Paper 1.1

Foundations of Neural Networks

PATRICK K. SIMPSON

GENERAL DYNAMICS ELECTRONICS DIVISION, SAN DIEGO, CA 92138

1. Introduction

Building intelligent systems that can model human behavior has captured the attention of the world for years. So, it is not surprising that a technology such as neural networks has generated great interest. This paper will provide an evolutionary introduction to neural networks by beginning with the key elements and terminology of neural networks, and developing the topologies, learning laws, and recall dynamics from this infrastructure. The perspective taken in this paper is largely that of an engineer, emphasizing the application potential of neural networks and drawing comparisons with other techniques that have similar motivations. As such, mathematics will be relied upon in many of the discussions to make points as precise as possible.

The paper begins with a review of what neural networks are and why they are so appealing. A typical neural network is immediately introduced to illustrate several of the key features. With this network as a reference, the evolutionary introduction to neural networks is then pursued. The fundamental elements of a neural network, such as input and output patterns, processing element, connections, and threshold operations, are described, followed by descriptions of neural network topologies, learning algorithms, and recall dynamics. A taxonomy of neural networks is presented that uses two of the key characteristics of learning and recall. Finally, a comparison of neural networks and similar nonneural information processing methods is presented.

2. What are neural networks, and what are they good for?

Neural networks are information processing systems. In general, neural networks can be thought of as "black box" devices that accept inputs and produce outputs. Some of the operations that neural networks perform include

- Classification—an input pattern is passed to the network, and the network produces a representative class as output.
- Pattern matching—an input pattern is passed to the network, and the network produces the corresponding output pattern.
- Pattern completion—an incomplete pattern is passed to the network, and the network produces an output pattern that has the missing portions of the input pattern filled in.
- Noise removal—a noise-corrupted input pattern is presented to the network, and the network removes some (or all) of the noise and produces a cleaner version of the input pattern as output.
- Optimization—an input pattern representing the initial values for a specific optimization problem is presented to the network, and the network produces a set of variables that represents a solution to the problem.
- Control—an input pattern represents the current state of a controller and the desired response for the controller, and the output is the proper command sequence that will create the desired response.

Neural networks consist of processing elements and weighted connections. Figure 1 illustrates a typical neural network. Each layer in a neural network consists of a collection of processing elements (PEs). Each PE in a neural network collects the values from all of its input connections, performs a predefined mathematical operation (typically a dot product followed by a PE function), and produces a single output value. The neural network in Fig. 1 has three layers: F_X, which consists of the PEs $\{x_1, x_2, x_3\}$; F_Y, which consists of the PEs $\{y_1, y_2\}$; and F_Z, which consists of the PEs $\{z_1, z_2, z_3\}$ (from bottom to top, respectively). The PEs are connected with weighted connections. In Fig. 1 there is a weighted connection from every F_X PE to every F_Y PE, and there is a weighted connection from every F_Y PE to every F_Z PE. Each weighted connection (often synonymously referred to as either a connection or a weight) acts as both a label and a value. As an example, in Fig. 1 the connection from the F_X PE x_1 to the F_Y PE y_2 is the connection weight w_{12} (the connection from x_1 to y_2). The connection weights store the information. The value of the connection weights is often determined by a neural network learning procedure (although sometimes they are predefined and hardwired into the network). It is through the adjustment of the connection weights that the neural network is able to learn. By performing the update operations for each of the PEs, the neural network is able to recall information.

There are several important features illustrated by the neural network shown in Fig. 1 that apply to all neural networks:

- Each PE acts independently of all others—each PE's output relies only on its constantly available inputs from the abutting connections.
- Each PE relies only on local information—the informa-

The author is now with ORINCON Corporation, 9363 Towne Centre Drive, San Diego, CA 92128.

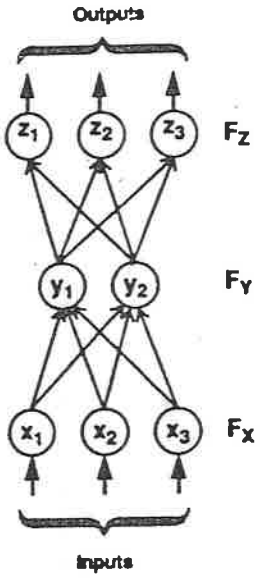


Fig. 1.

tion that is provided by the adjoining connections is all a PE needs to process; it does not need to know the state of any of the other PEs where it does not have an explicit connection.

- The large number of connections provides a large amount of redundancy and facilitates a distributed representation.

The first two features allow neural networks to operate efficiently in parallel. The last feature provides neural networks with inherent fault-tolerance and generalization qualities that are very difficult to obtain from typical computing systems. In addition to these features, through proper arrangement of the neural networks, introduction of a nonlinearity in the processing elements (i.e., adding a nonlinear PE function), and use of the appropriate learning rules, neural networks are able to learn arbitrary nonlinear mappings. This is a powerful attribute.

There are three primary situations where neural networks are advantageous:

1. Situations where only a few decisions are required from a massive amount of data (e.g., speech and image processing)
2. Situations where nonlinear mappings must be automatically acquired (e.g., loan evaluations and robotic control)
3. Situations where a near-optimal solution to a combinatorial optimization problem is required very quickly (e.g., airline scheduling and telecommunication message routing)

The foundations of neural networks consist of an understanding of the nomenclature and a firm comprehension of the rudimentary mathematical concepts used to describe and analyze neural network processing. In a broad sense, neural networks consist of three principle elements:

1. *Topology*—how a neural network is organized into layers and how those layers are connected.
2. *Learning*—how information is stored in the network.
3. *Recall*—how the stored information is retrieved from the network.

Each of these elements will be described in detail after discussing connections, processing elements, and PE functions.

3. Dissecting Neural Networks

Each neural network has at least two physical components: connections and processing elements. The combination of these two components creates a neural network. A convenient analogy is the directed graph, where the edges are analogous to the connections and the nodes are analogous to the processing elements. In addition to connections and processing elements, threshold functions and input/output patterns are also basic elements in the design, implementation, and use of neural networks. After a description of the terminology used to describe neural networks, each of these elements will be examined in turn.

3.1. Terminology

Neural network terminology remains varied, with a standard yet to be adopted (although there is an effort to create one (cf. Eberhart, 1990)). For clarity in further discussions, the terminology used within this paper will be described where appropriate. To illustrate some of the terminology introduced here, please refer to Fig. 2.

Input and output vectors (patterns) will be denoted by subscripted capital letters from the beginning of the alphabet. The input patterns will be denoted

$$A_k = (a_{k1}, a_{k2}, \cdots, a_{kn}); \qquad k = 1, 2, \cdots, m$$

and the output patterns

$$B_k = (b_{k1}, b_{k2}, \cdots, b_{kp}); \qquad k = 1, 2, \cdots, m.$$

The processing elements in a layer will be denoted by the same subscript variable. The collection of PEs in a layer form a vector, and these vectors will be denoted by capital letters from the end of the alphabet. In most cases, three layers of PEs will suffice. The input layer of PEs is denoted

$$F_X = (x_1, x_2, \cdots, x_n)$$

where each x_i receives input from the corresponding input pattern component a_{ki}. The next layer of PEs will be the F_Y PEs, then the F_Z PEs (if either layer is necessary). The dimensionality of these layers depends on its use. For the network in Fig. 2, for example, the second layer of the network is the output layer, so the number of F_Y PEs must match the dimensionality of output patterns. In this instance,

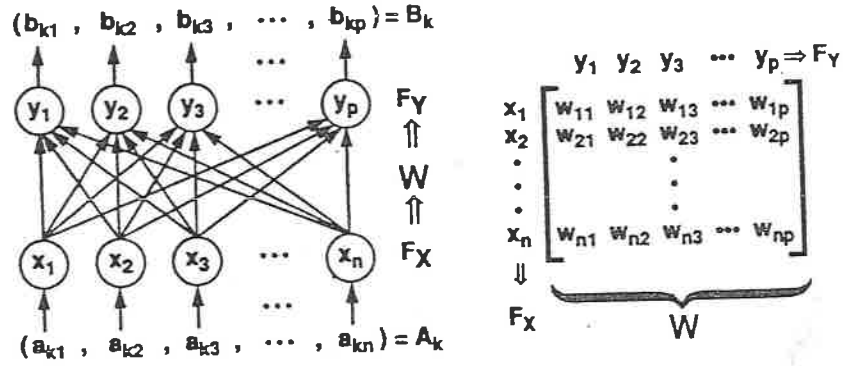


Fig. 2.

the output layer is denoted

$$F_Y = (y_1, y_2, \cdots, y_p)$$

where each y_j is correlated with the jth element of B_k.

Connection weights are stored in weight matrices. Weight matrices will be denoted by capital letters toward the middle of the alphabet, such as U, V, and W. For the example in Fig. 2, this two-layer neural network requires one weight matrix to fully connect the layer of n F_X PEs to the layer of p F_Y PEs. The matrix in Fig. 2 describes the full set of connection weights between F_X and F_Y, where the weight w_{ij} is the connection weight from the ith F_X PE, x_i, to the jth F_Y PE, y_j.

3.2. Input and Output Patterns

Neural networks cannot operate unless they have data. Some neural networks require only single patterns, and others require pattern pairs. Note that the dimensionality of the input pattern is not necessarily the same as the output pattern. When a network only works with single patterns, it is an autoassociative network. When a network works with pattern pairs, it is heteroassociative.

One of the key issues when applying neural networks is determining what the patterns should represent. For example, in speech recognition there are several different types of features that can be employed, including linear predictive coding coefficients, Fourier spectra, histograms of threshold crossings, cross-correlation values, and many others. The proper selection and representation of these features can greatly affect the performance of the network.

In some instances the representation of the features as a pattern vector is constrained by the type of processing the neural network can perform. Some networks can only process binary data, such as the Hopfield network (Hopfield, 1982; Amari, 1972), binary adaptive resonance theory (Carpenter and Grossberg, 1987a), and the brain-state-in-a-box (Anderson et al., 1977). Others can process real-valued data such as backpropagation (Werbos, 1974; Parker, 1982; Rumelhart, Hinton, and Williams, 1986), and learning vector quantization (Kohonen, 1984). Creating the best possible set of features and properly representing those features is the first step toward success in any neural network application (Anderson, 1990).

3.3. Connections

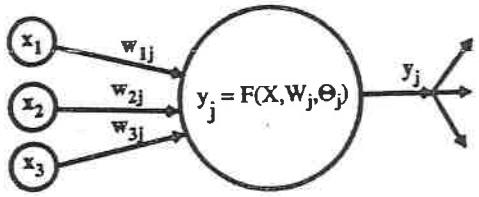
A neural network is equivalent to a directed graph (digraph). A digraph has edges (connections) between nodes (PEs) that allow information to flow in only one direction (the direction denoted by the arrow). Information flows through the digraph along the edges and is collected at the nodes. Within the digraph representation, connections serve a single purpose: they determine the direction of information flow. As an example, in Fig. 2 the information flows from the F_X layer through the connections W to the F_Y layer. Neural networks extend the digraph representation to include a weight with each edge (connection) that modulates the amount of output signal passed from one node (PE) down the connection to the adjacent node. For simplicity, the dual role of connections will be employed. A connection both defines the information flow through the network and modulates the amount of information passing between to PEs.

The connection weights are adjusted during a learning process that captures information. Connection weights that are positive-valued are *excitatory* connections. Those with negative values are *inhibitory* connections. A connection weight that has a zero value is the same as not having a connection present. By allowing only a subset of all the possible connections to have nonzero values, sparse connectivity between PEs can be simulated.

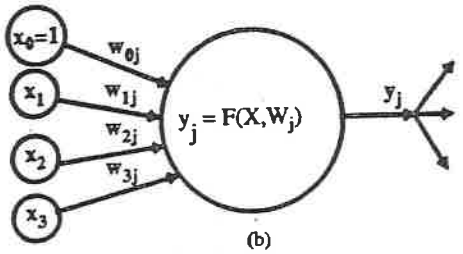
It is often desirable for a PE to have an internal bias value (threshold value). Part (a) of Fig. 3 shows the PE y_j with three connections from F_X $\{w_1, w_2, w_3\}$ and a bias value Θ_j. It is convenient to consider this bias value as an extra connection w_0 emanating from the F_X PE x_0, with the added constraint that x_0 is always equal to 1, as shown in part (b). This mathematically equivalent representation simplifies many discussions. Throughout the paper this method of representing the bias (threshold) values will be intrinsically employed.

3.4. Processing Elements

The PE is the portion of the neural network where all the computing is performed. Figure 3 illustrates the most com-



(a)



(b)

Fig. 3.

mon type of PE. A PE can have one input connection, as is the case when the PE is an input-layer PE and it receives only one value from the corresponding component of the input pattern, or it can have several weighted connections, as is the case of the F_Y PEs shown in Fig. 2, where there is a connection from every F_X PE to each F_Y PE. Each PE collects the information that has been sent down its abutting connections and produces a single output value. There are two important qualities that a PE must possess:

1. PEs require only local information. All the information necessary for a PE to produce an output value is present at the inputs and resides within the PE. No other information about other values in the network is required.
2. PEs produce only one output value. This single output value is propagated down the connections from the emitting PE to other receiving PEs, or it will serve as an output from the network.

These two qualities allow neural networks to operate in parallel. As was done with the connections, the value of the PE and its label are referred to synonymously. For example, the jth F_Y PE in Fig. 2 is y_j, and the value of that PE is also y_j.

There are several mechanisms for computing the output of a processing element. The output value of the PE shown in Fig. 3(b), y_j, is a function of the outputs of the preceding layer, $F_X = X = (x_1, x_2, \cdots, x_n)$ and the weights from F_X to y_j, $W_j = (w_{1j}, w_{2j}, \cdots, w_{nj})$. Mathematically, the output of this PE is a function of its inputs and its weights,

$$y_j = F(X, W_j). \tag{1}$$

Three examples of update functions follow.

3.4.1. Linear Combination. The most common computation performed by a PE is a linear combination (dot product) of the input values X with the abutting connection weights W_j, possibly followed by a nonlinear operation (cf. Simpson, 1990a; Hecht-Nielsen, 1990; Maren, Harston, and Pap, 1990). For the PE in Fig. 3(b), the output y_j is computed from the equation

$$y_j = f\left(\sum_{i=0}^{n} x_i w_{ij}\right) = f(X \cdot W_j) \tag{2}$$

where $W_j = (w_{1j}, w_{2j}, \cdots, w_{nj})$ and f is one of the nonlinear PE functions described in Section 3.4. The dot product update has a very appealing quality that is intrinsic to its computation. Using the relationship $A_k \cdot W_j = \cos(A_k, W_j)/\|A_k\| \|W_j\|$, one sees that the larger the dot product (assuming fixed length A_k and W_j), the more similar are the two vectors. Hence, the dot product can be viewed as a similarity measure.

3.4.2. Mean-Variance Connections. In some instances a PE will have two connections interconnecting PEs instead of just one, as shown in Fig. 4. One use of these dual connections is to allow one set of the abutting connections to represent the mean of a class and the other, the variance of the class (Lee and Kil, 1989; Robinson, Niranjan, and Fallside, 1988). In this case, the output value of the PE depends on the inputs and both sets of connections; that is, $y_j = F(X, V_j, W_j)$, where the mean connections are represented by $W_j = (w_{1j}, w_{2j}, \cdots, w_{nj})$ and the variance connections $V_j = (v_{1j}, v_{2j}, \cdots, v_{nj})$ for the PE y_j. With this scheme, the output of y_j is calculating the difference between the input X and the mean W_j, divided by the variance V_j, squaring the resulting quantity, and passing this value through a Gaussian nonlinear PE function to produce the final output value as follows:

$$y_j = g\left(\sum_{i=1}^{n}\left(\frac{w_{ij} - x_i}{v_{ij}}\right)^2\right) \tag{3}$$

where the Gaussian nonlinear PE function is

$$g(x) = \exp\left(\frac{-x^2}{2}\right) \tag{4}$$

Note that it is possible to remove one of the two connections in a mean-variance network, if the variance is known and

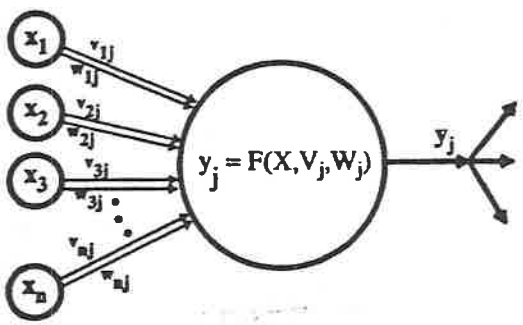


Fig. 4.

tationary, by dividing by the variance prior to neural network processing. Section 3.5.5 describes the Gaussian nonlinear PE function in greater detail.

3.4.3. Min–Max Connections. Another less common use of dual connections is to assign one of the abutting vectors, say V_j, to become the minimum bound for the class and the other vector, W_j, to become the maximum bound for the same class. Measuring the amount of the input pattern that falls within the bounds produces a min–max activation value (Simpson, 1991a). Figure 5 illustrates this notion by a graph representation for the min and the max points. The ordinate of the graph represents the value of each element of the min and max vectors, and the abscissa of the graph represents the dimensionality of the classification space. The input pattern X is compared with the bounds of the class. The amount of disagreement between the class bounds, V_j and W_j, and X is shown in the shaded regions. The measure of these shaded regions produces an activation value y_j.

Referring once again to Fig. 5, note that the max bound W_j is the maximum point allowed in class j and the min bound V_j is the minimum point allowed in class j. Measuring the degree to which X does not fall between V_j and W_j is done by measuring the relative amount of X that falls outside class j. One measure that was proposed (Simpson, 1990c) used Kosko's (1990a) fuzzy subsethood measures, which resulted in the equation

$$y_j = \big(1 - \text{supersethood}(X, W_j)\big)\big(1 - \text{subsethood}(X, V_j)\big) \tag{5}$$

When $y_j = 1$, X lies completely within the min–max bounds. When $y_j = 0$, X falls completely outside of the min–max bounds. When $0 < y_j < 1$, the value describes the degree to which X is contained by the min–max bounds. Although this is only one of many possible equations (cf. (59) and (60)), it does illustrate the use of min–max connections.

3.5. PE Functions

PE functions, also referred to as activation functions or squashing functions, map a PE's (possibly) infinite domain to a prespecified range. Although the number of PE functions possible is infinite, five are regularly employed by the majority of neural networks:

1. Linear PE function
2. Step PE function
3. Ramp PE function
4. Sigmoid PE function
5. Gaussian PE function

With the exception of the linear PE function, all of these functions introduce a nonlinearity in the network dynamics by bounding the output values within a fixed range. Each PE function is briefly described and shown in parts (a)–(e) of Fig. 6.

3.5.1. Linear PE Function. The linear PE function (see Fig. 6(a)) produces a linearly modulated output from the input x as described by the equation

$$f(x) = \alpha x \tag{6}$$

where x ranges over the real numbers and α is a positive scalar. If $\alpha = 1$, it is equivalent to removing the PE function completely.

3.5.2 Step PE Function. The step PE function (see Fig. 6(b)) produces only two values, β and δ. If the input to the PE function x equals or exceeds a predefined value θ, then the step PE function produces the value β; otherwise it produces the value $-\delta$, where β and δ are positive scalars. Mathematically this function is described as

$$f(x) = \begin{cases} \beta & \text{if } x \geq \theta \\ -\delta & \text{if } x < \theta \end{cases} \tag{7}$$

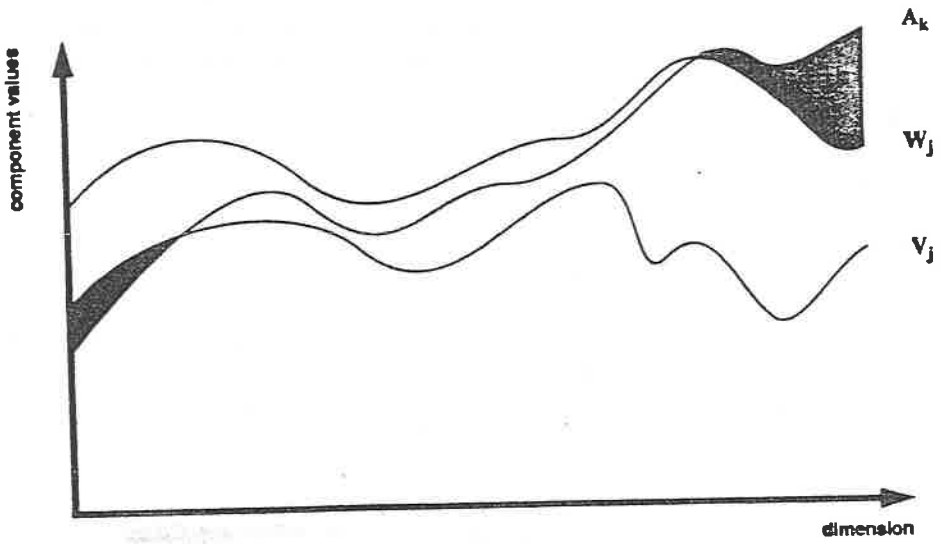


Fig. 5.

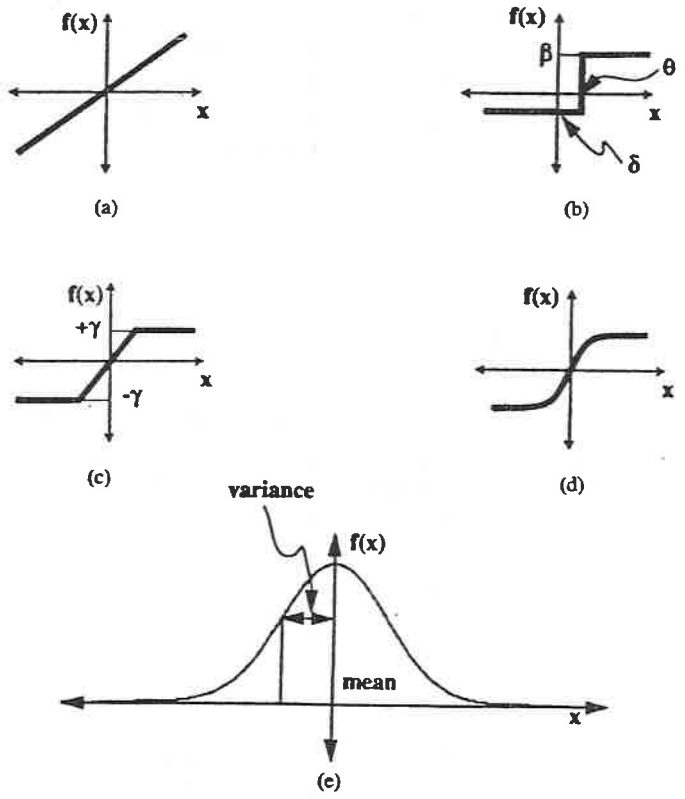


Fig. 6.

Typically, the step PE function produces a binary value in response to the sign of the input, emitting $+1$ if x is positive and 0 if it is not. For the assignments $\beta = 1$, $\delta = 0$, and $\theta = 0$, the step PE function becomes the binary step function

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

which is common to neural networks such as the Hopfield neural network (Amari, 1972; Hopfield, 1982) and the bidirectional associative memory (Kosko, 1988). One small variation of (8) is the bipolar PE function

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{9}$$

which replaces the 0 output value with a -1. In punish–reward systems such as the associative reward–penalty (Barto, 1985), the negative value is used to ensure changes, whereas a 0 will not.

3.5.3. Ramp PE Function. The ramp PE function (see Fig. 6(c)) is a combination of the linear and step PE functions. The ramp PE function places upper and lower bounds on the values that the PE function produces and allows a linear response between the bounds. These saturation points are symmetric around the origin and are discontinuous at the points of saturation. The ramp PE function is defined as

$$f(x) = \begin{cases} \gamma & \text{if } x \geq \gamma \\ x & \text{if } |x| < \gamma \\ -\gamma & \text{if } x \leq -\gamma \end{cases} \tag{10}$$

where γ is the saturation value for the function, and the points $x = \gamma$ and $x = -\gamma$ are where the discontinuities in f exist.

3.5.4. Sigmoid PE Function. The sigmoid PE function (see Fig. 6(d)) is a continuous version of the ramp PE function. The sigmoid (S-shaped) function is a bounded, monotonic, nondecreasing function that provides a graded, nonlinear response within a prespecified range.

The most common sigmoid function is the logistic function

$$f(x) = \frac{1}{1 + e^{-\alpha x}} \tag{11}$$

where $\alpha > 0$ (usually $\alpha = 1$), which provides an output value from 0 to 1. This function is familiar in statistics (as the Gaussian distribution function), chemistry (describing catalytic reactions), and sociology (describing human population growth). Note that a relationship between (11) and (8) exists. When $\alpha = \infty$ in (11), the slope of the sigmoid function between 0 and 1 becomes infinitely steep and, in effect, becomes the step function described by (8).

Two alternatives to the logistic sigmoid function are the hyperbolic tangent

$$f(x) = \tanh(x) \tag{12}$$

which ranges from -1 to 1, and the augmented ratio of squares

$$f(x) = \begin{cases} \dfrac{x^2}{1 + x^2} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

which ranges from 0 to 1.

3.5.5. Gaussian PE Function. The Gaussian PE function (see Fig. 6(e)) is a radial function (symmetric about the origin) that requires a variance value $v > 0$ to shape the Gaussian function. In some networks the Gaussian function is used in conjunction with a dual set of connections as described by (3), and in other instances (Specht, 1990) the variance is predefined. In the latter instance, the PE function is

$$f(x) = \exp\left(-x^2/v\right) \tag{14}$$

where x is the mean and v is the predefined variance.

4. Neural Network Topologies

The building blocks for neural networks are in place. Neural network topologies now evolve from the patterns, PEs, connections, and PE functions that have been described. Neural networks consist of layer(s) of PEs interconnected by weighted connections. The arrangement of the PEs, connections, and patterns into a neural network is referred to as a topology. After introducing some terminology, we describe six common neural network topologies.

4.1. Terminology

4.1.1. Layers. Neural networks are organized into layers of PEs. Within a layer, PEs are similar in two respects: 1) The connections that feed the layer of PEs are from the same source: for example, the PEs in the F_X layer in Fig. 2 all receive their inputs from the input pattern and the PEs in the layer F_Y all receive their inputs from the F_X PEs. 2) The PEs in each layer utilize the same type of update dynamics; for example, all the PEs will use the same type of connections and the same type of PE function.

4.1.2. Intralayer versus Interlayer Connections. There are two types of connections that a neural network employs: intralayer connections and interlayer connections. Intralayer connections (*intra* is Latin for "within") are connections between PEs in the same layer. Interlayer connections (*inter* is Latin for "among") are connections between PEs in different layers. It is possible to have neural networks that consist of one, or both, types of connections.

4.1.3 Feedforward versus Feedback Networks. When a neural network has connections that feed information in only one direction (e.g., input to output) without any feedback pathways in the network, it is a feedforward neural network. If the network has any feedback paths, where feedback is defined as any path through the network that would allow the same PE to be visited twice, then it is a feedback network.

4.2. Instars, Outstars, and the ADALINE

The two simplest neural networks are the instar and the outstar (Grossberg, 1982). The instar (see Fig. 7(a)) is the minimal pattern-encoding network. A simple example of an encoding procedure for the instar would take the pattern $A_k = (a_{k1}, a_{k2}, \cdots, a_{kn})$, normalize it, and use the values as the weights $W_j = (w_{1j}, w_{2j}, \cdots, w_{nj})$, as shown by the equation

$$v_{ij} = \frac{a_{ki}}{\sum_{i=1}^{n} a_{ki}} \tag{15}$$

for all $i = 1, 2, \cdots, n$.

The dual of the instar is the outstar (see Fig. 7(b)). The outstar is the minimal pattern recall neural network. An output pattern is generated from the outstar by using the equation

$$z_i = y_j w_{ji} \tag{16}$$

for all $i = 1, 2, \cdots, p$, where the weights are determined from (15) or one of the learning algorithms described in Section 5.

The ADALINE (ADAptive LInear NEuron, Widrow and Hoff, 1960) has the same topology as the instar (See Fig. 7(a)), but the weights V_j are adjusted by using the least-mean-square (LMS) algorithm (see Section 5.7.1). In the framework of adaptive signal processing, a similar topology with the same functionality is referred to as a finite impulse response (FIR) filter (Widrow and Stearns, 1985). Applications of the FIR filter to noise cancellation, echo cancellation, adaptive antennas, and control are numerous (Widrow and Winter, 1988).

4.3. Single-Layer Networks: Autoassociation, Optimization, and Contrast Enhancement

Beyond the instar/outstar neural networks, the minimal neural networks are the single-layer intraconnected neural networks. Figure 8 shows the topology of a one-layer neural network that consists of n F_X PEs. The connections are from each F_X PE to every other F_X PE, yielding a connection matrix with n^2 entries. The single-layer neural network accepts an n-dimensional input pattern in one of three ways:

1. PE initialization only—the input pattern is used to initialize the F_X PEs, and the input pattern does not influence the processing thereafter.
2. PE initialization and constant bias—the input pattern is used to initialize the F_X PEs, and the input remains as a constant valued-input bias throughout processing.
3. Constant bias only—the PEs are initialized to all zeroes, and the input pattern acts as a constant valued bias throughout processing.

One-layer neural networks are used to perform four types of pattern processing: pattern completion, noise removal, optimization, and contrast enhancement. The first two opera-

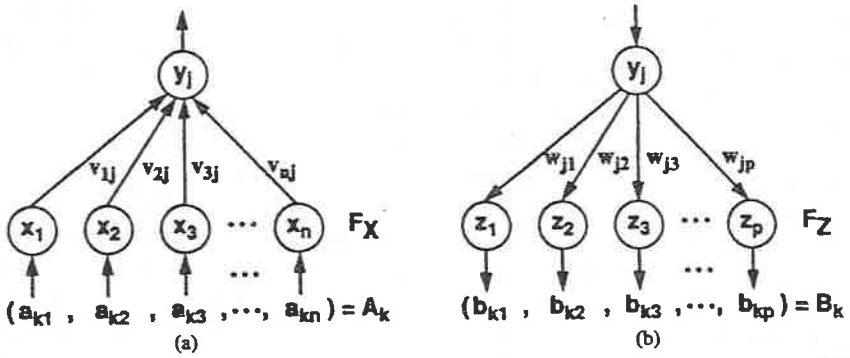


Fig. 7.

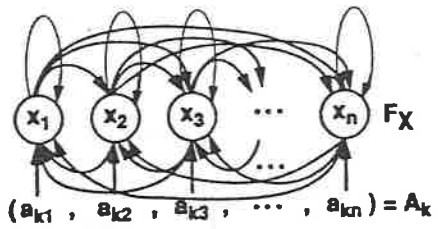


Fig. 8.

tions are performed by autoassociatively encoding patterns and (typically) using the input pattern for PE initialization only. The optimization networks are dynamical systems that stabilize to a state that represents a solution to an optimization problem and (typically) utilizes the inputs for both PE initialization and as constant biases. Contrast enhancement networks use the input patterns for PE initialization only and can operate in such a way that eventually only one PE remains active. Each of these one-layer neural networks is described in greater detail in the following paragraphs.

4.3.1. Pattern Completion. Pattern completion in a single-layer neural network is performed by presenting a partial pattern initially, and relying upon the neural network to complete the remaining portions. For example, assume a single-layer neural network has stored images of human faces. If half of a face is presented to the neural network as the initial state of the network, the neural network would complete the missing half of the face and output a complete face.

4.3.2. Noise Removal. Noise cancellation is similar to pattern completion in that a complete, noise-free response is desired from a pattern corrupted by noise. Fundamentally there is no difference between noise removal and pattern completion. The difference tends to be entirely operational. For the previous image-storage example, if a blurry or splotchy image is presented to the neural network, the output would be a crisp, clear image. Single-layer neural networks designed for pattern completion and noise cancellation include the discrete Hopfield network (Hopfield, 1982), the brain-state-in-a-box (Anderson et al., 1977), and the optimal linear associative memory (Kohonen, 1984).

4.3.3. Neural Optimization. One of the most prevalent uses of neural networks is neural optimization (Hopfield and Tank, 1985; Tank and Hopfield, 1986). Optimization is a technique for solving a problem by casting it into a mathematical equation that, when either maximized or minimized, solves the problem. Typical examples of problems approached by an optimization technique include scheduling, routing, and resource allocation. The neural optimization approach casts the optimization problem into the form of an energy function that describes the dynamics of a neural system. If the neural network dynamics are such that the network will always seek a stable state when the energy function is at a minimum, then the network will automatically find a solution. The inputs to the neural network are the initial state of the neural networks, and the final PE values represent the parameters of a solution.

4.3.4. Contrast Enhancement. Contrast enhancement in single-layer neural networks is achieved using on-center/off-surround connection values. The on-center connections are positive self-connections (i.e., $w_{ii} = \alpha(\alpha > 0)$ for all $i = 1, 2, \cdots, n$) that allow a pattern's activation value to grow by feeding back upon themselves. The off-surround connections are negative neighbor connections (i.e., $w_{ij} = -\beta$ $(\beta > 0)$ for all i not equal to j) that compete with the on-center connections. The competition between the positive on-center and the negative off-surround activation values are referred to as competitive dynamics. Contrast-enhancement neural networks take one of two forms: locally connected and globally connected. If the connections between the F_X PEs are only connected to a few of the neighboring PEs (see Fig. 9(a)), the result is a local competition that can result in several large activation values. If the off-surround connections are fully interconnected across the F_X layer (see Fig. 9(b)), the competition will yield a single winner.

4.4. Two-Layer Networks: Heteroassociation and Classification

Two-layer neural networks consist of a layer of n F_X PEs fully interconnected to a layer of p F_Y PEs, as shown in Fig. 10. The connections from the F_X to F_Y PEs form the $n \times p$ weight matrix W, where the entry w_{ij} represents the weight for the connection from the ith F_X PE, x_i, to the jth F_Y PE, y_j. There are three common types of two-layer neural networks: feedforward pattern matchers, feedback pattern matchers, and feedforward pattern classifiers.

4.4.1. Feedforward Pattern Matching. A two-layer feedforward pattern-matching neural network maps the input patterns A_k to the corresponding output patterns B_k, $k = 1, 2, \cdots, m$. The network in Fig. 10(a) illustrates the topology of this feedforward network. The two-layer feedforward neural network accepts the input pattern A_k and produces an output pattern $Y = (y_1, y_2, \cdots, y_p)$, which is the network's best estimate of the proper output, given A_k as the input. An optimal mapping between the inputs and the outputs is one that produces the correct response B_k when A_k is presented to the network, $k = 1, 2, \cdots, m$.

Most two-layer networks are concerned with finding the optimal linear mapping between the pattern pairs (A_k, B_k) (cf. Widrow and Winter, 1988; Kohonen, 1984), but there are other two-layer feedforward networks that also work with nonlinear mappings by extending the input patterns to include multiplicative combinations of the original inputs (Pao, 1989; Maren, Harsten, and Pap, 1990).

4.4.2. Feedback Pattern Matching. A two-layer feedback pattern-matching neural network, shown in Fig. 10(b), accepts inputs from either layer of the network, either the F_X or F_Y layer, and produces the output for the other layer (Kosko, 1988; Simpson, 1990a and b).

4.4.3. Feedforward Pattern Classification. A two-layer pattern classification neural network, shown in Fig. 10(c), maps an input pattern A_k to one of p classes. Representing

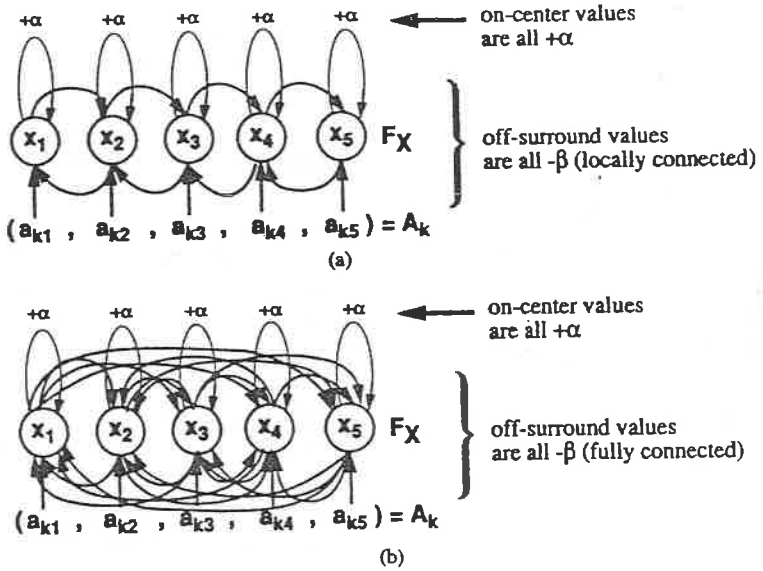


Fig. 9.

each class as a separate F_Y PE reduces the pattern classification task to selecting the F_Y PE that best responds to the input pattern. Most two-layer pattern classification systems utilize the competitive dynamics of global on-center/off-surround connections to perform the classification.

4.5. Multilayer Networks: Heteroassociation and Function Approximation

A multilayer neural network has more than two layers, possibly many more. A general description of a multilayer neural network is shown in Fig. 11, where there is an input layer of PEs, F_X, L hidden layers of F_Y PEs (Y_1, $Y_2, \cdots, Y_L$), and a final output layer, F_Z. The F_Y layers are called hidden layers because there are no direct connections between the input/output patterns to these PEs, rather they are always accessed through another set of PEs such as the input and output PEs. Although Fig. 11 shows connections only from one layer to the next, it is possible to have connections that skip over layers, that connect the input PEs to the output PEs, or that connect PEs together within the same layer. The added benefit of these PEs is not fully understood, but many applications are employing these types of topologies.

Multilayer neural networks are used for pattern classification, pattern matching, and function approximation. By adding a continuously differentiable PE function, such as a Gaussian or sigmoid function, it is possible for the network to learn practically any nonlinear mapping to any desired degree of accuracy (White, 1989).

The mechanism that allows such complex mappings to be acquired is not fully understood for each type of multilayer neural network, but in general the network partitions the input space into regions, and a mapping from the partitioned regions to the next space is performed by the next set of connections to the next layer of PEs, eventually producing an output response. This capability allows some very complex decision regions to be performed for classification and pattern-matching problems, as well as applications that require function approximation.

Several issues must be addressed when working with multilayer neural networks. How many layers are enough for a given problem? How many PEs are needed in each hidden layer? How much data is needed to produce a sufficient

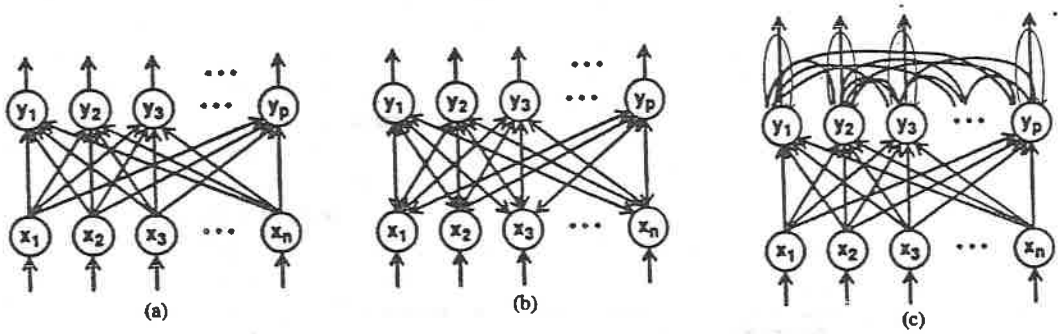


Fig. 10.

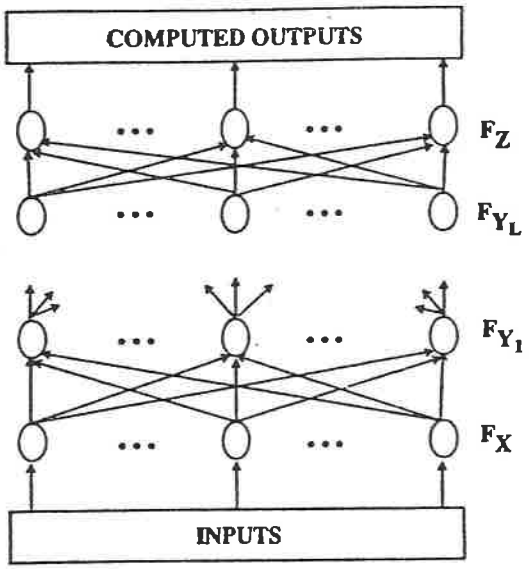


Fig. 11.

mapping from the input layer to the output layer? Some of these issues have been dealt with successfully. As an example, several researchers have proven that three layers are sufficient to perform any nonlinear mapping (with the exception of a few remote pathological cases) to any desired degree of accuracy with only one layer of hidden PEs (see White, 1989, for a review of this work). Although this is a very important result, it still does not indicate the proper number of hidden-layer PEs, or if the same solution can be obtained with more layers but fewer hidden PEs and connections overall.

There are several ways that multilayer neural networks can have their connection weights adjusted to learn mappings. The most popular technique is the backpropagation algorithm (Werbos, 1974; Parker, 1982; Rumelhart, Hinton, and Williams, 1986) and its many variants (see Simpson, 1990a for a list). Other multilayer networks include the neocognitron (Fukushima, 1988), the probabilistic neural network (Specht, 1990), the Boltzmann machine (Ackley, Hinton, and Sejnowski, 1985), and the Cauchy machine (Szu, 1986).

4.6. *Randomly Connected Networks*

Randomly connected neural networks are networks that have connection weights that are randomly assigned within a specific range. Some randomly connected networks have binary-valued connections. Realizing that a connection weight equal to zero is equivalent to no connection being present, binary-valued random connections create sparsely connected networks. Randomly connected networks are used in three ways:

1. Initial weights—The initial connection values for the network prior to training are preset to random values within a predefined range. This technique is used extensively in error-correction learning systems (see Sections 5.5–5.6).
2. Pattern preprocessing—A set of fixed random binary-valued connections are placed between the first two layers of a multilayer neural network as a pattern preprocessor. Such random connections can be used to increase the dimensionality of the space that is being used for mappings in an effort to improve the pattern-mapping capability. This approach was pioneered with the early Perceptron (Rosenblatt, 1962) and has been used recently in the sparse distributed memory (Kanerva, 1988).
3. Intelligence from randomness—Early studies in neural networks exerted a great deal of effort analyzing randomly connected binary-valued systems. The model of the brain as a randomly connected network of neurons prompted this research. These fixed-weight, nonadaptive systems have been studied extensively by Amari (1971) and Rozonoer (1969).

5. Neural Network Learning

Arguably the most appealing quality of neural networks is their ability to learn. Learning, in this context, is defined as a change in connection weight values that results in the capture of information that can later be recalled. Several procedures are available for changing the values of connection weights. After an introduction to some terminology, eight learning methods will be described. For continuity of discussion, the learning algorithms will be described in pointwise notation (as opposed to vector notation). In addition, the learning algorithms will be described with discrete-time equations (as opposed to continuous-time). Discrete-time equations are more accessible to digital computer simulations.

5.1. *Terminology*

5.1.1. Supervised versus Unsupervised Learning. All learning methods can be classified into two categories: supervised learning and unsupervised learning. Supervised learning is a process that incorporates an external teacher and/or global information. The supervised learning algorithms discussed in the following sections include error correction learning, reinforcement learning, stochastic learning, and hardwired systems. Examples of supervised learning include deciding when to turn off the learning, deciding how long and how often to present each association for training, and supplying performance (error) information. Supervised learning is further classified into two subcategories: structural learning and temporal learning. Structural learning is concerned with finding the best possible input/output relationship for each individual pattern pair. Examples of structural learning include pattern matching and pattern classification. The majority of the learning algorithms discussed on the following pages focus on structural learning. Temporal learning is concerned with capturing a sequence of patterns necessary to achieve some final outcome. In temporal learning, the current response of the network is dependent on previous inputs and

responses. In structural learning, there is no such dependence. Examples of temporal learning include prediction and control. The reinforcement learning algorithm to be discussed is an example of a temporal learning procedure.

Unsupervised learning, also referred to as self-organization, is a process that incorporates no external teacher and relies upon only local information during the entire learning process. Unsupervised learning organizes presented data and discovers its emergent collective properties. Examples of unsupervised learning in the following sections include Hebbian learning, principal component learning, differential Hebbian learning, min–max learning, and competitive learning.

5.1.2. Off-line versus On-line Learning. Most learning techniques utilize off-line learning. When the entire pattern set is used to condition the connections prior to the use of the network, it is called off-line learning. For example, the backpropagation training algorithm (see Section 5.7.2) is used to adjust connections in multilayer neural network, but it requires thousands of cycles through all the pattern pairs until the desired performance of the network has been achieved. Once the network is performing adequately, the weights are frozen and the resulting network is used in recall mode thereafter. Off-line learning systems have the intrinsic requirement that all the patterns have to be resident for training. Such a requirement does not make it possible to have new patterns automatically incorporated into the network as they occur; rather these new patterns must be added to the entire set of patterns and a retraining of the neural network must be done.

Not all neural networks perform off-line learning. Some networks can add new information "on the fly" nondestructively. If a new pattern needs to be incorporated into the network's connections, it can be done immediately without any loss of prior stored information. The advantage of off-line learning networks is they usually provide superior solutions to difficult problems such as nonlinear classification, but on-line learning allows the neural network to learn in situ. A challenge in the future of neural-network computing is the development of learning techniques that provide high-performance on-line learning without extreme costs.

5.2. Hebbian Correlations

The simplest form of adjusting connection-weight values in a neural network is based upon the correlation of PE activation values. The motivation for correlation-based adjustments has been attributed to Donald O. Hebb (1949), who hypothesized that the change in a synapses' efficacy (its ability to fire or, as we are simulating it in our neural networks, the connection weight) is prompted by a neuron's ability to produce an output signal. If a neuron A was active and A's activity caused a connected neuron B to fire, then the efficacy of the synaptic connection between A and B should be increased.

5.2.1. Unbounded PE Values and Weights. This form of learning, now commonly referred to as Hebbian learning, has been mathematically characterized as the correlation weight adjustment

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + a_{ki}b_{kj} \tag{17}$$

where $i = 1, 2, \cdots, n$, $j = 1, 2, \cdots, p$; x_i is the value of the ith PE in the F_X layer of a two-layer network; y_j is the value of the jth F_Y PE; and the connection weight between the two PEs is w_{ij}. In general, the values of the PEs can range over the real numbers, and the weights are unbounded. When the PE values and connection values are unbounded, these two-layer neural networks are amenable to linear systems theory. Neural networks like the linear associative memory (Anderson, 1970; Kohonen, 1972) employ this type of learning and analyze the capabilities of these networks with linear systems theory as a guide. The number of patterns that a network trained using (17) with unbounded weights and connections can produce is limited to the dimensionality of the input patterns (cf. Simpson, 1990a).

5.2.2. Bounded PE Values and Unbounded Weights. Recently, implementations that restrict the values of the PEs and/or the weights of (17) have been employed. These networks (called Hopfield networks because John Hopfield had excited people about their potential (Hopfield, 1982)), restrict the PE values to either binary $\{0, 1\}$ or bipolar $\{-1, +1\}$ values. Equation (17) is used for these types of correlations.

These discrete-valued networks typically involve some form of feedback recall, resulting in the need to show that every input will produce a stable response (output). Limiting the PE values during processing introduces nonlinearities in the system, eliminating some of the linear systems theory analyses that had previously been performed. Adding feedback into the recall process forms a discrete-valued, nonlinear, dynamical system. The single-layer versions of this learning rule are described as Hopfield nets (Hopfield, 1982), and two-layer versions as the bidirectional associative memory (Kosko, 1988). Some of the earlier analysis of these networks was performed by Amari (1972, 1977), who used the theory of statistical neurodynamics to show these networks were stable. Later, Hopfield (1982) found an alternative method to prove stability. Also, the number of patterns that neural networks of this form can store is limited (McEleice et al., 1987).

5.2.3. Bounded PE Values and Weights. Sometimes both the PE values and the weights are bounded. There are two forms of such systems. The first form is simply a running average of the amount of correlation between two PEs. The equation

$$w_{ij}^{\text{new}} = \frac{1}{k}\left(a_{ki}b_{kj} + (k-1)\,w_{ij}^{\text{old}}\right) \tag{18}$$

describes the average correlation during the presentation of the kth pattern pair (A_k, B_k), where $A_k = (a_{k1}, a_{k2}, \cdots, a_{kn})$; $B_k = (b_{k1}, b_{k2}, \cdots, b_{kp})$; and k is the cur-

rent pattern number, $k = 1, 2, \cdots, m$. The same information that was stored using (17) is stored using (18), the connection weights being simply bounded to the unit interval in the latter case.

The other example of the correlation neural network learning equation with bounded PE values and bounded weights is the sparse encoding equation, defined

$$w_{ij}^{\text{new}} = \begin{cases} 1 & \text{if } a_{ki}b_{kj} = 1 \\ 1 & \text{if } w_{ij}^{\text{old}} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (19)$$

This equation assigns a binary value to a connection if the PEs on each end of the connection have both had the value of 1 over the course of learning. The learning equation is equivalent to performing the logic operation

$$w_{ij}^{\text{new}} = (a_{ki} \cap b_{kj}) \cup w_{ij}^{\text{old}} \quad (20)$$

where $\cap$ and $\cup$ are the intersection and union operations, respectively.

Neural networks that have utilized this form of learning include the Learnmatrix (Steinbuch and Piske, 1963) and the Willshaw associative memory (Willshaw, 1980). This learning equation had a great deal of potential. Through the encoding of information in a binary vector (say, for example, only 32 components out of 1 million were set to 1, the others being set to 0), it is possible to store a tremendous amount of information in the network. The problem lies in creating the code necessary to perform such dense storage (cf. Hecht-Nielsen, 1990).

5.3. *Principal Component Learning*

Some neural networks have learning algorithms designed to produce, as a set of weights, the principal components of the input data patterns. The principal components of a set of data are found by first forming the covariance (or correlation) matrix of a set of patterns and then finding the minimal set of orthogonal vectors that span the space of the covariance matrix. Once the basis set has been found, it is possible to reconstruct any vector in the space with a linear combination of the basis vectors. The value of each scalar in the linear combination represents the "importance" of that basis vector (Lawley and Maxwell, 1963). It is possible to think of the basis vectors as feature vectors, and the combination of these feature vectors is used to construct patterns. Hence, the purpose of a principal component network is to decompose an input pattern into values that represent the relative importance of the features underlying the patterns.

The first work with principal component learning was done by Oja (1982), who reasoned that Hebbian learning with a feedback term that automatically constrained the weights would extract the principal components from the input data. The equation Oja uses is

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + b_{kj}(\alpha a_{ki} - \beta b_{kj} w_{ij}^{\text{old}}) \quad (21)$$

where a_{ki} is the ith component of the kth input pattern A_k, $i = 1, 2, \cdots, n$; b_{kj} is the jth component of the kth output pattern B_k, $j = 1, 2, \cdots, p$; $k = 1, 2, \cdots, m$; and α and β are nonzero constants.

A variant of the work by Oja has been developed by Sanger (1989) and is described by the equation

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \gamma_k \left(a_{ki}b_{kj} - b_{kj} \sum_{h=1}^{i} y_h w_{jh} \right) \quad (22)$$

where the variables are similar to those of (21) with the exception of the nonzero, time-decreasing learning parameter γ_k. Equations (21) and (22) are very similar; the key difference is that (22) includes more information in the feedback term and uses a decaying learning rate. There have been many analyses and applications of principal component networks. For a review of this work, see Oja (1989). It should be noted that both Oja's and Sanger's principal component networks only extract the first "one" principal component, and they are limited to networks with linear PEs.

5.4. *Differential Hebbian Learning*

Hebbian learning has been extended to capture the temporal changes that occur in pattern sequences. This learning law, called differential Hebbian learning, has been independently derived by Klopf (1986) in the discrete-time form, and by Kosko (1986b) in the continuous-time form. The general form, some variants, and some similar learning laws are outlined in the following sections. Several other combinations have been explored beyond those presented here. A more thorough examination of these Hebbian learning rules and others can be found in Barto (1984) and Tesauro (1986).

5.4.1. Basic Differential Hebbian Learning. Differential Hebbian learning correlates the changes in PE activation values with the equation

$$w_{ij}(t+1) = w_{ij}(t) + \Delta x_i(t) + \Delta y_j(t-1) \quad (23)$$

where $\Delta x_i(t) = x_i(t) - x_i(t-1)$ is the amount of change in the ith F_X PE at time t, and $\Delta y_j(t-1) = y_j(t-1) - y_j(t-2)$ is the amount of change in the jth F_Y PE at time $t-1$.

5.4.2. Drive-Reinforcement Learning. Klopf (1986) uses the more general case of (23) that captures changes in F_X PEs over the last k time steps and modulates each change by the corresponding weight value for the connection in a two-layer neural network. Klopf's equation is

$$w_{ij}(t+1) = w_{ij}(t) + \Delta y_j \sum_{h=1}^{k} \cdot \alpha(t-h) \mid w_{ij}(t-h) \mid \Delta x_i(t-h) \quad (24)$$

where $\alpha(t-h)$ is a decreasing function of time that regulates the amount of change, and $w_{ij}(t)$ is the connection value from the x_i to y_j at time t. Klopf refers to the presynaptic changes $\Delta x_i(t-h)$, $h = 1, 2, \cdots, k$, as drives

and to the postsynaptic change $\Delta y_j(t)$ as the reinforcement; hence the name drive-reinforcement learning.

5.4.3. Covariance Correlation. Sejnowski (1977) has proposed the covariance correlation of PE activation values in a two-layer neural network using the equation

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \mu\left[(a_{ki} - \bar{x}_i)(b_{kj} - \bar{y}_j)\right] \qquad (25)$$

where the bracketed terms represent the covariance, the difference between the expected (average) value of the PE activation values (x_i and y_j) and the input and output pattern values (a_{ki} and b_{kj}), respectively. The parameter $0 < \mu < 1$ is the learning rate. The overbar on the PE values represents the average value of the PE.

Sutton and Barto (1981) have proposed a similar type of covariance learning rule, suggesting the correlation of the expected value of x_i with the variance of y_j as expressed by the equation

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \mu\bar{x}_i(b_{kj} - \bar{y}_j) \qquad (26)$$

5.5 Competitive Learning

Competitive learning, introduced by Grossberg (1970) and Malsburg (1973) and extensively studied by Amari and Takeuchi (1978), Amari (1983), and Grossberg (1982), is a method of automatically creating classes for a set of input patterns. Competitive learning is a two-step procedure that couples the recall process with the learning process in a two-layer neural network (see Fig. 12). In Fig. 12 each F_x PE represents a component of the input pattern, and each F_Y PE represents a class (see also Section 4.3.4).

Step 1: Determine winning F_Y PE. An input pattern A_k is passed through the connections from the input layer F_X to the output layer F_Y in a feedforward fashion by using the dot-product update equation

$$y_j = \sum_{i=1}^{n} x_i w_{ij} \qquad (27)$$

where x_i is the ith PE in the input layer F_X, $i = 1, 2, \cdots, n$, y_j is the jth PE in the output layer F_Y, $j = 1, 2, \cdots, p$, and w_{ij} is the value of the connection weight between x_i and y_j. Each set of connections that abut an F_B PE, say y_j, is a reference vector $W_j = (w_{1j}, w_{2j}, \cdots, w_{nj})$ representing the class j. The reference vector W_j closest to the input A_k should provide the highest activation value. If the input patterns A_k, $k = 1, 2, \cdots, m$, and the reference vectors W_j, $j = 1, 2, \cdots, p$, are normalized to Euclidean unit length, then the following relationship holds:

$$0 \le \left(y_j = A_k \cdot W_j = \sum_{i=1}^{n} a_{ki} w_{ij}\right) \le 1 \qquad (28)$$

where the more similar A_k is to W_j the closer the dot product is to unity (see Section 3.4.1). The dot-product values y_j are used as the initial values for winner-take-all competitive interactions (see Section 4.3.4). The result of these interactions is identical to searching the F_Y PEs and finding the PE with the largest dot-product value. Using the equation

$$y_j = \begin{cases} 1 & \text{if } y_j > y_k \text{ for all } j \neq k \\ 0 & \text{otherwise} \end{cases} \qquad (29)$$

it is possible to find the F_y PE with the highest dot-product value, called the winning PE. The reference vector associated with the winning PE is the winning reference vector.

Step 2: Adjust winning F_Y PE's connection values. In competitive learning with winner-take-all dynamics like those previously described, there is only one set of connection weights adjusted—the connection weights of the winning reference vector. The equation that automatically adjusts the winning reference vector and no others is

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \alpha(t)\, y_j(a_{ki} - w_{ij}) \qquad (30)$$

where $\alpha(t)$ is a nonzero, decreasing function of time. The result of this operation is the motion of the reference vector toward the input vector. Over several presentations of the data vectors (on the order of $O(n^3)$ (Hertz, 1990)), the reference vectors will become the centroids of data clusters (Kohonen, 1986).

There have been several variations of this algorithm (cf. Simpson, 1990a), but one of the most important is the conscience mechanism (DeSieno, 1988). By adding a conscience to each F_Y PE , an F_Y PE is only allowed to become a winner if it has won equiprobably. The equiprobable winning constraint improves both the quality of solution and the learning time. Neural networks that employ competitive learning include self-organizing feature maps (Kohonen, 1984), adaptive resonance theory I (Carpenter and Grossberg, 1987a), and adaptive resonance theory II (Carpenter and Grossberg, 1987b).

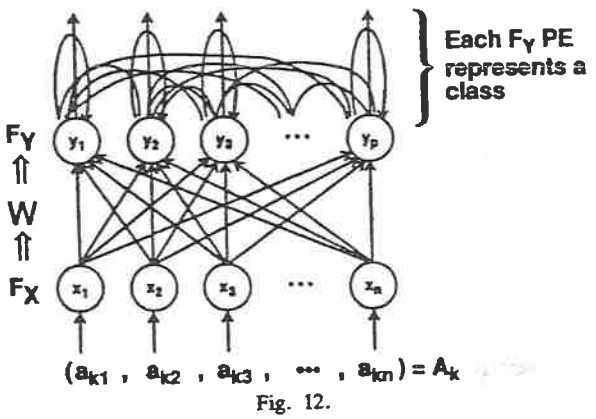


Fig. 12.

5.6. Min–Max Learning

Min–max classifier systems utilize a pair of vectors for each class (see Section 3.4.3). The class j is represented by the PE y_j and is defined by the abutting vectors V_j (the min vector) and W_j (the max vector). Learning in a min–max

neural system is done with the equation

$$v_{ij}^{new} = \min\left(a_{ki}, v_{ij}^{old}\right) \tag{31}$$

for the min vector and

$$w_{ij}^{new} = \max\left(a_{ki}, w_{ij}^{old}\right) \tag{32}$$

for the max vector. The min and max points are treated as bounds for a given membership/transfer function, providing a mechanism to easily adjust and analyze classes being formed in a neural network (Simpson, 1991a and 1992).

5.7. *Error Correction Learning*

Error correction learning adjusts the connection weights between PEs in proportion to the difference between the desired and computed values of each output layer PE. Two-layer error correction learning is able to capture linear mappings between input and output patterns. Multilayer error correction learning is able to capture nonlinear mappings between the inputs and outputs. In the following two sections, each of these learning techniques will be described.

5.7.1. Two-Layer Error Correction Learning. Consider the two-layer network in Fig. 13. Assume that the weights W are initialized to small random values (see Section 4.6). The input pattern A_k is passed through the connection weights W to produce a set of F_Y PE values $Y = (y_1, y_2, \cdots, y_p)$. The difference between the computed output values Y and the desired output pattern values B_k is the error. The error for each F_Y PE is computed from the equation

$$\delta_j = b_{kj} - y_j \tag{33}$$

The error is used to adjust the connection weights by using the equation

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\delta_j a_{ki} \tag{34}$$

where the positive-valued constant α is the learning rate.

The foundations for the learning rule described by (33) and (34) are solid. By realizing that the best solution can be attained when all the errors for a given pattern across all the output PEs, y_j, is minimized, we can construct the following

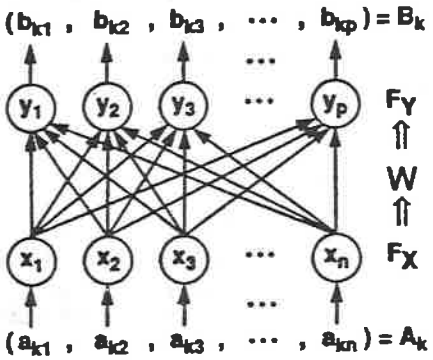


Fig. 13.

cost function:

$$E = \frac{1}{2}\sum_{j=1}^{p}\left(b_{kj} - y_j\right)^2 \tag{35}$$

When E is zero, the mapping from input to output is perfect for the given pattern. By moving in the opposite direction of the gradient of the cost function with respect to the weights, we can achieve the optimal solution (assuming each movement along the gradient α is sufficiently small). Restated mathematically, the two-layer error correction learning algorithm is computed as follows:

$$\begin{aligned}\frac{\partial E}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}}\left[\frac{1}{2}\sum_{j=1}^{p}\left(b_{kj} - \sum_{i=1}^{n} a_{ki}w_{ij}\right)^2\right] \\ &= \left(b_{kj} - \sum_{i=1}^{n} a_{ki}w_{ij}\right)a_{ki} \\ &= \left(b_{kj} - y_j\right)a_{ki}\end{aligned} \tag{36}$$

Although the cost function is only with respect to a single pattern, it has been shown (Widrow and Hoff, 1960) that the motion in the opposite direction of the gradient for each pattern, when taken in aggregate, acts as a noisy gradient motion that still achieves the proper end result.

The Perceptron (Rosenblatt, 1962) and the ADALINE (Widrow and Hoff, 1960), two of the most prominent early neural networks, employed error correction learning. In addition, the brain-state-in-a-box (Anderson et al., 1977) uses the two-layer error correction procedure previously described for one-layer autoassociative encoding.

5.7.2. Multilayer Error Correction Learning. A problem that once plagued error correction learning was its inability to extend learning beyond a two-layer network. With only a two-layer learning rule, only linear mappings could be acquired. There had been several attempts to extend the two-layer error correction learning algorithm to multiple layers, but the same problem kept arising: How much error is each hidden-layer PE responsible for in the output-layer PE error? Using the three-layer neural network in Fig. 14 to explain, the problem of multilayer learning (in this case three-layer learning) was to calculate the amount of error that each hidden-layer PE, y_j, should be credited with for an output-layer PE's error.

This problem, called the credit assignment problem (Barto, 1984; Minsky, 1961), was solved through the realization that a continuously differentiable PE function for the hidden-layer PEs would allow the chain rule of partial differentiation to be used to calculate weight changes for any weight in the network. For the three-layer network in Fig. 14, the output error across all the F_Z PEs is found by using the cost function

$$E = \frac{1}{2}\sum_{j=1}^{q}\left(b_{kj} - z_j\right)^2 \tag{37}$$

The output of an F_Z PE, z_j, is computed by using the

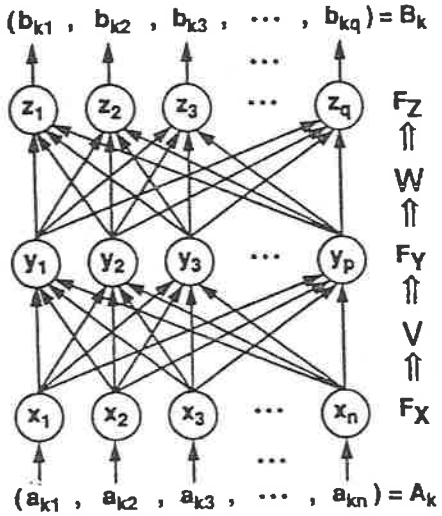


Fig. 14.

equation

$$z_j = \sum_{i=1}^{p} y_i w_{ij} \tag{38}$$

and each F_Y (hidden-layer) PE, y_i, is computed by using the equation

$$y_i = f\left(\sum_{h=1}^{n} a_{kh} v_{hi} \right) = f(r_i); \qquad r_i = \sum_{h=1}^{n} a_{kh} v_{hi} \tag{39}$$

The hidden-layer PE function is

$$f(\gamma) = \frac{1}{1 + e^{-\gamma}} \tag{40}$$

Using the same principle as described in the previous section, we perform the weight adjustments by moving along the cost function in the opposite direction of the gradient to a minimum (where the minimum is considered to be the input/output mapping producing the smallest amount of total error). The connection weights between the F_Y and F_Z PEs are adjusted by using the same form of the equation derived earlier for two-layer error correction learning, thereby yielding

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \left[\frac{1}{2} \sum_{j=1}^{q} (b_{kj} - z_j)^2 \right] \\ &= (b_{kj} - z_j) y_i \\ &= \delta_j y_i \end{aligned} \tag{41}$$

where the positive, constant-valued learning rate α has been added to adjust the amount of change made with each move down the gradient (see (43)).

Next, the adjustments to the connection weights between the F_X and F_Y PEs are found by using the chain rule of partial differentiation:

$$\frac{\partial E}{\partial v_{hi}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial r_i} \frac{\partial r_i}{\partial x_h} \frac{\partial x_h}{\partial v_{hi}} = -$$

$$= \sum_{l=1}^{q} (b_{kl} - y_l) y_l w_{hl} f'(r_i) a_{kh} \tag{42}$$

where β is a positive, constant-valued learning rate (see (44)). The multilayer version of this algorithm is commonly referred to as the backpropagation of errors learning rule, or simply backpropagation. Utilizing the chain rule, we can calculate weight changes for an arbitrary number of layers. The number of iterations that must be performed for each pattern in the data set is large, making this off-line learning algorithm very slow to train. From (41) and (42), the weight adjustment equations become

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} - \alpha \frac{\partial E}{\partial w_{ij}} \tag{43}$$

and

$$v_{hi}^{\text{new}} = v_{hi}^{\text{old}} - \beta \frac{\partial E}{\partial v_{hi}} \tag{44}$$

where α and β are positive-valued constants that regulate the amount of adjustments made with each gradient move.

Extending the backpropagation to utilize mean-variance connections (see Section 3.4.2) between the F_X and F_Y PEs is straightforward (Robinson, Niranjan, and Fallside, 1988). Figure 15 shows the topology of a three-layer mean-variance version of the multilayer error correction learning algorithm. The hidden layer F_Y PE values are computed with the equation

$$y_l = g(r_i); \qquad r_i = \sum_{h=1}^{n} \left(\frac{u_{hi} - a_{kh}}{v_{hi}} \right)^2 \tag{45}$$

where u_{hi} represents the mean connection strength between the hth F_X and ith F_Y PEs, v_{hi} is the variance connection strength between the hth F_X and ith F_Y PEs, and the PE function is the Gaussian function

$$g(x) = e^{-x/2} \tag{46}$$

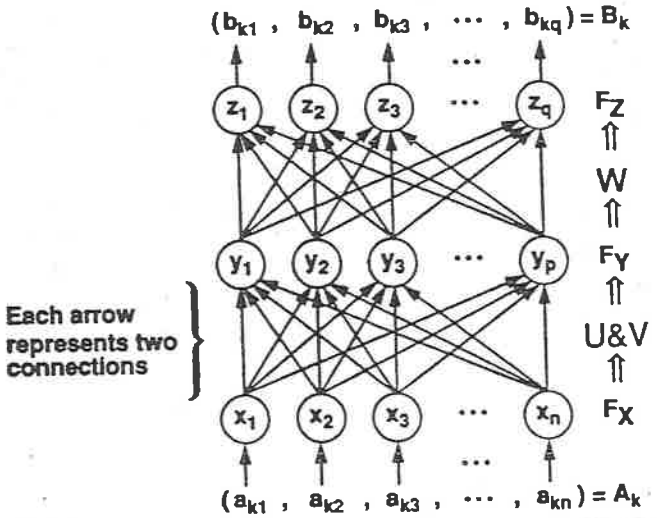


Fig. 15.

The output PE, F_Z, values are then formed from the linear combination of the hidden-layer Gaussians by using the equation

$$z_j = \sum_{i=1}^{p} y_i w_{ij} \tag{47}$$

where w_{ij} is the connection strength between the ith F_Y and jth F_Z PEs. Computing the gradients for each set of weights yields the following set of equations:

$$\frac{\partial E}{\partial u_{hi}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i}\frac{\partial y_i}{\partial r_i}\frac{\partial r_i}{\partial u_{hi}}$$

$$= \sum_{j=1}^{q} (b_{kj} - z_j) w_{ij} g'(r_i) \left(\frac{u_{hi} - a_{ki}}{v_{hi}^2} \right) \tag{48}$$

$$\frac{\partial E}{\partial v_{hi}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i}\frac{\partial y_i}{\partial r_i}\frac{\partial r_i}{\partial v_{hi}}$$

$$= \sum_{j=1}^{q} (b_{kj} - z_j) w_{ij} g'(r_i) \left(\frac{u_{hi} - a_{ki}}{v_{hi}^3} \right) \tag{49}$$

$$\frac{\partial E}{\partial w_{ij}} = (b_{kj} - z_j) y_i \tag{50}$$

From these equations, the update equations are found to be

$$u_{hi}^{\text{new}} = u_{hi}^{\text{old}} - \alpha \frac{\partial E}{\partial u_{hi}} \tag{51}$$

$$v_{hi}^{\text{new}} = v_{hi}^{\text{old}} - \beta \frac{\partial E}{\partial v_{hi}} \tag{52}$$

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} - \gamma \frac{\partial E}{\partial w_{ij}} \tag{53}$$

where α, β, and γ are nonzero constants.

The backpropagation algorithm was introduced by Werbos (1974) and rediscovered independently by Parker (1982) and Rumelhart, Hinton, and Williams (1986). The algorithm presented here has been brief. There are several variations on the algorithm (cf. Simpson, 1990a), including alternative multilayer topologies, methods of improving the learning time, methods for optimizing the number of hidden layers and the number of hidden-layer PEs in each hidden layer, and many more. Although many issues remain unresolved with the backpropagation of errors learning procedure, such as proper number of training parameters, existence of local minima during training, extremely long training time, and optimal number and configuration of hidden-layer PEs, the ability of this learning method to automatically capture nonlinear mappings remains a significant strength.

5.8. *Reinforcement Learning*

The initial idea for reinforcement learning was introduced by Widrow, Gupta, and Maitra (1973) and has been championed by Williams (1986). Reinforcement learning is similar to error correction learning in that weights are reinforced for properly performed actions and punished for poorly performed actions. The difference between these two supervised learning techniques is that error correction learning utilizes more specific error information by collecting error values from each output-layer PE, while reinforcement learning uses nonspecific error information to determine the performance of the network. Whereas error correction learning has a whole vector of values that it uses for error correction, only one value is used to describe the output layer's performance during reinforcement learning. This form of learning is ideal in situations where specific error information is not available, but overall performance information is, such as prediction and control.

A two-layer neural network such as that in Fig. 16 serves as a good framework for the reinforcement learning algorithm (although multilayer networks can also use reinforcement learning). The general reinforcement learning equation is

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \alpha(r - \theta_j) e_{ij} \tag{54}$$

where r is the scalar success/failure value provided by the environment, θ_j is the reinforcement PE value for the jth F_Y PE, e_{ij} is the canonical eligibility of the weight from the ith F_X PE to the jth F_Y PE, and $0 < \alpha < 1$ is a constant-valued learning rate. In error correction learning, gradient descent is performed in error space. Reinforcement learning performs gradient descent in probability space. The canonical eligibility of w_{ij} is dependent on a previously selected probability distribution that is used to determine if the computed output value equals the desired output value, and is defined as

$$e_{ij} = \frac{\partial}{\partial w_{ij}} \ln g_i \tag{55}$$

where g_i is the probability of the desired output equaling the computed output, defined as

$$g_i = \Pr(y_j = b_{kj} \mid W_j, A_k) \tag{56}$$

which is read as the probability that y_j equals b_{kj} given the input A_k and the corresponding weight vector W_j.

Neural networks that employ reinforcement learning include the adaptive heuristic critic (Barto, Sutton, and Anderson, 1983) and the associative reward–penalty neural network (Barto, 1985).

5.9. *Stochastic Learning*

Stochastic learning uses random processes, probability, and an energy relationship to adjust connection weights in a multilayered neural network. For the three-layer neural network in Fig. 14, the stochastic learning procedure is described as follows:

1. Randomly change the output value of a hidden-layer PE (the hidden-layer PEs utilize a binary step PE function).

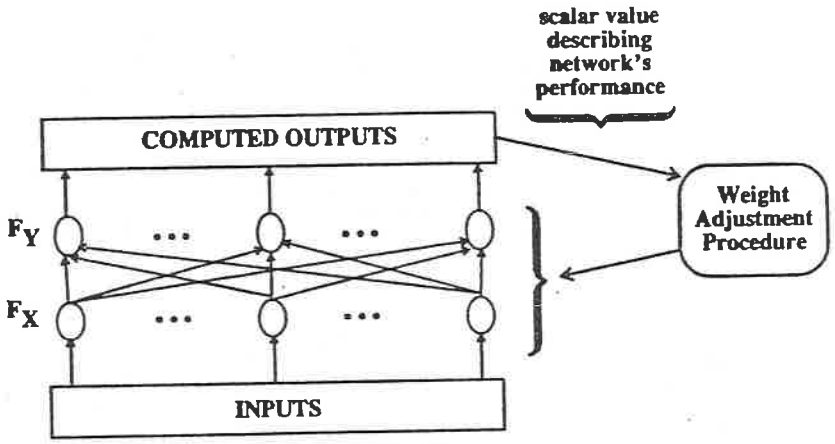


Fig. 16.

2. Evaluate the change by using the resulting difference in the neural network's energy as a guide. If the energy after the change is lower, keep the change. If the change in energy is not lower after the random change, accept the change according to a prechosen probability distribution.
3. After several random changes, the network will eventually become "stable." Collect the values of the hidden-layer PEs and the output-layer PEs.
4. Repeat steps 1–3 for each pattern pair in the data set; then use the collected values to statistically adjust the weights.
5. Repeat steps 1–4 until the network performance is adequate.

The probabilistic acceptance of higher energy states, despite a temporary increase in energy, allows the neural network to escape local energy minima in favor of a deeper energy minimum. This learning process, founded in simulated annealing (Kirkpatrick, Gelatt, and Vecchi, 1983), is governed by a "temperature" parameter that slowly decreases the number of probabilistically accepted higher energy states.

The Boltzmann machine (Ackley, Hinton, and Sejnowski, 1985) was the first neural network to employ stochastic learning. Szu (1986) has refined the procedure by employing the Cauchy distribution function in place of the Gaussian distribution function, thus resulting in a network that converges to a solution much quicker.

5.10. *Hardwired Systems*

Some neural networks have their connection weights predetermined for a specific problem. These weights are "hardwired" in that they do not change once they have been determined. The most popular hardwired systems are the neural optimization networks (Hopfield and Tank, 1985). Neural optimization works by designing a cost function that, when minimized, solves an unconstrained optimization problem. By translating the energy function into a set of weights and bias values, the neural network becomes a parallel optimizer. Given the initial values of the problem, the network will run to a stable solution. This technique has been applied to a wide range of problems (cf. Simpson, 1990a), including scheduling, routing, and resource optimization (see Section 4.3.3).

Two other types of hardwired networks include the avalanche matched filter (Grossberg, 1969; Hecht-Nielsen, 1990) and the probabilistic neural network (Specht, 1990). These networks are considered hardwired systems because the data patterns are normalized to unit length and used as connection weights. Despite the lack of an adaptive learning procedure, each of these neural networks is very powerful in its own right.

5.11. *Summary of Learning Procedures*

Several attributes of each of the neural network learning algorithms have been described. Table 1 describes six key attributes of the learning procedures discussed:

1. *Training time*—How long does it take the learning technique to adequately capture information (quick, slow, very slow, or extremely slow)?
2. *On-line / off-line*—Is the learning technique an on-line or an off-line learning algorithm?
3. *Supervised / unsupervised*—Is the learning technique a supervised or unsupervised learning procedure?
4. *Linear / nonlinear*—Is the learning technique capable of capturing nonlinear mappings?
5. *Structural / temporal*—Does the learning algorithm capture structural information, temporal information, or both?
6. *Storage capacity*—Is the information storage capacity good relative to the number of connections in the network?

The information provided in Table 1 is meant as a guide and is not intended to be a precise description of the qualities of each neural network. For a more detailed description of each neural network learning algorithm, please refer to Simpson,

TABLE 1

Learning Algorithm	Training Time	On-Line/ Off-Line	Supervised/ Unsupervised	Linear/ Nonlinear	Structural/ Temporal	Storage Capacity
Hebbian learning	Fast	On-line	Unsupervised	Linear	Structural	Poor
Principal component learning	Slow	Off-line	Unsupervised	Linear	Structural	Good
Differential Hebbian learning	Fast	On-line	Unsupervised	Linear	Temporal	Undetermined
Competitive learning	Slow	On-line	Unsupervised	Linear	Structural	Good
Min–max learning	Fast	On-line	Unsupervised	Nonlinear	Structural	Good
Two-layer error correction learning	Slow	Off-line	Supervised	Linear	Both	Good
Multilayer error correction learning	Very slow	Off-line	Supervised	Nonlinear	Both	Very good
Reinforcement learning	Extremely slow	Off-line	Supervised	Nonlinear	Both	Good
Stochastic learning	Extremely slow	Off-line	Supervised	Nonlinear	Structural	Good
Hardwired systems	Fast	Off-line	Supervised	Nonlinear	Structural	Good

(1990a), Hecht-Nielsen (1990), or Maren, Harsten, and Pap (1990).

6. Neural Network Recall

The previous section emphasized the storage of information through a wide range of learning procedures. In this section, the emphasis is on retrieving information already stored in the network. Some of the recall equations have been introduced as a part of the learning process. Others will be introduced here for the first time. The recall techniques described here fall into two broad categories: feedforward recall and feedback recall.

6.1. Feedforward Recall

Feedforward recall is performed in networks that do not have feedback connections. The most common feedforward recall technique is the linear combiner (see Section 3.4.1) followed by a PE function.

$$y_j = f\left(\sum_{i=1}^{n} x_i w_{ij}\right) \tag{57}$$

where the PE function f is one of those described in Section 3.5.

For a feedforward network using dual connections (see Section 3.4.2) where one set of connection weights W represents the mean and the other set of connection weights V represents the variance, the recall equation is

$$y_j = g\left(\sum_{i=1}^{n} \left(\frac{w_{ij} - x_i}{v_{ij}}\right)^2\right) \tag{58}$$

where g is the Gaussian PE function (see Section 3.5.5).

For a feedforward network using dual connections where one set of connection weights V represents the min vector and the other set of connection weights W represents the max vector (see Section 3.4.3), and the system is confined to the unit hypercube, there are two possible recall equations: the first is the "product of complements" based equation (Simpson, 1991b)

$$y_j = \left[1 - \frac{1}{n}\sum_{i=1}^{n} \max\left(0, \min\left(1, \gamma\left(v_{ji} - x_i\right)\right)\right)\right] \times \left[1 - \frac{1}{n}\sum_{i=1}^{n} \max\left(0, \min\left(1, \gamma\left(x_i - w_{ji}\right)\right)\right)\right] \tag{59}$$

and the other is the productless relative (Simpson, 1992)

$$y_j = \frac{1}{2n}\sum_{i=1}^{n} \left[\max\left(0, 1 - \max\left(0, \gamma \min\left(1, x_i - w_{ji}\right)\right)\right) + \max\left(0, 1 - \max\left(0, \gamma \min\left(1, v_{ji} - x_i\right)\right)\right)\right] \tag{60}$$

where x_i is the input layer F_x PE value, γ is a value regulating the sensitivity of the membership functions, and y_j is the output value of the jth F_Y PE. Referring to Fig. 5, (59) measures the degree to which the input pattern A_k falls between the min and max vectors of class j, where a value of 1 means that A_k falls completely between V_j and W_j, and the closer y_j is to 0 the greater the disparity between A_k and the class j, with a value of 0 meaning that A_k is completely outside of the class. Note that there are many other possible functions that can be used here. Also note that the relationship between neural networks and fuzzy sets is realized when each PE is seen as a separate fuzzy set (Simpson, 1992).

6.2. Feedback Recall

Those networks that have feedback connections employ a feedback recall equation of the form

$$x_j(t+1) = (1 - \alpha)x_j(t) + \beta \sum_{i=1}^{n} f\left(x_i(t)\right) w_{ij} + a_{ki} \tag{61}$$

where $x_j(t+1)$ is the value of the jth element in a single-layer neural network at time $t+1$, f is a monotonic nondecreasing function (e.g., sigmoid function), α is a positive constant that regulates the amount of decay a PE value has during a unit interval of time, β is a positive constant that

regulates the amount of feedback the other PEs provide the jth PE, and a_{ki} is the constant-valued input from the ith component of the kth input pattern.

One issue that arises in feedback recall systems is stability. Stability is achieved when a network's PEs cease to change in value after they have been given an initial set of inputs, A_k, and have processed for a while. If the network did not stabilize, it would not be of much use. Ideally, the initial inputs to the feedback neural network would represent the input pattern, and the stable state that the network reached would represent the nearest-neighbor output of the system.

An important theorem was presented by Cohen and Grossberg (1983), which proved that, for a wide class of neural networks under a set of minimal constraints, the network would become stable in a finite period of time for any initial conditions. This theorem dealt with systems that had fixed weights. In an extension to the Cohen-Grossberg theorem, Kosko (1990b) showed that a neural network could learn and recall at the same time and yet remain stable.

6.3. *Interpolation versus Nearest-Neighbor Responses*

In addition to recall operations being either feedforward or feedback, there is another important attribute associated with recall, namely output response. There are two types of neural network output response: nearest-neighbor and interpolative. Figure 17 illustrates the difference. Assume that the three face/disposition pairs in Fig. 17(a) have been stored in a neural network. If an input that is a combination of two of the faces is presented to the network, there are two ways that a neural network might respond. If the output is a combination of the two correct outputs associated with the given inputs, then the network has performed an interpolation (see Fig. 17(b)). On the contrary, the network might determine which of the stored faces is most closely associated with the input and respond with the associated output for that face (see Fig. 17(c)). The feedforward pattern-matching neural networks are typically interpolative response networks (e.g., backpropagation and linear associative memory). The feedforward pattern classification networks (e.g., learning vector quantization) and the feedback pattern-matching networks (e.g., Hopfield network and bidirectional associative memory) are typically nearest-neighbor response networks.

7. Neural Network Taxonomy

Several different topologies, learning algorithms, and recall equations have been described. Attempts at organizing the various configurations quickly become unwieldy unless some simple, yet accurate, taxonomy can be applied. The two most prevalent aspects of neural networks, learning supervision and information flow, seem ideally suited to address this need. Table 2 utilizes these criteria to organize the neural networks.

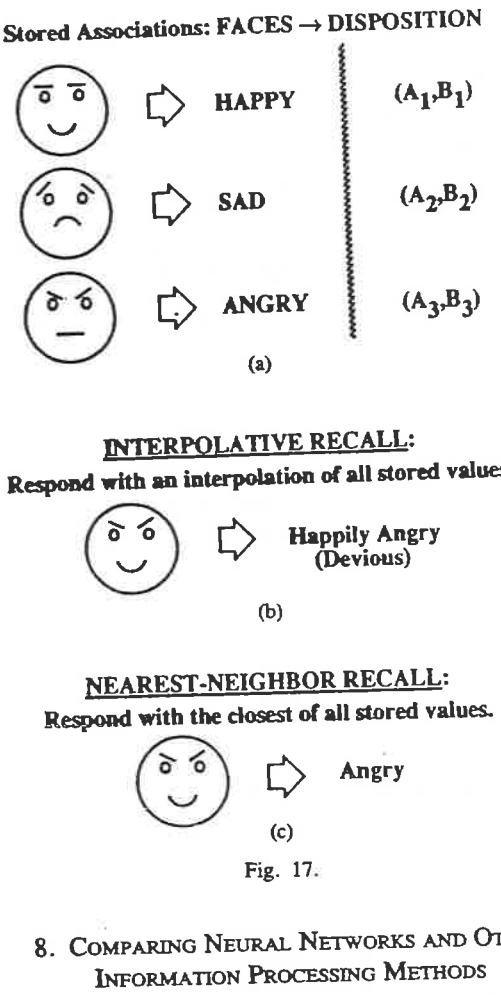


Fig. 17.

8. Comparing Neural Networks and Other Information Processing Methods

Several information processing techniques have capabilities similar to the neural network learning algorithms described. Despite the possibility of equally comparable solutions to a given problem, several additional aspects of a neural network solution are appealing, including fault-tolerance through the large number of connections, parallel implementations that allow fast processing, and on-line adaptation that allows the networks to constantly change according to the needs of the environment. The following sections briefly describe some of the alternative methods that are used for pattern recognition, clustering, control, and statistical analysis.

8.1. *Stochastic Approximation*

The method of stochastic approximation was first introduced by Robbins and Monro (1951) as a method for finding a mapping between inputs and outputs when the inputs and outputs are extremely noisy (i.e., the inputs and outputs are stochastic variables). The stochastic approximation technique has been shown to be identical to the two-layer error correction algorithm presented in Section 5.7.1 (Kohonen, 1984) and the three-layer error correction algorithm presented in Section 5.7.2 (White, 1989).

TABLE 2

Learning	RECALL INFORMATION FLOW: Feedback	RECALL INFORMATION FLOW: Feedforward
Unsupervised	Hopfield networks (Amari, 1972; Hopfield, 1982) ART1 & ART2 (Carpenter and Grossberg, 1987a, b) Bidirectional associative memory (Kosko, 1988; Simpson, 1990b) Principal,component networks (Oja, 1982; Sanger, 1989)	Linear associative memory (Anderson, 1970; Kohonen, 1972) Associative reward–penalty (Barto, 1985) Adaptive heuristic critic (Barto, Sutton, and Anderson, 1983) Drive-reinforcement learning (Klopf, 1986) Learning vector quantization (Kohonen, 1984) Fuzzy min–max classifier (Simpson 1991a) Learnmatrix (Steinbuch and Piske, 1963; Willshaw, 1980)
Supervised	Brain-state-in-a-box (Anderson et al., 1977) Neural optimization (Hopfield and Tank, 1985)	Boltzmann machine (Ackley, Hinton, and Sejnowski, 1985) Neocognitron (Fukushima, 1988) Avalanche matched filter (Grossberg, 1969; Hecht-Nielsen, 1990) Sparse distributed memory (Kanerva, 1988) Gaussian potential function network (Lee and Kil, 1989) Backpropagation (Werbos, 1974; Parker, 1982, Rumelhart et al., 1986) Perceptron (Rosenblatt, 1962) Probabilistic neural network (Specht, 1990) Cauchy machine (Szu, 1986) ADALINE (Widrow and Hoff, 1960)

8.2. *Kalman Filters*

A Kalman filter is a technique for estimating, or predicting, the next state of a system based upon a moving average of measurements driven by additive white noise. The Kalman filter requires a model of the relationship between the inputs and the outputs to provide feedback that allows the system to continuously perform its estimation. Kalman filters are primarily used for control systems. Singhal and Wu (1989) have developed a method of using a Kalman filter to train the weights of a multilayer neural network. In some recent work, Ruck et al. (1990) have shown that the backpropagation algorithm is a special case of the extended Kalman filter algorithm, and have provided several comparative examples of the two training algorithms on a variety of data sets.

8.3. *Linear and Nonlinear Regression*

Linear regression is a technique for fitting a line to a set of data points such that the total distance between the line and the data points is minimized. This technique, used widely in statistics (Spiegel, 1975), is similar to the two-layer error correction learning algorithm described in Section 5.7.1.

Nonlinear regression is a technique for fitting curves (nonlinear surfaces) to data points. White (1990) points out that the PE function used in many error correction learning algorithms is a family of curves, and the adjustment of the weights that minimizes the overall mean-squared error is equivalent to curve fitting. In this sense, the backpropagation algorithm described in Section 5.7.2 is an example of an automatic nonlinear regression technique.

8.4. *Correlation*

Correlation is a method of comparing two patterns. One pattern is the template and the other is the input. The correlation between the two patterns is the dot product. Correlation is used extensively in pattern recognition (Young and Fu, 1986) and signal processing (Elliot, 1987). In pattern recognition the templates and inputs are normalized, allowing the dot-product operation to provide similarities based upon the angles between vectors. In signal processing the correlation procedure is often used for comparing templates with a time series to determine when a specific sequence occurs (this technique is commonly referred to as cross-correlation or matched filters). The Hebbian learning techniques described in Section 5.2 are correlation routines that store correlations in a matrix and compare the stored correlations with the input pattern by using inner products.

8.5. *Bayes Classification*

The purpose of pattern classification is to determine to which class a given pattern belongs. If the class boundaries are not clearly separated and tend to overlap, the classification system must find the boundary between the classes that minimizes the average misclassification (error). The smallest possible error is referred to as the Bayes error, and a

classifier that provides the Bayes error is called a Bayes classifier (Fukunuga, 1986). Two methods are often used for designing Bayes classifiers: the Parzen approach and k-nearest neighbors. The Parzen approach utilizes a uniform kernel (typically the Gaussian function) to approximate the probability density function of the data. A neural network implementation of this approach (see Section 4.5) is the probabilistic neural network (Specht, 1990). The k-nearest-neighbors approach uses k vectors to approximate the underlying distribution of the data. The learning vector quantization network (Kohonen, 1984) is similar to the k-nearest-neighbors approach (see Section 5.5).

8.6. Vector Quantization

The purpose of vector quantization is to produce a code from an n-dimensional input pattern. The code is passed across a channel and then used to reconstruct the original input with minimal distortion. There have been several techniques proposed to perform vector quantization (Gray, 1984), with one of the most successful being the LBG algorithm (Linde, Buzo, and Gray, 1980). The learning vector quantization (see Section 5.5) is a method of developing a set of reference vectors from a data set and is very similar to the LBG algorithm. A comparison of these two techniques can be found in Ahalt et al. (1990).

8.7. Radial Basis Functions

A radial basis function is a function that is symmetric about a given mean (e.g., a Gaussian function). In pattern classification, a radial basis function is used in conjunction with a set of n-dimensional reference vectors, where each reference vector has a radial basis function that constrains its response. An input pattern is processed through the basis functions to produce an output response. The mean-variance connection topologies that employ the backpropagation algorithm (Lee and Kil, 1989; Robinson, Niranjan, and Fallside, 1988) as described in Section 5.7.2 are methods of automatically producing the proper sets of basis functions (by adjustment of the variances) and their placement (by adjustment of their means).

8.8. Machine Learning

Neural networks are not the only method of learning that has been proposed for machines (although they are the most biologically related). Numerous machine learning procedures have been proposed during the past 30 years. Carbonell (1990) classifies machine learning into four major paradigms (p. 2):

> [I]nductive learning (e.g., acquiring concepts from sets of positive and negative examples), analytic learning (e.g., explanation-based learning and certain forms of analogical and case-based learning methods), genetic algorithms (e.g., classifier systems), and connectionist learning methods (e.g., nonrecurrent "backprop" hidden layer neural networks).

It is possible that some of the near-term applications might find it useful to combine two or more of these machine learning techniques into a coherent solution. It has only been recently that this type of approach has even been considered.

References

Ackley, D., G. Hinton, and T. Sejnowski (1985), "A learning algorithm for Boltzmann machines," *Cognitive Sci.*, vol. 9, pp. 147–169.

Ahalt, S., A. Krishnamurthy, P. Chen, and D. Melton (1990), "Competitive learning algorithms for vector quantization," *Neural Networks*, vol. 3, pp. 277–290.

Amari, S. (1971), "Characteristics of randomly connected threshold-element networks and network systems," *Proc. IEEE*, vol. 59, pp. 35–47, Jan.

Amari, S. (1972), "Learning patterns and pattern sequences by self-organizing nets of threshold elements," *IEEE Trans. Computer*, vol. C-21, pp. 1197–1206, Nov.

Amari, S. (1977), "Neural theory of association and concept formation," *Biol. Cybernet.*, vol. 26, pp. 175–185.

Amari, S. (1983), "Field theory of self-organizing neural nets," *IEEE Trans. Systems, Man, Cybernet.*, vol. SMC-13, pp. 741–748, Sept./Oct.

Amari, S. and M. Takeuchi (1978), "Mathematical theory on formation of category detecting nerve cells," *Biol. Cybernet.*, vol. 29, pp. 127–136.

Anderson, J. (1970), "Two models for memory organization using interactive traces," *Math. Biosci.*, vol. 8, pp. 137–160.

Anderson, J. (1990), "Knowledge representation in neural networks," *AI Expert*, Fall.

Anderson, J., J. Silverstein, S. Ritz, and R. Jones (1977), "Distinctive features, categorical perception, and probability learning: Some applications of a neural model," *Pysch. Rev.*, vol. 84, pp. 413–451.

Barto, A. (1984), "Simulation experiments with goal-seeking adaptive elements," Air Force Wright Aeronautical Laboratory, Technical Report AFWAL-TR-84-1022.

Barto, A. (1985), "Learning by statistical cooperation of self-interested neuron-like computing units," *Human Neurobiol.*, vol. 4, pp. 229–256.

Barto, A., R. Sutton, and C. Anderson (1983), "Neuron-like adaptive elements that can solve difficult learning control problems," *IEEE Trans. Systems, Man, Cybernet.*, vol. SMC-13, pp. 834–846, Sept./Oct.

Carbonell, J. (1990), "Introduction: Paradigms for machine learning," in *Machine Learning: Paradigms and Methods*, J. Carbonell, Ed., Cambridge, MA: MIT/Elsevier, pp. 1–10.

Carpenter, G. and S. Grossberg (1987a), "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Understanding*, vol. 37, pp. 54–115.

Carpenter, G. and S. Grossberg (1987b), "ART2: Self-organization of stable category recognition codes for analog input patterns," *Appl. Optics*, vol. 26, pp. 4919–4930.

Cohen, M. and S. Grossberg (1983), "Absolute stability of global pattern formation and parallel storage by competitive neural networks," *IEEE Trans. Systems, Man, Cybernet.*, vol. SMC-13, p. 815–825, Sept./Oct.

DeSieno, D. (1988), "Adding a conscience to competitibe learning," in *Proc. 1988 Int. Conf. Neural Networks*, vol. I, pp. 117–124.

Eberhart, R. (1990), "Standardization of neural network terminology," *IEEE Trans. Neural Networks*, vol. 1, pp. 244–245, June.

Elliot, D., Ed. (1987), *Handbook of Digital Signal Processing: Engineering Applications*, San Diego, CA: Academic Press.

Fukunuga, K. (1986), "Statistical pattern classification," in *Handbook of Pattern Recognition and Image Proc.*, T. Young and K. Fu, Eds., San Diego, CA: Academic Press, pp. 3–32.

Fukushima, K. (1988), "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, pp. 119–130.

Gray, R. (1984), "Vector quantization," *IEEE ASSP Mag.*, vol. 1, no. 2, pp. 4–29, Apr.

Grossberg, S. (1969), "On the serial learning of lists," *Math. Biosci.*, vol. 4, pp. 201–253.

Grossberg, S. (1970), "Neural pattern discrimination," *J. Theoret. Biol.*, vol. 27, pp. 291–337.

Grossberg, S. (1982), *Studies of Mind and Brain*, Boston: Reidel.

Hebb, D. (1949), *Organization of Behavior*, New York: Wiley.

Hecht-Nielsen, R. (1990), *Neurocomputing*, Reading, MA: Addison-Wesley.

Hertz, J. et al. (1990), *Introduction to the Theory of Neural Computation*, Reading, MA: Addison-Wesley.

Hopfield, J. (1982), "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. U.S.A.*, vol. 79, pp. 2554–2558.

Hopfield, J. and D. Tank (1985), "'Neural' computation of decisions in optimization problems," *Biol. Cybernet.*, vol. 52, pp. 141–152.

Kanerva, P. (1988), *Sparse Distributed Memory*, Cambridge, MA: MIT Press.

Kirkpatrick, S., C. Gelatt, and M. Vecchi (1983), "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680.

Klopf, A. (1986), "Drive-reinforcement model of a single neuron function: An alternative to the Hebbian neuron model," in *AIP Conf. Proc. 151: Neural Networks for Computing*, J. Denker, Ed., New York: American Institute of Physics, pp. 265–270.

Kohonen, T. (1972), "Correlation matrix memories," *IEEE Trans. Computer*, vol. C-21, pp. 353–359, Apr.

Kohonen, T. (1984), *Self-organization and Associative Memory*, Berlin: Springer-Verlag.

Kohonen, T. (1986), "Learning vector quantization for pattern recognition," Helsinki University of Technology, Technical Report No. TKK-F-A601.

Kosko, B., (1986a), "Fuzzy entropy and conditioning," *Information Sci.*, vol. 40, pp. 165–174.

Kosko, B. (1986b), "Differential Hebbian learning," in *AIP Conf. Proc. 151: Neural Networks for Computing*, J. Denker, Ed., New York: American Institute of Physics, pp. 277–282.

Kosko, B. (1988), "Bidirectional associative memories," *IEEE Trans. Systems, Man, Cybernet.*, vol. SMC-18, pp. 42–60, Jan./Feb.

Kosko, B. (1990a), "Fuzziness vs. probability," *Int. J. General Systems*, vol. 17, pp. 211–240.

Kosko, B. (1990b), "Unsupervised learning in noise," *IEEE Trans. Neural Networks*, vol. 1, pp. 44–57, Mar.

Lawley, D. and A. Maxwell (1963), *Factor Analysis as a Statistical Method*, London: Butterworths.

Lee, S. and R. Kil (1989), "Bidirectional continuous associator based on Gaussian potential function network," in *Proc. IEEE/INNS Int. Joint Conf. Neural Networks*, vol. I, pp. 45–54.

Linde, Y., A. Buzo, and R. M. Gray (1980), "An algorithm for vector quantizer design," *IEEE Trans. Communications*, vol. 28, no. 1, pp. 84–95.

Malsburg, C. v. d. (1973), "Self-organization of orientation sensitive cells in the striate cortex," *Kybernetik*, vol. 14, pp. 85–100.

Maren, A., C. Harston, and R. Pap (1990), *Handbook of Neural Computing Applications*, San Diego, CA: Academic Press.

McEliece, R., E. Posner, E. Rodemich, and S. Venkatesh (1987), "The capacity of the Hopfield associative memory," *IEEE Trans. Information Theory*, vol. IT-33, pp. 461–482, July.

Minsky, M. (1961), "Steps toward AI," *Proc. of the IRE*, vol. 49, pp. 5–30.

Oja, E. (1982), "A simplified neuron model as a principal component analyzer," *J. Math. Biol.*, vol. 15, pp. 267–273.

Oja, E. (1989), "Neural networks, principle components, and subspaces," *Int. J. Neural Networks*, vol. 1, pp. 61–68.

Pao, Y. (1989), *Adaptive Pattern Recognition and Neural Networks*, Reading, MA: Addison-Wesley.

Parker, D. (1982), "Learning logic," Stanford University, Dept. of Electrical Engineering, Invention Report 581-64, Oct.

Robbins, H. and S. Monro (1951), "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407.

Robinson, A., M. Niranjan, and F. Fallside (1988), "Generalizing the nodes of the error propagation network," Cambridge University Engineering Department, Technical Report CUED/F-INENG/TR.25.

Rosenblatt, F. (1962), *Principles of Neurodynamics*, Washington, DC: Spartan Books.

Rozonoer, L. (1969), "Random logic networks I, II, III," in *Automatic Remote Control*, vols. 5–7, pp. 137–147, 99–109, and 129–136.

Ruck, D., S. Rogers, M. Kabrisky, P. Maybeck, and M. Oxley (1990), "Comparative analysis of backpropagation and the extended Kalman filter for training multilayer perceptrons," *IEEE Trans. Pattern Anal. Machine Intelligence*, in review.

Rumelhart, D., G. Hinton, and R. Williams (1986), "Learning representations by backpropagating errors," *Nature*, vol. 323, pp. 533–536.

Sanger, T. (1989), "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks*, vol. 2, pp. 459–473.

Sejnowski, T. (1977), "Storing covariance with nonlinearly interacting neurons," *J. Math. Biol.*, vol. 4, pp. 303–321.

Simpson, P. (1990a), *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations*, Elmsford, NY: Pergamon Press.

Simpson, P. (1990b), "Higher-ordered and intraconnected bidirectional associative memories," *IEEE Trans. Systems, Man, Cybernet.*, vol. 20, pp. 637–653, May/June.

Simpson, P. (1990c), "Fuzzy adaptive resonance theory," presented at Southern Illinois Neuroengineering Workshop, Sept., and published as General Dynamics Technical Report GDE-ISG-PKS-010, Apr. (revised Nov. 1990).

Simpson, P. (1991a), "Fuzzy min-max classification with neural networks," *Heuristics*, vol. 4, no. 7, pp. 1–9.

Simpson, P. (1991b), "Fuzzy min-max neural networks," in *Proc. 1991 Int. Joint Conf. Neural Networks* (Singapore), pp. 1658–1669.

Simpson, P. (1992), "Fuzzy min-max neural networks: I. Classification," *IEEE Trans. Neural Networks*, in press.

Singhal, S. and L. Wu (1989), "Training multi-layer perceptrons with the extended Kalman algorithm," in *Advances in Neural Information Processing Systems 1*, D. Touretzky, Ed., San Mateo, CA: Kaufmann, pp. 133–140.

Specht, D. (1990), "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109–118.

Spiegel, M. (1975), *Schaum's Outline of Theory and Problems of Probability and Statistics*, New York: McGraw-Hill.

Steinbuch, K. and U. Piske (1963), "Learning matrices and their applications," *IEEE Trans. Electronic Computers*, vol. EC-12, pp. 846–862, Dec.

Sutton, R. and A. Barto (1981), "Toward a modern theory of adaptive networks: Expectation and prediction," *Psych. Rev.*, vol. 88, pp. 135–171.

Szu, H. (1986), "Fast simulated annealing," in *AIP Conf. Proc. 151: Neural Networks for Computing*, J. Denker, Ed., New York: American Institute of Physics, pp. 420–425.

Tank, D. and J. Hopfield (1986), "Simple 'neural' optimization networks: A/D convertor, signal decision circuit, and a linear programming circuit," *IEEE Trans. Circuits Systems*, vol. CAS-33, pp. 533–541, May.

Tesauro, G. (1986), "Simple neural models of classical conditioning," *Biol. Cybernet.*, vol. 55, pp. 187–200.

Werbos, P. (1974), "Beyond regression," Ph.D. dissertation, Harvard University, Cambridge, MA.

White, H. (1989), "Learning in neural networks: A statistical perspective," *Neural Computation*, vol. 1, pp. 425–464.

White, H. (1990), "Neural network learning and statistics," *AI Expert*, Fall.

Widrow, B. and M. Hoff (1960), "Adaptive switching circuits," in *1960 WESCON Convention Record: Part IV*, pp. 96–104.

Widrow, B., N. K. Gupta, and S. Maitra (1973), "Punish/reward: Learning with a critic in adaptive threshold systems," *IEEE Trans. Syst., Man, Cybernetics*, vol. SMC-3, no. 5, pp. 455–465.

Widrow, B. and S. Stearns (1985), *Adaptive Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall.

Widrow, B. and R. Winter (1988), "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Computer Mag.*, pp. 25–39, Mar.

Williams, R. (1986), "Reinforced learning in connection to networks: A mathematical analysis," University of California, Institute for Cognitive Science, Technical Report No. 8605.

Willshaw, D. (1980), "Holography, associative memory, and inductive generalization," in *Parallel Models of Associative Memory*, J. Anderson and G. Hinton, Eds., Hillsdale, NJ: Lawrence Erlbaum, pp. 103–122.

Young, T. and K. Fu, Eds. (1986), *Handbook of Pattern Recognition and Image Processing*, San Diego, CA: Academic Press.