Chapter 7

Artificial Neural Networks

Biological systems perform pattern recognition using interconnections of large numbers of cells called neurons. The large number of parallel neural connections makes the human information processing system adaptable, context-sensitive, error-tolerant, large in memory capacity, and real-time responsive. These characteristics of the human brain provide an alternative model to the more common serial, single-processor signal processing architecture. Although each human neuron is relatively slow in processing information (on the order of milliseconds), the overall processing of information in the human brain is completed in a few hundred milliseconds. The processing speed of the human brain suggests that biological computation involves a small number of serial steps, each massively parallel. Artificial neural networks attempt to mimic the perceptual or cognitive power of humans using the parallel-processing paradigm. Table 7.1 compares the features of artificial neural networks and the more conventional von Neumann serial signal processing architecture.

Table 7.1 Comparison of artificial neural network and von Neumann architectures.

Artificial Neural Network	von Neumann
No separate arithmetic and memory units and thus no von Neumann bottleneck	Separate arithmetic and memory units
Simple devices densely interconnected	Many microcomputers connected in parallel
Programmed by specifying the architecture and the learning rules used to modify the interconnection weights	Programmed with high-level, assembly, or machine languages
Finds approximate solutions quickly	Must be specifically programmed to find each type of desired solution
Fault tolerance may be achieved through the normal artificial neural network architecture	Fault tolerant through specific programming or use of parallel computers

7.1 Applications of artificial neural networks

Artificial neural network applications include recognition of visual images of shapes and orientations under varied conditions; speech recognition where pitch, rate, and volume vary from sample to sample; and adaptive control. These applications typically involve character recognition, image processing, and direct and parallel implementations of matching and search algorithms.[1,2]

Artificial neural networks can be thought of as a trainable nonalgorithmic, blackbox suitable for solving problems that are generally ill defined and require large amounts of processing through massive parallelism. These problems possess the following characteristics:

- A high-dimensional problem space;
- Complex interactions between problem variables;
- Solution spaces that may be empty, contain a unique solution, or (most typically) contain a number of useful solutions.

The computational model provided by artificial neural networks has the following attributes:

- A variable interconnection of simple elements or units;
- A learning approach based on modifying interelement connectivity as a function of training data;
- Use of a training process to store information in an internal structure that enables the network to correctly classify new similar patterns and thus exhibit the desired associative or generalization behavior;
- A dynamic system whose state (e.g., unit outputs and interconnection weights) changes with time in response to external inputs or an initial unstable state.

7.2 Adaptive linear combiner

The basic building block of nearly all artificial neural networks is the adaptive linear combiner[1] shown in Figure 7.1. Its output s_k is a linear combination of all its inputs. In a digital implementation, an input signal vector or input pattern vector $\mathbf{X}_k = [x_{0k}, x_{1k}, x_{2k}, \ldots, x_{nk}]^T$ and a desired response d_k (a known response to the special input used to train the combiner) are applied at time k. The symbol T indicates a transpose operation. The components of the input vector are weighted by a set of coefficients called the weight vector $\mathbf{W}_k = [w_{0k}, w_{1k}, w_{2k}, \ldots, w_{nk}]^T$. The output of the network is given by the weighted input vector, denoted by the

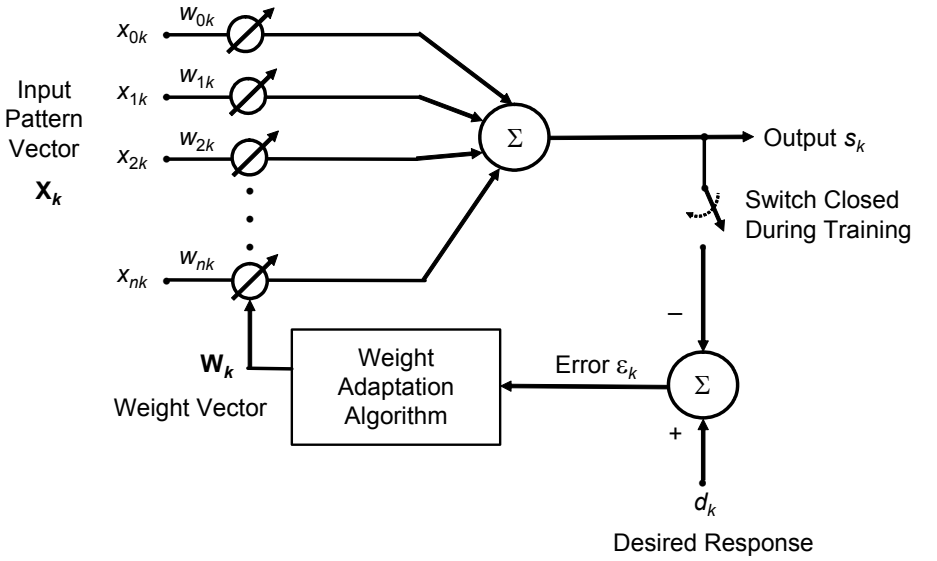


Figure 7.1 Adaptive linear combiner.

inner product $s_k = \mathbf{X}_k^T \mathbf{W}_k$. The components of $\mathbf{X}_k$ may be either analog or binary. The weights are continuously variable positive or negative numbers.

During the training process, a number of input patterns and corresponding desired responses are presented to the linear combiner. An adaptation algorithm is used to automatically adjust the weights so that the output responses to the input patterns are as close as possible to their respective desired responses. The simple least mean square (LMS) algorithm is commonly used to adapt the weights in linear neural networks. This algorithm evaluates and minimizes the sum of squares of the linear errors ε_k over the training pattern set. The linear error is defined as the difference between the desired response d_k and the linear output s_k at time k.

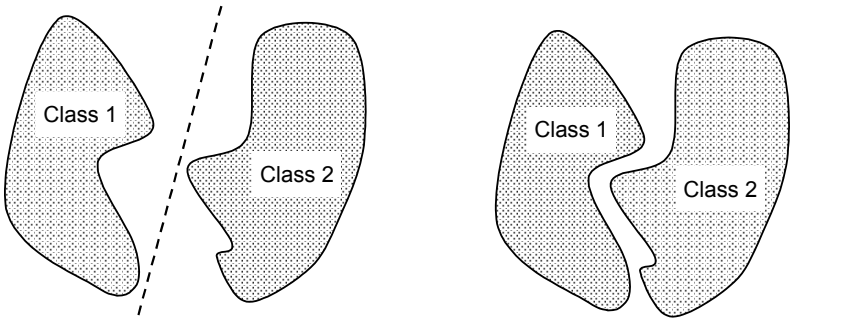
7.3 Linear classifiers

Both linear and nonlinear artificial neural networks have been developed. The nonlinear classifiers can correctly classify a larger number of input patterns and are not limited to only linearly separable forms of patterns. They are discussed later in the chapter.

Figure 7.2 illustrates the difference between linearly and nonlinearly separable pattern pairs. Linear separability requires that the patterns to be classified be sufficiently separated from each other such that the decision surfaces are hyperplanes. Figure 7.2(a) illustrates this requirement for a two-dimensional, single-layer perceptron (discussed further in Section 7.8.5). If the two patterns

move too close to each other, as in Figure 7.2(b), they become nonlinearly separable.

One type of linear classifier used in many artificial neural networks is the adaptive linear element or Adaline developed by Widrow and Hoff.[3] This adaptive threshold logic device contains an adaptive linear combiner cascaded with a hard-limiting quantizer as shown in Figure 7.3. Adalines may also be constructed without the nonlinear output device. The quantizer produces a binary ±1 output $y_k = \text{sgn}(s_k)$ where sgn represents the signum function $s_k/|s_k|$. Thus, the output of the summing node of the neuron is +1 if the hard limiter input is



(a) Linearly separable pattern class pair (b) Nonlinearly separable pattern class pair

Figure 7.2 Linearly and nonlinearly separable pattern pairs.

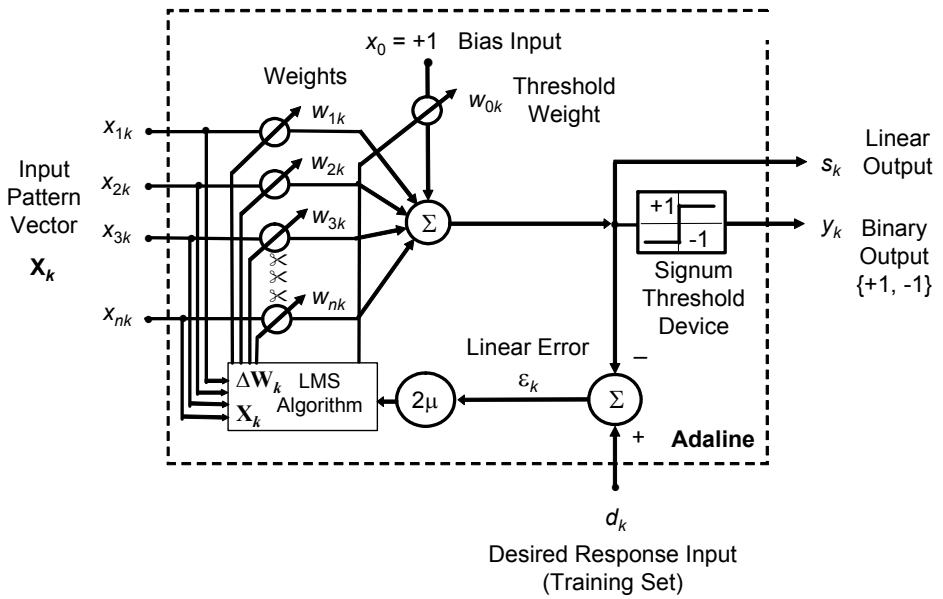


Figure 7.3 Adaptive linear element (Adaline).

positive and –1 if it is negative. The threshold weight w_{0k} connected to the constant input $x_0 = +1$ controls the threshold level of the quantizer.

An adaptive algorithm is utilized to adjust the weights of the Adaline so that it responds correctly to as many input patterns as possible in a training set that has binary desired responses. Once the weights are adjusted, the response of the trained Adaline is tested by applying new input patterns that were not part of the training set. If the Adaline produces correct responses with some high probability, then generalization is said to have occurred.

7.4 Capacity of linear classifiers

The average number of random patterns with random binary desired responses that an Adaline can learn to classify correctly is approximately equal to twice the number of weights. This number is called the statistical pattern capacity C_s of the Adaline. Thus,

$$C_s = 2N_w. \tag{7-1}$$

Furthermore, the probability that a training set is linearly separable is a function of the number N_p of input patterns in the training set and the number N_w of weights including the threshold weight. The probability of linear separability is plotted in Figure 7.4 as a function of the ratio N_p to N_w for several values of N_w. As the number of weights increases, the statistical pattern capacity of the Adaline becomes an accurate estimate of the number of responses it can learn.[4]

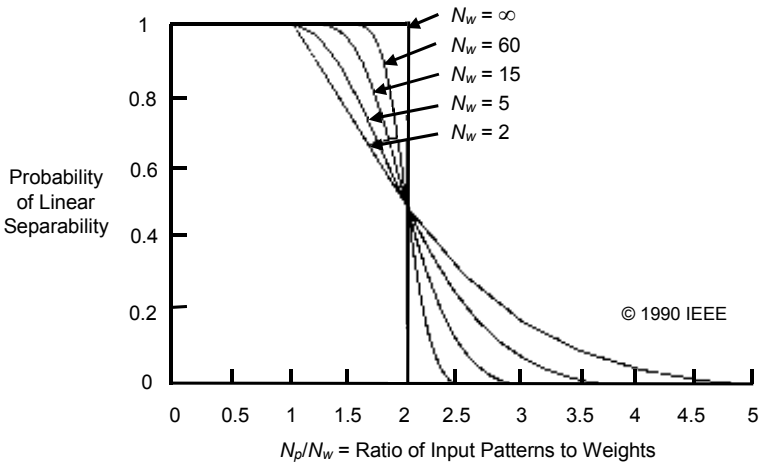


Figure 7.4 Probability of training pattern separation by an Adaline. (B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation," *Proc. IEEE*, 78(9), 1415-1442 [Sept. 1990].)

Figure 7.4 also demonstrates that a problem is guaranteed to have a solution if the number of patterns is equal to or less than half of the statistical pattern capacity, i.e., if the number of patterns is equal to or less than the number of weights. This number of patterns is called the deterministic pattern capacity C_d of the Adaline. The capacity results apply to randomly selected training patterns. Since the training set patterns in most problems of practical interest are not random, but exhibit some statistical regularity, the number of patterns learned often far exceeds the statistical capacity. The increase in the number of learned patterns is due to the regularities that make generalization possible, allowing the Adaline to learn many of the training patterns before they are even presented.

7.5 Nonlinear classifiers

The nonlinear classifier possesses increased capacity and the ability to separate patterns that have nonlinear boundaries. Two types of nonlinear classifiers are described below: the multiple adaptive linear element classifier or Madaline and the multielement, multilayer feedforward network.

7.5.1 Madaline

The Madaline was originally used to analyze retinal stimuli by connecting the inputs to a layer of adaptive Adalines, whose outputs were connected to a fixed logic device that generated the output. An adaptation of this network is illustrated in Figure 7.5 using two Adalines connected to an AND threshold logic output device.

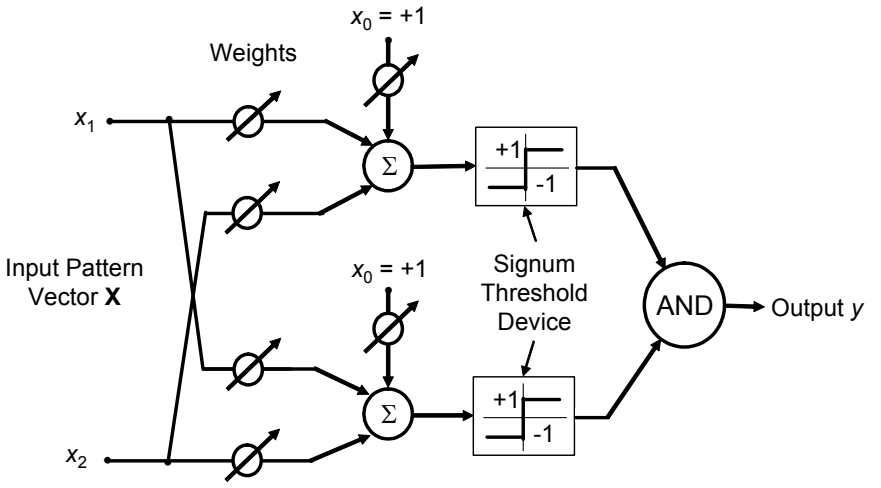
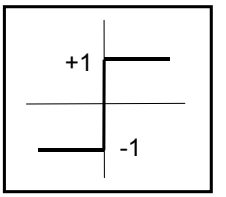
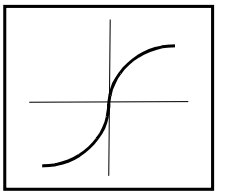


Figure 7.5 Madaline constructed of two Adalines with an AND threshold logic output.

Other types of Madalines may be constructed with many more inputs, many more Adalines in the first layer, and with various logic devices in the second layer. Although the adaptive elements in the original Madalines used the hard-limiting signum quantizers, other nonlinear networks, including the backpropagation network discussed later in this chapter, use differentiable nonlinearities such as sigmoid or *S*-shaped functions illustrated in Figure 7.6.



a. Signum



b. Sigmoid

Figure 7.6 Threshold functions used in artificial neural networks.

The input-output relation for the signum function is denoted by

$$y_k = \text{sgn}(s_k), \tag{7-2}$$

where

$$\text{sgn}(s_k) = \frac{s_k}{|s_k|} \tag{7-3}$$

and s_k and y_k are the linear and binary outputs of the network, respectively.

Figure 7.7 shows implementations of three threshold logic output functions, namely, AND, OR, and MAJORITY vote taker. The weight values in the figure implement these three functions, but the weights are not unique.

For the sigmoid function, the input-output relation is given by

$$y_k = \text{sgm}(s_k). \tag{7-4}$$

A typical sigmoid function is modeled by the hyperbolic tangent as

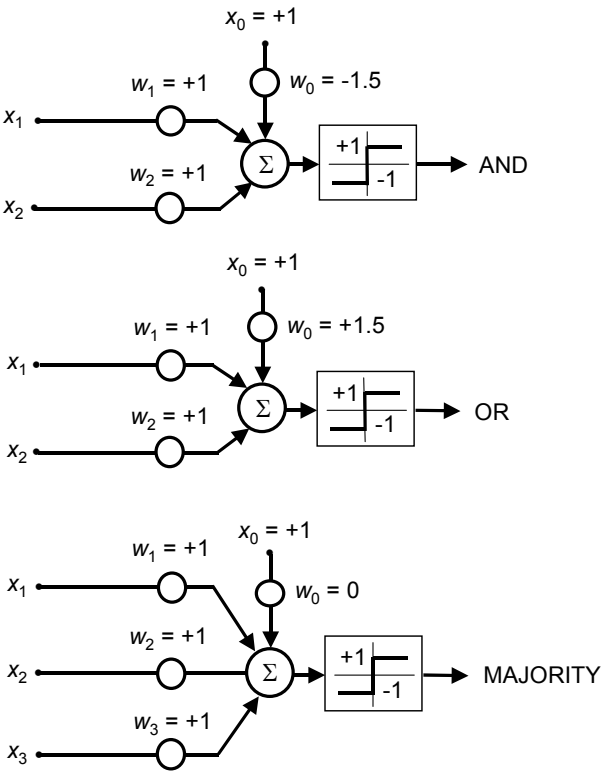


Figure 7.7 Fixed-weight Adaline implementations of AND, OR, and MAJORITY threshold logic functions.

$$y_k = \tanh(s_k) = (1 - e^{-2s_k})/(1 + e^{-2s_k}). \tag{7-5}$$

However, sigmoid functions can be generalized in neural network applications to include any smooth nonlinear function at the output of a linear adaptive element.[2]

7.5.2 Feedforward network

Typical feedforward neural networks have many layers and usually all are adaptive. Examples of nonlinear, layered feedforward networks include multilayer perceptrons and radial-basis function networks,[5] whose characteristics are described later in Table 7.4. A fully connected, three-layer feedforward network is illustrated in Figure 7.8.

Adalines are used in Figure 7.8 to represent an artificial neuron-processing element that connects inputs to a summing node. The inputs are subject to modification by the adjustable weights. The output of the summing node may then pass through a hard or soft limiter. In a fully connected network, each

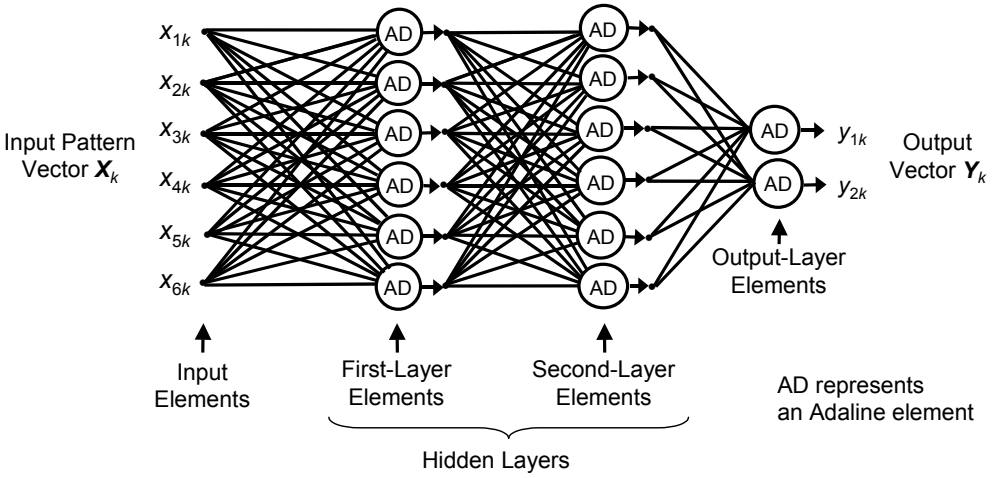


Figure 7.8. Three-layer fully connected feedforward neural network.

processing element receives inputs from every output in the preceding layer. During training, the response of each output element in the network is compared with a corresponding desired response. Error signals associated with the output elements are easily computed, allowing for straightforward adaptation or training of the output layer. However, obtaining error signals for hidden-layer processing elements, i.e., elements in layers other than the output layer, requires more complex learning rules such as the backpropagation algorithm.

In general, a feedforward network is composed of a hierarchy of processing elements. The processing elements are organized in a series of two or more mutually exclusive sets of layers. The input elements are a holding place for the values applied to the network. These elements do not implement a separate mapping or conversion of input data and their weights are insignificant. The last, or output layer, permits the final state of the network to be read. Between these two extremes are zero or more layers of hidden elements. The hidden layers remap the inputs and results of previous layers and, thereby, produce a more separable or more easily classifiable representation of the data. In the architecture of Figure 7.8, links or weights connect each element in one layer to only those in the next higher layer. An implied directionality exists in these connections, whereby the output of one element, scaled by the connecting weight, is fed forward to provide a portion of the activation for the elements in the next higher layer. Forms of feedforward networks, other than that of Figure 7.8, have been developed. In one, the processing elements receive signals directly from each input component and from the output of each preceding processing element.[1]

7.6 Capacity of nonlinear classifiers

The average number of random patterns, having random binary responses, that a Madeline network represented by Figure 7.5 can learn to classify is equal to the capacity per Adaline, or processing element, multiplied by the number of Adalines in the network. Therefore, the statistical capacity C_s of the Madaline is approximately equal to twice the total number of adaptive weights. Although the Madeline and the Adeline have roughly the same capacity per adaptive weight, the Madaline can separate sets with nonlinear separation boundaries.

The capacity of a feedforward signum network with an arbitrary number of layers is dependent on the number of weights N_w and the number of outputs N_y.[6] For a two-layer fully connected feedforward network of signum Adalines with N_x inputs (excluding bias inputs) and N_y outputs, the minimum number of weights N_w is bounded by

$$\frac{N_y N_p}{1+\log_2 N_p} \le N_w < N_y\left(\frac{N_p}{N_x}+1\right)(N_x+N_y+1)+N_y \tag{7-6}$$

when the network is required to learn to map any set of N_p patterns in the general position* into any set of binary desired response vectors with N_y outputs. The statistical and deterministic capacities given above for the linear classifier are also dependent upon the input patterns being in general position. If the patterns are not in general position, the capacity results represent upper bounds to the actual capacity that can be obtained.[1,4]

For a two-layer feedforward signum network with at least five times as many inputs and hidden elements as outputs, the deterministic pattern capacity is bounded *below* by a number slightly smaller than N_w/N_y. For any feedforward network with a large ratio of weights to outputs (at least several thousand), the deterministic pattern capacity is bounded *above* by a number slightly larger than $N_w/N_y \log_2(N_w/N_y)$. Thus, the deterministic pattern capacity C_d of a two-layer network is bounded by

$$(N_w/N_y) - K_1 \le C_d \le N_w/N_y \log_2(N_w/N_y) + K_2, \tag{7-7}$$

*Patterns are in general position with respect to an Adaline that does not contain a threshold weight if any subset of pattern vectors that contains no more than N_w members forms a linearly independent set. Equivalently, the patterns are in general position if no set of N_w or more input points in the N_w-dimensional pattern space lay on a homogeneous hyperplane. For an Adaline with a threshold weight, general position occurs when 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.

where K_1 and K_2 are positive numbers that are small if the network is large with few outputs relative to the number of inputs and hidden elements. Equation (7-7) also bounds the statistical capacity of a two-layer signum network.

The following rules of thumb are useful for estimating pattern capacity:

- Single-layer network capacities serve as capacity estimates for multilayer networks;
- The capacity of sigmoid (soft limiting) networks cannot be less than that of signum networks of equal size;
- For good generalization, i.e., classification of patterns not presented during training, the training set pattern size should be several times larger than the network's capacity such that $N_p >> N_w / N_y$. Other estimates for the training set size needed for good generalization are given in Appendix E.

Finding the optimum number of hidden elements for a feedforward network is problem dependent and often involves considerable engineering judgment. While intuition may suggest that more hidden elements will improve the generalization capability of the network, excessively large numbers of hidden elements may be counterproductive. A difficulty that may arise if the number of hidden elements grows too large is depicted in Figure 7.9.

The accuracy of the output decision made by this particular artificial neural network quickly approaches a limiting value. The training time rapidly falls when the number of hidden elements is kept below some value that is problem specific. As the number of hidden elements is increased further, the training time increases rapidly, while the accuracy grows much more slowly.[2] The specific values shown in this figure are not general results, but rather apply to a specific neural network application.[7]

7.7 Supervised and unsupervised learning

The description of learning algorithms as supervised or unsupervised originates from pattern recognition theory. Supervised learning uses pattern class information; unsupervised learning does not. Learning seeks to accurately estimate $p(\mathbf{X})$, the probability density function that describes the continuous distribution of patterns $\mathbf{X}$ in the pattern space. The supervision in supervised learning provides information about $p(\mathbf{X})$. However, the information may be

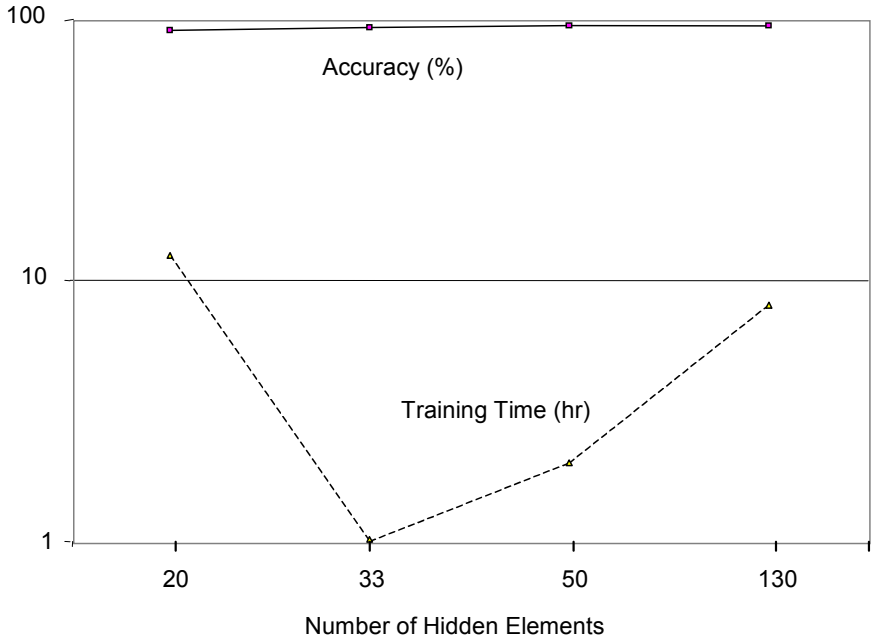


Figure 7.9 Effect of number of hidden elements on feedforward neural network training time and output accuracy for a specific problem. (Adapted from R. Gaborski, "An intelligent character recognition system based on neural networks," Research Magazine, Eastman Kodak Company, Rochester, NY [Spring 1990].)

inaccurate. Unsupervised learning makes no assumptions about $p(\mathbf{X})$. It uses minimal information.[8]

Supervised learning algorithms depend on the class membership of each training sample x. Class membership information allows supervised learning algorithms to detect pattern misclassifications and compute an error signal or vector, which reinforces the learning process.

Unsupervised learning algorithms use unlabeled pattern samples and blindly process them. Unsupervised learning algorithms often have less computational complexity and less accuracy than supervised learning algorithms.[8] Unsupervised learning algorithms learn quickly, often on a single pass of noisy data. Thus, unsupervised learning is applied to many high-speed real-time problems where time, information, or computational precision is limited.

Examples of supervised learning algorithms include the steepest descent and error correction algorithms that estimate the gradient or error of an unknown mean-squared performance measure. The error depends on the unknown probability density function $p(\mathbf{X})$.

Unsupervised learning may occur in several ways. It may adaptively form clusters of patterns or decision classes that are defined by their centroids. Other unsupervised neural networks evolve attractor basins in the pattern state space. Attractor basins correspond to pattern classes and are defined by their width, position, and number.

7.8 Supervised learning rules

Figure 7.10 shows the taxonomy used by Widrow and Lehr to summarize the supervised learning rules developed to train artificial neural networks that incorporate adaptive linear elements.[1] The rules are first separated into steepest descent and error correction categories, then into layered network and single element categories, and finally into nonlinear and linear rules.

Steepest descent or gradient rules alter the weights of a network during each pattern presentation with the objective of reducing mean squared error (MSE) averaged over all training patterns. Although other gradient approaches are available, MSE remains the most popular. Error correction rules, on the other hand, alter the weights of a network to reduce the error in the output response to the current training pattern. Both types of rules use similar training procedures. However, since they are based on different objectives, they may have significantly different learning characteristics. Error correction rules are most often applied when training objectives are not easily quantified or when a problem does not lend itself to tractable analysis.

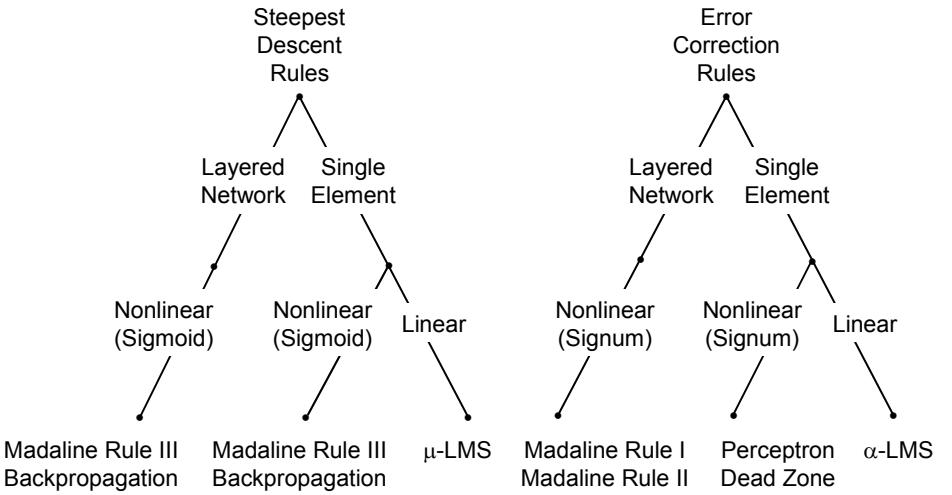


Figure 7.10 Learning rules for artificial neural networks that incorporate adaptive linear elements. (Adapted from B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation," *Proc. IEEE*, 78(9), 1415-1442 [Sept. 1990].)

7.8.1 μ-LMS steepest descent algorithm

The μ-LMS steepest descent algorithm performs approximate steepest descent on the MSE surface in weight space. Since this surface is a quadratic function of the weights, it is convex in shape and possesses a unique minimum. Steepest descent algorithms adjust the network weights by computing or estimating the error between the network output and the desired response to a known input. The weight adjustment is proportional to the gradient formed by the partial derivative of the error with respect to the weight, but in the direction opposite to the gradient.

The algebraic expression for updating the weight vector is given by

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\, \varepsilon_k\, \mathbf{X}_k. \tag{7-8}$$

Stability and speed of convergence are controlled by the learning constant μ. If μ is too small, the μ-LMS algorithm moves very slowly down the estimated mean square error surface and learning may be prohibitively slow. If μ is too large, the algorithm may leap recklessly down the estimated mean square error surface and the learning may never converge. In this case, the weight vector may land randomly at points that correspond to first larger and then smaller values of the total mean square error surface.[8]

The learning constant should vary inversely with system uncertainty. The more uncertain the sampling or training environment the smaller the value of μ should be to avoid divergence of the training process. The learning constant can be larger to speed convergence when there is less uncertainty in the sampling environment.

If the input patterns are independent over time, the mean and variance of the weight vector converge for most practical purposes if

$$0 < \mu < 1/\text{trace}\,[\mathbf{R}_k], \tag{7-9}$$

where trace $[\mathbf{R}_k]$ equals the sum of the diagonal elements of $\mathbf{R}_k$ which, in turn, is equal to the average signal power of the $\mathbf{X}_k$-vector or $E[\mathbf{X}_k{}^T\mathbf{X}_k]$. The variable $\mathbf{R}_k$ may also be viewed as the autocorrelation matrix of the input vectors $\mathbf{X}_k$ when the input patterns are independent.

7.8.2 α-LMS error correction algorithm

Using a fixed input pattern, the α-LMS algorithm optimizes the weights to reduce the error between the network output and the desired response by a factor α. The weight vector update is found as

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha\, \varepsilon_k \frac{\mathbf{X}_k}{|\mathbf{X}_k|^2} \tag{7-10}$$

and the error reduction factor as

$$\Delta\varepsilon_k = -\alpha\, \varepsilon_k. \tag{7-11}$$

The negative sign indicates that the change in error is in the direction opposite to the error itself. Stability and speed of convergence of the algorithm are controlled by the value of α. When the input pattern vectors are independent over time, stability is ensured for most practical purposes when $0 < \alpha < 2$. Values of α greater than 1 overcorrect the error, while total error correction corresponds to $\alpha = 1$. A practical range for α lies between 0.1 and 1.0. When all input patterns are equal in length, the α-LMS algorithm minimizes mean square error and is best known for this property.

7.8.3 Comparison of the μ-LMS and α-LMS algorithms

Both the μ-LMS and α-LMS algorithms rely on the least mean square instantaneous gradient for their implementation. The α-LMS is self-normalizing, with α determining the fraction of the instantaneous error corrected with each iteration, whereas μ-LMS is a constant coefficient linear algorithm that is easier to analyze. The α-LMS is similar to the μ-LMS with a continually variable learning constant. Although the α-LMS is somewhat more difficult to implement and analyze, experiments show that it is a better algorithm than the μ-LMS when the eigenvalues of the input autocorrelation function matrix **R** are highly disparate. In this case, the α-LMS gives faster convergence for a given difference between the gradient estimate and the true gradient. This difference is propagated into the weights as "gradient noise." The μ-LMS has the advantage that it will always converge in the mean to the minimum MSE solution, while the α-LMS may converge to a somewhat biased solution.[1]

7.8.4 Madaline I and II error correction rules

The Madaline I error correction training rule applies to a two-layer Madaline network such as the one depicted in Figure 7.5. The first layer consists of hard-limited signum Adaline elements. The outputs of these elements are connected to a second layer containing a single fixed-threshold logic element, e.g., AND, OR, or MAJORITY vote taker. The weights of the Adalines are initially set to small random values. The Madaline I rule adapts the input elements in the first layer such that the output of the threshold logic element is in the desired state as specified by a training pattern. No more Adaline elements are adapted than necessary to correct the output decision. The elements whose linear outputs are nearest to zero are adapted first, as they require the smallest weight changes to

reverse their output responses. Whenever an Adaline is adapted, the weights are changed in the direction of its input vector because this provides the required error correction with minimal weight change.

The Madaline II error correction rule applies to multilayer binary networks with signum thresholds. Training is similar to training with the Madaline I 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 training, the first-layer Adaline with the smallest linear output is adapted first by inverting its binary output. If the number of output errors produced by the training patterns is reduced by the trial adaptation, the weights of the selected elements are changed by the α-LMS error correction algorithm in a direction that reinforces the bit reversal with minimum disturbance to the weights. If the trial adaptation does not improve the network response, the weight adaptation is not performed. After finishing with the first element, other Adalines in the first layer with sufficiently small linear outputs are adapted. After exhausting all possibilities in the first layer, the next layer elements are adapted, and so on. When the final layer is reached and the α-LMS algorithm has adapted all appropriate elements, a new training pattern is selected at random and the procedure is repeated.

7.8.5 Perceptron rule

In some cases, the α-LMS algorithm may fail to separate training patterns that are linearly separable. In these situations, nonlinear rules such as Rosenblatt's α-perceptron rule may be suitable.[9]

Rosenblatt's perceptron, shown in Figure 7.11, is a feedforward network with one output neuron that learns the position of a separating hyperplane in pattern space. The first layer of fixed threshold logic devices processes a number of input patterns that are sparsely and randomly connected to it. The outputs of the first layer feed a second layer composed of a single adaptive linear threshold element or neuron. The adaptive element is similar to the Adaline, with two exceptions: its input signals are binary {0, 1} and no threshold weight is used.

The adaptive threshold element of the perceptron is illustrated in Figure 7.12. Weights are adapted only if the output decision y_k disagrees with the desired binary response d_k to an input training pattern, whereas the α-LMS algorithm corrects the weights on every trial. The perceptron weight adaptation algorithm adds the input vector to the weight vector of the adaptive threshold element when the quantizer error is positive and subtracts the input vector from the weight vector when the error is negative. The quantizer error, indicated in Figure 7.12 as $\tilde{\varepsilon}_k$, is given by

$$\tilde{\varepsilon}_k = d_k - y_k. \tag{7-12}$$

The perceptron rule is identical to the α-LMS algorithm except that the perceptron uses half the quantizer error, $\tilde{\varepsilon}_k/2$, in place of the normalized linear error $\varepsilon_k/|\mathbf{X}_k|^2$ of the α-LMS algorithm. Thus, the perceptron rule gives the weight vector update as

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha(\tilde{\varepsilon}_k/2)\mathbf{X}_k. \tag{7-13}$$

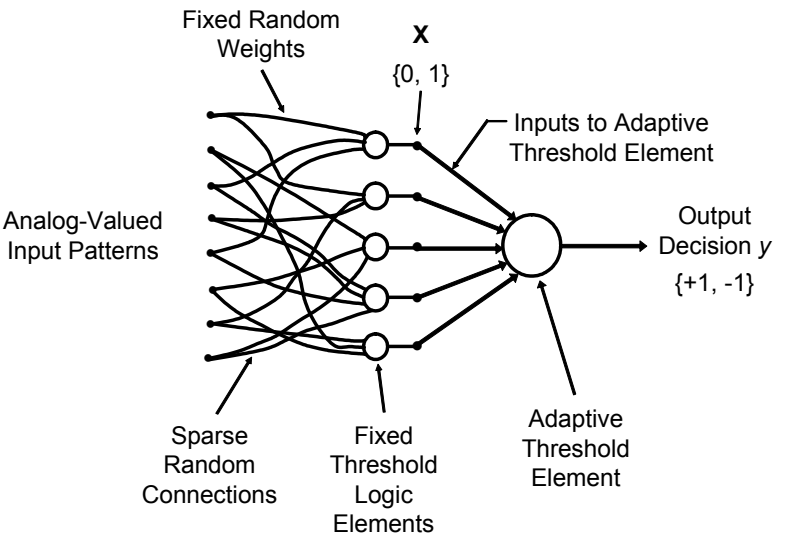


Figure 7.11 Rosenblatt's perceptron.

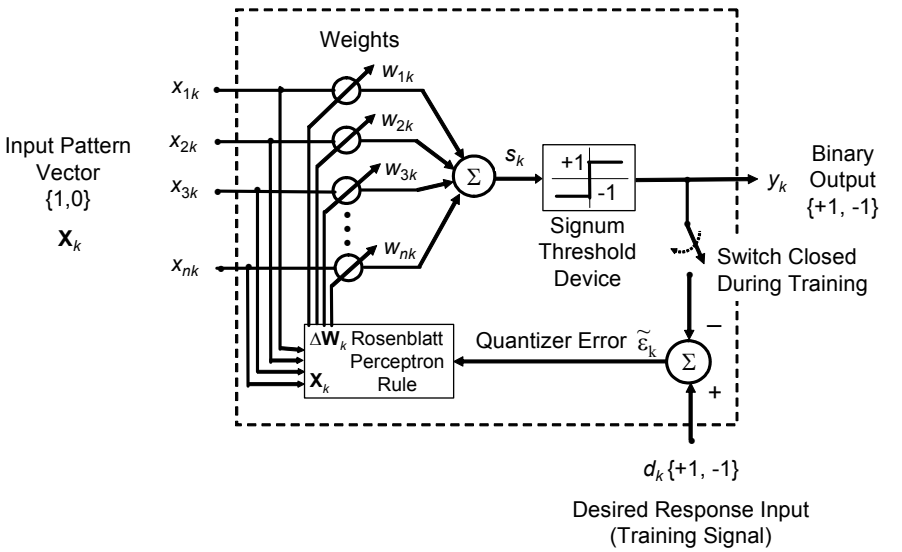


Figure 7.12 Adaptive threshold element of perceptron.

Normally α is set equal to 1. In contrast to α-LMS, α does not affect the stability of the perceptron algorithm. It affects the convergence time only if the initial weight vector is nonzero. While the α-LMS algorithm may be applied to either analog or binary desired responses, the perceptron may only be used with binary desired responses. Although the perceptron was developed in the late 1950s, its widespread application was not extensive because its classification ability was dependent on training with linearly separable patterns and a training algorithm for the multilayer case did not exist. The multilayer feedforward networks and the backpropagation algorithm have helped to remedy these constraints.

Lippmann discusses generalized perceptron architectures with layer configurations similar to those shown in Figure 7.8.[10] With one output node and a hard-limiting nonlinearity, no more than three layers (two hidden layers and one output layer) are required since a three-layer perceptron network can generate arbitrary complex decision regions. The number of nodes in the second hidden layer must be greater than one when decision regions are disconnected or meshed and cannot be formed from one convex area. In the worst case, the number of second layer nodes is equal to the number of disconnected regions in the input distributions. The typical number of nodes in the first hidden layer must be sufficient to provide three or more edges for each convex area generated by every second-layer node. Therefore, there should typically be more than three times as many nodes in the first as the second hidden layer.

Alternatively, Cybenko proved that one hidden layer in a perceptron is sufficient for performing any arbitrary transformation, given enough nodes.[11,12] However, a single layer may not be optimum in the sense of learning time or ease of implementation.

7.8.6 Backpropagation algorithm

The backpropagation algorithm is a stochastic steepest descent learning rule used to train single- or multiple-layer nonlinear networks. The algorithm overcomes some limitations of the perceptron rule by providing a framework for computing the weights of hidden layer neurons. The algorithm's stochastic nature implies a search for a random minimum mean square error separating surface rather than an unknown deterministic mean square error surface. Therefore, the back-propagation algorithm may converge to local error minima or may not converge at all if a poor choice of initial weights is made. The backpropagation algorithm reduces to the μ-LMS algorithm if all neural elements are linear and if the feedforward topology from input to output layers contains no hidden neurons.

Expressed in biological nomenclature, the backpropagation algorithm recursively modifies the synapses between neuronal fields, i.e., input, hidden, and output layers. The algorithm first modifies the synapses between the output layer and the penultimate layer of hidden or interior neurons. Then the algorithm applies

this information to modify the synapses between the penultimate hidden layer and the preceding hidden layer, and so on, until the synapse between the first hidden layer and the input layer is reached.

After the initial small randomly chosen values for the weights are selected, training begins by presenting an input pattern vector **X** to the network. The input values in **X** are swept forward through the network to generate an output response vector **Y** and to compute the errors at the output of each layer, including the hidden layers. The effects of the errors are then swept backwards through the network. The backward sweep (1) associates a mean square error derivative δ with each network element in each layer, (2) computes a gradient from each δ, and (3) updates the weights of each element based upon the gradient for that layer and element. A new pattern is then presented to the network and the process is repeated. Training continues until all patterns in the training set are exhausted. Calculations associated with the backward sweep through the network are roughly as complex as those associated with the forward pass.[1,2,8] The objective of the backpropagation algorithm is not to reduce the mean square error derivatives at each layer in the network. Rather the goal is to reduce the mean squared error (the sum of the squares of the difference between the desired response and actual output at each element in the output layer) at the network output.

When a sigmoid nonlinearity is used in an artificial neural network trained with the backpropagation algorithm, the change in the weight connecting a source neuron i in layer L–1 to a destination neuron j in layer L is given by

$$\Delta w_{ij} = \eta\, \delta_{pj}\, y_{pi}, \tag{7-14}$$

where $p = p^{\text{th}}$ presentation vector, η = learning constant, δ_{pj} = gradient at neuron j, and y_{pi} = actual output of neuron i.[5,10,13,14]

The expression for the gradient δ_{pj} is dependent on whether the weight connects to an output neuron or a hidden neuron. Accordingly, for output neurons

$$\delta_{pj} = (t_{pj} - y_{pj})\, y_{pj}\, (1 - y_{pj}), \tag{7-15}$$

where t_{pj} is the desired signal at the output of the j^{th} neuron. For hidden neurons,

$$\delta_{pj} = y_{pj}\, (1 - y_{pj}) \sum_{k} \delta_{pk}\, w_{kj}\ . \tag{7-16}$$

Thus, the new value for the weights at period $(k + 1)$ is given by

$$w_{ij}(k + 1) = w_{ij}(k) + \Delta w_{ij} = w_{ij}(k) + \eta\, \delta_{pj}(k)\, y_{pi}(k), \tag{7-17}$$

where δ_{pj} is selected from (7-15) or (7-16).

When applying the backpropagation algorithm, the initial weights are normally set to small random numbers. Multilayer networks are sensitive to the initial weight selection, and the algorithm will not function properly if the initial weight values are either zero or poorly chosen nonzero values. In fact, the network may not learn the set of training examples. If this occurs, learning should be restarted with other values for the initial random weights.

The speed of training is affected by the learning constant that controls the step size along which the steepest descent path or gradient proceeds. When broad minima that yield small gradients are present, a larger value of the learning constant gives more rapid convergence. For applications with steep and narrow minima, a small value of the learning constant avoids overshooting the solution. Thus, the learning constant should be chosen experimentally to suit each problem. Learning constants between 10^{-3} and 10 have been reported in literature.[13] Large learning constants can dramatically increase the learning speed, but the solution may overshoot and not stabilize at any network minimum error.

A momentum term, which takes into account the effect of past weight changes, is often added to Eq. (7-17) to obtain more rapid convergence in particular problem domains. Momentum smoothes the error surface in weight space by filtering out high frequency variations. The momentum constant α determines the emphasis given to this term as shown in the modified expression for the weight update, namely

$$w_{ij}(k+1) = w_{ij}(k) + \eta\, \varepsilon_{pj}\, y_{pi} + \alpha[w_{ij}(k) - w_{ij}(k-1)], \tag{7-18}$$

where $0 < \alpha < 1$.

7.8.7 Madaline III steepest descent rule

The Madaline III steepest descent rule is used in networks containing sigmoid Adalines. It avoids some of the problems that occur when backpropagation is used with inaccurate realizations of sigmoid functions and their derivatives. Madaline Rule III works similarly to Madaline Rule II. All the Adalines in the Madaline Rule III network are adapted. However, Adalines whose analog sums are closest to zero will usually be adapted most strongly since the sigmoid has its maximum slope at zero, contributing to high gradient values. As with Madaline Rule II, the objective is to change the weights for the given input training pattern to reduce the sum square error at the network output. The weight vectors of the Adaline elements are selected for adaptation in the LMS direction according to their capabilities for reducing the sum square error at the output. The weight vectors are adjusted in proportion to a small perturbation signal Δs that is added

to the sum s_k at the output of the weight vector (as in Figure 7.12). The effect of the perturbation on output y_k and error $\tilde{\varepsilon}_k$ is noted.

The instantaneous gradient is computed in one of two ways, leading to two forms of the Madaline III algorithm for updating the weights. These are

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \mu \left[\frac{\Delta(\tilde{\varepsilon}_k)^2}{\Delta s}\right] \mathbf{X}_k \tag{7-19}$$

and

$$\mathbf{W}_{k+1} = \mathbf{W}_k - 2\mu\, \tilde{\varepsilon}_k \left[\frac{\Delta \tilde{\varepsilon}_k}{\Delta s}\right] \mathbf{X}_k. \tag{7-20}$$

The learning constant μ is similar to that used in the μ-LMS algorithm. The two forms for updating the weights are equivalent for small perturbations Δs.

7.8.8 Dead zone algorithms

Mays developed two algorithms that incorporate dead zones into the training process.[14] These are the increment adaptation rule and the modified relaxation adaptation rule. Increment adaptation associates a dead zone with the linear outputs s_k, where the dead zone is set equal to $\pm\gamma$ about zero. The dead zone reduces sensitivity to weight errors. If the linear output is outside the dead zone, the weight update follows a variant of the perceptron rule given by

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha\, \tilde{\varepsilon}_k \frac{\mathbf{X}_k}{2|\mathbf{X}_k|^2} \quad \text{if } |s_k| \geq \gamma. \tag{7-21}$$

If the linear output is inside the dead zone, the weights are adapted by another variant of the perceptron rule as

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha\, d_k \frac{\mathbf{X}_k}{|\mathbf{X}_k|^2} \quad \text{if } |s_k| < \gamma, \tag{7-22}$$

where $\tilde{\varepsilon}_k$ is given by Eq. (7-12) and d_k is the desired response defined by the training pattern.

Mays proved that if the training patterns are linearly separable, increment adaptation would always converge and separate the patterns in a finite number of steps. If the training set is not linearly separable, the increment adaptation rule typically performs better than the perceptron rule because a sufficiently large

dead zone causes the weight vector to adapt away from zero when any reasonably good solution exists.[1]

The modified relaxation adaptation rule uses the linear error ε_k, depicted in Figures 7.1 and 7.3 for the α-LMS algorithm, to update the weights. The modified relaxation rule differs from the α-LMS in that a dead zone is created. If the quantizer output y_k is correct and the linear output s_k falls outside the dead zone, the weights are not updated. In this case

$$\mathbf{W}_{k+1} = \mathbf{W}_k \text{ if } \varepsilon_k = 0 \text{ and } |s_k| \geq \gamma. \tag{7-23}$$

If the quantizer output is incorrect or if the linear output falls within the dead zone $\pm\gamma$, the weights are updated following the α-LMS algorithm according to

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha\, \varepsilon_k \frac{\mathbf{X}_k}{|\mathbf{X}_k|^2} \text{ otherwise.} \tag{7-24}$$

7.9 Generalization

Generalization permits the neuron to respond "sensibly" to patterns not encountered during training. Generalization is implemented through a firing rule that determines whether and how a neuron should fire for any input pattern. An example of a firing rule that uses the Hamming distance to decide when a neuron should fire is given below.

Suppose an artificial neural network having three input nodes $X1$, $X2$, $X3$ is trained with patterns that cause the neuron to fire (i.e., the 1-taught set) and others that prevent firing (i.e., the 0-taught set). Patterns not in the training set cause the node to fire if they have more input elements in common with the "nearest" pattern in the 1-taught set than with the nearest pattern in the 0-taught set and vice versa. A tie causes a random output from the neuron.

The truth table in Table 7.2 reflects teaching the neuron to output 1 when input $X1$, $X2$, $X3$ is 111 or 101 and to output 0 when the input is 000 or 001.

Table 7.2 Truth table after training by 1-taught and 0-taught sets.

$X1$	0	0	0	0	1	1	1	1
$X2$	0	0	1	1	0	0	1	1
$X3$	0	1	0	1	0	1	0	1
Output Y	0	0	0/1	0/1	0/1	1	0/1	1

When the input pattern 010 is applied after training, the Hamming distance rule says that 010 differs from 000 in 1 element, from 001 in 2 elements, from 101 in 3 elements, and from 111 in 2 elements. The nearest pattern is 000, which belongs to 0 set. Therefore, the neuron does not fire when the input is equal to 010 since 000 is a member of the 0-taught set.

When the input pattern 011 is applied after training, the Hamming distance rule asserts that 011 is equally distant from its nearest patterns 001 and 111 by 1 element. Since these patterns belong to different output sets, the output of the neuron stays undefined.

When the input pattern 100 is applied after training, the Hamming distance rule shows that 100 is equally distant from its nearest training set patterns 000 and 101 by 1 element. Since these patterns belong to different output sets, the output of the neuron stays undefined.

Applying the Hamming distance rule to the input pattern 110 after training shows that 110 differs from the nearest training set pattern 111 by 1 element. Therefore, the neuron fires when the input is equal to 111 since 111 is a member of the 1-taught training set.

The truth table in Table 7.3 gives the results of the generalization process. Evidence of generalization by the neuron is shown by the different outputs for the 010 and 110 inputs as compared with the original output shown in Table 7.2.

Table 7.3 Truth table after neuron generalization with a Hamming distance firing rule.

$X1$	0	0	0	0	1	1	1	1
$X2$	0	0	1	1	0	0	1	1
$X3$	0	1	0	1	0	1	0	1
Output Y	0	0	0	0/1	0/1	1	1	1

7.10 Other artificial neural networks and processing techniques

Other types of artificial neural networks have been developed in addition to the adaptive linear element (Adaline), multiple adaptive linear element (Madaline), perceptron, and multilayer adaptive linear element feedforward networks. These include the multilayer perceptron, Kohonen self-organizing network, Grossberg adaptive resonance network, counterpropagation network, and Hopfield network. The Kohonen, Grossberg, and counterpropagation networks use unsupervised learning.[2,5,10,15,16] The characteristics and applications of these networks are summarized in Table 7.4. Another artificial neural network architecture, derived from a statistical hierarchical mixture density model, emulates the expectation-

maximization algorithm, which finds the maximum likelihood estimates of parameters used to define mixture density models. These models find application in classifying objects contained in images.[17]

Minimizing object classification error can also be accomplished by combining artificial neural network classifiers or by passing data through a series of individual neural networks. In the first approach, several neural networks are selected, each of which has the best classification performance for a particular class. Then the networks are combined with optimal linear weights.[18] Several criteria such as minimum squared error (MSE) and minimum classification error (MCE) are available to generate and evaluate the effectiveness of these weights. The MSE approach is optimal when the distribution of each class is normal, an assumption that may not always hold. Therefore, the MSE criterion does not generally lead to the optimal solution in a Bayes sense. However, the MCE criterion has the property of being able to construct a classifier with minimum error probability for classes characterized by different basis functions.

An example of the second approach is provided by analysis of data from a multichannel visible and IR scanning radiometer (MVISR). This sensor receives a combination of 10 channels of visible, short wavelength infrared, and thermal infrared energy.[19] Different channels of data are incrementally passed through three stages of artificial neural network classification to separate the signals into classes that produce images of cloud cover, cold ice clouds, sea ice, water, and cloud shadows. Each fully connected feedforward network stage computes an image-specific normalized dynamic threshold for a specific wavelength band based on the mean and maximum values of the input data. Image classification occurs by comparing each threshold against the normalized image data entered for that stage.

Artificial neural network pattern classifiers based on Dempster-Shafer evidential theory have also been developed.[20] Reference patterns are utilized to train the network to determine the class membership of each input pattern in each reference pattern. Membership is expressed in terms of a basic probability assignment (i.e., belief mass). The network combines the basic probability assignments, i.e., evidence of class membership, of the input pattern vector with respect to all reference prototypes using Dempster's rules. Thus, the output of the network assigns belief masses to all classes represented in the reference patterns and to the frame of discernment. The belief mass assigned to the frame of discernment represents the partial lack of information for decision making. The belief mass allocations may be used to implement various decision rules, including those for ambiguous pattern rejection.

A radial basis function network consisting of one input layer, two hidden layers, and one output layer can be used to implement the above technique. Hidden layer 1 computes the distances between the input vector and each reference class

according to some metric. Hidden layer 2 converts the distance metric into a bpa for each class. The output layer combines the basic probability assignments of the input vector to each class according to Dempster's rules. The weight vector is optimized by minimizing the MSE between the classifier outputs and the reference values.

7.11 Summary

Artificial neural networks are commonly applied to solve problems that involve complex interactions between input variables. These applications include target classification, speech synthesis, speech recognition, pattern mapping and recognition, data compression, data association, optical character recognition, and system optimization. The adaptive linear combiner is a basic building block of linear and nonlinear artificial neural networks. Generally, nonlinear classifiers can correctly classify a larger number of input patterns than linear classifiers. The statistical capacity or number of random patterns that a linear classifier can learn to classify is approximately equal to twice the number of weights in the processing element. The statistical capacity of nonlinear Madaline networks is also equal to twice the number of weights in the processing elements. However, the Madaline contains more than one processing element and, hence, has a greater capacity than the linear classifier. The capacities of more complex nonlinear classifiers, such as multilayer feedforward networks, can be bounded and approximated by the expressions discussed in this chapter.

Learning or training rules for single element and multilayer linear and nonlinear classifier networks are utilized to adapt the weights during training. In supervised training, the network weights are adjusted to minimize the error between the network output and the desired response to a known input. Linear classifier training rules include the μ-LMS and α-LMS algorithms. Nonlinear classifier training rules include the perceptron, backpropagation, Madaline, and dead zone algorithms. The backpropagation algorithm permits optimization of not only the weights in output layer elements of feedforward networks, but also those in the hidden layer elements. Generalization, through which artificial neural networks attempt to properly respond to input patterns not seen during training, is performed by firing rules, one of which is based on the Hamming distance.

The key operating principles and applications of the multilayer perceptron, Kohonen self-organizing network, Grossberg adaptive resonance network, counter-propagation network, and Hopfield network have been presented. The Kohonen, Grossberg, and counterpropagation networks are examples of systems that use unsupervised learning based on processing unlabeled samples. These systems adaptively cluster patterns into decision classes. Other artificial neural networks implement algorithms such as expectation maximization and Dempster-Shafer to optimize image classification. Still others combine individual networks

optimized for particular classes into one integrated system. These individual networks are combined using optimal linear weights.

Comparisons of the information needed to apply classical inference, Bayesian inference, Dempster-Shafer evidential theory, artificial neural networks, voting logic, and fuzzy logic fusion algorithms to a target identification application are found in Chapter 11.

Table 7.4 Properties of other artificial neural networks.

Type	Key Operating Principles	Applications
• Multilayer perceptron[10]	• A multilayer feedforward network. • Uses signum or sigmoid threshold nonlinearities. • Trained with supervised learning. • Errors are minimized using the backpropagation algorithm to update the weights applied to the input data by the hidden and output network layers. • No more than 3 layers are required because a three-layer perceptron can generate arbitrary complex decision regions.[10] • Number of weights equals the number of hidden-layer neurons.	• Accommodates complex decision regions in the feature space • Target classification • Speech synthesis • Nonlinear regression
• Radial-basis function network[5]	• Provides regularization, i.e., a stabilized solution using a nonnegative function to embed prior information (e.g., training examples that provide smoothness constraints on the input-output mapping), which converts an ill-posed problem into a well-posed problem. • The radial-basis function neural network is a regularization network with a multilayer feedforward network structure. • It minimizes a cost function that is proportional to the difference between the desired and actual network responses. • In one form of radial-basis function networks, the actual response is written as a linear superposition of the products of weights and multivariate Gaussian basis functions with centers located at the input data points and widths equal to the standard deviation of the data. • The Gaussian radial-basis function for each hidden element computes the Euclidean norm between the input vector and the center of that element. • Approximate solutions for the cost function utilize a number of basis functions less than the number of input data points to reduce computational complexity. • Trained with supervised learning.	• Target classification • Image processing • Speech recognition • Time series analysis • Adaptive equalization to reduce effects of imperfections in communications channels • Radar point source location • Medical diagnosis

Table 7.4 Properties of other artificial neural networks (continued).

Type	Key Operating Principles	Applications
• Kohonen network[23–25]	• Feedforward network that works with an unsupervised learning paradigm (processes unlabeled data, i.e., data where the desired classification is unknown). • Uses a mathematical transformation to convert input data vectors into output graphs, maps, or clustering diagrams. • Individual neural network clusters self-organize to reflect input pattern similarity. • Overall structure of the network can be viewed as an array of matched filters that competitively adjust input element weights based on current weights and goodness of match of the output to the training set input. • Output nodes are extensively inter-connected with many local connections. • Trained with winner-take-all algorithm. The winning node is rewarded with a weight adjustment, while the weights of the other nodes are unaffected. Winning node is the one whose output cluster most closely matches the input. • Network can also be trained with multiple winner unsupervised learning where the K neurons best matching the input vector are allowed to have their weights adjusted. The outputs of the winning neurons can be adjusted to sum to unity.	• Speech recognition
• Grossberg adaptive resonance network[26–28]	• Unsupervised learning paradigm that employs feedforward and feedback computations. • Teaches itself new categories and continues storing information without rejecting pieces of information that are temporarily useless, as they may be needed later. Pattern or feature information is stored in clusters. • Uses two layers – an input layer and an output layer. The output layer itself has two sublayers: a comparison layer for short-term memory and a recognition layer for long-term memory. • One adaptive resonance theory network learning algorithm (ART1) performs an offline search through encoded clusters, exemplars, and by trying to find a sufficiently close match of the input pattern to a stored cluster. If no match is found, a new class is created.	• Pattern recognition • Target classification

Table 7.4 Properties of other artificial neural networks (continued).

Type	Key Operating Principles	Applications
• Counter-propagation network[29–31]	• The counterpropagation network consists of two layers that map input data vectors into bipolar binary responses (-1, +1). It allows propagation from the input to a classified output, as well as propagation in the reverse direction. • First layer is a Kohonen layer trained in unsupervised winner-take-all mode. Input vectors belonging to the same cluster activate the same neuron in the Kohonen layer. The outputs of the Kohonen layer neurons are binary unipolar values 0 and 1. The first layer organizes data, allowing, for example, faster training to perform associative mapping than is typical of other two-layer networks. • Second layer is a Grossberg layer that orders the mapping of the input vectors into the bipolar binary outputs. The result is a network that behaves as a lookup memory table.	• Target classification • Pattern mapping and association • Data compression
• Hopfield network[32–34] (a type of associative memory network)	• Associative memories belong to a class of neural networks that learn according to a specific recording algorithm. They usually require *a priori* information and their connectivity (weight) matrices are frequently formed in advance. The network is trained with supervised learning. • Writing into memory produces changes in neural interconnections. Transformation of the input signals by the network allows information to be stored in memory for later output. • No usable addressing scheme exists in associative memory since all memory information is spatially distributed and superimposed throughout the network. • Every neuron is connected to all other neurons. • Network convergence is relatively insensitive to the fraction of elements (15 to 100%) updated at each step. • Each node receives inputs that are processed through a hard limiter. The outputs of the nodes (±1) are multiplied by the weight assigned to the nodes connected by the weight.	• Problems with binary inputs • Data association • Optimization problems • Optical character recognition

Table 7.4 Properties of other artificial neural networks (continued).

Type	Key Operating Principles	Applications
• Hopfield network[32–34] (continued)	• The minimum number of nodes is seven times the number of memories to be stored. • The asymptotic capacity C_a of auto-associative networks is bounded by $n/(4 \ln n) < C_a < n/(2 \ln n)$, where n is the number of neurons.	

References

1. B. Widrow and M.A. Lehr, "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation," *Proc. IEEE*, 78(9), 1415-1442 (Sep. 1990).
2. R. Schalkoff, *Pattern Recognition: Statistical, Structural, and Neural Approaches*, John Wiley and Sons, New York, NY (1992).
3. B. Widrow and M.E. Hoff, "Adaptive switching circuits," 1960 IRE Wescon Conv. Rec., Part 4, 96-104 (Aug. 1960). [Reprinted in J.A. Anderson and E. Rosenfeld (Eds.), *Neuro-Computing: Foundations of Research*, MIT Press, Cambridge, MA (1988)].
4. N.J. Nilsson, *Learning Machines*, McGraw-Hill, New York, NY (1965).
5. S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd Ed., Prentice Hall PTR, Upper Saddle River, NJ (1998).
6. E.B. Baum, "On the capabilities of multilayer perceptrons," *J. Complex.*, 4, 193-215 (Sep. 1988).
7. R. Gaborski, "An intelligent character recognition system based on neural networks," *Research Magazine*, Eastman Kodak Company, Rochester, NY (Spring 1990).
8. B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, Englewood Cliffs, NJ (1992).
9. F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington, DC (1962).
10. R.P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag*., 4, 4-22 (Apr. 1987).
11. G. Cybenko, *Approximations by Superpositions of a Sigmoidal Function*, Research Note, Computer Science Department, University of Illinois, Urbana (Oct. 1988).
12. G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, **2**, 303-314 (1989).
13. J.M. Zurada, *Introduction to Artificial Neural Systems*, West Publishing Comp., St. Paul, MN (1992).
14. C.H. Mays, *Adaptive threshold logic*, Ph.D. thesis, Tech. Rep. 1557-1, Stanford Electron. Labs., Stanford, CA (Apr. 1963).
15. J.A. Anderson and E. Rosenfeld, Eds., *Neurocomputing: Foundations of Research*, M.I.T. Press, Cambridge, MA (1988).
16. B. Kosko, "Unsupervised learning in noise," *IEEE Trans. Neural Networks*, 1(1), 44-57 (Mar. 1990).
17. V.P. Kumar and E.S. Manolakos, "Unsupervised statistical neural networks for model-based object recognition," *IEEE Trans. Sig. Proc.*, 45(11), 2709-2718 (Nov. 1997).
18. N. Ueda, "Optimal linear combination of neural networks for improving classification performance," *IEEE Trans. Pattern Anal. and Mach. Intel.*, 22(2), 207-215 (Feb. 2000).

19. T.J. McIntire and J.J. Simpson, "Arctic sea ice, cloud, water, and lead classification using neural networks and 1.6-mm data," *IEEE Trans. Geosci. and Rem. Sensing*, 40(9), 1956-1972 (Sep. 2002).
20. T. Denoeux, "A neural network classifier based on Dempster-Shafer theory," *IEEE Trans. Sys., Man, and Cybern.–Part A: Systems and Humans*, SMC-30(2), 131-150 (Mar. 2000).
23. T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, 43, 59-69 (1982).
24. T. Kohonen, "Analysis of a simple self-organizing process," *Biol. Cybern.*, 44, 135-140 (1982).
25. T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, Berlin (1984).
26. G.A. Carpenter and S. Grossberg, "Neural dynamics of category learning and recognition: Attention, memory consolidation, and amnesia," in J. Davis, R. Newburgh, and E. Wegman, Eds., *Brain Structure, Learning, and Memory*, AAAS Symposium Series (1986).
27. G.A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Comput. Vis., Graph., Image Process.*, 37, 54-115 (1987).
28. G.A. Carpenter and S. Grossberg, "ART 2: Self-organization of stable category recognition codes for analog input patterns," *Appl. Opt.*, 26(3), 4919-4930 (Dec. 1987).
29. R. Hecht-Nielsen, "Counterpropagation networks," *Appl. Opt.*, 26(3), 4979-4984 (Dec. 1987).
30. R. Hecht-Nielsen, "Applications of counterpropagation networks," *Neural Net.*, 1, 131-139 (1988).
31. R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Reading, MA, 1990.
32. J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci.*, 79 (Biophysics), 2554-2558 (Apr. 1982).
33. J.J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Nat. Acad. Sci.*, 81 (Biophysics), 3088-3092 (May 1984).
34. J.J. Hopfield and D.W. Tank, "Computing with neural circuits: a model," *Science*, 233, 625-633 (Aug. 1986).