Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition

Bernard Widrow, Stanford University
Rodney Winter, United States Air Force

The fields of adaptive signal processing and adaptive neural networks have been developing independently but have the adaptive linear combiner (ALC) in common. With its inputs connected to a tapped delay line, the ALC becomes a key component of an adaptive filter. With its output connected to a quantizer, the ALC becomes an adaptive threshold element or adaptive neuron.

Adaptive filters have enjoyed great commercial success in the signal processing field. All high-speed modems now use adaptive equalization filters. Long-distance telephone and satellite communications links are being equipped with adaptive echo cancelers to filter out echo, allowing simultaneous two-way communications. Other applications include noise canceling and signal prediction.

Adaptive threshold elements, on the other hand, are the building blocks of neural networks. Today neural nets are the focus of widespread research interest. Areas of investigation include pattern recognition and trainable logic. Neural network systems have not yet had the commercial impact of adaptive filtering.

The commonality of the ALC to adaptive signal processing and adaptive neural networks suggests the two fields have much to share with each other. This article describes practical applications of the ALC in signal processing and pattern recognition.

A new multilayer adaptation algorithm that descrambles output and reproduces original patterns is advancing the practicality of neural-network pattern-recognition systems.

The adaptive linear combiner

The ALC shown in Figure 1 is the basic building block for most adaptive systems. The output is a linear combination of the many input signals. The weighting coefficients comprise a weight vector. The input signals comprise an input signal vector. The output signal is the inner product or dot product of the input signal vector with the weight vector. The output signal is compared with a special input signal called the desired response, and the difference is the error signal. To optimize performance, the ALC's weighting coefficients or weights are generally adjusted to minimize the mean square of the error signal. Of the many adaptive algorithms to adjust the weights automatically, the most popular is the Widrow-Hoff LMS (least mean square) algorithm devised in 1959.[1]

Adaptive filters. Digital signals generally originate from sampling continuous input signals by analog-to-digital conversion. Digital signals are often filtered by means of a tapped delay line or transversal filter, as shown in Figure 2a. The sampled input signal is applied to a string of delay boxes, each delaying the signal by one sampling period. An ALC is seen connected to the taps between the delay boxes. The filtered output is a linear combination of the current and past input signal samples. By varying the weights, the impulse response from input to output is directly controllable. Since the frequency response is the Fourier transform of the impulse response, controlling the impulse response controls the frequency response. The weights are usually adjusted so that the output signal will provide the best least-squares match over time to the desired-response input signal.

The literature reports many other forms

0018-9162/88/0300-0025$01.00 © 1988 IEEE

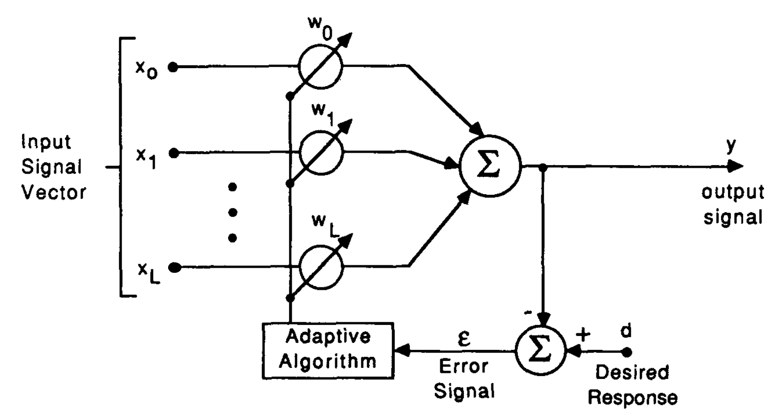


Figure 1. Adaptive linear combiner (ALC).

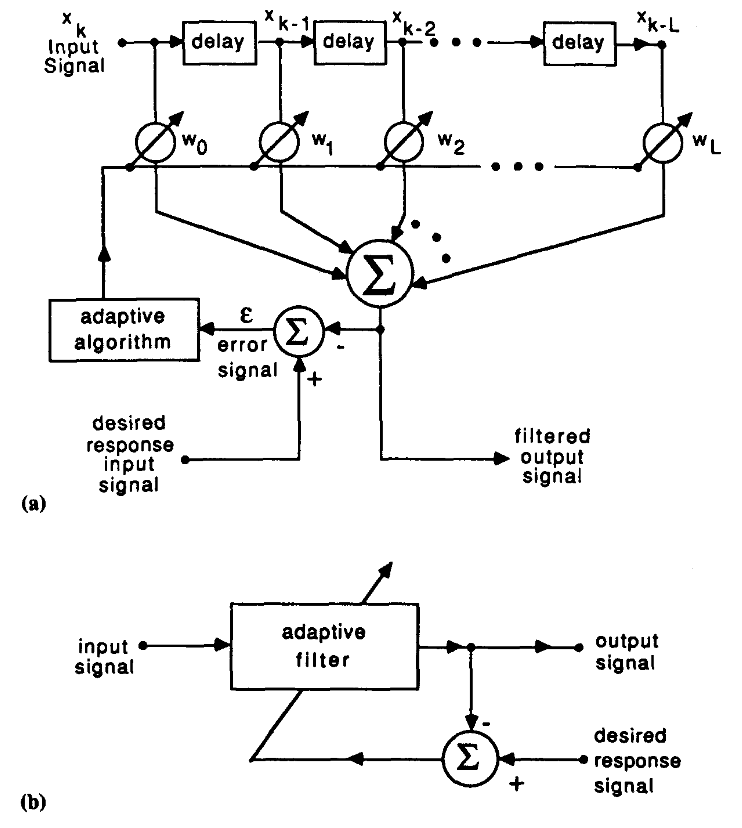


Figure 2. Adaptive digital filter: (a) details of a tapped-delay-line digital filter; (b) symbolic representation of an adaptive filter.

of adaptive filters.[1] Some use feedback to obtain both poles and zeros. The filter of Figure 2a realizes only zeros. Adaptive filters based on lattice structures achieve more rapid convergence under certain conditions. However, the simplest, most robust, and most widely used filter is that of Figure 2a adapted by the LMS algorithm. Figure 2b shows a symbolic representation of an adaptive filter.

Adaptive threshold element. Figure 3 shows an adaptive threshold element, a key component in adaptive pattern recognition systems. It consists of an adaptive linear combiner cascaded with a quantizer. The output of the ALC is quantized to produce a binary "decision." Most often, the inputs are binary and the desired response is binary. As such, the adaptive threshold element is trainable and capable of implementing binary logic functions. The LMS algorithm was originally developed to train the adaptive threshold element of Figure 3. This element was called an adaptive linear neuron or Adaline.[2]

The adaptive threshold element was an early neuronal model. The adaptive weights were analogous to synapses. The input vector components related to the dendritic inputs. The quantized output was analogous to the axonal output. The output decision was determined by a weighted sum of the inputs, in much the same way real neurons were believed to behave.

Adaptive signal processing

The adaptive filter of Figure 2b has an input signal and produces an output signal. The desired response is supplied during training. A question naturally arises: If the desired response were known and available, why would one need the adaptive filter? Put another way, how would one obtain the desired response in a practical application? There is no general answer to these questions, but studying successful examples provides some insight.

Example 1—system modeling. In many engineering and scientific applications, a system of unknown structure has observable input and output signals. One way of obtaining knowledge about the unknown system's dynamic response is to apply its input to an adaptive filter and to use its output as the adaptive filter's desired response. (See Figure 4.) The adaptive fil-

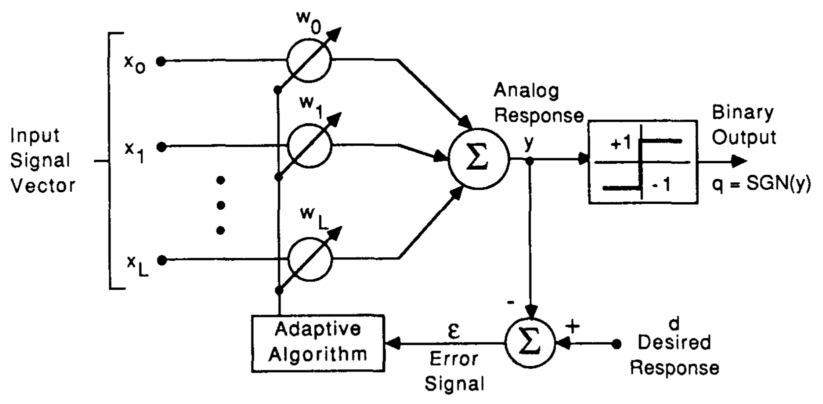


Figure 3. Adaptive threshold element (Adaline).

ter develops an impulse response to match that of the unknown system since the filter and the system develop similar outputs when driven by the same input.

Example 2—statistical prediction. One can estimate future values of time-correlated digital signals from present and past input samples. Wiener[3] has developed optimal linear least-squares filtering techniques for signal prediction. When the signal's autocorrelation function is known, Wiener's theory yields the impulse response of the optimal filter. More often than not, however, the autocorrelation function is unknown and may be time-variable. One could use a correlator to measure the autocorrelation function and plug this into Wiener theory to get the optimal impulse response, or one could get the optimal prediction filter directly by adaptive filtering. Figure 5 illustrates the latter approach.

In this figure, the input signal delayed by Δ time units is fed to an adaptive filter. The undelayed input serves as the desired response for this adaptive filter. The filter weights adapt and converge to produce a best least-squares estimate of the present input signal, given an input that is this very signal delayed by Δ. The optimal weights are copied into a "slave filter" whose input is undelayed and whose output therefore is a best least-squares prediction of the input Δ time units into the future.

Example 3—noise canceling. Separating a signal from additive noise is a common

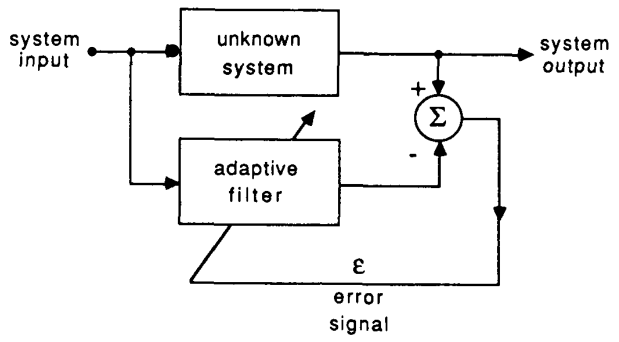


Figure 4. System modeling.

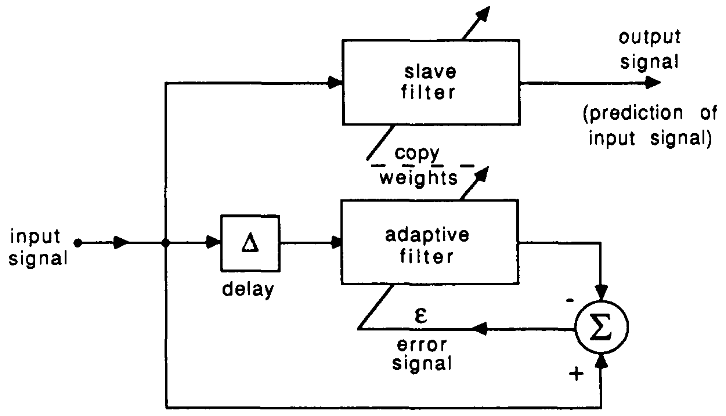


Figure 5. An adaptive statistical predictor.

problem in signal processing. Figure 6a shows a classical approach to this problem using optimal Wiener or Kalman filtering.[3] The purpose of the optimal filter is to pass the signal s without distortion while stopping the noise n_0. In general, this cannot be done perfectly. Even with the best filter, the signal is distorted, and some noise goes through to the output.

Figure 6b shows another approach to the problem using adaptive filtering. This approach is viable only when an additional "reference input" is available containing noise n_1, which is correlated with the original corrupting noise n_0. In Figure 6b, the adaptive filter receives the reference noise, filters it, and subtracts the result from the noisy "primary input," $s+n_0$. For this adaptive filter, the noisy input $s+n_0$ acts as the desired response. The "system output" acts as the error for the adaptive filter. Adaptive noise canceling generally performs much better than the classical approach since the noise is subtracted out rather than filtered out.

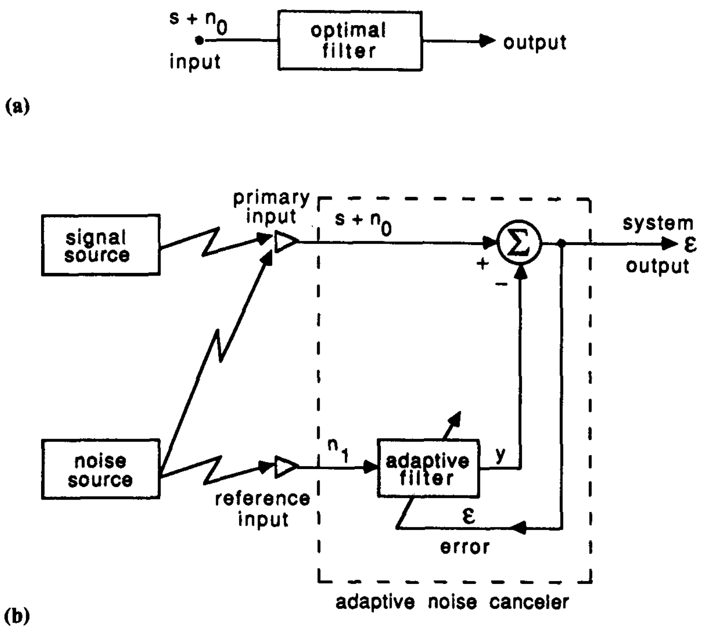


Figure 6. Separation of signal and noise: (a) classical approach; (b) adaptive noise-canceling approach.

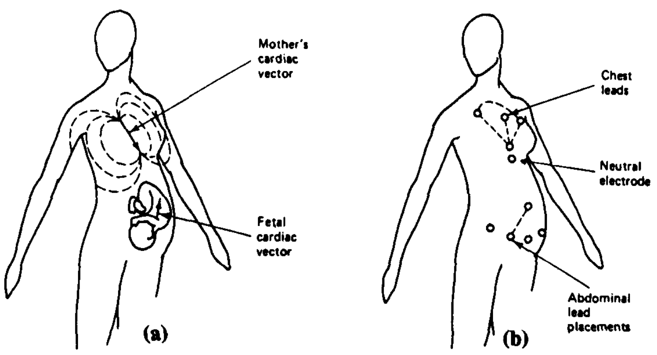


Figure 7. Canceling maternal heartbeat in fetal electrocardiography: (a) cardiac electric field vectors of mother and fetus; (b) placement of leads.

One might think that some prior knowledge of the signal s or of the noises n_0 and n_1 would be necessary before the filter could adapt to produce the noise-canceling signal y. A simple argument will show, however, that little or no prior knowledge of s, n_0, or n_1 or of their interrelationships is required.

Assume that s, n_0, n_1 and y are statistically stationary and have zero means. Assume that s is uncorrelated with n_0 and n_1 and suppose that n_1 is correlated with n_0. The output is

$$\varepsilon = s + n_0 - y \tag{1}$$

Squaring, one obtains

$$\varepsilon^2 = s^2 + (n_0-y)^2 + 2s(n_0-y) \tag{2}$$

Taking expectations of both sides of Equation 2, and realizing that s is uncorrelated with n_0 and with y, yields

$$\begin{aligned} E[\varepsilon^2] &= E[s^2] + E[(n_0-y)^2] \\ &\quad + 2E[s(n_0-y)] \\ &= E[s^2] + E[(n_0-y)^2] \end{aligned} \tag{3}$$

Adapting the filter to minimize $E[\varepsilon^2]$ will not affect the signal power $E[s^2]$. Accordingly, the minimum output power is

$$E_{\min}[\varepsilon^2] = E[s^2] + E_{\min}[(n_0-y)^2] \tag{4}$$

When the filter is adjusted so that $E[\varepsilon^2]$ is minimized, $E[(n_0-y)^2]$ is therefore also minimized. The filter output y is then a best least-squares estimate of the primary noise n_0. Moreover, when $E[(n_0-y)^2]$ is minimized, $E[(\varepsilon-s)^2]$ is also minimized, since, from Equation 1,

$$(\varepsilon-s) = (n_0-y) \tag{5}$$

Adjusting or adapting the filter to minimize the total output power is tantamount to causing the output ε to be a best least-squares estimate of the signal s for the given structure and adjustability of the adaptive filter and for the given reference input.

There are many practical applications for adaptive noise canceling techniques. One involves canceling interference from the mother's heart when attempting to record clear fetal electrocardiograms. Figure 7 shows the location of the fetal and maternal hearts and the placement of the input leads. The abdominal leads provide the primary input (containing fetal ECG and interfering maternal ECG signals), and the chest leads provide the reference input (containing pure interference, the maternal ECG). Figure 8 shows the results. The maternal ECG from the chest leads was adaptively filtered and subtracted from the abdominal signal, leaving the fetal ECG. This was an interesting problem since the fetal and maternal ECG signals had spectral overlap. The two hearts were electrically isolated and worked independently, but the second harmonic frequency of the maternal ECG was close to the fundamental of the fetal ECG. Ordinary filtering techniques would have great difficulty with this problem.

Example 4—adaptive echo canceling. Echo is a natural phenomenon in long-distance telephone circuits because of amplification in both directions and series coupling of telephone transmitters and receivers at each end of the circuit. Echo suppressors, used to break the feedback, give one-way communication to the party speaking first. To avoid switching effects and to permit simultaneous two-way transmission of voice and data, adaptive echo cancelers are replacing echo suppressors worldwide (see Figure 9).

In Figure 9, the delay boxes represent transmission delays in the long line. Note that separate circuits are normally used in each direction because the repeater amplifiers used to overcome transmission loss are one-way devices. Hybrid transformers prevent incoming signals from coupling through the telephone set and passing as outgoing signals. Hybrids are balanced to do this by designing them for the average, local telephone circuit. Since each local circuit has its own length and electrical characteristics, the hybrid cannot do its job perfectly. Using an adaptive filter to cancel any incoming signal that might leak through the hybrid eliminates the possibility of echo. The circuit of Figure 9 works well, allowing simultaneous two-way communication without echo.

Example 5—inverse modeling. Figure 4 showed use of an adaptive filter for direct modeling of an unknown system to obtain a close approximation to its impulse and frequency responses. By changing the configuration, it is possible to use the adaptive filter for inverse modeling to obtain the reciprocal of the unknown system's trans-

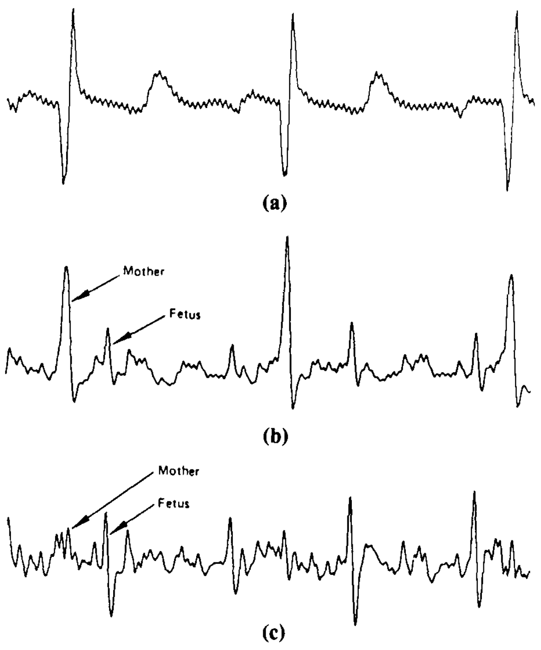


Figure 8. Result of fetal ECG experiment (bandwidth, 3-35 Hz; sampling rate, 256 Hz): (a) reference input (chest lead); (b) primary input (abdominal lead); (c) noise canceler output.

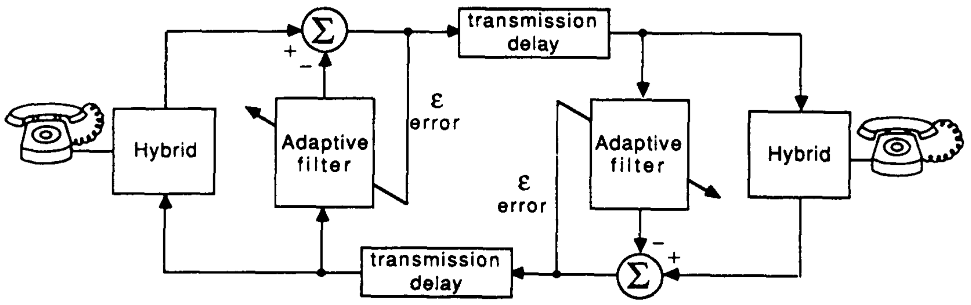


Figure 9. Long-distance system with adaptive echo cancellation.

fer function. The idea is illustrated in Figure 10. The output of the unknown system is the input to the adaptive filter. The unknown system's input delayed by Δ time units is the desired response of the adaptive filter. For simplicity, assume that Δ is set to zero delay. To make the error small, the cascade of the unknown system and the adaptive filter needs a unity transfer function. Therefore, using an adaptive algorithm to make the error small causes the adaptive filter to develop a transfer function that is the inverse of that of the unknown system.

If the response of the unknown system contains a delay or is nonminimum phase, allowing a nonzero delay Δ will be highly advantageous. Including Δ delays the inverse impulse response but yields a much lower mean-square error. In some applications, one would like to make this delay as small as possible. In other applications, this delay is of no concern except that it should be chosen to minimize the mean-square error. Applications for inverse modeling exist in the field of adaptive control, in geophysical signal processing, where it is called "deconvolution," and in telecommunications for channel equalization.

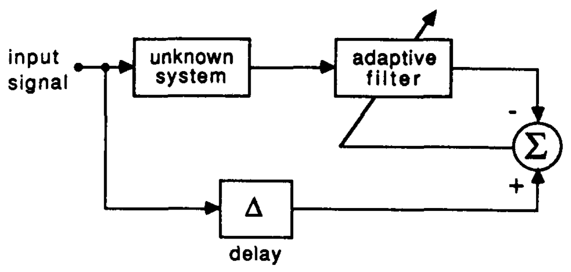


Figure 10. Inverse modeling.

Example 6—channel equalization. Telephone channels, radio channels, and even fiber optic channels can have non-flat frequency responses and nonlinear phase responses in the signal passband. Sending digital data at high speed through these channels often results in a phenomenon called "intersymbol interference," caused by signal pulse smearing in the dispersive

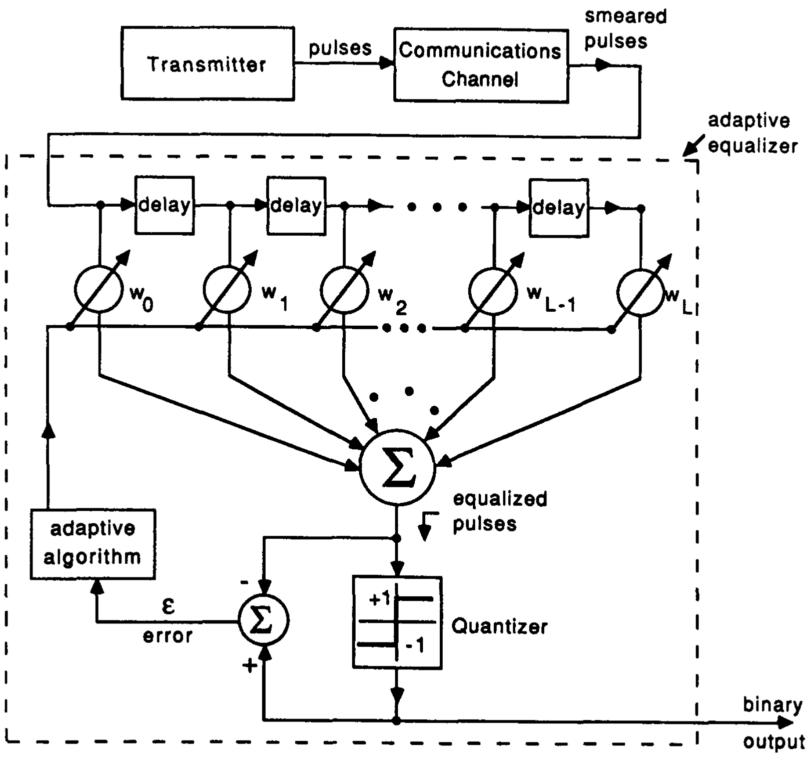


Figure 11. Adaptive channel equalizer with decision-directed learning.

medium. Equalization in data modems combats this phenomenon by filtering incoming signals. A modem's adaptive filter, by adapting itself to become a channel inverse, can compensate for the irregularities in channel magnitude and phase response.

The adaptive equalizer in Figure 11 consists of a tapped delay line with an adaptive linear combiner connected to the taps. Deconvolved signal pulses appear at the weighted sum, which is quantized to provide a binary output corresponding to the original binary data transmitted through the channel. The ALC and its quantizer comprise a single Adaline. Any least-squares algorithm can adapt the weights, but the telecommunications industry uses the LMS algorithm almost exclusively.

In operation, the weight at a central tap is generally fixed at unit value. Initially, all other weights are set to zero so that the equalizer has a flat frequency response and a linear phase response. Without equalization, telephone channels can provide quantized binary outputs that reproduce the transmitted data stream with error rates of 10^{-1} or less. As such, the quantized binary output can be used as the desired response to train the neuron. It is a noisy desired response initially. Sporadic errors cause adaptation in the wrong direction, but on average, adaptation proceeds correctly. As the neuron learns, noise in the desired response diminishes. Once the adaptive equalizer converges, the error rate will typically be 10^{-6} or less. The method, called "decision-directed" learning, was invented by Robert W. Lucky of AT&T Bell Labs.[4]

Figure 12a shows the analog response of a telephone channel carrying high-speed binary pulse data. Figure 12b shows an "eye" pattern, which is the same signal after going through a converged adaptive equalizer. Equalization opens the eye and allows clear separation of $+1$ and -1 pulses. Using a modem with an adaptive equalizer enables transmitting about four times as much data through the same channel with the same reliability as without equalization.

Integrated services digital network (ISDN), a new concept now in development and deployment, makes high-speed digital communication possible through ordinary local copper telephone circuits. ISDN requires both adaptive equalization and adaptive echo canceling at each line termination. The number of adaptive filters to be used in the world's telecommunications plant will be massive.

Adaptive pattern recognition

The adaptive threshold element of Figure 3 can be used for pattern recognition and as a trainable logic device. It can be trained to classify input patterns into two categories. For these applications, the zeroth weight, w_0, has a constant input $x_0 = +1$ which does not change from input pattern to pattern. Varying the zeroth weight varies the threshold level of the quantizer.

Linear separability. With n binary inputs and one binary output, a single neuron of the type shown in Figure 3 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 output. A single neuron is capable of realizing only a small subset of these functions, known as the linearly separable logic functions.[5] These are the set of logic functions that can be obtained with all possible settings of the weight values.

Figure 13 shows a two-input neuron, and Figure 14 shows all of its possible binary inputs in pattern vector space. In this space, the coordinate axes are the components of the input pattern vector. The neuron separates the input patterns into two categories, depending on the values of the input-signal weights and the bias weight. A critical thresholding condition occurs when the analog response y equals zero:

$$y = x_1 w_1 + x_2 w_2 + w_0 = 0 \qquad (6)$$

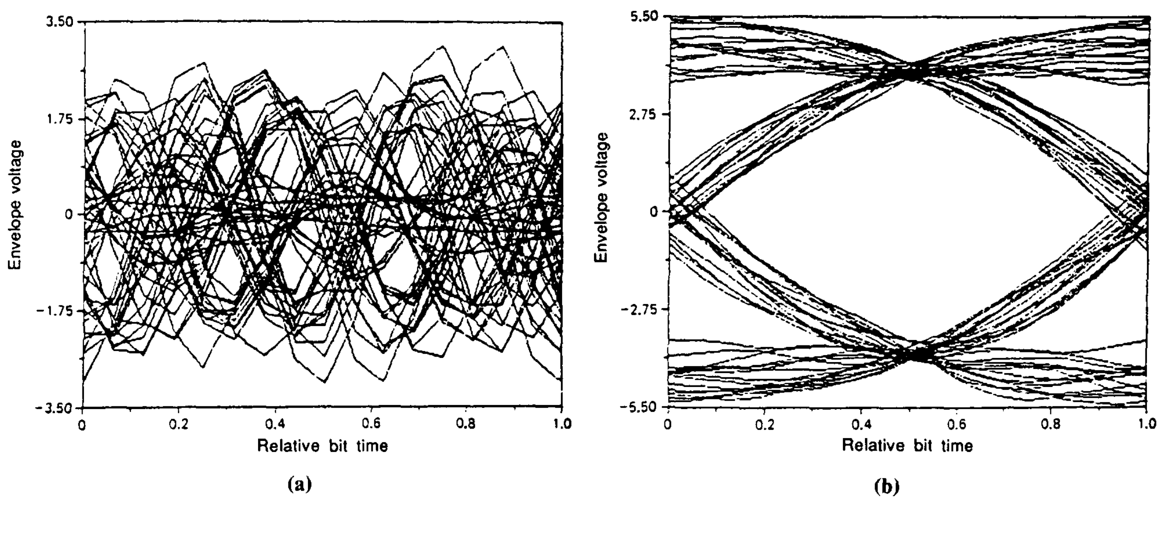


Figure 12. Eye patterns produced by overlaying cycles of the received waveform: (a) before equalization; (b) after equalization.

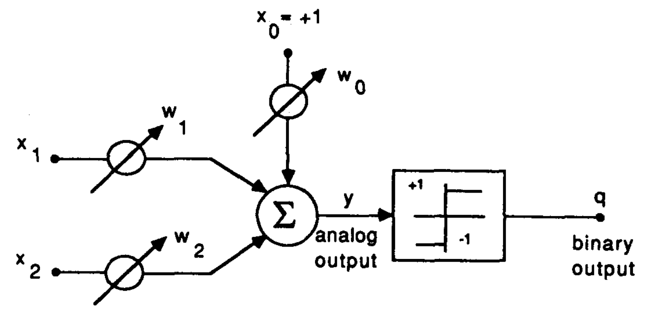


Figure 13. A two-input neuron.

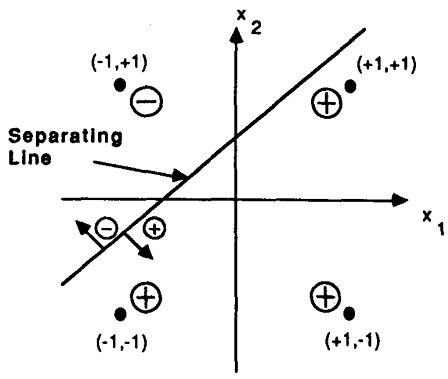


Figure 14. Separating line in pattern space.

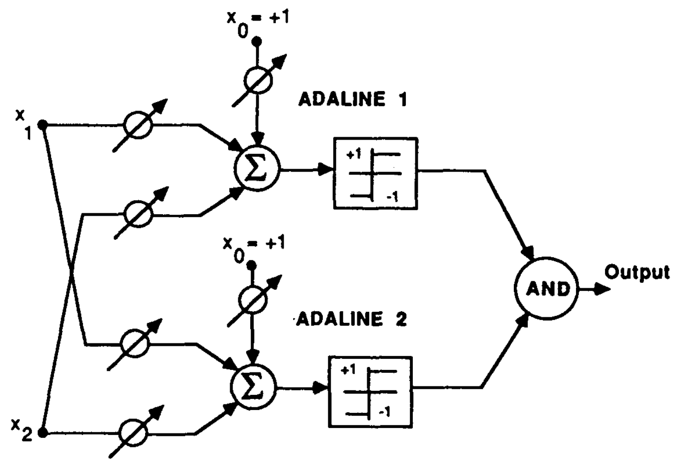


Figure 15. A two-neuron form of Madaline (many Adalines).

$$\therefore x_2 = -(w_0/w_2) - (w_1/w_2)\,x_1 \qquad (7)$$

Figure 14 graphs this linear relation, which comprises a separating line having slope and intercept of

$$\begin{aligned} slope &= -(w_1/w_2) \\ intercept &= -(w_0/w_2) \end{aligned} \qquad (8)$$

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.

The input/output mapping obtained in Figure 14 illustrates an example of a linearly separable function. An example of a nonlinearly separable function with two inputs is the following:

$$\begin{aligned} (+1, +1) &\rightarrow +1 \\ (+1, -1) &\rightarrow -1 \\ (-1, -1) &\rightarrow +1 \\ (-1, +1) &\rightarrow -1 \end{aligned} \qquad (9)$$

No single line exists that can achieve this separation of the input patterns.

With two inputs, a single neuron can realize almost all possible logic functions. With many inputs, however, only a small fraction of all possible logic functions are linearly separable. The single neuron can realize only linearly separable functions and generally cannot realize most functions. However, combinations of neurons or networks of neurons can be used to realize nonlinearly separable functions.

Nonlinear separability—Madaline networks. In the early 1960s at Stanford, Ridgway[6] initiated an approach to the implementation of nonlinearly separable logic functions. He connected retinal inputs to adaptive neurons in a single layer and, in turn, connected their outputs to a fixed logic device providing the system output. Methods for adapting such nets were developed at that time. In the example network shown in Figure 15, two Adalines are connected to an AND logic device to provide an output. Systems of this type were called Madalines (many Adalines). Today such systems would be called small neural nets.

With weights suitably chosen, the separating boundary in pattern space for the system of Figure 15 would be as shown in Figure 16. This separating boundary implements the nonlinearly separable logic function of Equation 9.

Madalines were constructed with many more inputs, with many more neurons in the first layer, and with fixed logic devices such as AND, OR, and MAJority vote-taker in the second layer. Those three functions, as illustrated in Figure 17, are in themselves threshold logic functions. The weights given will implement these functions, but the weight choices are not unique.

Layered neural nets. The Madalines of the 1960s had adaptive first layers and fixed threshold functions for the second (output) layers.[6,7] The feed-forward neural nets of the 1980s have many layers, and all layers are adaptive. The back-propagation method of Rumelhart et al.[8] is perhaps the best-known example of a multilayer network. A three-layer feed-forward adaptive network is illustrated in Figure 18.

Adapting the neurons in the output layer is simple, since the desired responses for the entire network (given with each input training pattern) are the desired responses for the corresponding output neurons. Given the desired responses, adaptation of the output layer can be a straightforward exercise of the LMS algorithm. The fundamental difficulty associated with adapting a layered network lies in obtaining desired responses for the neurons in the layers other than the output layer. The back-propagation algorithm (first reported by Werbos[9] and later rediscovered by Parker[10] and by Rumelhart et al.[8]) is one method for establishing desired responses for the neurons in the "hidden layers," those layers whose neuronal outputs do not appear directly at the system output (see Figure 18). There is nothing unique about the choice of desired outputs for the hidden layers.

Generalization in layered networks is a key issue. The question is, how well do multilayered networks perform with inputs for which they were not specifically trained? The question of generalization is important, and theorists are developing some good examples where useful generalizations take place. Many different algorithms may be needed for the adaptation of multilayered networks to produce required generalizations. Without generalization, neural nets will be of little engineering significance. Merely learning the training patterns can be accomplished by storing these patterns and their associated desired responses in a look-up table.

The layered networks of Rumelhart et al.[8] use neuronal elements like the Adaline of Figure 3, except that the quantizer

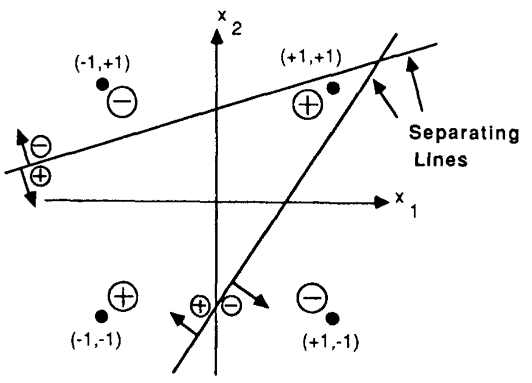


Figure 16. Separating boundaries for the Madaline of Figure 15.

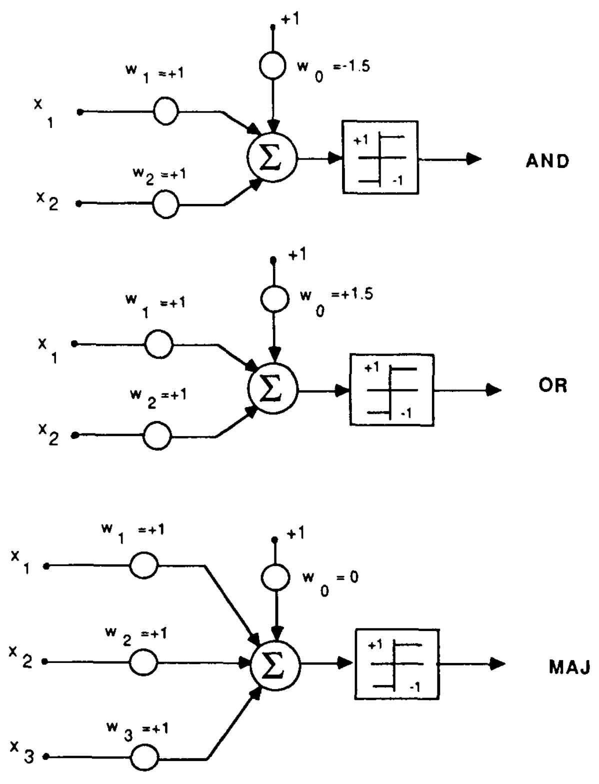


Figure 17. Neuronal implementation of AND, OR, and MAJ logic functions.

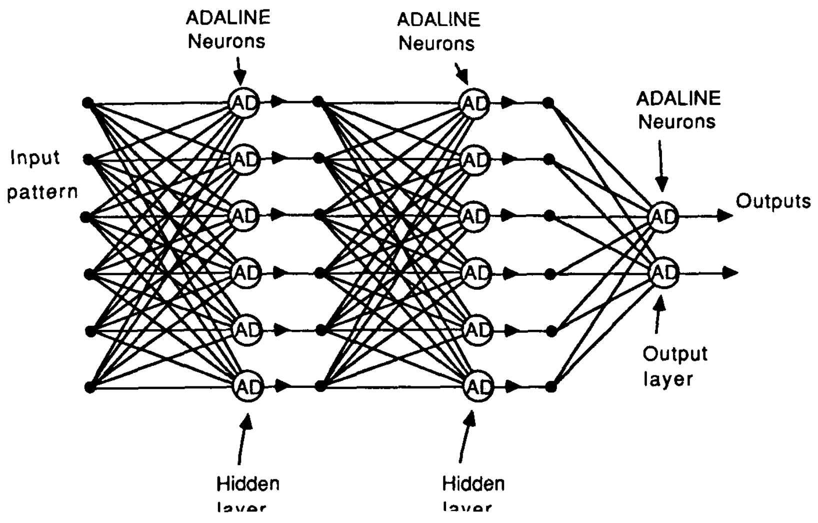


Figure 18. A three-layer adaptive neural network.

or threshold device is a soft-limiting "sigmoid" function rather than the hard-limiting "signum" function of the Adaline. The various back-propagation algorithms for adapting layered networks of neurons require differentiability along the network's signal paths and cannot work with the Adaline element's hard limiter. The sigmoid function has the necessary differentiability. However, it presents implementational difficulties if the neural net is ultimately constructed digitally. For this reason, we developed a new algorithm for adapting layered networks of Adaline neurons with hard-limiting quantizers. The new algorithm is an extension of the original Madaline adaptation rule[6,7] and is called Madaline rule II or MRII. The idea is to adapt the network to properly respond to the newest input pattern while minimally disturbing the responses already trained-in for the previous input patterns. Unless this principle is practiced, it is difficult for the network to simultaneously store all of the required pattern responses.

LMS or Widrow-Hoff delta rule for the single neuron. The LMS algorithm applied to the adaptation of the weights of a single neuron embodies a *minimal disturbance principle*. This algorithm can be written as

$$W_{k+1} = W_k + \alpha \varepsilon_k X_k / |X_k|^2 \qquad (10)$$

The time index or adaption 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, X_k is the present input pattern vector, and ε_k is the present error (that is, the difference between the desired response d_k and the analog output before adaptation). Applying the above recursion formula to each adaption cycle reduces the error by the fraction α. That is, at the kth cycle, the error is

$$\varepsilon_k = d_k - X_k^T W_k \qquad (11)$$

Changing the weights changes (reduces) the error:

$$\begin{aligned}\Delta\varepsilon_k &= \Delta(d_k - X_k^T W_k) \\ &= -X_k^T \Delta W_k \qquad (12)\end{aligned}$$

In accordance with the LMS rule (Equation 10), the weight change is as follows:

$$\begin{aligned}\Delta W_k &= W_{k+1} - W_k \\ &= \alpha \varepsilon_k X_k / |X_k|^2 \qquad (13)\end{aligned}$$

Combining Equations 12 and 13, we obtain

$$\begin{aligned}\Delta\varepsilon_k &= -X_k^T \alpha \varepsilon_k X_k / |X_k|^2 \\ &= -\alpha \varepsilon_k X_k^T X_k / |X_k|^2 \qquad (14) \\ &= -\alpha \varepsilon_k\end{aligned}$$

Therefore, the error is reduced by a factor of α as the weights are changed while holding the input pattern fixed. Putting in 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 choice of α controls stability and speed of convergence.[1] Stability requires that

$$2 > \alpha > 0 \qquad (15)$$

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

$$1.0 > \alpha > 0.1 \qquad (16)$$

The weights change proportionately with their inputs in accordance with the LMS algorithm (see Equation 10). With the usual binary inputs 1 and 0, no adaptation occurs for weights with 0 inputs.

Thus, the symmetric inputs + 1 and − 1 are generally preferred.

To verify that the LMS rule embodies the minimal disturbance principle, refer to Equation 13. The weight-change vector ΔW_k is chosen to be parallel to the input pattern vector X_k. From Equation 12, the change in the error is equal to the negative dot product of X_k with ΔW_k, thus achieving the needed error correction with the smallest magnitude of weight vector change. When adapting to respond properly to a new input pattern, the responses to previous training patterns are therefore minimally disturbed, on the average. The algorithm also minimizes mean square error,[1] the property for which it is best known.

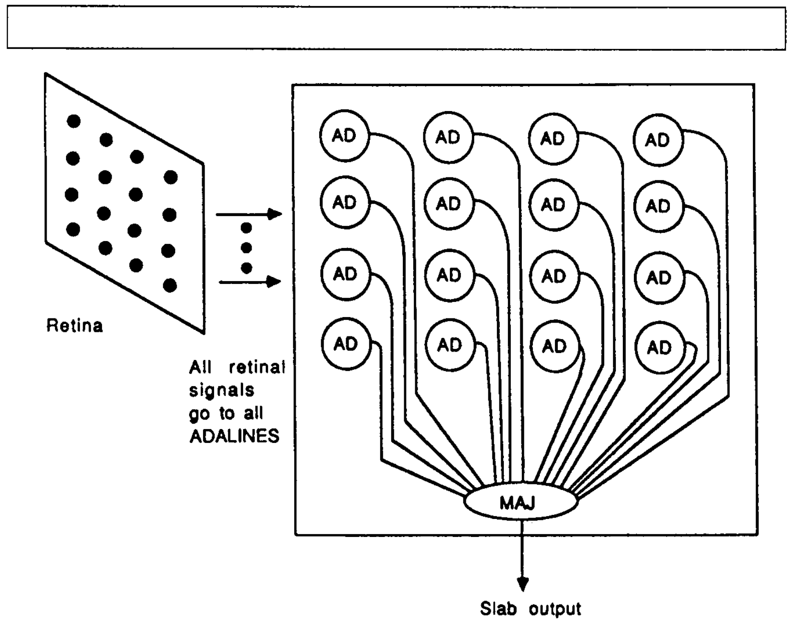


Figure 19. One slab of a left-right, up-down translation invariant network.

Adaptation of layered neural nets by the MRII rule. The minimal disturbance principle can be applied to the adaptation of Figure 18's layered neural network. Presenting an input pattern and its associated desired responses to the network, the training objective is to reduce the number of errors to as low a level as possible. Accordingly, when the first training pattern is presented, the first layer will adapt as required to reduce the number of response errors at the final output layer. In accordance with the minimal disturbance principle, the first-layer neuron whose analog response is closest to zero is given a trial adaptation in the direction to reverse its binary output. When the reversal takes place, the second-layer inputs change, the second-layer outputs change, and consequently the network outputs change. A check is made to see if this reduces the number of output errors for the current input pattern. If so, the trial change is accepted. If not, the weights are restored to their previous values, and the first-layer neuron whose analog response is next closest to zero is trial adapted, reversing its response. If this reduces the number of output errors, the change is accepted. If not, the weights are restored, and one goes on to adaptively switch the neuron with an analog response next closest to zero, and so on, disturbing the neurons as little as possible. After adapting all neurons whose output reversals reduced the number of output errors, neurons are then chosen in pair combinations and trial adaptations are made which can be accepted if output errors are reduced. After adapting the first-layer neurons in singles, pairs, triples, etc., up to a predetermined limit in combination size (simulation results indicate pairwise trials are sufficient in layers having up to 25 Adalines), the second layer is adapted to further reduce the number of network output errors. The method of choosing the neurons to be adapted in the second layer is the same as that for the first layer. If further error reduction is needed, the output layer is then adapted. This is straightforward, since the output-layer desired responses are the given desired responses for the entire network. After adapting the output layer, the responses will be correct. The next input pattern vector and its associated desired responses are then applied to the neural network, and the adaptive process resumes.

When training the network to respond correctly to the various input patterns, the "golden rule" is *give the responsibility to the neuron or neurons that can most easily assume it.* In other words, *don't rock the boat* any more than necessary to achieve the desired training objective. (Simulation results using this minimal-disturbance MRII algorithm are presented later.)

Application of layered networks to pattern recognition. It would be useful to devise a neural net configuration that could be trained to classify an important set of training patterns as required, but have these network responses be invariant to translation, rotation, and scale change of the input pattern within the field of view. It should not be necessary to train the system with the specific training patterns of interest in all combinations of translation, rotation, and scale.

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

Invariance to up-down, left-right pattern translation. Figure 19 shows a planar network configuration (a "slab" of neurons) that could be used to map a retinal image into a single-bit output so that, with proper weights in the network's neurons, the response will be insensitive to left-right and/or up-down translation. The same slab structure can be replicated, with different weights, to allow the retinal pattern to be independently mapped into additional single-bit outputs, all insensitive to left-right, up-down translation.

Figure 20 illustrates the general idea. A retinal image having a given number of

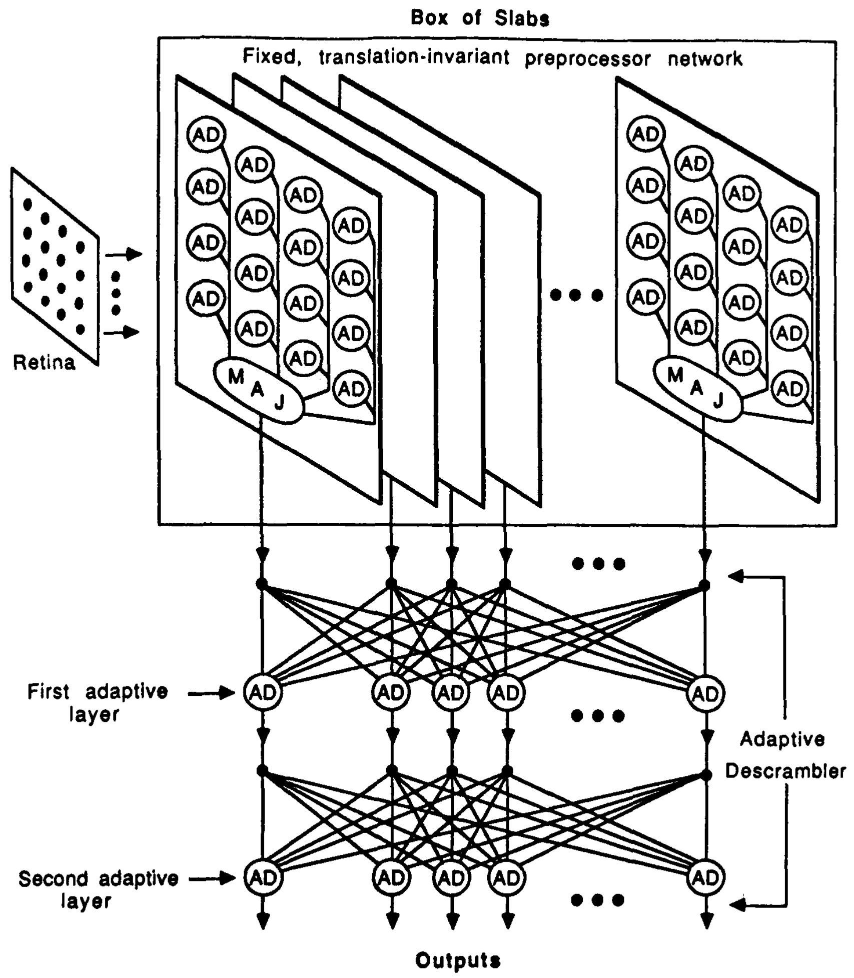


Figure 20. A translation-invariant preprocessor network and an adaptive two-layer descrambler network.

pixels can be mapped through an array of slabs into a different image having the same, more, or fewer pixels, depending on the number of slabs used. In any event, the mapped image is insensitive to up-down, left-right translation of the original image. The mapped image in Figure 20 is fed to a set of Adaline neurons that can be easily trained to provide output responses to the original image as required. This amounts to a "descrambling" of the preprocessor's outputs. The descrambler's output responses classify the original input images and, at the same time, are insensitive to their left-right, up-down translations.

In the systems of Figures 19 and 20, the elements labeled "AD" are Adalines. Those labeled "MAJ" are majority vote-takers. (If the number of input lines to MAJ is even and there is a tie vote, these elements are biased to give a positive response.) The AD elements are adaptive neurons and the MAJ elements are fixed neurons, as in Figure 17.

In the system shown in Figure 19, the structuring of the weights so that the output is insensitive to left-right and up-down translation needs further explanation. Let the weights of each Adaline be arranged in a square array and the corresponding retinal pixels arrayed in a square pattern. Let the square matrix (W_1) designate the array of weights of the upper-left Adaline, and let $T_{D1}(W_1)$ be the array of weights of the next lower Adaline. The operator T_{D1} represents "translate down one," so the second set of weights is the same as the topmost set, but translated down en masse by one pixel. The bottom row wraps around to comprise the top row. The patterns on the retina itself wrap around on a cylinder when they undergo translation. The weights of the next lower Adaline are $T_{D2}(W_1)$, and those of the next lower Adaline are $T_{D3}(W_1)$. Returning to the upper-left Adaline, let its neighbor to the right be designated by $T_{R1}(W_1)$, with T_{R1} being a "translate right one" operator. The pattern of weights for the entire array of Adalines in Figure 19 is

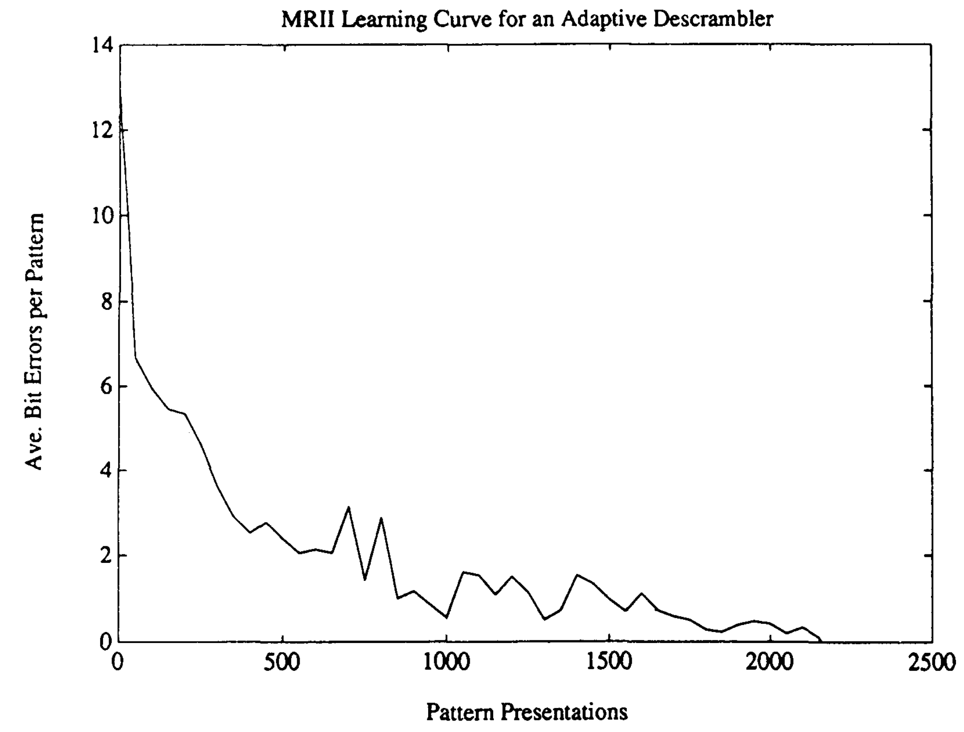


Figure 21. Learning curve for a two-layer, 25 × 25 adaptive descrambler.

$$\begin{array}{cccc} (W_1) & T_{R1}(W_1) & T_{R2}(W_1) & T_{R3}(W_1) \\ T_{D1}(W_1) & T_{R1}T_{D1}(W_1) & T_{R2}T_{D1}(W_1) & T_{R3}T_{D1}(W_1) \\ T_{D2}(W_1) & T_{R1}T_{D2}(W_1) & T_{R2}T_{D2}(W_1) & T_{R3}T_{D2}(W_1) \\ T_{D3}(W_1) & T_{R1}T_{D3}(W_1) & T_{R2}T_{D3}(W_1) & T_{R3}T_{D3}(W_1) \end{array} \tag{17}$$

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

The set of "key" weights (W_1) can be randomly chosen. Once chosen, they can be translated according to Equation 17 to fill out the array of weights for the system of Figure 19. This array of weights can be incorporated as the weights for the first slab of Adalines shown in Figure 20. The weights for the second slab would require the same translational symmetries, but be based on a different randomly chosen set of key weights (W_2). The mapping function of the second slab would therefore be distinct from that of the first slab.

The translational symmetries in the weights called for in Figure 19 could be fixed and manufactured in, or they could be arrived at through training. If, when designing an application-specific pattern recognition system, one knew that translational invariance would be required, it would make sense to manufacture the appropriate symmetry into a fixed weight system, leaving only the final-output Adaline layers plastic and trainable (see Figure 20). Such a preprocessor would definitely work, would provide very high speed response without scanning and searching for pattern location and alignment, and would be an excellent application of neural nets.

Invariance to rotation. Figure 20 represents a system for preprocessing retinal patterns with a translation-invariant fixed neural net followed by a two-layer adaptive descrambler net. The system can be expanded to incorporate rotational invariance. Suppose that all input patterns can be presented in "normal" vertical orientation, approximately centered within the field of view of the retina. Suppose further that all input patterns can be presented when rotated from normal by 90, 180, and 270 degrees. Thus, each pattern can be presented in all four rotations and in all possible left-right, up-down translations. The number of combinations would be large. The problem is to design a neural net preprocessor that is invariant to translation and to rotation by 90 degrees.

Begin with a single slab of Adaline elements, as shown in Figure 19, producing a majority output that is insensitive to translation of the input pattern on the retina. Next, replicate this slab four-fold, and let the majority outputs feed into a single majority output element. In the first slab, (W_1) designates the upper-left Adaline's matrix of weights. (See Equation 17 for the weight matrices of all first-slab Adalines.)

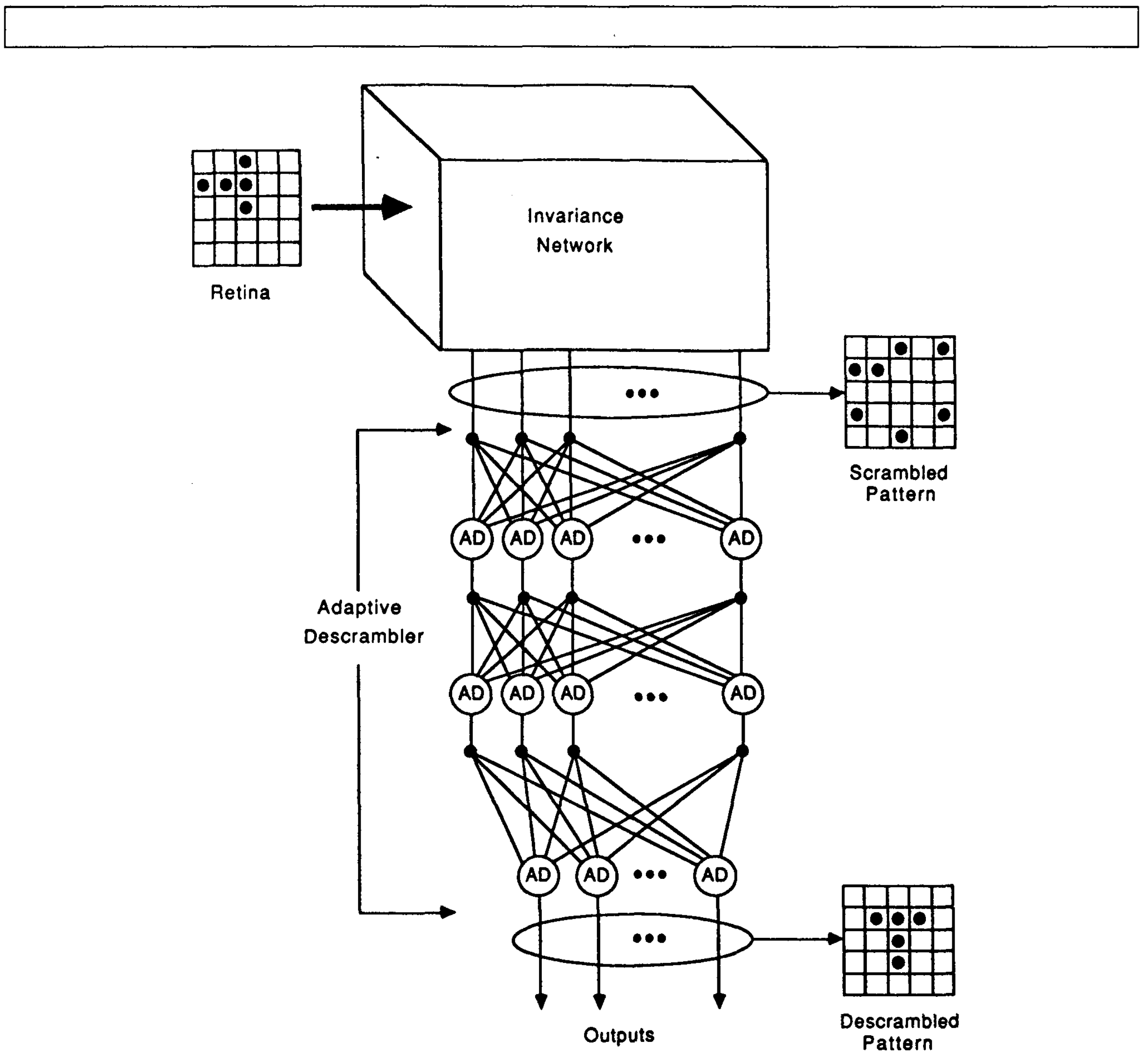


Figure 22. A Madaline system for pattern recognition.

In the second slab, the upper-left Adaline's weight matrix corresponds to the first-slab weight matrix rotated 90 degrees clockwise. This can be designated by $R_{C1}(W_1)$, and the corresponding third- and fourth-slab weight matrices can be designated by $R_{C2}(W_1)$ and $R_{C3}(W_1)$. Thus, the weight matrices of the upper-left Adalines begin with (W_1) in the first slab, and are rotated clockwise by 90 degrees in the second slab, by 180 degrees in the third slab, and by 270 degrees in the fourth slab. The weight matrices of all slabs are translated right and down, in the fashion of Equation 17, starting with the Adalines in the upper left-hand corner. For example, the array of weight matrices for the second slab is

$$\begin{matrix} R_{C1}(W_1) & T_{R1}R_{C1}(W_1) & T_{R2}R_{C1}(W_1) & T_{R3}R_{C1}(W_1) \\ T_{D1}R_{C1}(W_1) & T_{R1}T_{D1}R_{C1}(W_1) & T_{R2}T_{D1}R_{C1}(W_1) & T_{R3}T_{D1}R_{C1}(W_1) \\ T_{D2}R_{C1}(W_1) & T_{R1}T_{D2}R_{C1}(W_1) & T_{R2}T_{D2}R_{C1}(W_1) & T_{R3}T_{D2}R_{C1}(W_1) \\ T_{D3}R_{C1}(W_1) & T_{R1}T_{D3}R_{C1}(W_1) & T_{R2}T_{D3}R_{C1}(W_1) & T_{R3}T_{D3}R_{C1}(W_1) \end{matrix} \tag{18}$$

Clearly, translating the pattern on the retina does not change the majority output response. Rotating the pattern 90 degrees causes an interchange of the roles of the slabs in making their responses, but, since the output majority element weights them equally, the output response is unchanged. Insensitivity to 45-degree rotation can be accomplished by using more slabs; thus, a complete neural network providing invariance to rotation and translation could be constructed. Each translation-invariant slab of Figure 20 would need to be replaced by the rotation-invariant multiple slab and majority-element system described above.

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

Simulation results. The system of Figure 20 was computer simulated. The training set consisted of 36 patterns, each arranged on a 5×5 pixel retina in "standard" position. Twenty-five slabs, each with 25 Adalines having weights fixed according to Equation 17, were used in the translation-invariant preprocessor. The preprocessor output represented a scrambled version of the input pattern. The nature of this scrambling was determined by the choice of the key weight matrices $(W_1), \ldots, (W_{25})$. These key weights were chosen randomly, the only requirement was that the input pattern to preprocessor output map be one-to-one. (This choice of weights produced a very noise-intolerant mapping. We are investigating methods of training in the key weights, using MRII to customize them to the training set.)

We used MRII to train the descrambler, a two-layer system with 25 Adalines in each layer. The initial descrambler weights were chosen randomly and independently, distributed uniformly on the interval $(-1, +1)$. Patterns were presented in random order, each pattern being equally likely of being the next presented. The desired response used was the training pattern in standard position. The system as a whole would then recognize any trained-in pattern in any translated position on the input retina and reproduce it in standard position at the output. Figure 21, a typical descrambler learning curve, graphs the number of incorrect pixels at the output, averaged over the training set, every 50 pattern presentations.

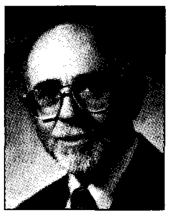
Much work on MRII remains to be done, including detailed studies of its convergence properties and its ability to produce generalizations. Preliminary results are very encouraging. Applying the algorithm to problems will lead to insights that will, we hope, allow a mathematical analysis of the algorithm.

The concept of using an invariance preprocessor followed by a descrambler is a potentially powerful one. We plan to apply the concept to a speech recognition problem. When a word is spoken by different people or even by the same person at different times, the sounds produced differ greatly but remain recognizable as the same word—at least to a human. Therefore, those sounds must have properties that are invariant from utterance to utterance. We believe a system similar to the one in Figure 20 would be useful in developing a multiuser speech recognition system.

The general pattern-recognition concept we've described involves use of an invariance net followed by a trainable classifier. Figure 22 illustrates the key ideas. The invariance net can be trained or designed to produce a set of outputs that are insensitive to translation, rotation, scale change, etc., of the retinal pattern. These outputs are scrambled, but the adaptive layers can be trained to descramble them and reproduce the original patterns in "standard" position, orientation, and scale. Multilayer adaptation algorithms are essential to making such a scheme work, and we've devised a new Madaline adaptation rule—MRII—for that purpose. Our preliminary experimental results indicate that it works and is effective. □

References

1. B. Widrow and S.D. Stearns, *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, N.J., 1985.
2. B. Widrow, "Generalization and Information Storage in Networks of Adaline 'Neurons'," in *Self-Organizing Systems 1962*, M.C. Yovitz, G.T. Jacobi, and G.. Goldstein, eds., Spartan Books, Washington, DC, 1962, pp. 435-461.
3. T. Kailath, *Lectures on Wiener and Kalman Filtering*, Springer Verlag, New York, 1981.
4. R.W. Lucky, "Automatic Equalization for Digital Communication," *Bell Syst. Tech. J.*, Vol. 44, Apr. 1965, pp. 547-588.
5. P.M. Lewis II and C.L. Coates, *Threshold Logic*, John Wiley and Sons, New York, 1967.
6. W.C. Ridgway III, *An Adaptive Logic System with Generalizing Properties*, PhD thesis, Stanford Electronics Labs. Rep. 1556-1, Stanford University, Stanford, Calif., Apr. 1962.
7. N. Nilsson, *Learning Machines*, McGraw-Hill, New York,1965.
8. D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing*, Vol. I and II, MIT Press, Cambridge, Mass., 1986.
9. P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, Harvard, Aug. 1974.
10. D.B. Parker, *Learning Logic*, Tech. Rep. TR-47, Center for Computational Research in Economics and Management Science, MIT, Apr. 1985.



Bernard Widrow is a professor of electrical engineering at Stanford University. Before joining the Stanford faculty in 1959, he was with the Massachusetts Institute of Technology, Cambridge. He is presently engaged in research and teaching in systems theory, pattern recognition, adaptive filtering, and adaptive control systems. He is an 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).

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. A fellow of the IEEE and of the American Association for the Advancement of Science, Widrow received the IEEE Alexander Graham Bell Medal in 1986 for exceptional contributions to the advancement of telecommunications.

Rodney Winter is a captain in the United States Air Force and a graduate student attending Stanford University through the Civilian Institutes Program of the Air Force Institute of Technology. His research interests include signal processing, pattern recognition, and adaptive systems. His doctoral thesis topic is the application of neural networks to optimal nonlinear filtering. His prior duties were those of a pilot, most recently in the F-106 interceptor.

Winter received his BSEE and MSEE degrees from Purdue University in 1977. He is a member of Eta Kappa Nu and a student member of the IEEE.

The authors' address is Information Systems Laboratory, Dept. of Electrical Engineering, Stanford University, Stanford, CA 94305.