anderson and E. Rosenfeld (Eds.), *Neurocomputing: Foundations of Research,* The MIT Press, Cambridge, MA, 1988.
48. N. Nilsson, *Learning Machines,* McGraw-Hill, New York, 1965.
49. D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel Distributed Processing,* Vol. 1 and 2, The MIT Press, Cambridge, MA, 1986.
50. B. Moore, "Art 1 and pattern clustering," In *Proceedings of the 1988 Connec-*

tionist Models Summer School, D. Touretzky, G. Hinton, and T. Sejnowski, (Eds.), Morgan Kaufmann, San Mateo, CA, June 17–26 1988. pp. 174–185.
51. *DARPA Neural Network Study,* AFCEA International Press, Fairfax, VA, 1988.
52. D. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural networks," In *Proceedings of the International Joint Conference on Neural Networks,* Washington, DC, June 1989, Vol. II, pp. 357–363.
53. T.J. Sejnowski and C.R. Rosenberg, *Nettalk: A Parallel Network that Learns to Read Aloud,* Technical Report JHU/EECS-86/01, Johns Hopkins University, 1986.
54. T.J. Sejnowski and C.R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Syst.,* **1,** 145–168 (1987).
55. P.M. Shea and V. Lin, "Detection of explosives in checked airline baggage using an artificial neural system," In *Proceedings of the International Joint Conference on Neural Networks,* Washington, DC, June 1989, Vol. II, pp. 31–34.
56. D.G. Bounds, P.J. Lloyd, B. Mathew, and G. Waddell, "A multilayer perceptron network for the diagnosis of low back pain," In *Proceedings of the IEEE Second International Conference on Neural Networks,* San Diego, CA, July 1988, Vol. II, pp. 481–489.
57. G. Bradshaw, R. Fozzard, and L. Ceci, "A connectionist expert system that actually works," In *Advances in Neural Information Processing Systems I,* D.S. Touretzky (Ed.) (Morgan Kaufmann, San Mateo, CA, 1989), pp. 248–255.
58. N. Mokhoff, "Neural nets making the leap out of lab," *Electronic Engineering Times,* January 22, 1990, p. 1.
59. C.A. Mead, *Analog VLSI and Neural Systems* (Addison-Wesley, Reading, MA, 1989).
60. B. Widrow and M.E. Hoff, Jr., "Adaptive switching circuits," in *1960 IRE Western Electric Show and Convention Record, Part 4,* August 23, 1960, pp. 96–104.
61. B. Widrow and M.E. Hoff, Jr., *Adaptive Switching Circuits,* Technical Report 1553-1, Stanford Electron. Labs., Stanford, CA, June 30, 1960.
62. P.M. Lewis II and C. Coates, *Threshold Logic,* Wiley, New York, 1967.
63. T.M. Cover, *Geometrical and Statistical Properties of Linear Threshold Devices,* Ph.D. Thesis, Technical Report 6107-1, Stanford Electron. Labs., Stanford, CA, May 1964.
64. R.J. Brown, *Adaptive Multiple-Output Threshold Systems and Their Storage Capacities,* Ph.D. Thesis, Technical Report 6771-1, Stanford Electron. Labs., Stanford, CA, June 1964.
65. D.F. Specht, *Generation of Polynomial Discriminant Functions for Pattern Recognition,* Ph.D. Thesis, Technical Report 6764-5, Stanford Electron. Labs., Stanford, CA, May 1966.
66. D.F. Specht, "Vectorcardiographic diagnosis using the polynomial discriminant method of pattern recognition," *IEEE Trans. Biomed. Eng.* **BME-14,** 90–95 (1967).
67. D.F. Specht, "Generation of polynomial discriminant functions for pattern recognition," *IEEE Trans. Electron. Comput.* **EC-16,** 308–319 (1967).
68. A.R. Barron, "Adaptive learning networks: Development and application in the united states of algorithms related to gmdh," In *Self-Organizing Methods in Modeling,* S.J. Farlow (Ed.), Marcel Dekker Inc., New York, 1984, pp. 25–65.
69. A.R. Barron, "Predicted squared error: A criterion for automatic model selection," In *Self-Organizing Methods in Modeling,* S.J. Farlow (Ed.), Marcel Dekker Inc., New York, 1984, pp. 87–103.
70. A.R. Barron and R.L. Barron, "Statistical learning networks: A unifying view," In *1988 Symposium on the Interface: Statistics and Computing Science,* Reston, VA, April 21–23, 1988, pp. 192–203.
71. A.G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE Trans. Syst. Man Cybern.,* **SMC-1,** 364–378 (1971).

72. Y.-H. Pao, "Functional link nets: Removing hidden layers," *AI Expert* 60–68 (1989).
73. C.L. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Appl. Optics* **26**(23), 4972–4978 (1987).
74. M.E. Hoff, Jr, *Learning Phenomena in Networks of Adaptive Switching Circuits*, Ph.D. Thesis, Technical Report 1554-1, Stanford Electron. Labs., Stanford, CA, July 1962.
75. W.C. Ridgway III, *An Adaptive Logic System with Generalizing Properties*, Ph.D. Thesis, Technical Report 1556-1, Stanford Electron. Labs., Stanford, CA, April 1962.
76. F.H. Glanz, *Statistical Extrapolation in Certain Adaptive Pattern-Recognition Systems*, Ph.D. Thesis, Technical Report 6767-1, Stanford Electron. Labs., Stanford, CA, May 1965.
77. B. Widrow, "Adaline and madaline—1963, plenary speech," In *Proceedings of the IEEE First International Conference on Neural Networks*, San Diego, CA, June 23, 1987. Vol. I, pp. 145–158.
78. B. Widrow, *An Adaptive "Adaline" Neuron Using Chemical "Memistors"*, Technical Report 1553-2, Stanford Electron. Labs., Stanford, CA, October 17, 1960.
79. M.L. Minsky and S.A. Papert, *Perceptrons: An Introduction to Computational Geometry* expanded ed., The MIT Press, Cambridge, MA, 1988.
80. E.B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computation*, **1**(1), 151–160 (1989).
81. C.L. Giles, R.D. Griffin, and T. Maxwell, "Encoding geometric invariances in higher order neural networks," In *Neural Information Processing Systems*, D.Z. Anderson (Ed.), American Institute of Physics, New York, 1988, pp. 301–309.
82. J.J. Hopfield and D.W. Tank, "Neural computations of decisions in optimization problems," *Biolog. Cybernetics*, **52**, 141–152 (1985).
83. K.S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, **1**(1), 4–27 (1990).
84. C.H. Mays, *Adaptive Threshold Logic*, Ph.D. Thesis, Technical Report 1557-1, Stanford Electron. Labs., Stanford, CA, April 1963.
85. F. Rosenblatt, "On the convergence of reinforcement procedures in simple perceptrons," *Cornell Aeronautical Laboratory Report VG-1196-G-4*, Buffalo, New York, February 1960.
86. H. Block, "The perceptron: A model for brain functioning, I." *Reviews of Modern Physics*, **34**, 123–135 (1962).
87. B. Widrow and R.G. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Computer*, 25–39 (1988).
88. R.G. Winter, *Madaline Rule II: A New Method for Training Networks of Adalines*, Ph.D. Thesis, Stanford University, Stanford, CA, January 1989.
89. E. Walach and B. Widrow, "The least mean fourth (lmf) adaptive algorithm and its family," *IEEE Trans. on Information Theory* **IT-30**(2), 275–283 (1984).
90. E.B. Baum and F. Wilczek, "Supervised learning of probability distributions by neural networks," In *Neural Information Processing Systems*, D.Z. Anderson, (Ed.), American Institute of Physics, New York, 1988, pp. 52–61.
91. S.A. Solla, E. Levin, and M. Fleisher, "Accelerated learning in layered neural networks," *Complex Syst.* **2**, 625–640 (1988).
92. D.B. Parker, "Optimal algorithms for adaptive neural networks: Second order back propagation, second order direct propagation, and second order hebbian learning," In *Proceedings of the IEEE First International Conference on Neural Networks*, San Diego, CA, June 1987, Vol. II, pp. 593–600.
93. A.J. Owens and D.L. Filkin, "Efficient training of the back propagation network by solving a system of stiff ordinary differential equations," In *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, June 1989, Vol. II, pp. 381–386.

94. D.G. Leunberger, *Linear and Nonlinear Programming* 2nd ed., Addison-Wesley, Reading, MA, 1984.
95. A. Kramer and A. Sangiovanni-Vincentelli, "Efficient parallel learning algorithms for neural networks," In *Advances in Neural Information Processing Systems I,* D.S. Touretzky (Ed.), Morgan Kaufmann, San Mateo, CA, 1989, pp. 40–48.
96. R.V. Southwell, *Relaxation Methods in Engineering Science,* Oxford, New York, 1940.
97. D.J. Wilde, *Optimum Seeking Methods,* Prentice-Hall, Englewood Cliffs, NJ, 1964.
98. N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series, with Engineering Applications,* Wiley, New York, 1949.
99. T. Kailath, "A view of three decades of linear filtering theory," *IEEE Trans. Inform. Theory* **IT-20,** 145–181 (1974).
100. H. Bode and C. Shannon, "A simplified derivation of linear least squares smoothing and prediction theory," *Proc. IRE* **38,** 417–425 (1950).
101. L.L. Horowitz and K.D. Senne, "Performance advantage of complex LMS for controlling narrow-band adaptive arrays," *IEEE Trans. Circuits, Systems,* June 1981, Vol. CAS-28, pp. 562-576.
102. D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," In *Proceedings of the International Joint Conference on Neural Networks,* San Diego, CA, June 1990.
103. B. Widrow and M.A. Lehr, "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proc. IEEE,* September 1990, pp. 1415–1442.
104. J.J. Shynk and S. Roy, "The lms algorithm with momentum updating," In *ISCAS 88,* Espoo, Finland, June 1988.
105. S.E. Fahlman, "Faster learning variations on backpropagation: An empirical study," In *Proceedings of the 1988 Connectionist Models Summer School,* D. Touretzky, G. Hinton, and T. Sejnowski (Eds.), Morgan Kaufmann, San Mateo, CA, June 17–26 1988 pp. 38–51.
106. R.A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks* **1**(4), 295–307 (1988).
107. F.J. Pineda, "Generalization of backpropagation to recurrent neural networks," *Phys. Rev. Lett.* **18**(59), 2229–2232 (1987).
108. L.B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," In *Proceedings of the IEEE First International Conference on Neural Networks,* San Diego, CA, June 1987, Vol. II, pp. 609–618.
109. R.J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," ICS Report 8805, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA 92093, October 1988.
110. S.A. White, "An adaptive recursive digital filter," In *Proc. 9th Asilomar Conf. Circuits Syst. Comput.,* November 1975, p. 21.
111. B. Pearlmutter, "Learning state space trajectories in recurrent neural networks," In *Proceedings of the 1988 Connectionist Models Summer School,* D. Touretzky, G. Hinton, and T. Sejnowski (Eds.), Morgan Kaufmann, San Mateo, CA, June 17–26 1988, pp. 113–117.
112. M. Holler *et al.,* "An electrically trainable artificial neural network (etann) with 10240 "floating gate" synapses," In *Proceedings of the International Joint Conference on Neural Networks,* Washington, DC, June 1989, Vol. II, pp. 191–196.
113. D. Andes, B. Widrow, M. Lehr, and E. Wan, "MRIII: A robust algorithm for training analog neural networks," In *Proceedings of the International Joint Conference on Neural Networks,* Lawrence Erlbaum, Washington, DC, January 1990, Vol. I, pp. 533–536.
114. V.V. Tolat and B. Widrow, "An adaptive broom balancer with visual inputs," In *Proceedings of the IEEE Second International Conference on Neural Networks,* San Diego, CA, July 1988, Vol. II, pp. 641–647.