30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation

BERNARD WIDROW, FELLOW, IEEE, AND MICHAEL A. LEHR

Fundamental developments in feedforward artificial neural networks from the past thirty years are reviewed. The central theme of this paper 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 underlying 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 that disturbs stored information to the smallest extent possible.

I. Introduction

This year marks the 30th anniversary of the Perceptron rule and the LMS algorithm, two early rules for training adaptive elements. Both algorithms were 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 is growing rapidly. One early development was Steinbuch's Learning Matrix [1], a pattern recognition machine based on linear discriminant functions. At the same time, 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] 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].

Widrow devised a reinforcement learning algorithm called "punish/reward" or "bootstrapping" [10], [11] in the mid-1960s. 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-look-up system based on models of human memory [13], [14].

In the 1970s Grossberg developed his Adaptive Resonance Theory (ART), a number of novel hypotheses about the underlying principles governing biological neural systems [15]. These ideas served as the basis for later work by Carpenter and Grossberg involving three classes of ART architectures: ART 1 [16], ART 2 [17], and ART 3 [18]. These are self-organizing neural implementations of pattern clustering algorithms. Other important theory on self-organizing systems was pioneered by Kohonen with his work on feature maps [19], [20].

In the early 1980s, Hopfield and others introduced outer product rules as well as equivalent approaches based on the early work of Hebb [21] for training a class of recurrent (signal feedback) networks now called Hopfield models [22], [23]. More recently, Kosko extended some of the ideas of Hopfield and Grossberg to develop his adaptive Bidirectional Associative Memory (BAM) [24], a network model employing differential as well as Hebbian and competitive learning laws. Other significant models from the past decade include probabilistic ones such as Hinton, Sejnowski, and Ackley's Boltzmann Machine [25], [26] 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 [27], weather forecasting [28], and adaptive controls [29]. Work then switched to adaptive filtering and adaptive signal processing [30] after attempts to develop learning rules for networks with multiple adaptive layers were unsuccessful. Adaptive signal processing proved to

Manuscript received September 12, 1989; revised April 13, 1990. This work was sponsored by SDIO Innovative Science and Technology office and managed by ONR under contract no. N00014-86-K-0718, by the Dept. of the Army Belvoir RD&E Center under contracts no. DAAK 70-87-P-3134 and no. DAAK-70-89-K-0001, by a grant from the Lockheed Missiles and Space Co., by NASA under contract no. NCA2-389, and by Rome Air Development Center under contract no. F30602-88-D-0025, subcontract no. E-21-T22-S1.

The authors are with the Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305-4055, USA.

IEEE Log Number 9038824.

0018-9219/90/0900-1415$01.00 © 1990 IEEE

be a fruitful avenue for research with applications involving adaptive antennas [31], adaptive inverse controls [32], adaptive noise cancelling [33], and seismic signal processing [30]. Outstanding work by 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 [34], [35] and to adaptive echo cancellers for long-distance telephone and satellite circuits [36]. 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 [37].[1] Unfortunately, Werbos's work remained almost unknown in the scientific community. In 1982, Parker rediscovered the technique [39] and in 1985, published a report on it at M.I.T. [40]. Not long after Parker published his findings, Rumelhart, Hinton, and Williams [41], [42] also rediscovered the technique and, largely as a result of the clear framework within which they presented their ideas, they finally succeeeded 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.[2] In digital implementations, the hard-limiting quantizer is more easily computed than any of the differentiable nonlinearities used in backpropagation networks. In 1987, Widrow, Winter, and Baxter looked back at the original Madaline I algorithm with the goal of developing a new technique that could adapt multiple layers of adaptive elements using the simpler hard-limiting quantizers. The result was Madaline Rule II [43].

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 and 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, are not discussed in this paper. The algorithms described are limited primarily to those developed in our laboratory at Stanford, and to related techniques developed elsewhere, the most important of which is the backpropagation algorithm. Section II explores fundamental concepts, Section III discusses adaptation and the minimal disturbance principle, Sections IV and V cover error correction rules, Sections VI and VII delve into steepest-descent rules, and Section VIII provides a summary.

Information about the neural network paradigms not discussed in this paper can be obtained from a number of other sources, such as the concise survey by Lippmann [44], and the collection of classics by Anderson and Rosenfeld [45]. Much of the early work in the field from the 1960s is carefully reviewed in Nilsson's monograph [46]. A good view of some of the more recent results is presented in Rumelhart and McClelland's popular three-volume set [47]. A paper by Moore [48] presents a clear discussion about ART 1 and some of Grossberg's terminology. Another resource is the DARPA Study report [49] which gives a very comprehensive and readable "snapshot" of the field in 1988.

II. Fundamental Concepts

Today we can build computers and other machines that 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 rivaling modern workstations. Nonetheless, many problems remain 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. An infinite number of three-dimensional scenes 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

[1] 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 [38].

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

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 [50], speech generation [51], [52], and undersea mine detection [49]. Small networks have also been used successfully in airport explosive detection [53], expert systems [54], [55], and scores of other applications. Furthermore, efforts to develop parallel neural network hardware are meeting with some success, and such hardware should be available in the future for attacking more difficult problems, such as speech recognition [56], [57].

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.

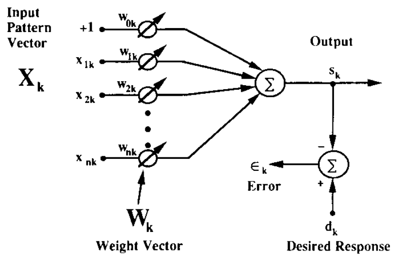


Fig. 1. Adaptive linear combiner.

A. The Adaptive Linear Combiner

The adaptive linear combiner is diagrammed in Fig. 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 $\boldsymbol{X}_k = [x_0, x_{1_k}, x_{2_k}, \cdots, x_{n_k}]^T$ 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 $\boldsymbol{W}_k = [w_{0_k}, w_{1_k}, w_{2_k}, \cdots, w_{n_k}]^T$. The sum of the weighted inputs is then computed, producing a linear output, the inner product $s_k = \boldsymbol{X}_k^T \boldsymbol{W}_k$. The components of $\boldsymbol{X}_k$ 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 reponses. In signal processing applications, the most popular method for adapting the weights is the simple LMS (least mean square) algorithm [58], [59], often called the Widrow-Hoff delta rule [42]. 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 paper.

B. A Linear Classifier—The Single Threshold Element

The basic building block used in many neural networks is the "adaptive linear element," or Adaline[3] [58] (Fig. 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 that 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.

Linear Separability: With n binary inputs and one binary

[3] 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 in that it contains not only the neural cell body, but also the input synapses and a mechanism for training them.

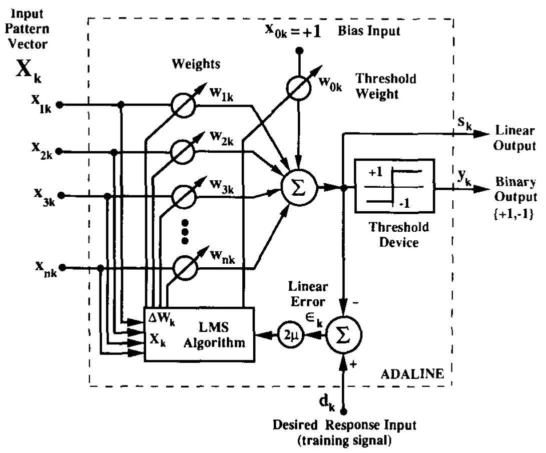


Fig. 2. Adaptive linear element (Adaline).

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 [60]. These are the set of logic functions that can be obtained with all possible weight variations.

Figure 3 shows a two-input Adaline element. Figure 4 represents all possible binary inputs to this element with four large dots in pattern vector space. In this space, the components of the input pattern vector lie along the coordinate axes. The Adaline separates input patterns into two categories, depending on the values of the weights. A critical thresholding condition occurs when the linear output s equals zero:

$$s = x_1 w_1 + x_2 w_2 + w_0 = 0, \tag{1}$$

therefore

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}. \tag{2}$$

Figure 4 graphs this linear relation, which comprises a separating line having slope and intercept given by

$$\text{slope} = -\frac{w_1}{w_2}$$

$$\text{intercept} = -\frac{w_0}{w_2}. \tag{3}$$

The three weights determine slope, intercept, and the side of the separating line that corresponds to a positive output. The opposite side of the separating line corresponds to a negative output. For Adalines with four weights, the separating boundary is a plane; with more than four weights, the boundary is a hyperplane. Note that if the bias weight is zero, the separating hyperplane will be homogeneous—it will pass through the origin in pattern space.

As sketched in Fig. 4, the binary input patterns are classified as follows:

$$\begin{aligned} (+1, +1) &\rightarrow +1 \\ (+1, -1) &\rightarrow +1 \\ (-1, -1) &\rightarrow +1 \\ (-1, +1) &\rightarrow -1 \end{aligned} \tag{4}$$

This is an example of a linearly separable function. An example of a function which is not linearly separable is the two-input exclusive NOR function:

$$\begin{aligned} (+1, +1) &\rightarrow +1 \\ (+1, -1) &\rightarrow -1 \\ (-1, -1) &\rightarrow +1 \\ (-1, +1) &\rightarrow -1 \end{aligned} \tag{5}$$

No single straight line exists that can achieve this separation of the input patterns; thus, without preprocessing, no single Adaline can implement the exclusive NOR function.

With two inputs, a single Adaline can realize 14 of the 16 possible logic functions. With many inputs, however, only a small fraction of all possible logic functions is realizable, that is, linearly separable. Combinations of elements or networks of elements can be used to realize functions that are not linearly separable.

Capacity of Linear Classifiers: The number of training patterns or stimuli that an Adaline can learn to correctly classify is an important issue. Each pattern and desired output combination represents an inequality constraint on the weights. It is possible to have inconsistencies in sets of simultaneous inequalities just as with simultaneous equalities. When the inequalities (that is, the patterns) are determined at random, the number that can be picked before an inconsistency arises is a matter of chance.

In their 1964 dissertations [61], [62], T. M. Cover and R. J. Brown both showed that the average number of random patterns with random binary desired responses that can be

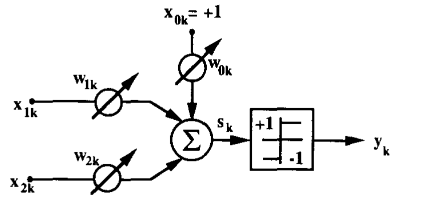


Fig. 3. Two-input Adaline.

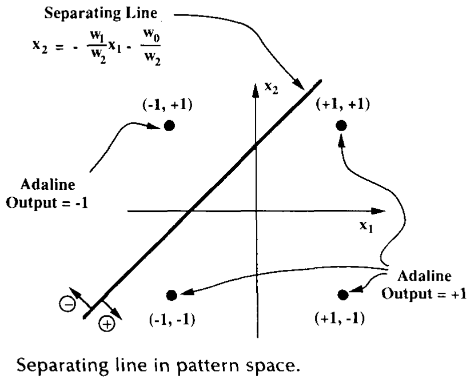


Fig. 4. Separating line in pattern space.

absorbed by an Adaline is approximately equal to twice the number of weights.[4] This is the *statistical pattern capacity* C_s of the Adaline. As reviewed by Nilsson [46], both theses included an analytic formula describing the probability that such a training set can be separated by an Adaline (i.e., it is linearly separable). The probability is a function of N_p, the number of input patterns in the training set, and N_w, the number of weights in the Adaline, including the threshold weight, if used:

$$P_{\text{Separable}} = \begin{cases} 2^{-(N_p-1)} \displaystyle\sum_{i=0}^{N_w-1} \binom{N_p-1}{i} & \text{for } N_p > N_w \\ 1 & \text{for } N_p \leq N_w. \end{cases} \tag{6}$$

In Fig. 5 this formula was used to plot a set of analytical curves, which show the probability that a set of N_p random patterns can be trained into an Adaline as a function of the ratio N_p/N_w. Notice from these curves that as the number of weights increases, the statistical pattern capacity of the Adaline $C_s = 2N_w$ becomes an accurate estimate of the number of responses it can learn.

Another fact that can be observed from Fig. 5 is that a problem is guaranteed to have a solution if the number of patterns is equal to (or less than) half the statistical pattern capacity; that is, if the number of patterns is equal to the number of weights. We will refer to this as the *deterministic pattern capacity* C_d of the Adaline. An Adaline can learn *any* two-category pattern classification task involving no more patterns than that represented by its deterministic capacity, $C_d = N_w$.

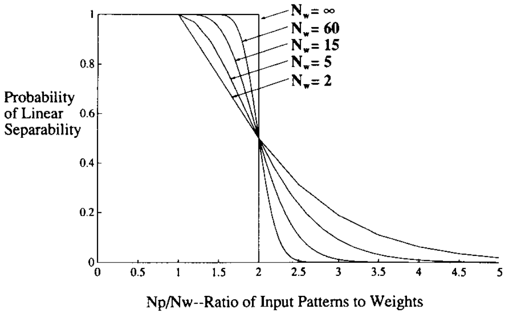


Fig. 5. Probability that an Adaline can separate a training pattern set as a function of the ratio N_p/N_w.

Both the statistical and deterministic capacity results depend upon a mild condition on the positions of the input patterns: the patterns must be in general position with respect to the Adaline.[5] If the input patterns to an Adaline are continuous valued and smoothly distributed (that is, pattern positions are generated by a distribution function containing no impulses), general position is assured. The general position assumption is often invalid if the pattern vectors are binary. Nonetheless, even when the points are not in general position, the capacity results represent useful upper bounds.

The capacity results apply to randomly selected training patterns. In most problems of interest, the patterns in the training set are not random, but exhibit some statistical regularities. These regularities are what make generalization possible. The number of patterns that an Adaline can learn in a practical problem often far exceeds its statistical capacity because the Adaline is able to generalize within the training set, and learns many of the training patterns before they are even presented.

C. Nonlinear Classifiers

The linear classifier is limited in its capacity, and of course is limited to only linearly separable forms of pattern discrimination. More sophisticated classifiers with higher capacities are nonlinear. Two types of nonlinear classifiers 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.

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 Fig. 6 which contains only linear and quadratic input functions. The critical thresholding condition for this system is

$$s = w_0 + x_1w_1 + x_1^2w_{11} + x_1x_2w_{12} + x_2^2w_{22} + x_2w_2 = 0. \tag{7}$$

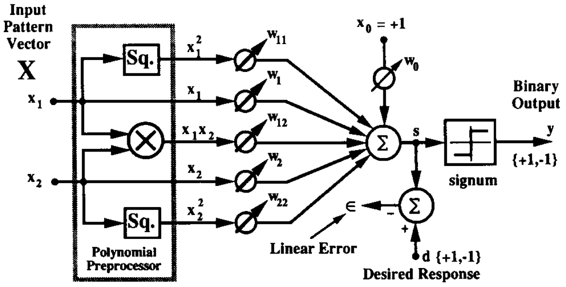


Fig. 6. Adaline with inputs mapped through nonlinearities.

With proper choice of the weights, the separating boundary in pattern space can be established as shown, for example, in Fig. 7. This represents a solution for the exclusive NOR function of (5). Of course, all of the linearly separable functions are also realizable. The use of such nonlinearities can be generalized for more than two inputs and for higher degree polynomial functions of the inputs. Some of the first work in this area was done by Specht [66]-[68] 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

[4] Underlying theory for this result was discovered independently by a number of researchers including, among others, Winder [63], Cameron [64], and Joseph [65].

[5] Patterns are in general position with respect to an Adaline with no threshold weight if any subset of pattern vectors containing no more than N_w members forms a linearly independent set or, equivalently, if no set of N_w or more input points in the N_w-dimensional pattern space lie on a homogeneous hyperplane. For the more common case involving an Adaline with a threshold weight, general position means that no set of N_w or more patterns in the (N_w − 1)-dimension pattern space lie on a hyperplane not constrained to pass through the origin [61], [46].

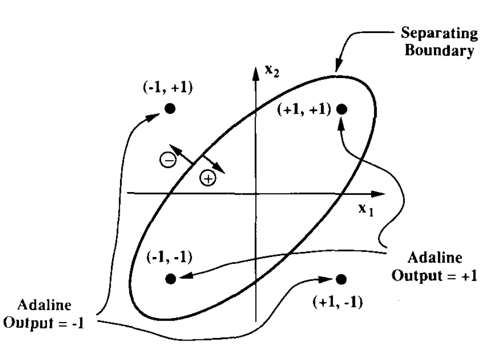


Fig. 7. Elliptical separating boundary for realizing a function which is not linearly separable.

by Barron and Barron [69]-[71] and by Ivankhnenko [72] 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 (that is, 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 [30]. 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 multidimensional Taylor series expansion of the desired 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) multidimensional 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 [73], [74]. 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 multilayer networks as single-layer networks with trainable preprocessors which are essentially self-optimizing.

Madaline I

One of the earliest trainable layered neural networks with multiple adaptive elements was the Madaline I structure of Widrow [2] and Hoff [75]. Mathematical analyses of Madaline I were developed in the Ph.D. theses of Ridgway [76], Hoff [75], and Glanz [77]. In the early 1960s, a 1000-weight Madaline I was built out of hardware [78] 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 [79].

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 Fig. 8. Two Adalines are connected to an AND logic device to provide an output.

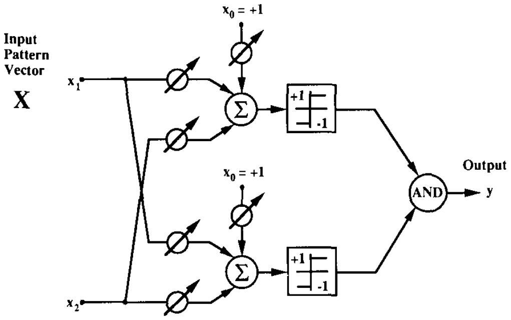


Fig. 8. Two-Adaline form of Madaline.

With weights suitably chosen, the separating boundary in pattern space for the system of Fig. 8 would be as shown in Fig. 9. This separating boundary implements the exclusive NOR function of (5).

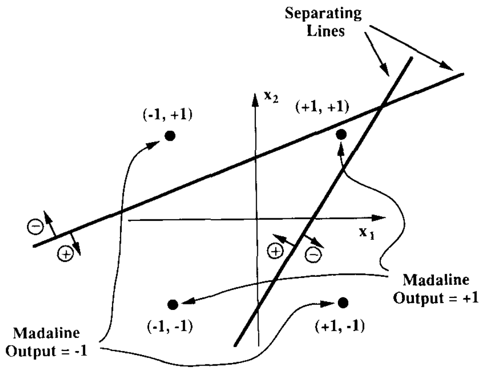


Fig. 9. Separating lines for Madaline of Fig. 8.

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 (Fig. 10) are all threshold logic functions. The given weight values will implement these three functions, but the weight choices are not unique.

Feedforward Networks

The Madalines of the 1960s had adaptive first layers and fixed threshold functions in the second (output) layers [76],

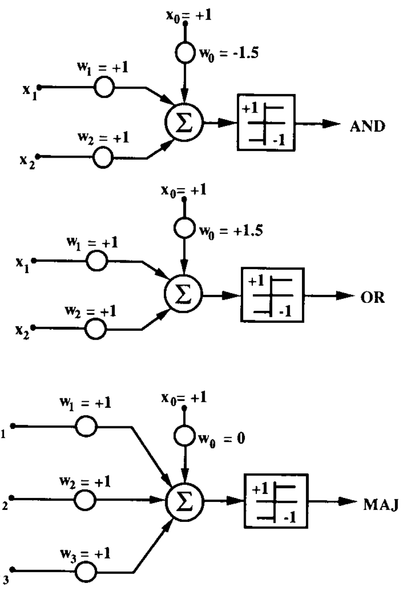


Fig. 10. Fixed-weight Adaline implementations of AND, OR, and MAJ logic functions.

[46]. The feedfoward neural networks of today often have many layers, and usually all layers are adaptive. The backpropagation networks of Rumelhart *et al.* [47] are perhaps the best-known examples of multilayer networks. A fully connected three-layer[6] feedforward adaptive network is illustrated in Fig. 11. In a fully connected layered network, each Adaline receives inputs from every output in the preceding layer.

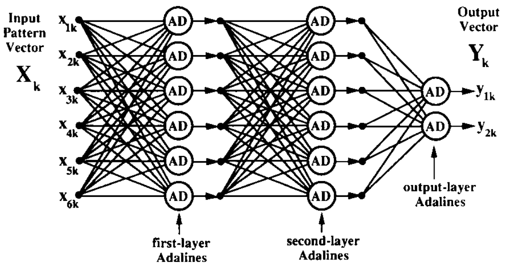


Fig. 11. Three-layer adaptive neural network.

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.

There is no reason why a feedforward network must have the layered structure of Fig. 11. In Werbos's development of the backpropagation algorithm [37], 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 "σ-π" networks that incorporate a polynomial preprocessor for each Adaline [47], [80].

One characteristic that is often desired in pattern recognition problems is invariance of the network output to changes in the position and size of the input pattern or image. Various techniques have been used to achieve translation, rotation, scale, and time invariance. One method involves including in the training set several examples of each exemplar transformed in size, angle, and position, but with a desired response that depends only on the original exemplar [78]. Other research has dealt with various Fourier and Mellin transform preprocessors [81], [82], as well as neural preprocessors [83]. Giles and Maxwell have developed a clever averaging approach, which removes unwanted dependencies from the polynomial terms in high-order threshold logic units (polynomial discriminant functions) [74] and high-order neural networks [80]. Other approaches have considered Zernike moments [84], graph matching [85], spatially repeated feature detectors [9], and time-averaged outputs [86].

Capacity of Nonlinear Classifiers

An important consideration that should be addressed when comparing various network topologies concerns the amount of information they can store.[7] Of the nonlinear classifiers mentioned above, the pattern capacity of the Adaline driven by a fixed preprocessor composed of smooth nonlinearities is the simplest to determine. If the inputs to the system are smoothly distributed in position, the outputs of the preprocessing network will be in general position with respect to the Adaline. Thus, the inputs to the Adaline will satisfy the condition required in Cover's Adaline capacity theory. Accordingly, the deterministic and statistical pattern capacities of the system are essentially equal to those of the Adaline.

The capacities of Madaline I structures, which utilize both the majoritiy element and the OR element, were experimentally estimated by Koford in the early 1960s. Although the logic functions that can be realized with these output elements are quite different, both types of elements yield essentially the same statistical storage capacity. The average number of patterns that a Madaline I network can learn to classify was found to be equal to the capacity per Adaline multiplied by the number of Adalines in the structure. The statistical capacity C_s is therefore approximately equal to twice the number of adaptive weights. Although the Madaline and the Adaline have roughly the same capacity per adaptive weight, without preprocessing the Adaline can separate only linearly separable sets, while the Madaline has no such limitation.

[6] In Rumelhart *et al.*'s terminology, this would be called a four-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.

[7] We should emphasize that the information referred to here corresponds to the maximum number of binary input/output mappings a network achieve with properly adjusted weights, not the number of bits of information that can be stored directly into the network's weights.

A great deal of theoretical and experimental work has been directed toward determining the capacity of both Adalines and Hopfield networks [87]-[90]. Somewhat less theoretical work has been focused on the pattern capacity of multilayer feedforward networks, though some knowledge exists about the capacity of two-layer networks. Such results are of particular interest because the two-layer network is surprisingly powerful. With a sufficient number of hidden elements, a signum network with two layers can implement any Boolean function.[8] Equally impressive is the power of the two-layer sigmoid network. Given a sufficient number of hidden Adaline elements, such networks can implement any continuous input-output mapping to arbitrary accuracy [92]-[94]. Although two-layer networks are quite powerful, it is likely that some problems can be solved more efficiently by networks with more than two layers. Nonfinite-order predicate mappings (such as the connectedness problem [95]) can often be computed by small networks using signal feedback [96].

In the mid-1960s, Cover studied the capacity of a feedforward signum network with an arbitrary number of layers[9] and a single output element [61], [97]. He determined a lower bound on the minimum number of weights N_w needed to enable such a network to realize any Boolean function defined over an arbitrary set of N_p patterns in general position. Recently, Baum extended Cover's result to multi-output networks, and also used a construction argument to find corresponding upper bounds for the special case of the two-layer signum network [98]. Consider a two-layer fully connected feedforward network of signum Adalines that has N_x input components (excluding the bias inputs) and N_y output components. If this network is required to learn to map any set containing N_p patterns that are in general position to any set of binary desired response vectors (with N_y components), it follows from Baum's results[10] that the minimum requisite number of weights N_w can be bounded by

$$\frac{N_y N_p}{1 + \log_2(N_p)} \le N_w < N_y\left(\frac{N_p}{N_x} + 1\right)(N_x + N_y + 1) + N_y. \tag{8}$$

From Eq. (8), it can be shown that for a two-layer feedforward network with several times as many inputs and hidden elements as outputs (say, at least 5 times as many), the deterministic pattern capacity is bounded below by something slightly smaller than N_w/N_y. It also follows from Eq. (8) that the pattern capacity of any feedforward network with a large ratio of weights to outputs (that is, N_w/N_y at least several thousand) can be bounded above by a number of somewhat larger than $(N_w/N_y)\log_2(N_w/N_y)$. Thus, the deterministic pattern capacity C_d of a two-layer network can be bounded by

$$\frac{N_w}{N_y} - K_1 \le C_d \le \frac{N_w}{N_y}\log_2\left(\frac{N_w}{N_y}\right) + K_2 \tag{9}$$

where K_1 and K_2 are positive numbers which are small terms if the network is large with few outputs relative to the number of inputs and hidden elements.

It is easy to show that Eq. (8) also bounds the number of weights needed to ensure that N_p patterns can be learned with probability 1/2, except in this case the lower bound on N_w becomes: $(N_yN_p - 1)/(1 + \log_2(N_p))$. It follows that Eq. (9) also serves to bound the statistical capacity C_s of a two-layer signum network.

It is interesting to note that the capacity bounds (9) encompass the deterministic capacity for the single-layer network comprising a bank of N_y Adalines. In this case each Adaline would have N_w/N_y weights, so the system would have a deterministic pattern capacity of N_w/N_y. As N_y becomes large, the statistical capacity also approaches N_w/N_y (for N_x finite). Until further theory on feedforward network capacity is developed, it seems reasonable to use the capacity results from the single-layer network to estimate that of multilayer networks.

Little is known about the number of binary patterns that layered sigmoid networks can learn to classify correctly. The pattern capacity of sigmoid networks cannot be smaller than that of signum networks of equal size, however, because as the weights of a sigmoid network grow toward infinity, it becomes equivalent to a signum network with a weight vector in the same direction. Insight relating to the capabilities and operating principles of sigmoid networks can be winnowed from the literature [99]-[101].

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 will have little need for a neural network. The relationship between generalization and pattern capacity represents a fundamental trade-off in neural network applications: the Adaline's inability to realize all functions is in a sense a strength rather than the fatal flaw envisioned by some critics of neural networks [95], 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 (i.e., $N_p \gg N_w/N_y$). This can be understood intuitively by noting that if the number of degrees of freedom in a network (i.e., N_w) is larger than the number of constraints associated with the desired response function (i.e., N_yN_p), the training procedure will be unable to completely constrain the weights in the network. Apparently, this allows effects of initial weight conditions to interfere with learned information and degrade the trained network's ability to generalize. A detailed analysis of generalization performance of signum networks as a function of training set size is described in [102].

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 [51], [52]. With the exception of work on the traveling salesman problem with Hopfield networks [103], this was the first neural network

[8] This can be seen by noting that any Boolean function can be written in the sum-of-products form [91], and that such an expression can be realized with a two-layer network by using the first-layer Adalines to implement AND gates, while using the second-layer Adalines to implement OR gates.

[9] Actually, the network can be an arbitrary feedforward structure and need not be layered.

[10] The upper bound used here is Baum's loose bound: minimum number hidden nodes $\le N_y\lceil N_p/N_x\rceil < N_y(N_p/N_x + 1)$.

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 more 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 to 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 that can be used to control 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; all other components are set to zero.[11]

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 paper 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 [50], system identification [50], [104], signal processing [30], or decision making [55].

III. Adaptation—The Minimal Disturbance Principle

The iterative algorithms described in this paper 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 multilayer 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.

IV. Error Correction Rules—Single Threshold Element

As adaptive algorithms evolved, principally two kinds of on-line rules have come to exist. *Error-correction rules* alter the weights of a network to correct error in the output response to the present input pattern. *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 that 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 proportional to the error itself. The one linear rule, α-LMS, is described next.

[11] The input representation often has a considerable impact on the success of a network. In NETtalk, the inputs are sparsely coded in 29 components. One might consider instead choosing a 5-bit binary representation of the 7-bit ASCII code. It should be clear, however, that in this case the sparse representation helps simplify the network's job of interpreting input characters as 29 distinct symbols. Usually the appropriate input encoding is not difficult to decide. When intuition fails, however, one sometimes must experiment with different encodings to find one that works well.

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

$$W_{k+1} = W_k + \alpha \frac{\epsilon_k X_k}{|X_k|^2}. \tag{10}$$

The time index or adaptation cycle number is k. W_{k+1} is the next value of the weight vector, W_k is the present value of the weight vector, and 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 $s_k = W_k^T X_k$ before adaptation:

$$\epsilon_k \triangleq d_k - W_k^T X_k. \tag{11}$$

Changing the weights yields a corresponding change in the error:

$$\Delta\epsilon_k = \Delta(d_k - W_k^T X_k) = -X_k^T \Delta W_k. \tag{12}$$

In accordance with the α-LMS rule of Eq. (10), the weight change is as follows:

$$\Delta W_k = W_{k+1} - W_k = \alpha \frac{\epsilon_k X_k}{|X_k|^2}. \tag{13}$$

Combining Eqs. (12) and (13), we obtain

$$\Delta\epsilon_k = -\alpha \frac{\epsilon_k X_k^T X_k}{|X_k|^2} = -\alpha\epsilon_k. \tag{14}$$

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 [30]. For input pattern vectors independent over time, stability is ensured for most practical purposes if

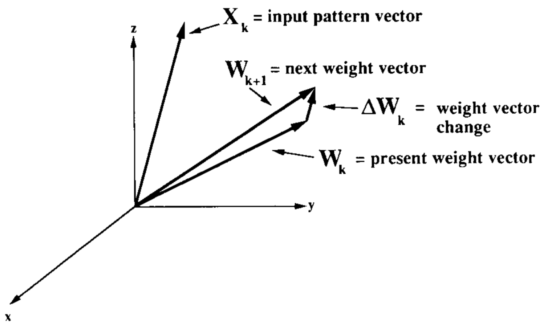
$$0 < \alpha < 2. \tag{15}$$

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

$$0.1 < \alpha < 1.0. \tag{16}$$

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 $|X_k|^2$. With binary ± 1 inputs, $|X_k|^2$ 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.

Figure 12 provides a geometrical picture of how the α-LMS rule works. In accord with Eq. (13), W_{k+1} equals W_k added to ΔW_k, and ΔW_k is parallel with the input pattern vector X_k. From Eq. (12), the change in error is equal to the negative dot product of X_k and ΔW_k. Since the α-LMS algorithm selects ΔW_k to be collinear with X_k, the desired error correction is achieved with a weight change of the smallest possible magnitude. When adapting to respond properly to a new input pattern, the responses to previous training patterns are therefore minimally disturbed, on the average.



z
X_k = input pattern vector
W_{k+1} = next weight vector
ΔW_k = weight vector change
W_k = present weight vector
y
x

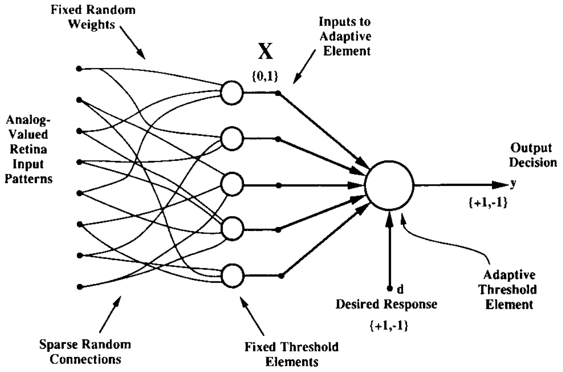
Fig. 12. Weight correction by the LMS rule.

The α-LMS algorithm corrects error, and if all input patterns are all of equal length, it minimizes mean-square error [30]. The algorithm is best known for this property.

B. Nonlinear Rules

The α-LMS algorithm is a linear rule that makes error corrections that are proportional to the error. It is known [105] 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 next sections, we describe early nonlinear rules, which were devised by Rosenblatt [106], [5] and Mays [105]. These nonlinear rules also make weight vector changes collinear with the input pattern vector (the direction which causes minimal disturbance), changes that are based on the linear error but are not directly proportional to it.

The Perceptron Learning Rule: The Rosenblatt α-Perceptron [106], [5], diagrammed in Fig. 13, 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.



Fixed Random Weights
X
{0,1}
Inputs to Adaptive Element
Analog-Valued Retina Input Patterns
Output Decision
y
{+1,-1}
Adaptive Threshold Element
d
Desired Response
{+1,-1}
Sparse Random Connections
Fixed Threshold Elements

Fig. 13. Rosenblatt's α-Perceptron.

It is interesting to note that Rosenblatt's Perceptron learning rule was first presented in 1960 [106], and Widrow and Hoff's LMS rule was first presented the same year, a few months later [59]. These rules were developed independently in 1959.

The adaptive threshold element of the α-Perceptron is shown in Fig. 14. Adapting with the Perceptron rule makes use of the "quantizer error" $\tilde{\tilde{\epsilon}}_k$, defined to be the difference between the desired response and the output of the quantizer

$$\tilde{\tilde{\epsilon}}_k \triangleq d_k - y_k. \tag{17}$$

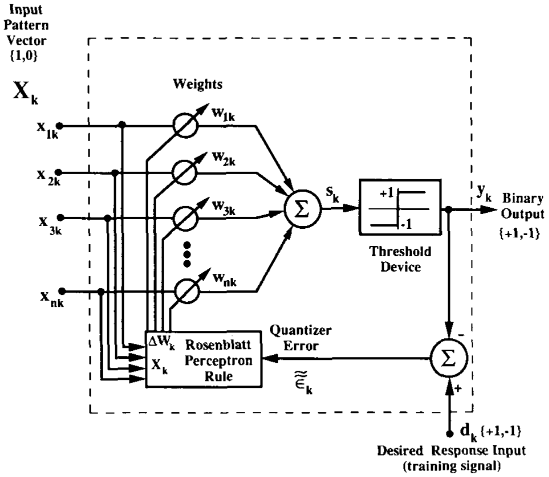


Fig. 14. The adaptive threshold element of the Perceptron.

The Perceptron rule, sometimes called the Perceptron convergence procedure, does not adapt the weights if the output decision y_k is correct, that is, if $\tilde{\tilde{\epsilon}}_k = 0$. 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 $\tilde{\tilde{\epsilon}}_k$ is positive, or subtracting the input vector from the weight vector when the error $\tilde{\tilde{\epsilon}}_k$ is negative. Thus, half the product of the input vector and the quantizer error $\tilde{\tilde{\epsilon}}_k$ 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 quantizer error $\tilde{\tilde{\epsilon}}_k/2$ is used in place of the normalized linear error $\epsilon_k/|X_k|^2$ of the α-LMS rule. The Perceptron rule is nonlinear, in contrast to the LMS rule, which is linear (compare Figs. 2 and 14). Nonetheless, the Perceptron rule can be written in a form very similar to the α-LMS rule of Eq. (10):

$$W_{k+1} = W_k + \alpha \frac{\tilde{\tilde{\epsilon}}_k}{2} X_k. \tag{18}$$

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], [107], [46], [105]. 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[12] so that even if α is very small, each adaptation can dramatically affect the switching function implemented by the Perceptron.

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—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.[13] This rule is identical to that stated in Eq. (18) except the 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.

Mays's Algorithms: In his Ph.D. thesis [105], Mays described an "increment adaptation" rule[14] 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. 14). 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 $\alpha/|X_k|^2$ 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

$$W_{k+1} = \begin{cases} W_k + \alpha \tilde{\tilde{\epsilon}}_k \dfrac{X_k}{2|X_k|^2} & \text{if } |s_k| \geq \gamma \\ W_k + \alpha d_k \dfrac{X_k}{|X_k|^2} & \text{if } |s_k| < \gamma \end{cases} \tag{19}$$

where $\tilde{\tilde{\epsilon}}_k$ is the quantizer error of Eq. (17).

With the dead zone $\gamma = 0$, Mays's increment adaptation algorithm reduces to a normalized version of the Percep-

[12] 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 situation occurs only rarely if the weight vector is much longer than the weight increment vector.

[13] The terms "fixed-increment" and "absolute correction" are due to Nilsson [46]. Rosenblatt referred to methods of these types, respectively, as quantized and nonquantized learning rules.

[14] The increment adaptation rule was proposed by others before Mays, though from a different perspective [107].

tron rule (18). 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 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

$$W_{k+1} = \begin{cases} W_k & \text{if } \tilde{\epsilon}_k = 0 \text{ and } |s_k| \geq \gamma \\ W_k + \alpha\epsilon_k \dfrac{X_k}{|X_k|^2} & \text{otherwise} \end{cases} \tag{20}$$

where $\tilde{\epsilon}_k$ is the quantizer error of Eq. (17).

If the dead zone γ is set to ∞, this algorithm reduces to the α-LMS algorithm (10). Mays showed that, for dead zone $0 < \gamma < 1$ and learning rate $0 < \alpha \leq 2$, 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 believe 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. Multilayer network adaptation procedures that use some of these algorithms are discussed in the following.

V. 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 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.

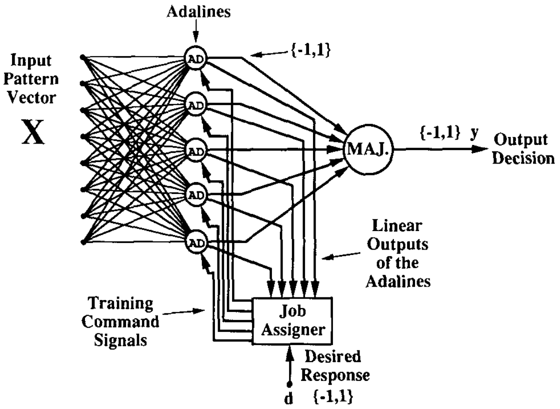


Fig. 15. A five-Adaline example of the Madaline I architecture.

Figure 15 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. 15 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—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 differing 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 Fig. 15, 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 weights 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 [76] 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 adaptive system of Fig. 15 was suggested by common sense, and was found to work well in simulations. Ridgway found that the probability that a given Adaline will be adapted in response to an input pattern is greatest if that element had taken such responsibility during the previous adapt cycle when the pattern was most recently presented. The division of responsibility stabilizes at the same time that the responses of individual elements stabilize to their share of the load. When the training problem is not perfectly separable by this system, the adaptation process tends to minimize error probability, although it is possible for the algorithm to "hang up" on local optima.

The Madaline structure of Fig. 15 has 2 layers—the first layer consists of adaptive logic elements, the second of fixed logic. A variety of fixed-logic devices could be used for the second layer. A variety of MRI adaptation rules were devised by Hoff [75] that can be used with all possible fixed-logic output elements. An easily described training procedure results when the output element is an OR gate. During training, if the desired output for a given input pattern is +1, only the one Adaline whose linear output is closest to zero would be adapted if any adaptation is needed—in other words, if all Adalines give −1 outputs. If the desired output is −1, all elements must give −1 outputs, and any giving +1 outputs must be adapted.

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 multilayer binary networks by Winter and Widrow with the introduction of Madaline Rule II (MRII) [43], [83], [108]. A typical two-layer MRII network is shown in Fig. 16. The weights in both layers are adaptive.

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.

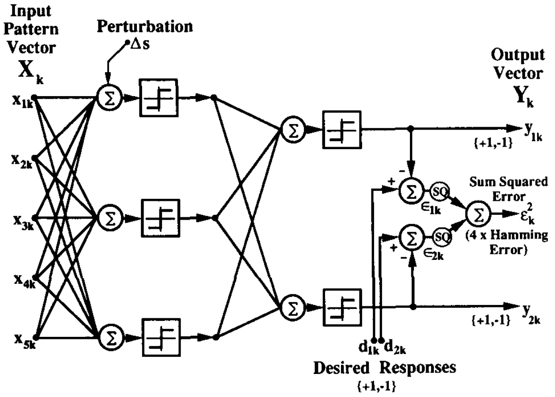


Fig. 16. Typical two-layer Madaline II architecture.

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. 16). If the output Hamming error is reduced by this bit inversion, that is, 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 that 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, and so on, 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.

VI. Steepest-Descent Rules—Single Threshold Element

Thus far, we have described a variety of adaptation rules that act to reduce 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 [109]–[111]. The most popular approaches to MSE 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 [30], [112]–[114] and conjugate gradient [114], [115] 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 [116], [117]. 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 W_0 for the system's weight vector. The gradient of the MSE function is measured and the weight vector is altered in the direction corresponding 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

$$W_{k+1} = W_k + \mu(-\nabla_k) \tag{21}$$

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 $W = W_k$.

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.

Mean-Square Error Surface of the Linear Combiner: In this section we demonstrate that the MSE surface of the linear combiner of Fig. 1 is a quadratic function of the weights, and thus easily traversed by gradient descent.

Let the input pattern 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 MSE since it is time-varying. The only possibility is an ensemble average, defined below.

At the kth iteration, let the weight vector be W_k. Squaring and expanding Eq. (11) yields

$$\epsilon_k^2 = (d_k - X_k^T W_k)^2 \tag{22}$$

$$= d_k^2 - 2d_k X_k^T W_k + W_k^T X_k X_k^T W_k. \tag{23}$$

Now assume an ensemble of identical adaptive linear combiners, each having the same weight vector W_k at the kth iteration. Let each combiner have individual inputs X_k and d_k derived from stationary ergodic ensembles. Each combiner will produce an individual error ϵ_k represented by Eq. (23). Averaging Eq. (23) over the ensemble yields

$$E[\epsilon_k^2]_{W=W_k} = E[d_k^2] - 2E[d_k X_k^T] W_k + W_k^T E[X_k X_k^T] W_k. \tag{24}$$

Defining the vector P as the crosscorrelation between the desired response (a scalar) and the X-vector[15] then yields

$$P^T \triangleq E[d_k X_k^T] = E[d_k, d_k x_{1k}, \cdots, d_k x_{nk}]^T. \tag{25}$$

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

$$R \triangleq E[X_k X_k^T]$$

$$= E\begin{bmatrix} 1 & x_{1k} & \cdots & x_{nk} \\ x_{1k} & x_{1k}x_{1k} & \cdots & x_{1k}x_{nk} \\ \vdots & \vdots & & \vdots \\ x_{nk} & x_{nk}x_{1k} & \cdots & x_{nk}x_{nk} \end{bmatrix}. \tag{26}$$

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

$$\xi_k \triangleq E[\epsilon_k^2]_{W=W_k}$$

$$= E[d_k^2] - 2P^T W_k + W_k^T R W_k. \tag{27}$$

Note that the MSE is a quadratic function of the weights. It is a convex hyperparaboloidal surface, a function that never goes negative. Figure 17 shows a typical MSE 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 MSE over the training set when the Adaline's weights are 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.

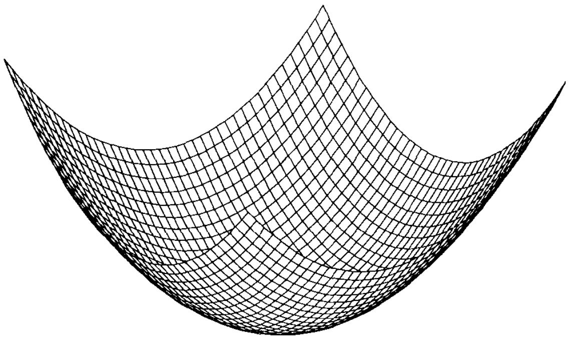


Fig. 17. Typical mean-square-error surface of a linear combiner.

The gradient ∇_k of the MSE function with $W = W_k$ is obtained by differentiating Eq. (27):

$$\nabla_k \triangleq \left\{ \begin{matrix} \dfrac{\partial E[\epsilon_k^2]}{\partial w_{0k}} \\ \vdots \\ \dfrac{\partial E[\epsilon_k^2]}{\partial w_{nk}} \end{matrix} \right\}_{W=W_k} = -2P + 2RW_k. \tag{28}$$

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

This is a linear function of the weights. The optimal weight vector $\mathbf{W}^*$, generally called the Wiener weight vector, is obtained from Eq. (28) by setting the gradient to zero:

$$\mathbf{W}^* = \mathbf{R}^{-1}\mathbf{P}. \tag{29}$$

This is a matrix form of the Wiener-Hopf equation [118]-[120]. 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 $\mathbf{P}$.

The μ-LMS Algorithm: The μ-LMS algorithm works by performing approximate steepest descent on the MSE surface in weight space. Because it is a quadratic function of the weights, this surface is convex and has a unique (global) minimum.[16] An instantaneous gradient based upon the square of the instantaneous linear error is

$$\hat{\nabla}_k = \frac{\partial \epsilon_k^2}{\partial \mathbf{W}_k} = \left\{\begin{matrix} \dfrac{\partial \epsilon_k^2}{\partial w_{0k}} \\ \vdots \\ \dfrac{\epsilon_k^2}{\partial w_{nk}} \end{matrix}\right\}. \tag{30}$$

LMS works by using this crude gradient estimate in place of the true gradient ∇_k of Eq. (28). Making this replacement into Eq. (21) yields

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k - \mu \frac{\partial \epsilon_k^2}{\partial \mathbf{W}_k}. \tag{31}$$

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 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. (31) and replacing the linear error by definition (11) gives

$$\begin{aligned} \mathbf{W}_{k+1} &= \mathbf{W}_k - 2\mu\epsilon_k \frac{\partial \epsilon_k}{\partial \mathbf{W}_k} \\ &= \mathbf{W}_k - 2\mu\epsilon_k \frac{\partial(d_k - \mathbf{W}_k^T\mathbf{X}_k)}{\partial \mathbf{W}_k}. \end{aligned} \tag{32}$$

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

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\epsilon_k\mathbf{X}_k. \tag{33}$$

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 [30] for most practical purposes if

$$0 < \mu < \frac{1}{\text{trace } [\mathbf{R}]} \tag{34}$$

where trace $[\mathbf{R}] = \Sigma$(diagonal elements of $\mathbf{R}$) is the average signal power of the $\mathbf{X}$-vectors, that is, $E(\mathbf{X}^T\mathbf{X})$. With μ set within this range,[17] 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 [30].

In the μ-LMS algorithm, and other iterative steepest-descent procedures, use of the instantaneous gradient 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

$$\begin{aligned} -\sum_{l=0}^{K-1} \frac{\partial \epsilon_{k+l}^2}{\partial \mathbf{W}_{k+l}} &\simeq -K\left(\frac{1}{K}\sum_{l=0}^{K-1} \frac{\partial \epsilon_{k+l}^2}{\partial \mathbf{W}_k}\right) \\ &= -K\frac{\partial}{\partial \mathbf{W}_k}\left(\frac{1}{K}\sum_{l=0}^{K-1} \epsilon_{k+l}^2\right) \\ &\simeq -K\frac{\partial \xi}{\partial \mathbf{W}_k} \end{aligned} \tag{35}$$

where ξ denotes the MSE function. Thus, on average the weights follow the true gradient. It is shown in [30] that the instantaneous gradient is an unbiased estimate of the true gradient.

Comparison of μ-LMS and α-LMS: We have now presented two forms of the LMS algorithm, α-LMS (10) in Section IV-A and μ-LMS (33) in the last section. 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 $\mathbf{R}$ are highly disparate, giving faster convergence for a given level of gradient noise[18] 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 MSE solution, while α-LMS may converge to a somewhat biased solution.

We begin with α-LMS of Eq. (10):

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha\frac{\epsilon_k\mathbf{X}_k}{|\mathbf{X}_k|^2} \tag{36}$$

Replacing the error with its definition (11) and rearranging terms yields

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha\frac{(d_k - \mathbf{W}_k^T\mathbf{X}_k)\mathbf{X}_k}{|\mathbf{X}_k|^2} \tag{37}$$

$$= \mathbf{W}_k + \alpha\left(\frac{d_k}{|\mathbf{X}_k|} - \mathbf{W}_k^T\frac{\mathbf{X}_k}{|\mathbf{X}_k|}\right)\frac{\mathbf{X}_k}{|\mathbf{X}_k|}. \tag{38}$$

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,[19]

$$\begin{aligned} \tilde{\mathbf{X}}_k &\triangleq \frac{\mathbf{X}_k}{|\mathbf{X}_k|} \\ \tilde{d}_k &\triangleq \frac{d_k}{|\mathbf{X}_k|}. \end{aligned} \tag{39}$$

[16] If the autocorrelation matrix of the pattern vector set has m zero eigenvalues, the minimum MSE solution will be an m-dimensional subspace in weight space [30].

[17] Horowitz and Senne [121] have proven that (34) is not sufficient in general to guarantee 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 $[\mathbf{R}]$).

[18] Gradient noise is the difference between the gradient estimate and the true gradient.

[19] The idea of a normalized training set was suggested by Derrick Nguyen.

Eq. (38) then becomes

$$W_{k+1} = W_k + \alpha(\tilde{\tilde{d}}_k - W_k^T\tilde{\tilde{X}}_k)\tilde{\tilde{X}}_k. \tag{40}$$

This is the μ-LMS rule of Eq. (33) 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 (39). The solution that will be reached by the μ-LMS algorithm is the Wiener solution of this training set

$$\tilde{\tilde{W}}^* = (\tilde{\tilde{R}})^{-1}\tilde{\tilde{P}} \tag{41}$$

where

$$\tilde{\tilde{R}} = E[\tilde{\tilde{X}}_k\tilde{\tilde{X}}_k^T] \tag{42}$$

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

$$\tilde{\tilde{P}} = E[\tilde{\tilde{d}}_k\tilde{\tilde{X}}_k] \tag{43}$$

is the crosscorrelation 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. (15) can be computed from the corresponding range for μ given in Eq. (34) by replacing R and μ in Eq. (34) by $\tilde{\tilde{R}}$ and $\alpha/2$, respectively, and then noting that trace$[\tilde{\tilde{R}}]$ is equal to one:

$$0 < \alpha < \frac{2}{\text{trace}[\tilde{\tilde{R}}]}, \text{ or}$$

$$0 < \alpha < 2. \tag{44}$$

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 =.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.

Fig. 18 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:

$$y_k = \tanh\ (s_k) = \left(\frac{1 - e^{-2s_k}}{1 + e^{-2s_k}}\right). \tag{45}$$

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

$$\tilde{\epsilon}_k \triangleq d_k - y_k = d_k - \text{sgm}\ (s_k). \tag{46}$$

Backpropagation for the Sigmoid Adaline: Our objective is to minimize $E[(\tilde{\epsilon}_k)^2]$, 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. (33).

Referring to Fig. 18, the instantaneous gradient estimate obtained during presentation of the kth input vector X_k is given by

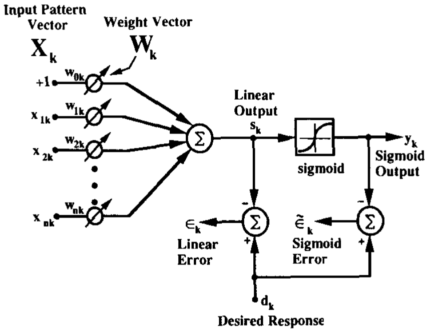


Fig. 18. Adaline with sigmoidal nonlinearity.

$$\hat{\nabla}_k = \frac{\partial(\tilde{\epsilon}_k)^2}{\partial W_k} = 2\tilde{\epsilon}_k\frac{\partial\tilde{\epsilon}_k}{\partial W_k}. \tag{47}$$

Differentiating Eq. (46) yields

$$\frac{\partial\tilde{\epsilon}_k}{\partial W_k} = -\frac{\partial\ \text{sgm}\ (s_k)}{\partial W_k} = -\text{sgm}'\ (s_k)\frac{\partial s_k}{\partial W_k}. \tag{48}$$

We may note that

$$s_k = X_k^T W_k. \tag{49}$$

Therefore,

$$\frac{\partial s_k}{\partial W_k} = X_k. \tag{50}$$

Substituting into Eq. (48) gives

$$\frac{\partial\tilde{\epsilon}_k}{\partial W_k} = -\text{sgm}'\ (s_k)\,X_k. \tag{51}$$

Inserting this into Eq. (47) yields

$$\hat{\nabla}_k = -2\tilde{\epsilon}_k\ \text{sgm}'\ (s_k)\,X_k. \tag{52}$$

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

$$W_{k+1} = W_k + \mu(-\hat{\nabla}_k) \tag{53}$$

$$= W_k + 2\mu\tilde{\epsilon}_k\ \text{sgm}'\ (s_k)\,X_k. \tag{54}$$

Algorithm (54) is the backpropagation algorithm for the sigmoid Adaline element. The backpropagation name makes more sense when the algorithm is utilized in a lay-

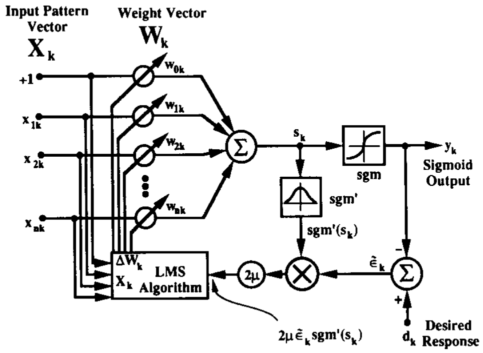


Fig. 19. Implementation of backpropagation for the sigmoid Adaline element.

ered network, which will be studied below. Implementation of algorithm (54) is illustrated in Fig. 19.

If the sigmoid is chosen to be the hyperbolic tangent function (45), then the derivative sgm′ (s_k) is given by

$$\begin{aligned} \text{sgm}'(s_k) &= \frac{\partial(\tanh(s_k))}{\partial s_k} \\ &= 1 - (\tanh(s_k))^2 = 1 - y_k^2. \end{aligned} \tag{55}$$

Accordingly, Eq. (54) becomes

$$W_{k+1} = W_k + 2\mu\tilde{\epsilon}_k(1 - y_k^2)X_k. \tag{56}$$

Madaline Rule III for the Sigmoid Adaline: The implementation of algorithm (54) (Fig. 19) 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 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 Fig. 20. 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 $\tilde{\epsilon}_k$ is noted.

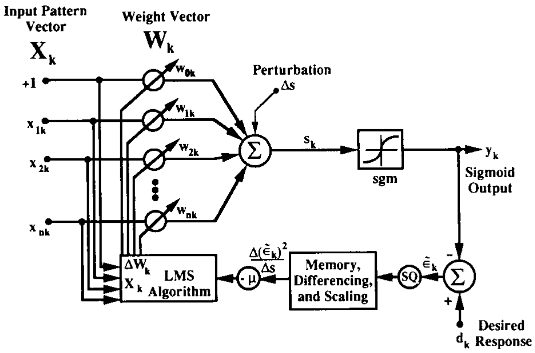


Fig. 20. Implementation of the MRIII algorithm for the sigmoid Adaline element.

An instantaneous estimated gradient can be obtained as follows:

$$\hat{\nabla}_k = \frac{\partial(\tilde{\epsilon}_k)^2}{\partial W_k} = \frac{\partial(\tilde{\epsilon}_k)^2}{\partial s_k}\frac{\partial s_k}{\partial W_k} = \frac{\partial(\tilde{\epsilon}_k)^2}{\partial s_k}X_k. \tag{57}$$

Since Δs is small,

$$\hat{\nabla}_k \simeq \left(\frac{\Delta(\tilde{\epsilon}_k)^2}{\Delta s}\right)X_k. \tag{58}$$

Another way to obtain an approximate instantaneous gradient by measuring the effects of the perturbation Δs can be obtained from Eq. (57).

$$\hat{\nabla}_k = \frac{\partial(\tilde{\epsilon}_k)^2}{\partial s_k}X_k = 2\tilde{\epsilon}_k\frac{\partial\tilde{\epsilon}_k}{\partial s_k}X_k \simeq 2\tilde{\epsilon}_k\left(\frac{\Delta\tilde{\epsilon}_k}{\Delta s}\right)X_k. \tag{59}$$

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:

$$W_{k+1} = W_k - \mu\left(\frac{\Delta(\tilde{\epsilon}_k)^2}{\Delta s}\right)X_k \tag{60}$$

or,

$$W_{k+1} = W_k - 2\mu\tilde{\epsilon}_k\left(\frac{\Delta\tilde{\epsilon}_k}{\Delta s}\right)X_k. \tag{61}$$

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. (60) is illustrated in Fig. 20.

Regarding algorithm (61), some changes can be made to establish a point of interest. Note that, in accord with Eq. (46),

$$\tilde{\epsilon}_k = d_k - y_k. \tag{62}$$

Adding the perturbation Δs causes a change in ϵ_k equal to

$$\Delta\tilde{\epsilon}_k = -\Delta y_k. \tag{63}$$

Now, Eq. (61) may be rewritten as

$$W_{k+1} = W_k + 2\mu\tilde{\epsilon}_k\left(\frac{\Delta y_k}{\Delta s}\right)X_k. \tag{64}$$

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

$$\begin{aligned} W_{k+1} &\simeq W_k + 2\mu\tilde{\epsilon}_k\frac{\partial y_k}{\partial s_k}X_k && (65) \\ &= W_k + 2\mu\tilde{\epsilon}_k \,\text{sgm}'(s_k)X_k. && (66) \end{aligned}$$

This is identical to the backpropagation algorithm (54) 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.

MSE Surfaces of the Adaline: Fig. 21 shows a linear combiner connected to both sigmoid and signum devices. Three errors, ϵ, $\tilde{\epsilon}_k$, and $\tilde{\tilde{\epsilon}}$ are designated in this figure. They are:

$$\begin{aligned} \text{linear error} &= \epsilon = d - s \\ \text{sigmoid error} &= \tilde{\epsilon} = d - \text{sgm}(s) \\ \text{signum error} &= \tilde{\tilde{\epsilon}} = d - \text{sgn}(\text{sgm}(s)) \\ &= d - \text{sgn}(s). \end{aligned} \tag{67}$$

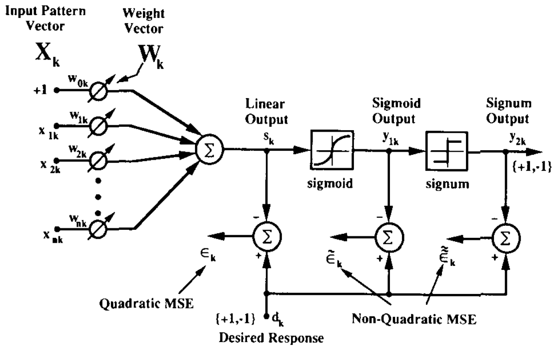


Fig. 21. The linear, sigmoid, and signum errors of the Adaline.

To demonstrate the nature of the square error surfaces associated with these three types of error, a simple experiment with a two-input Adaline was performed. The Adaline was driven by a typical set of input patterns and their associated binary $\{+1, -1\}$ desired responses. The sigmoid function used was the hyperbolic tangent. The weights could have been adapted to minimize the mean-square error of ϵ, $\tilde{\epsilon}$, or $\tilde{\tilde{\epsilon}}$. The MSE surfaces of $E[(\epsilon)^2]$, $E[(\tilde{\epsilon})^2]$, $E[(\tilde{\tilde{\epsilon}})^2]$ plotted as functions of the two weight values, are shown in Figs. 22, 23, and 24, respectively.

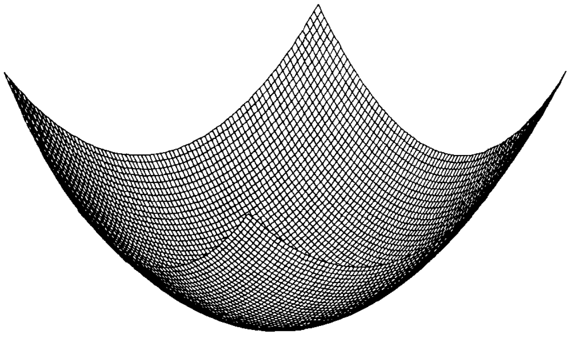


Fig. 22. Example MSE surface of linear error.

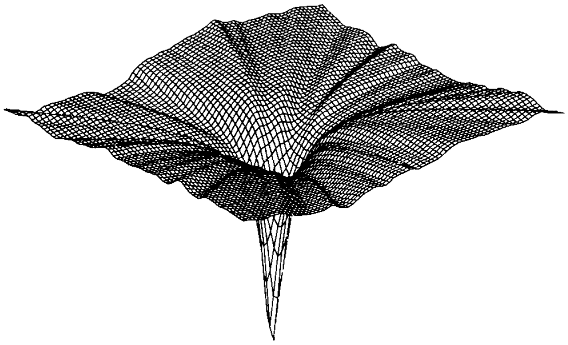


Fig. 23. Example MSE surface of sigmoid error.

Although the above experiment is not all encompassing, we can infer from it that minimizing the mean square of the linear error is easy and minimizing the mean square of the sigmoid error is more difficult, but typically much easier than minimizing the mean square of the signum error. Only the linear error is guaranteed to have an MSE surface with a unique global minimum (assuming invertible $\boldsymbol{R}$-matrix). The other MSE surfaces can have local optima [122], [123].

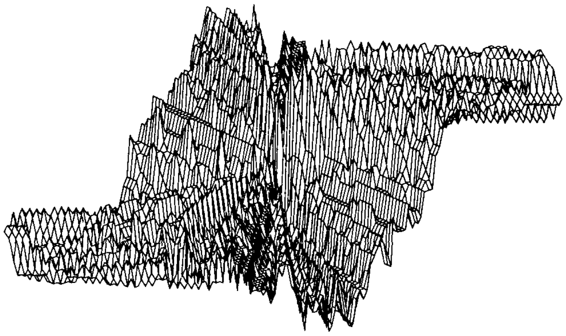


Fig. 24. Example MSE surface of signum error.

In nonlinear neural networks, gradient methods generally work better with sigmoid rather than signum nonlinearities. Smooth nonlinearities are required by the MRIII and backpropagation techniques. Moreover, sigmoid networks are capable of forming internal representations that are more complex than simple binary codes and, thus, these networks can often form decision regions that are more sophisticated than those associated with similar signum networks. In fact, if a noiseless infinite-precision sigmoid Adaline could be constructed, it would be able to convey an infinite amount of information at each time step. This is in contrast to the maximum Shannon information capacity of one bit associated with each binary element.

The signum does have some advantages over the sigmoid in that it is easier to implement in hardware and much simpler to compute on a digital computer. Furthermore, the outputs of signums are binary signals which can be efficiently manipulated by digital computers. In a signum network with binary inputs, for instance, the output of each linear combiner can be computed without performing weight multiplications. This involves simply adding together the values of weights with +1 inputs and subtracting from this the values of all weights that are connected to −1 inputs.

Sometimes a signum is used in an Adaline to produce decisive output decisions. The error probability is then proportional to the mean square of the output error $\tilde{\tilde{\epsilon}}$. To minimize this error probability approximately, one can easily minimize $E[(\epsilon)^2]$ instead of directly minimizing $E[(\tilde{\tilde{\epsilon}})^2]$ [58]. However, with only a little more computation one could minimize $E[(\tilde{\epsilon})^2]$ and typically come much closer to the objective of minimizing $E[(\tilde{\tilde{\epsilon}})^2]$. The sigmoid can therefore be used in training the weights even when the signum is used to form the Adaline output, as in Fig. 21.

VII. 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.

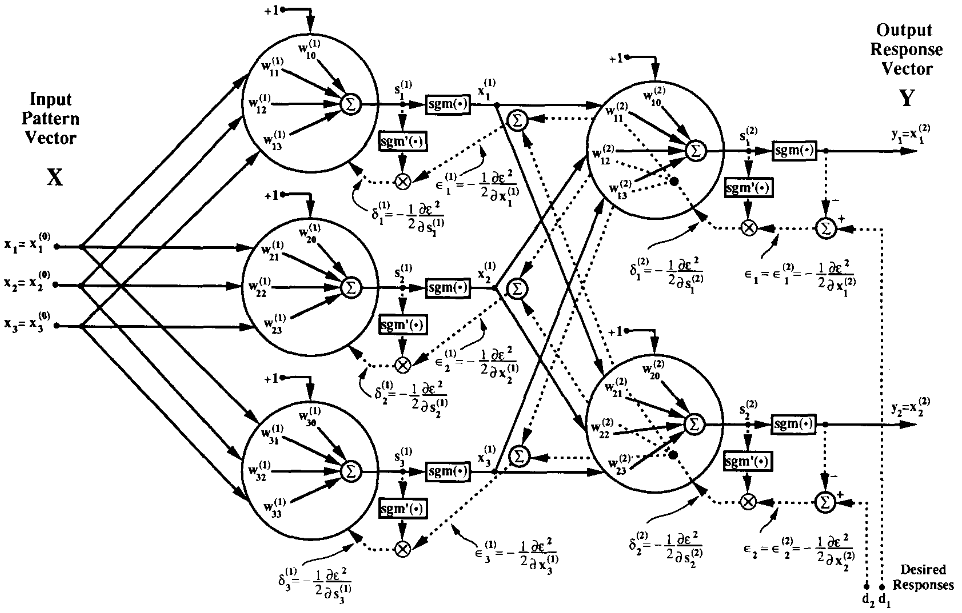

Input
Pattern
Vector
X
$x_1 = x_1^{(0)}$
$x_2 = x_2^{(0)}$
$x_3 = x_3^{(0)}$
+1
$w_{10}^{(1)}$
$w_{11}^{(1)}$
$w_{12}^{(1)}$
$w_{13}^{(1)}$
$w_{20}^{(1)}$
$w_{21}^{(1)}$
$w_{22}^{(1)}$
$w_{23}^{(1)}$
$w_{30}^{(1)}$
$w_{31}^{(1)}$
$w_{32}^{(1)}$
$w_{33}^{(1)}$
Σ
$s_1^{(1)}$
$s_2^{(1)}$
$s_3^{(1)}$
sgm(•)
sgm'(•)
$x_1^{(1)}$
$x_2^{(1)}$
$x_3^{(1)}$
$\epsilon_1^{(1)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial x_1^{(1)}}$
$\epsilon_2^{(1)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial x_2^{(1)}}$
$\epsilon_3^{(1)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial x_3^{(1)}}$
$\delta_1^{(1)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_1^{(1)}}$
$\delta_2^{(1)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_2^{(1)}}$
$\delta_3^{(1)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_3^{(1)}}$
$w_{10}^{(2)}$
$w_{11}^{(2)}$
$w_{12}^{(2)}$
$w_{13}^{(2)}$
$w_{20}^{(2)}$
$w_{21}^{(2)}$
$w_{22}^{(2)}$
$w_{23}^{(2)}$
$s_1^{(2)}$
$s_2^{(2)}$
$\delta_1^{(2)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_1^{(2)}}$
$\delta_2^{(2)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_2^{(2)}}$
$\epsilon_1 = \epsilon_1^{(2)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial x_1^{(2)}}$
$\epsilon_2 = \epsilon_2^{(2)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial x_2^{(2)}}$
+
−
Output
Response
Vector
Y
$y_1 = x_1^{(2)}$
$y_2 = x_2^{(2)}$
Desired
Responses
d_2 d_1

Fig. 25. Example two-layer backpropagation network architecture.

A. Backpropagation for Networks

The publication of the backpropagation technique by Rumelhart *et al.* [42] 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 exclusively 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 present an informal derivation of the algorithm and illustrate how it works for the simple network shown in Fig. 25.

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

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \boldsymbol{W}_k} = \begin{Bmatrix} \dfrac{\partial \varepsilon_k^2}{\partial w_{1k}} \\ \vdots \\ \dfrac{\varepsilon_k^2}{\partial w_{mk}} \end{Bmatrix}. \tag{68}$$

Now, however, $\boldsymbol{W}_k$ is a long m-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 N_y outputs of the network. Thus

$$\varepsilon_k^2 = \sum_{i=1}^{N_y} \epsilon_{ik}^2. \tag{69}$$

In the network example shown in Fig. 25, the sum square error is given by

$$\varepsilon^2 = (d_1 - y_1)^2 + (d_2 - y_2)^2 \tag{70}$$

where we now suppress the time index k for convenience.

In its simplest form, backpropagation training begins by presenting an input pattern vector $\boldsymbol{X}$ to the network, sweeping forward through the system to generate an output response vector $\boldsymbol{Y}$, 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.[20]

We can get some idea about what is involved in the calculations associated with the backpropagation algorithm by examining the network of Fig. 25. 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 Fig. 26. The solid lines in these diagrams represent forward signal paths through the network,

[20] 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 [100]. 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.

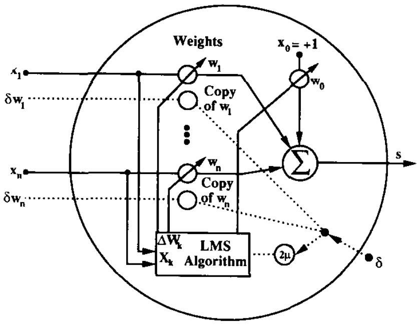


Fig. 26. Detail of linear combiner and associated circuitry in backpropagation network.

and the dotted lines represent the separate backward paths that are used in association with calculations of the square error derivatives δ. From Fig. 25, we see that the calculations associated with the backward sweep are of a complexity 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 no weight multiplications in the first layer.

As stated earlier, 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[21]

$$\delta_j^{(l)} \triangleq -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_j^{(l)}}. \tag{71}$$

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 instantaneous square-error derivatives are first computed for each element in the output layer. The calculation is simple. As an example, below we derive the required expression for $\delta_1^{(2)}$, the derivative associated with the top Adaline element in the output layer of Fig. 25. We begin with the definition of $\delta_1^{(2)}$ from Eq. (71)

$$\delta_1^{(2)} \triangleq -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_1^{(2)}}. \tag{72}$$

Expanding the squared-error term ε^2 by Eq. (70) yields

$$\delta_1^{(2)} = -\frac{1}{2}\frac{\partial((d_1 - y_1)^2 + (d_2 - y_2)^2)}{\partial s_1^{(2)}} \tag{73}$$

$$= -\frac{1}{2}\frac{\partial(d_1 - \text{sgm }(s_1^{(2)}))^2}{\partial s_1^{(2)}}$$

$$-\frac{1}{2}\frac{\partial(d_2 - \text{sgm }(s_1^{(2)}))^2}{\partial s_1^{(2)}}. \tag{74}$$

We note that the second term is zero. Accordingly,

$$\delta_1^{(2)} = -\frac{1}{2}\frac{\partial(d_1 - \text{sgm }(s_1^{(2)}))^2}{\partial s_1^{(2)}}. \tag{75}$$

Observing that d_1 and $s_1^{(2)}$ are independent yields

$$\delta_1^{(2)} = -(d_1 - \text{sgm }(s_1^{(2)}))\frac{\partial(-\text{sgm }(s_1^{(2)}))}{\partial s_1^{(2)}} \tag{76}$$

$$= (d_1 - \text{sgm }(s_1^{(2)}))\ \text{sgm}'\ (s_1^{(2)}). \tag{77}$$

We denote the error $d_1 - \text{sgm }(s_1^{(2)})$, by $\epsilon_1^{(2)}$. Therefore,

$$\delta_1^{(2)} = \epsilon_1^{(2)}\ \text{sgm}'\ (s_1^{(2)}). \tag{78}$$

Note that this corresponds to the computation of $\delta_1^{(2)}$ as illustrated in Fig. 25. The value of δ associated with the other output element in the figure can be expressed in an analogous fashion. Thus each square-error derivative δ in the output layer is computed by multiplying the output error associated with that element by the derivative of the associated sigmoidal nonlinearity. Note from Eq. (55) that if the sigmoid function is the hyperbolic tangent, Eq. (78) becomes simply

$$\delta_1^{(2)} = \epsilon_1^{(2)}(1 - (y_1)^2). \tag{79}$$

Developing expressions for the square-error derivatives associated with hidden layers is not much more difficult (refer to Fig. 25). We need an expression for $\delta_1^{(1)}$, the square-error derivative associated with the top element in the first layer of Fig. 25. The derivative $\delta_1^{(1)}$ is defined by

$$\delta_1^{(1)} \triangleq -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_1^{(1)}}. \tag{80}$$

Expanding this by the chain rule, noting that ε^2 is determined entirely by the values of $s_1^{(2)}$ and $s_2^{(2)}$, yields

$$\delta_1^{(1)} = -\frac{1}{2}\left(\frac{\partial \varepsilon^2}{\partial s_1^{(2)}}\frac{\partial s_1^{(2)}}{\partial s_1^{(1)}} + \frac{\partial \varepsilon^2}{\partial s_2^{(2)}}\frac{\partial s_2^{(2)}}{\partial s_1^{(1)}}\right). \tag{81}$$

Using the definitions of $\delta_1^{(2)}$ and $\delta_2^{(2)}$, and then substituting expanded versions of Adaline linear outputs $s_1^{(2)}$ and $s_2^{(2)}$ gives

$$\delta_1^{(1)} = \delta_1^{(2)}\frac{\partial s_1^{(2)}}{\partial s_1^{(1)}} + \delta_2^{(2)}\frac{\partial s_2^{(2)}}{\partial s_1^{(1)}} \tag{82}$$

$$= \delta_1^{(2)}\frac{\partial}{\partial s_1^{(1)}}\left(w_{10}^{(2)} + \sum_{i=1}^{3} w_{1i}^{(2)}\ \text{sgm }(s_i^{(1)})\right)$$

$$+ \delta_2^{(2)}\frac{\partial}{\partial s_1^{(1)}}\left(w_{20}^{(2)} + \sum_{i=1}^{3} w_{2i}^{(2)}\ \text{sgm }(s_i^{(1)})\right). \tag{83}$$

Noting that $\partial[\text{sgm }(s_i^{(l)})]/\partial s_j^{(l)} = 0,\ i \neq j$, leaves

$$\delta_1^{(1)} = \delta_1^{(2)}w_{11}^{(2)}\ \text{sgm}'\ (s_1^{(1)}) + \delta_2^{(2)}w_{21}^{(2)}\ \text{sgm}'\ (s_1^{(1)}) \tag{84}$$

$$= [\delta_1^{(2)}w_{11}^{(2)} + \delta_2^{(2)}w_{21}^{(2)}]\ \text{sgm}'\ (s_1^{(1)}). \tag{85}$$

Now, we make the following definition:

$$\epsilon_1^{(1)} \triangleq \delta_1^{(2)}w_{11}^{(2)} + \delta_2^{(2)}w_{21}^{(2)}. \tag{86}$$

Accordingly,

$$\delta_1^{(1)} = \epsilon_1^{(1)}\ \text{sgm}'\ (s_1^{(1)}). \tag{87}$$

Referring to Fig. 25, we can trace through the circuit to verify that $\delta_1^{(1)}$ is computed in accord with Eqs. (86) and (87).

[21] In Fig. 25, 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.

The easiest way to find values of δ for all the Adaline elements in the network is to follow the schematic diagram of Fig. 25.

Thus, 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 that connects it to the given Adaline. These weighted square-error derivatives are then added together, producing an error term $\epsilon^{(l)}$, 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. This is easily shown at each layer by repeating the chain rule argument associated with Eq. (81).

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

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k}. \tag{88}$$

This can be written as

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k} = \frac{\partial \varepsilon_k^2}{\partial s_k}\frac{\partial s_k}{\partial \mathbf{W}_k}. \tag{89}$$

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

$$\frac{\partial s_k}{\partial \mathbf{W}_k} = \frac{\partial \mathbf{W}_k^T\mathbf{X}_k}{\partial \mathbf{W}_k} = \mathbf{X}_k. \tag{90}$$

Therefore,

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial s_k}\mathbf{X}_k. \tag{91}$$

For this element,

$$\delta_k = -\frac{1}{2}\frac{\partial \varepsilon_k^2}{\partial s_k}. \tag{92}$$

Accordingly,

$$\hat{\nabla}_k = -2\delta_k\mathbf{X}_k. \tag{93}$$

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

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k + 2\mu\delta_k\mathbf{X}_k. \tag{94}$$

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. Eq. (94) and the means for finding δ_k comprise the general weight update rule of the backpropagation algorithm.

There is a great similarity between Eq. (94) and the μ-LMS algorithm (33), 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. (77) into Eq. (94) reveals the Adaline which provides output y_1 in Fig. 25 is updated by the rule

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\epsilon_1^{(2)}\,\text{sgm}'\,(s_1^{(2)})\mathbf{X}_k. \tag{95}$$

This rule turns out to be identical to the single Adaline version (54) 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 [42] 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. (94) is replaced by a pair of equations:

$$\Delta\mathbf{W}_k = 2\mu(1-\eta)\delta_k\mathbf{X}_k + \eta\Delta\mathbf{W}_{k-1} \tag{96}$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \Delta\mathbf{W}_k. \tag{97}$$

where the momentum constant $0 \le \eta < 1$ 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 caused either by gradient noise or high spatial frequencies in the MSE surface. The factor $(1-\eta)$ in Eq. (96) 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 paper. A detailed analysis of stability issues associated with momentum updating for the μ-LMS algorithm, for instance, has been described by Shynk and Roy [124].

In our experience, the momentum technique used alone is usually of little value. We have found, however, that it is often useful to apply the technique in situations that require relatively "clean"[22] 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 Eq. (96) and (97) with

$$\Delta_k = \delta_k\mathbf{X}_k + \eta\Delta_{k+1} \tag{98}$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \frac{\mu\Delta_k}{\sqrt{\sum_{\text{all Adalines}} |\Delta_k|^2}}. \tag{99}$$

where again $0 < \eta < 1$. The weight updates determined by Eqs. (98) and (99) can help a network find a solution when a relatively flat local region in the MSE surface is encoun-

[22] "Clean" gradient estimates are those with little gradient noise.

tered. 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. (96) through (99). Other methods for speeding up backpropagation training include Fahlman's popular quickprop method [125], as well as the delta-bar-delta approach reported in an excellent paper by Jacobs [126].[23]

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 derived for training recurrent networks to learn static associations [127], [128]. More interesting is the on-line technique of Williams and Zipser [129] 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 [104]. These on-line methods are generalizations of a well-known steepest-descent algorithm for training linear IIR filters [130], [30].

An equivalent technique that is usually far less computationally intensive but best suited for off-line computation [37], [42], [131], called "backpropagation through time," has been used by Nguyen and Widrow [50] to enable a neural network to learn without a teacher how to back up a computer-simulated trailer truck to a loading dock (Fig. 27). 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 that 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 [132]. The method described in this section is a generalization of the single Adaline MRIII technique (60). The multi-element generalization of the other single element MRIII rule (61) is described in [133].

The MRIII algorithm can be readily described by referring to Fig. 28. 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 [133], 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.

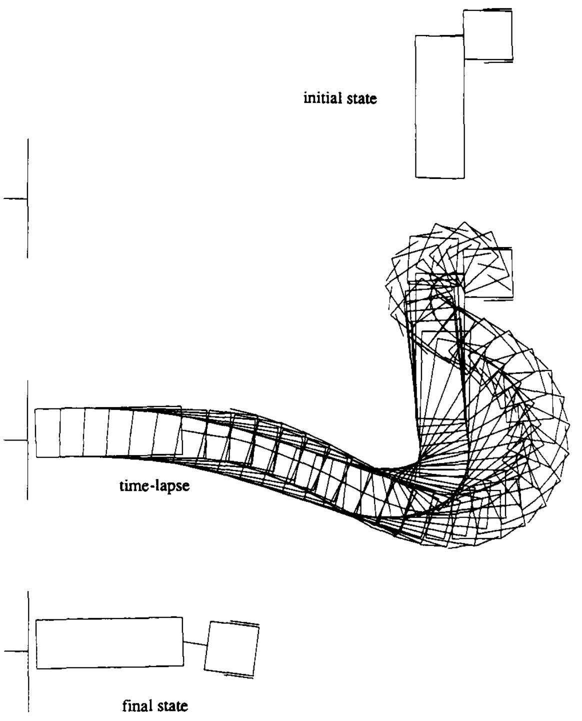


Fig. 27. Example truck backup sequence.

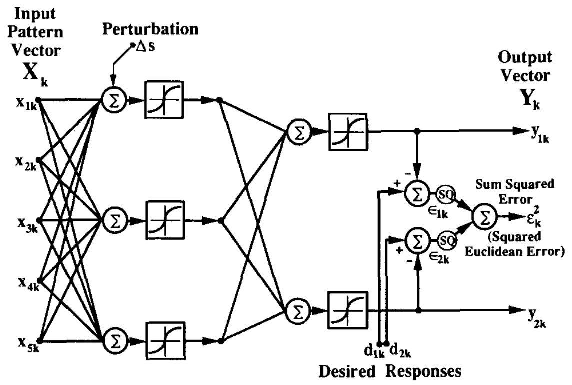


Fig. 28. Example two-layer Madaline III architecture.

Assume that an input pattern X and its associated desired output responses d_1 and d_2 are presented to the network of Fig. 28. At this point, we measure the sum squared output response error $\varepsilon^2 = (d_1 - y_1)^2 + (d_2 - y_2)^2 = \epsilon_1^2 + \epsilon_2^2$. We then 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, $\Delta(\varepsilon^2) = \Delta(\epsilon_1^2 + \epsilon_2^2)$. An easily measured ratio is

$$\frac{\Delta(\varepsilon^2)}{\Delta s} = \frac{\Delta(\epsilon_1^2 + \epsilon_2^2)}{\Delta s} \simeq \frac{\partial(\varepsilon^2)}{\partial s}. \tag{100}$$

[23] Jacob's paper, like many other papers in the literature, assumes for analysis that the true gradients rather than instantaneous gradients are used to update the weights, that is, 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 Jacob's paper 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.

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

$$\hat{\nabla}_k = \frac{\partial(\varepsilon_k^2)}{\partial \boldsymbol{W}_k} = \frac{\partial(\varepsilon_k^2)}{\partial s_k}\frac{\partial s_k}{\partial \boldsymbol{W}_k} = \frac{\partial(\varepsilon_k^2)}{\partial s_k}\boldsymbol{X}_k. \tag{101}$$

Replacing the derivative with a ratio of differences yields

$$\hat{\nabla}_k \simeq \frac{\Delta(\varepsilon_k^2)}{\Delta s}\boldsymbol{X}_k. \tag{102}$$

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 VI-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 (102) can be used to optimize the weight vector in accord with the method of steepest descent:

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - \mu\frac{\Delta(\varepsilon_k^2)}{\Delta s}\boldsymbol{X}_k. \tag{103}$$

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 equation (103), 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, 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 with MRII

MRIII was derived from MRII by replacing the signum nonlinearities with sigmoids. The similarity of these algorithms becomes evident when comparing Fig. 28, representing MRIII, with Fig. 16, representing MRII.

The MRII network is highly discontinuous and nonlinear. Using an instantaneous gradient to adjust the weights is not possible. In fact, from the MSE surface for the signum Adaline presented in Section VI-B, it is clear that even gradient descent techniques that use the true gradient could run into severe problems with local minima. 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 $\boldsymbol{X}$-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, that is, the Adalines that 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 1/4 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 $\boldsymbol{X}$-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 VI-B, we argued that for the sigmoid Adaline element, the MRIII algorithm (61) is essentially equivalent to the backpropagation algorithm (54). 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, then MRIII is essentially equivalent to backpropagation. That is, to the extent that the sample derivative $\Delta\varepsilon_k^2/\Delta s$ from Eq. (103) is equal to the analytical derivtive $\partial\varepsilon_k^2/\partial s_k$ from Eq. (91), 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 multilayer 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 could require a lot 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 Fig. 25 with Fig. 28 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. 26). 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.

E. MSE Surfaces of Neural Networks

In Section VI-B, "typical" mean-square-error surfaces of sigmoid and signum Adalines were shown, indicating that sigmoid Adalines are much more conducive to gradient approaches than signum Adalines. The same phenomena result when Adalines are incorporated into multi-element networks. The MSE surfaces of MRII networks are reasonably chaotic and will not be explored here. In this section we examine only MSE surfaces from a typical backpropagation training problem with a sigmoidal neural network.

In a network with more than two weights, the MSE surface is high-dimensional and difficult to visualize. It is possible, however, to look at slices of this surface by plotting the MSE surface created by varying two of the weights while holding all others constant. The surfaces plotted in Figs. 29 and 30 show two such slices of the MSE surface from a typical learning problem involving, respectively, an untrained sigmoidal network and a trained one. The first surface resulted from varying two first-layer weights of an untrained network. The second surface resulted from varying the same two weights after the network was fully trained. The two surfaces are similar, but the second one has a deeper minimum which was carved out by the backpropagation learning process. Figs. 31 and 32 resulted from varying a different set of two weights in the same network. Fig. 31 is the result from varying a first-layer weight and third-layer weight in the untrained network, whereas Fig. 32 is the surface that resulted from varying the same two weights after the network was trained.

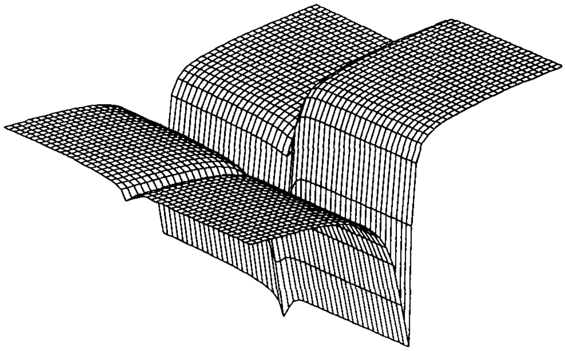


Fig. 29. Example MSE surface of untrained sigmoidal network as a function of two first-layer weights.

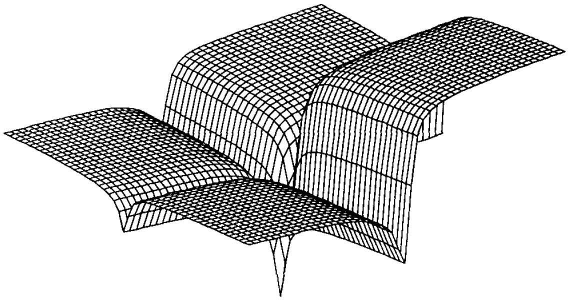


Fig. 30. Example MSE surface of trained sigmoidal network as a function of two first-layer weights.

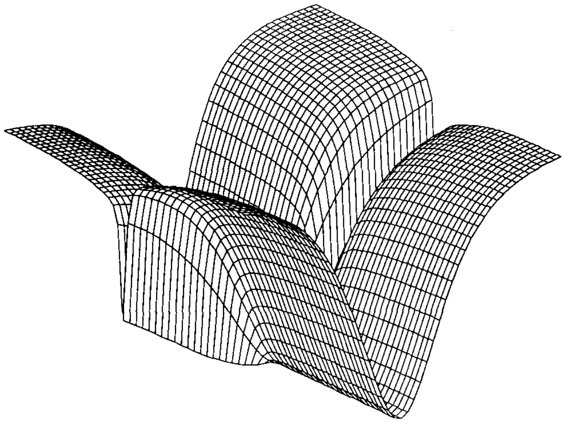


Fig. 31. Example MSE surface of untrained sigmoidal network as a function of a first-layer weight and a third-layer weight.

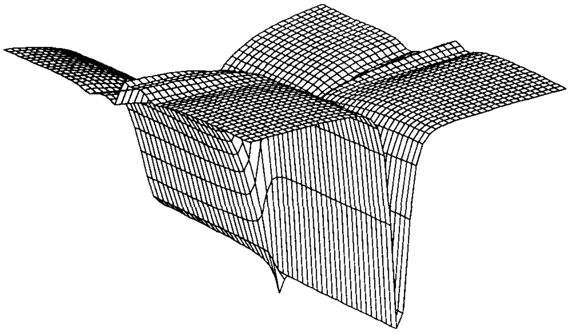


Fig. 32. Example MSE surface of trained sigmoidal network as a function of a first-layer weight and a third-layer weight.

By studying many plots, it becomes clear that backpropagation and MRIII will be subject to convergence on local optima. The same is true for MRII. The most common remedy for this is the sporadic addition of noise to the weights or gradients. Some of the "simulated annealing" methods [47] do this. Another method involves retraining the network several times using differnt random initial weight values until a satisfactory solution is found.

Solutions found by people in everyday life are usually not optimal, but many of them are useful. If a local optimum yields satisfactory performance, often there is simply no need to search for a better solution.

VIII. Summary

This year is the 30th anniversary of the publication of the Perceptron rule by Rosenblatt and the LMS algorithm by Widrow and Hoff. It has also been 16 years since Werbos first published the backpropagation algorithm. These learning rules and several others have been studied and compared. 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' 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

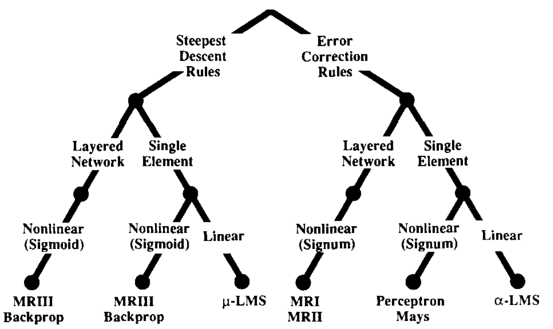


Fig. 33. Learning rules.

backpropagation algorithm. Fig. 33 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.

References

[1] K. Steinbuch and V. A. W. Piske, "Learning matrices and their applications," *IEEE Trans. Electron. Comput.*, vol. EC-12, pp. 846-862, Dec. 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. Washington, DC: Spartan Books, 1962, pp. 435-461.

[3] L. Stark, M. Okajima, and G. H. Whipple, "Computer pattern recognition techniques: Electrocardiographic diagnosis," *Commun. Ass. Comput. Mach.*, vol. 5, pp. 527-532, Oct. 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, Nov. 1958*, vol. 1 pp. 421-456. London: HM Stationery Office, 1959.

[5] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Washington, DC: Spartan Books, 1962.

[6] C. von der Malsburg, "Self-organizing of orientation sensitive cells in the striate cortex," *Kybernetik*, vol. 14, pp. 85-100, 1973.

[7] S. Grossberg, "Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors," *Biolog. Cybernetics*, vol. 23, pp. 121-134, 1976.

[8] K. Fukushima, "Cognitron: A self-orgainizing multilayered neural network," *Biolog. Cybernetics*, vol. 20, pp. 121-136, 1975.

[9] —, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biolog. Cybernetics*, vol. 36, pp. 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, Cybernetics*, vol. SMC-3, pp. 455-465, Sept. 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, Cybernetics*, vol. SMC-13, pp. 834-846, 1983.

[13] J. S. Albus, "A new approach to manipulator control: the cerebellar model articulation controller (CMAC)," *J. Dyn. Sys., Meas., Contr.*, vol. 97, pp. 220-227, 1975.

[14] W. T. Miller, III, "Sensor-based control of robotic manipulators using a general learning algorithm." *IEEE J. Robotics Automat.*, vol. RA-3, pp. 157-165, Apr. 1987.

[15] S. Grossberg, "Adaptive pattern classification and universal recoding, II: Feedback, expectation, olfaction, and illusions," Biolog. Cybernetics, vol. 23, pp. 187-202, 1976.

[16] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115, 1983.

[17] —, "Art 2: Self-organization of stable category recognition codes for analog output patterns," *Applied Optics*, vol. 26, pp. 4919-4930, Dec. 1, 1987.

[18] —, "Art 3 hierarchical search: Chemical transmitters in self-organizing pattern recognition architectures," in *Proc. Int. Joint Conf. on Neural Networks*, vol. 2, pp. 30-33, Wash., DC, Jan. 1990.

[19] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biolog. Cybernetics*, vol. 43, pp. 59-69, 1982.

[20] —, Self-Organization and Associative Memory. New York: Springer-Verlag, 2d ed., 1988.

[21] D. O. Hebb, *The Organization of Behavior.* New York: Wiley, 1949.

[22] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci.*, vol. 79, pp. 2554-2558, Apr. 1982.

[23] —, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad. Sci.*,\vol. 81, pp. 3088-3092, May 1984.

[24] B. Kosko, "Adaptive bidirectional associative memories," *Appl. Optics*, vol. 26, pp. 4947-4960, Dec. 1, 1987.

[25] G. E. Hinton, R. J. Sejnowski, and D. H. Ackley, "Boltzmann machines: Constraint satisfaction networks that learn," Tech. Rep. CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science, 1984.

[26] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machines," in *Parallel Distributed Processing*, vol. 1, ch. 7, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA, M.I.T. Press, 1986.

[27] L. R. Talbert *et al.*, "A real-time adaptive speech-recognition system," Tech. rep., Stanford University, 1963.

[28] M. J. C. Hu, *Application of the Adaline System to Weather Forecasting.* Thesis, Tech. Rep. 6775-1, Stanford Electron. Labs., Stanford, CA, June 1964.

[29] B. Widrow, "The original adaptive neural net broom-balancer," *Proc. IEEE Intl. Symp. Circuits and Systems*, pp. 351-357, Phila., PA, May 4-7 1987.

[30] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

[31] B. Widrow, P. Mantey, L. Griffiths, and B. Goode, "Adaptive antenna systems," *Proc. IEEE*, vol. 55, pp. 2143-2159, Dec. 1967.

[32] B. Widrow, "Adaptive inverse control," *Proc. 2d Intl. Fed. of Automatic Control Workshop*, pp. 1-5, Lund, Sweden, July 1-3, 1986.

[33] B. Widrow, *et al.*, "Adaptive noise cancelling: Principles and applications," *Proc. IEEE*, vol. 63, pp. 1692-1716, Dec. 1975.

[34] R. W. Lucky, "Automatic equalization for digital communication," *Bell Syst. Tech. J.*, vol. 44, pp. 547-588, Apr. 1965.

[35] R. W. Lucky, *et al.*, *Principles of Data Communication.* New York: McGraw-Hill, 1968.

[36] M. M. Sondhi, "An adaptive echo canceller," *Bell Syst. Tech. J.*, vol. 46, pp. 497-511, Mar. 1967.

[37] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Ph.D. thesis, Harvard University, Cambridge, MA, Aug. 1974.

[38] Y. le Cun, "A theoretical framework for back-propagation," in *Proc. 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. June 17-26, pp. 21-28. San Mateo, CA; Morgan Kaufmann.

[39] D. Parker, "Learning-logic," Invention Report S81-64, File 1, Office of Technology Licensing, Stanford University, Stanford, CA, Oct. 1982.

[40] —, "Learning-logic," Technical Report TR-47, Center for

Computational Research in Economics and Management Science, M.I.T., Apr. 1985.

[41] 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, Sept. 1985.

[42] —, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, vol. 1, ch. 8, D. E. Rumelhart and J. L. McClelland, Eds., Cambridge, MA: M.I.T. Press, 1986.

[43] B. Widrow, R. G. Winter, and R. Baxter, "Learning phenomena in layered neural networks," *Proc. 1st IEEE Intl. Conf. on Neural Networks*, vol. 2, pp. 411-429, San Diego, CA, June 1987.

[44] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, Apr. 1987.

[45] J. A. Anderson and E. Rosenfeld, Eds., *Neurocomputing: Foundations of Research*. Cambridge, MA: M.I.T. Press, 1988.

[46] N. Nilsson, *Learning Machines*. New York: McGraw-Hill, 1965.

[47] D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing*. Cambridge, MA: M.I.T. Press, 1986.

[48] B. Moore, "Art 1 and pattern clustering," in *Proc. 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds., June 17-26 1988, pp. 174-185, San Mateo, CA: Morgan Kaufmann.

[49] *DARPA Neural Network Study*. Fairfax, VA: AFCEA International Press, 1988.

[50] D. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural networks," *Proc. Intl. Joint Conf. on Neural Networks, vol. 2, pp. 357-363, Wash., DC, June 1989.*

[51] T. J. Sejnowski and C. R. Rosenberg, "Nettalk: a parallel network that learns to read aloud," Tech. Rep. JHU/EECS-86/01, Johns Hopkins University, 1986.

[52] —, "Parallel networks that learn to pronounce English text," *Complex Systems*, vol. 1, pp. 145-168, 1987.

[53] P. M. Shea and V. Lin, "Detection of explosives in checked airline baggage using an artificial neural system," *Proc. Intl. Joint Conf. on Neural Networks*, vol. 2, pp. 31-34, Wash., DC, June 1989.

[54] D. G. Bounds, P. J. Lloyd, B. Mathew, and G. Waddell, "A multilayer perceptron network for the diagnosis of low back pain," *Proc. 2d IEEE Intl. Conf. on Neural Networks*, vol. 2, pp. 481-489, San Diego, CA, July 1988.

[55] 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. San Mateo, CA: Morgan Kaufmann, 1989, pp. 248-255.

[56] N. Mokhoff, "Neural nets making the leap out of lab," *Electronic Engineering Times*, p. 1, Jan. 22, 1990.

[57] C. A. Mead, *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.

[58] B. Widrow and M. E. Hoff, Jr., "Adaptive switching circuits." *1960 IRE Western Electric Show and Convention Record, Part 4*, pp. 96-104, Aug. 23, 1960.

[59] —, "Adaptive switching circuits," Tech. Rep. 1553-1, Stanford Electron. Labs., Stanford, CA June 30, 1960.

[60] P. M. Lewis II and C. Coates, *Threshold Logic*. New York: Wiley, 1967.

[61] T. M. Cover, *Geometrical and Statistical Properties of Linear Threshold Devices*. Ph.D. thesis, Tech. Rep. 6107-1, Stanford Electron. Labs., Stanford, CA, May 1964.

[62] R. J. Brown, *Adaptive Multiple-Output Threshold Systems and Their Storage Capacities*. Thesis, Tech. Rep. 6771-1, Stanford Electron. Labs., Stanford, CA, June 1964.

[63] R. O. Winder, *Threshold Logic*. Ph.D. thesis, Princeton University, Princeton, NJ, 1962.

[64] S. H. Cameron, "An estimate of the complexity requisite in a universal decision network," *Proc. 1960 Bionics Symposium*, Wright Air Development Division Tech. Rep. 60-600, pp. 197-211, Dayton, OH, Dec. 1960.

[65] R. D. Joseph, "The number of orthants in *n*-space intersected by an *s*-dimensional subspace," *Tech. Memorandum 8*, Project PARA, Cornell Aeronautical Laboratory, Buffalo, New York 1960.

[66] D. F. Specht, *Generation of Polynomial Discriminant Functions for Pattern Recognition*. Ph.D. thesis, Tech. Rep. 6764-5, Stanford Electron. Labs., Stanford, CA, May 1966.

[67] —, "Vectorcardiographic diagnosis using the polynomial discriminant method of pattern recognition," *IEEE Trans. Biomed. Eng.*, vol. BME-14, pp. 90-95, Apr. 1967.

[68] —, "Generation of polynomial discriminant functions for pattern recognition," IEEE Trans. Electron. Comput., vol. EC-16, pp. 308-319, June 1967.

[69] 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., New York: Marcel Dekker Inc., 1984, pp. 25-65.

[70] —, "Predicted squared error: A criterion for automatic model selection," *Self-Organizing Methods in Modeling*, in S. J. Farlow, Ed. New York: Marcel Dekker Inc., 1984, pp. 87-103.

[71] A. R. Barron and R. L. Barron, "Statistical learning networks: A unifying view," *1988 Symp. on the Interface: Statistics and Computing Science*, pp. 192-203, Reston, VA, Apr. 21-23, 1988.

[72] A. G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE Trans. Syst., Man, Cybernetics*, SMC-1, pp. 364-378, Oct. 1971.

[73] Y. H. Pao, "Functional link nets: Removing hidden layers." *AI Expert*, pp. 60-68, Apr. 1989.

[74] C. L. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Applied Optics*, vol. 26, pp. 4972-4978, Dec. 1, 1987.

[75] M. E. Hoff, Jr., *Learning Phenomena in Networks of Adaptive Switching Circuits*. Ph.D. thesis, Tech. Rep. 1554-1, Stanford Electron. Labs., Stanford, CA, July 1962.

[76] W. C. Ridgway III, *An Adaptive Logic System with Generalizing Properties*. Ph.D. thesis, Tech. Rep. 1556-1, Stanford Electron. Labs., Stanford, CA, April 1962.

[77] F. H. Glanz, *Statistical Extrapolation in Certain Adaptive Pattern-Recognition Systems*. Ph.D. thesis, Tech. Rep. 6767-1, Stanford Electron. Labs., Stanford, CA, May 1965.

[78] B. Widrow, "Adaline and Madaline—1963, plenary speech," *Proc. 1st IEEE Intl. Conf. on Neural Networks*, vol. 1, pp. 145-158, San Diego, CA, June 23, 1987.

[79] —, "An adaptive "adaline" neuron using chemical 'memistors.'" Tech. Rep. 1553-2, Stanford Electron. Labs., Stanford, CA, Oct. 17, 1960.

[80] C. L. Giles, R. D. Griffin, and T. Maxwell, "Encoding geometric invariances in higher order neural networks," *Neural Information Processing Systems*, in D. Z. Anderson, Ed. New York: American Institute of Physics, 1988, pp. 301-309.

[81] D. Casasent and D. Psaltis, "Position, rotation, and scale invariant optical correlation," *Appl. Optics*, vol. 15, pp. 1795-1799, July 1976.

[82] W. L. Reber and J. Lyman, "An artificial neural system design for rotation and scale invariant pattern recognition," *Proc. 1st IEEE Intl. Conf. on Neural Networks*, vol. 4, pp. 277-283, San Diego, CA, June 1987.

[83] B. Widrow and R. G. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Computer*, pp. 25-39, Mar. 1988.

[84] A. Khotanzad and Y. H. Hong, "Rotation invariant pattern recognition using zernike moments," *Proc. 9th Intl. Conf. on Pattern Recognition*, vol. 1, pp. 326-328, 1988.

[85] C. von der Malsburg, "Pattern recognition by labeled graph matching," *Neural Networks*, vol. 1, pp. 141-148, 1988.

[86] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time delay neural networks," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. ASSP-37, pp. 328-339, Mar. 1989.

[87] C. M. Newman, "Memory capacity in neural network models: Rigorous lower bounds," *Neural Networks*, vol. 1, pp. 223-238, 1988.

[88] Y. S. Abu-Mostafa and J. St. Jacques, "Information capacity of the hopfield model," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 461-464, 1985.

[89] Y. S. Abu-Mostafa, "Neural networks for computing?" in *Neural Networks for Computing, Amer. Inst. of Phys. Conf. Proc. No. 151*, J. S. Denker, Ed. New York: American Institute of Physics, 1986, pp. 1-6.

[90] S. S. Venkatesh, "Epsilon capacity of neural networks," in

Neural Networks for Computing, Amer. Inst. of Phys. Conf. Proc. No. 151, J. S. Denker, Ed. New York: American Institute of Physics, 1986, pp. 440-445.

[91] J. D. Greenfield, *Practical Digital Design Using IC's. 2d ed., New York: Wiley, 1983.*

[92] M. Stinchombe and H. White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," *Proc. Intl. Joint Conf. on Neural Networks*, vol. 1, pp. 613-617, Wash., DC, June 1989.

[93] G. Cybenko, "Continuous valued neural networks with two hidden layers are sufficient," Tech. Rep., Dept. of Computer Science, Tufts University, Mar. 1988.

[94] B. Irie and S. Miyake, "Capabilities of three-layered perceptrons," *Proc. 2d IEEE Intl. Conf. on Neural Networks*, vol. 1, pp. 641-647, San Diego, CA, July 1988.

[95] M. L. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA: M.I.T. Press, expanded ed., 1988.

[96] M. W. Roth, "Survey of neural network technology for automatic target recognition," *IEEE Trans. Neural Networks*, vol. 1, pp. 28-43, Mar. 1990.

[97] T. M. Cover, "Capacity problems for linear machines,"*Pattern Recognition*, in L. N. Kanal, Ed. Wash., DC: Thompson Book Co., 1968, pp. 283-289, part 3.

[98] E. B. Baum, "On the capabilities of multilayer perceptrons," *J. Complexity*, vol. 4, pp. 193-215, Sept. 1988.

[99] A. Lapedes and R. Farber, "How neural networks work," Tech. Rep. LA-UR-88-418, Los Alamos Nat. Laboratory, Los Alamos, NM, 1987.

[100] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," *Proc. Intl. Joint Conf. on Neural Networks*, San Diego, CA, June 1990.

[101] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, 1989.

[102] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computation*, vol. 1, pp. 151-160, 1989.

[103] J. J. Hopfield and D. W. Tank, "Neural computations of decisions in optimization problems," *Biolog. Cybernetics*, vol. 52, pp. 141-152, 1985.

[104] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4-27, Mar. 1990.

[105] C. H. Mays, *Adaptive Threshold Logic.* Ph.D. thesis, Tech. Rep. 1557-1, Stanford Electron. Labs., Stanford, CA, Apr. 1963.

[106] F. Rosenblatt, "On the convergence of reinforcement procedures in simple perceptrons," *Cornell Aeronautical Laboratory Report VG-1196-G-4*, Buffalo, NY, Feb. 1960.

[107] H. Block, "The perceptron: A model for brain functioning, I," *Rev. Modern Phys.*, vol. 34, pp. 123-135, Jan. 1962.

[108] R. G. Winter, *Madaline Rule II: A New Method for Training Networks of Adalines.* Ph.D. thesis, Stanford University, Stanford, CA, Jan. 1989.

[109] E. Walach and B. Widrow, "The least mean fourth (lmf) adaptive algorithm and its family," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 275-283, Mar. 1984.

[110] E. B. Baum and F. Wilczek, "Supervised learning of probability distributions by neural networks," in *Neural Information Processing Systems*, D. Z. Anderson, Ed. New York: American Institute of Physics, 1988, pp. 52-61.

[111] S. A. Solla, E. Levin, and M. Fleisher, "Accelerated learning in layered neural networks," *Complex Systems*, vol. 2, pp. 625-640, 1988.

[112] D. B. Parker, "Optimal algorithms for adaptive neural networks: Second order back propagation, second order direct propagation, and second order Hebbian learning," *Proc. 1st IEEE Intl. Conf. on Neural Networks*, vol. 2, pp. 593-600, San Diego, CA, June 1987.

[113] A. J. Owens and D. L. Filkin, "Efficient training of the back propagation network by solving a system of stiff ordinary differential equations," *Proc. Intl. Joint Conf. on Neural Networks*, vol. 2, pp. 381-386, Wash., DC, June 1989.

[114] D. G. Luenberger, *Linear and Nonlinear Programming.* Reading, MA: Addison-Wesley, 2d ed., 1984.

[115] 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., pp. 40-48, San Mateo, CA: Morgan Kaufmann, 1989.

[116] R. V. Southwell, *Relaxation Methods in Engineering Science.* New York: Oxford, 1940.

[117] D. J. Wilde; *Optimum Seeking Methods.* Englewood Cliffs, NJ: Prentice-Hall, 1964.

[118] N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series, with Engineering Applications.* New York: Wiley, 1949.

[119] T. Kailath, "A view of three decades of linear filtering theory," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 145-181, Mar. 1974.

[120] H. Bode and C. Shannon, "A simplified derivation of linear least squares smoothing and prediction theory," *Proc. IRE*, vol. 38, pp. 417-425, Apr. 1950.

[121] L. L. Horowitz and K. D. Senne, "Performance advantage of complex LMS for controlling narrow-band adaptive arrays," *IEEE Trans. Circuits, Systems*, vol. CAS-28, pp. 562-576, June 1981.

[122] E. D. Sontag and H. J. Sussmann, "Backpropagation separates when perceptrons do," *Proc. Intl. Joint Conf. on Neural Networks*, vol. 1, pp. 639-642, Wash., DC, June 1989.

[123] —, "Backpropagation can give rise to spurious local minima even for networks without hidden layers," *Complex Systems*, vol. 3, pp. 91-106, 1989.

[124] J. J. Shynk and S. Roy, "The lms algorithm with momentum updating," *ISCAS 88*, Espoo, Finland, June 1988.

[125] S. E. Fahlman, "Faster learning variations on backpropagation: An empirical study," in *Proc. 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. June 17-26, 1988, pp. 38-51, San Mateo, CA: Morgan Kaufmann.

[126] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation, *Neural Networks*, vol. 1, pp. 295-307, 1988.

[127] F. J. Pineda, "Generalization of backpropagation to recurrent neural networks," *Phys. Rev. Lett.*, vol. 18, pp. 2229-2232, 1987.

[128] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," *Proc. 1st IEEE Intl. Conf. on Neural Networks*, vol. 2, pp. 609-618, San Diego, CA, June 1987.

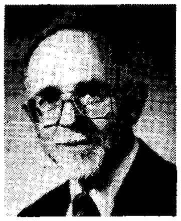
[129] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," ICS Report 8805, Inst. for Cog. Sci., University of California at San Diego, La Jolla, CA, Oct. 1988.

[130] S. A. White, "An adaptive recursive digital filter," *Proc. 9th Asilomar Conf. Circuits Syst. Comput.*, p. 21, Nov. 1975.

[131] B. Pearlmutter, "Learning state space trajectories in recurrent neural networks," in *Proc. 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. June 17-26, 1988, pp. 113-117. San Mateo, CA: Morgan Kaufmann.

[132] M. Holler, *et al.*, "An electrically trainable artificial neural network (etann) with 10240 'floating gate' synapses," *Proc. Intl. Joint Conf. on Neural Networks*, vol. 2, pp. 191-196, Wash., DC, June 1989.

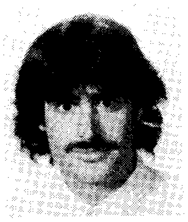
[133] D. Andes, B. Widrow, M. Lehr, and E. Wan, "MRIII: A robust algorithm for training analog neural networks, *Proc. Intl. Joint Conf. on Neural Networks*, vol. 1, pp. 533-536, Wash., DC, Jan. 1990.



Bernard Widrow (Fellow, IEEE) received the S.B., S.M., and Sc.D. degrees from the Massachusetts Institute of Technology in 1951, 1953, and 1956, respectively.

He was with M.I.T. until he joined the Stanford University faculty in 1959, where he is now a Professor of electrical engineering. He is presently engaged in research and teaching in neural networks, pattern recognition, adaptive filtering, and adaptive control systems. He is associate editor of the journals *Adaptive Control and Signal Processing, Neural Networks, Information Sciences,* and *Pattern Recognition* and coauthor with S. D. Stearns of *Adaptive Signal Processing* (Prentice Hall).

Dr. Widrow received the SB, SM and ScD degrees from MIT in 1951, 1953, and 1956. He is a member of the American Association of University Professors, the Pattern Recognition Society, Sigma Xi, and Tau Beta Pi. He is a fellow of the American Association for the Advancement of Science. He is president of the International Neural Network Society. He received the IEEE Alexander Graham Bell Medal in 1986 for exceptional contributions to the advancement of telecommunications.



Michael A. Lehr was born in New Jersey on April 18, 1964. He received the B.E.E. degree in electrical engineering at the Georgia Institute of Technology in 1987, graduating top in his class. He received the M.S.E.E. from Stanford University in 1986.

From 1982 to 1984, he worked on two-way radio development at Motorola in Ft. Lauderdale, Florida, and from 1984 to 1987 he was involved with naval sonar system development and test at IBM in Manassas, Virginia. Currently, he is a doctoral candidate in the Department of Electrical Engineering at Stanford University. His research interests include adaptive signal processing and neural networks.

Mr. Lehr holds a General Radiotelephone Operator License (1981) and Radar Endorsement (1982), and is a member of Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi.