

**Figure 6.7** Example of an asynchronous update of a corrupted negative image of a bit map of digit 4: (a) key vector, (b) after first cycle, (c) after second cycle, and d) after third cycle.

operation. Indeed, storing complementary patterns  $\mathbf{s}'^{(m)}$  instead of original patterns  $\mathbf{s}^{(m)}$  results in the weights as follows:

$$w'_{ij} = (1 - \delta_{ij}) \sum_{m=1}^p (2s_i'^{(m)} - 1)(2s_j'^{(m)} - 1) \quad (6.22)$$

The reader can easily verify that substituting

$$s_i'^{(m)} = 1 - s_i^{(m)} \quad (6.23)$$

into (6.22) results in  $w'_{ij} = w_{ij}$ . Figure 6.7 shows four example convergence steps for an associative memory consisting of 120 neurons with a stored binary bit map of digit 4. Retrieval of a stored pattern initialized as shown in Figure 6.7(a) terminates after three cycles of convergence as illustrated in Figure 6.7(d). It can be seen that the recall has resulted in the true complement of the bit map originally stored. The reader may notice similarities between Figures 5.2 and 6.7.

**EXAMPLE 6.1**

In this example we will discuss the simplest recurrent associative memory network known, which at the same time, seems to be the most commonly used electronic network in the world. Although the network has been popular for more than half a century, we will take a fresh look at it and investigate its performance as of a neural memory. The bistable flip-flop is the focus of our discussion. In fact, most digital computers use this element as a basic memory cell and as a basic computing device.

The flip-flop network is supposed to find and indefinitely remain in one of the two equilibrium states:  $s^{(1)} = [1 \ -1]^t$  or  $s^{(2)} = [-1 \ 1]^t$ .

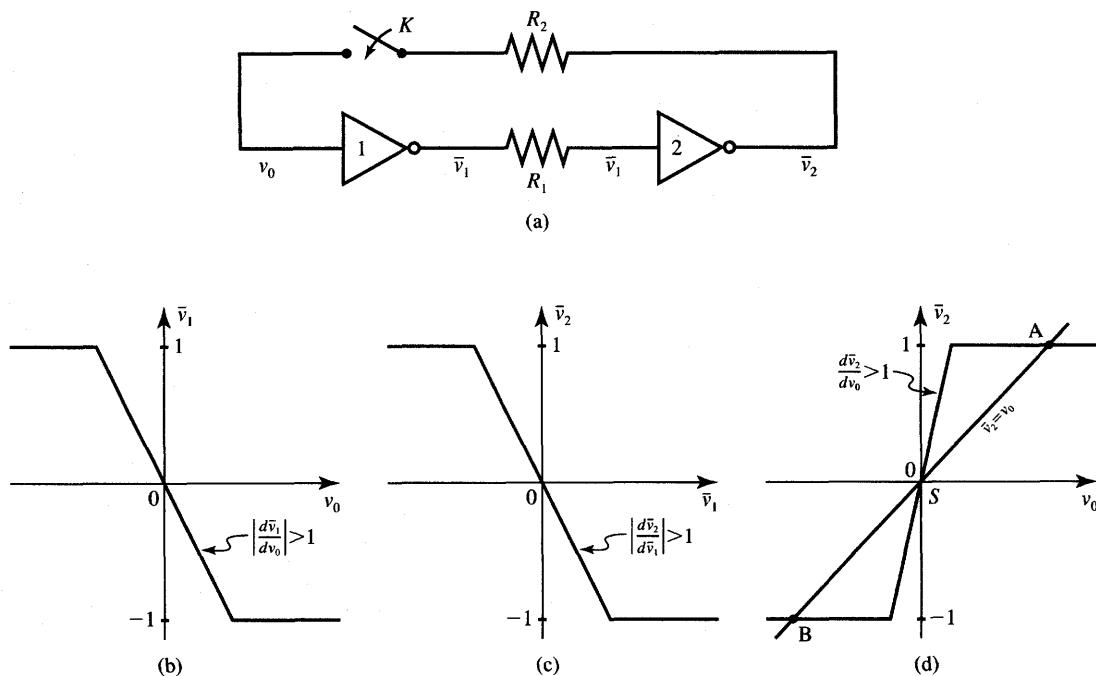
Before we design a bistable flip-flop using the associative memory concept, let us review its basic physical properties. The network as shown in Figure 6.8(a) consists of two cascaded inverting neurons. Consider the feedback loop switch  $K$  to be open initially. Each of the neurons represented by amplifiers with saturation is characterized by the two identical voltage transfer characteristics as shown in Figures 6.8(b) and (c).

Resistances  $R_1$  and  $R_2$  have to be positive, but their values are of no significance since the ideal neurons absorb no input current. By connecting two neurons in cascade, the output  $\bar{v}_2$  is a function of  $\bar{v}_1$ , and  $v_1$  is a function of  $v_0$ . Thus,  $\bar{v}_2$  can be expressed as a composite function of  $v_0$ . The cascade of two amplifiers has a composed characteristic  $\bar{v}_2[\bar{v}_1(v_0)]$  as illustrated in Figure 6.8(d). When the switch  $K$  closes and enforces the condition  $\bar{v}_2 = v_0$  represented by the unity-slope straight line on the figure, the circuit immediately starts evolving toward one of the equilibrium points. Note that the three potential equilibrium points are  $A$ ,  $B$ , and  $S$ .

The origin  $S$  of the coordinates here is an unstable equilibrium point. If any perturbation arises in either direction with the network momentarily at  $S$ , it is going to be magnified by the positive feedback within the network so the transition will finally end in either  $A$  or  $B$ . Thus, the physical memory network will never stabilize at  $S$ , which is supposedly the saddle point of its energy function. Note that the equilibrium states for high-gain amplifiers are, for points  $A$  and  $B$ , respectively:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \cong \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \cong \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad (6.24)$$

The dynamics of the equilibrium search, which is essential for the network's physical operation during the transition period, has been suppressed in this description for simplicity. The issue of dynamics was extensively discussed in Chapter 5. A trace of capacitance and neuron input conductance is, in fact, indispensable for this circuit to respond. The associative memory discussed here performs the input/output mapping, and discrete-time formalism is used only to characterize it. The physical output update process, however,



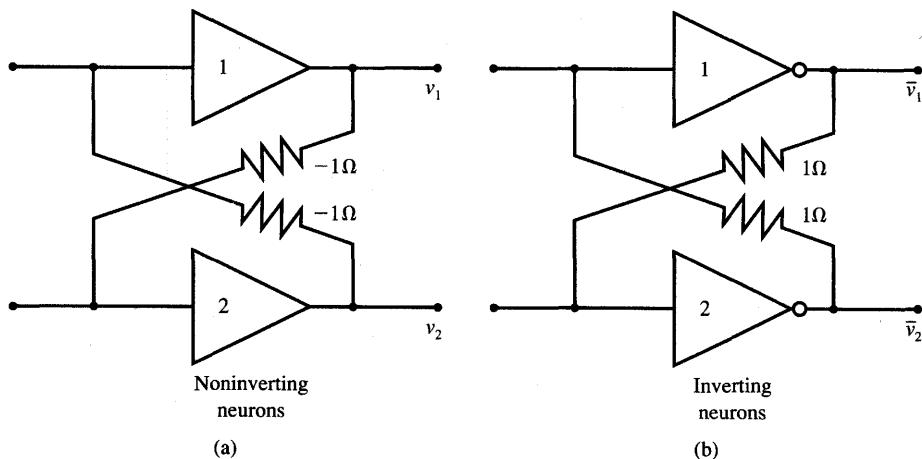
**Figure 6.8** Bistable flip-flop: (a) circuit diagram, (b) voltage transfer characteristic of amplifier 1, (c) voltage transfer characteristic of amplifier 2, and (d) joint characteristics and equilibrium states.

follows the continuous-time behavior of the model, which uses the continuous activation function of the neurons.

Knowing now the basic physical principles of the network, we can approach the problem of a bistable neural memory design with better understanding. Since the learning (storage) rule is invariant under the binary complement operation, it is sufficient to store either of the two patterns  $s^{(1)}$  and  $s^{(2)}$  only. From (6.20a) we obtain the weight matrix

$$\mathbf{W} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \quad (6.25)$$

This results in the bistable flip-flop circuit shown in Figure 6.9(a) with weights of  $w_{12} = w_{21} = -1 \Omega$ . The circuit has negative resistances, which are difficult to implement. It has, however, noninverting neurons. By transforming noninverting neurons into inverting neurons and by changing the signs of the resistances, an equivalent circuit is obtained as shown in Figure 6.9(b). Although the equivalency is valid strictly for a hard-limiting activation function and zero input conductance of the neurons, both circuits of



**Figure 6.9** Bistable flip-flop as an associative memory: (a) circuit diagram and (b) realizable circuit diagram.

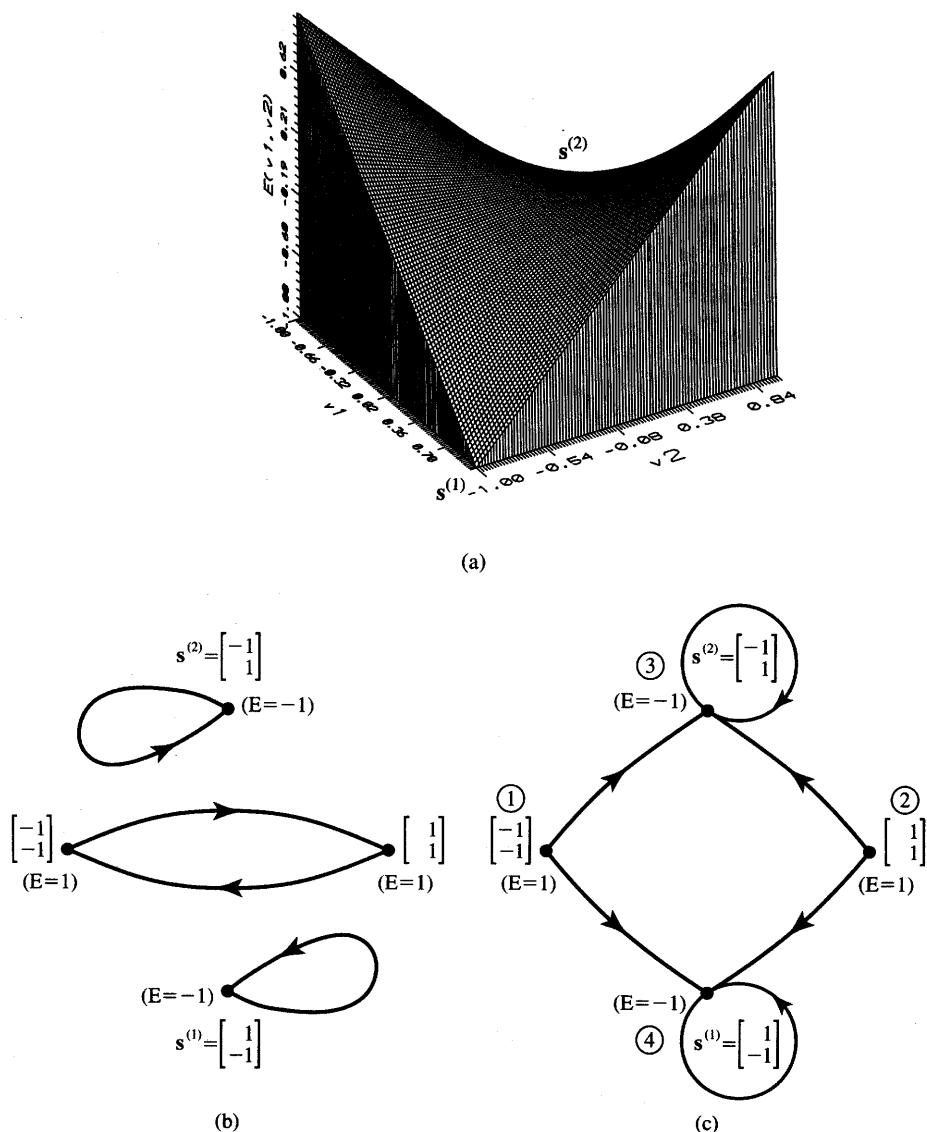
Figure 6.9 can be considered equivalent for all practical purposes. As we can see, the memory circuit obtained is identical to the basic flip-flop circuit configuration introduced earlier in Figure 6.8(a).

To complete our study of the network, let us consider its energy function. Substituting (6.25) into (6.19), the energy function of this memory network is obtained as

$$E = v_1 v_2 \quad (6.26)$$

The energy function is shown in Figure 6.10(a). It consists of two surfaces monotonically decreasing toward  $s^{(1)}$  and  $s^{(2)}$ , where  $s^{(1)} = [1 \ -1]^t$  and  $s^{(2)} = [-1 \ 1]^t$ . The ridge of the function is at  $v_1 = v_2$ . It can easily be seen that for the initial condition  $v_1^0 < v_2^0$ , the update process will result in stable output pattern  $s^{(2)}$ . For  $v_1^0 > v_2^0$ , the network will stabilize at output  $s^{(1)}$ . The figure shows that this network has a virtually perfect recall ability to the closer of the two memories stored. These properties refer to the update by the continuous-time network employing neurons with the continuous activation function.

Let us inspect the performance of the memory operating in the discrete-time mode. The memory cell now employs two bipolar binary neurons. In Section 5.2 we considered the recurrent memory network with the weight matrix as in (6.25). We then showed that the synchronous transitions of the network were oscillations between  $[-1 \ -1]^t$  and  $[1 \ 1]^t$ , provided the updates originated at one of these outputs. The transition map for this case is depicted in Figure 6.10(b). It can be seen from the map that the energy



**Figure 6.10** Energy function and transition maps for the bistable flip-flop: (a) energy surface, (b) transition map for synchronous update, and (c) transition map for asynchronous update.

minimization property does not hold for the synchronous update mode since there are lower energy levels that cannot be reached. This is the case for oscillatory updates between the energy levels of value 1 never reaching the lower value of -1.

Consider now the asynchronous updates for this circuit. Starting at each of the first four initial vectors  $v^0$  we obtain in a single update step:

1. For  $v^0 = [-1 \ -1]^t$  and the first neuron allowed to update

$$v^1 = \Gamma \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} = s^{(1)}$$

Energy has dropped from 1 to  $-1$  as a result of the update and the network has settled at  $s^{(1)}$ . Starting again at  $v^0$  as above but with the second neuron allowed to update, we obtain

$$v^1 = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} = s^{(2)}$$

and we have the same energy drop from 1 to  $-1$  while the network settled at  $s^{(2)}$ .

2. For  $v^0 = [1 \ 1]^t$  and the first neuron allowed to update

$$v^1 = \Gamma \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} = s^{(2)}$$

Energy has dropped from 1 to  $-1$  as a result of this update. Allowing the second neuron to update, we obtain

$$v^1 = \Gamma \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} = s^{(1)}$$

and we have the same energy drop from 1 to  $-1$ .

3. For  $v^0 = [-1 \ 1]^t$  we obtain

$$v^1 = \Gamma \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} = s^{(2)}$$

independent of the update sequence. The energy value remains at the lowest level of  $-1$ .

4. For  $v^0 = [1 \ -1]^t$  we obtain

$$v^1 = \Gamma \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} = s^{(1)}$$

independent of the update sequence. As in step 3, energy remains at the lowest level of  $-1$ .

The summary of the transitions in the asynchronous mode for this simple associative memory is illustrated in Figure 6.10(c). The figure shows the discussed transitions between the four vertices of the graph. Case numbers correspond to initial node numbers which are circled on the figure.

Because of the network size, the presented example has been more educational than of any practical value. The practical importance of recurrent associative memory is for larger size networks. In fact, many of its attractive properties related to recall and noise suppression are valid under statistical independence of stored vectors and for large  $n$  values. As stated before, the network is able to produce correct stored states when an incomplete or noisy version of the stored vector is applied at the input and after the network is allowed to update its output asynchronously.

■ *Summary of the Recurrent Associative Memory Storage and Retrieval Algorithm (RAMSRA)*

Given are  $p$  bipolar binary vectors:

$$\left\{ \mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \dots, \mathbf{s}^{(p)} \right\}, \text{ where } \mathbf{s}^{(m)} \text{ is } (n \times 1), \text{ for } m = 1, 2, \dots, p$$

Initializing vector  $\mathbf{v}^0$  is  $(n \times 1)$

**Storage**

**Step 1:** Weight matrix  $\mathbf{W}$  is  $(n \times n)$ :

$$\mathbf{W} \leftarrow \mathbf{0}, \quad m \leftarrow 1$$

**Step 2:** Vector  $\mathbf{s}^{(m)}$  is stored:

$$\mathbf{W} \leftarrow \mathbf{s}^{(m)} \mathbf{s}^{(m)t} - \mathbf{I}$$

**Step 3:** If  $m < p$  then  $m \leftarrow m + 1$ , and go to Step 2, otherwise, go to Step 4.

**Step 4:** Recording of vectors is completed. Output weights  $\mathbf{W}$ .

**Recall**

**Step 1:** Cycle counter  $k$  is initialized,  $k \leftarrow 1$ . Update counter  $i$  within cycle is initialized,  $i \leftarrow 1$  and the network is initialized,  $\mathbf{v} \leftarrow \mathbf{v}^0$ .

**Step 2:** Integers  $1, 2, \dots, n$ , are arranged in an ordered random sequence  $\alpha_1, \alpha_2, \dots, \alpha_n$  for this cycle. (See note at end of list.)

**Step 3:** Neuron  $i$  is updated by computing  $v_{new_{\alpha_i}}$

$$net_{\alpha_i} = \sum_{j=1}^n w_{\alpha_i j} v_j$$

$$v_{new_{\alpha_i}} = \text{sgn}(net_{\alpha_i})$$

**Step 4:** If  $i < n$ , then  $i \leftarrow i + 1$ , and go to Step 3; otherwise, go to Step 5.

**Step 5:** If  $v_{new_{\alpha_i}} = v_{\alpha_i}$ , for  $i = 1, 2, \dots, n$ , then no updates occur in this cycle; thus, recall is complete; output  $k$  and  $v_{new_1}, v_{new_2}, \dots, v_{new_n}$ ; otherwise  $k \leftarrow k + 1$  and go to Step 2.

**■ NOTE:** Recall is much simpler to implement without randomizing the update sequence. In such case, Step 2 of the algorithm is eliminated and we have  $\alpha_1 = 1, \alpha_2 = 2, \dots, \alpha_n = n$ , or simply  $\alpha_i = i$  at each update cycle. Elimination of Step 2 can, however, result in reduced efficiency of retrieval.

### Performance Considerations

Hopfield autoassociative memory is often referred to in the literature as an error correcting decoder in that, given an input vector that is equal to the stored memory plus random errors, it produces as output the original memory that is closest to the input. The reason why the update rule proposed by Hopfield can reconstruct a noise-corrupted or incomplete pattern can be understood intuitively. The memory works best for large  $n$  values and this is our assumption for further discussion of memory's performance evaluation. Let us assume that a pattern  $s^{(m')}$  has been stored in the memory as one of  $p$  patterns. This pattern is now at the memory input. The activation value of the  $i$ 'th neuron for the update rule (6.17) for retrieval of pattern  $s^{(m')}$  has the following form:

$$net_i^{(m')} = \sum_{j=1}^n w_{ij} s_j^{(m')} \quad (6.27a)$$

or, using (6.20b) and temporarily neglecting the contribution coming from the nullification of the diagonal, we obtain

$$net_i^{(m')} \cong \sum_{j=1}^n \sum_{m=1}^p s_i^{(m)} s_j^{(m)} s_j^{(m')} \quad (6.27b)$$

Thus, we can write

$$net_i^{(m')} \cong \sum_{m=1}^p s_i^{(m)} \sum_{j=1}^n s_j^{(m)} s_j^{(m')} \quad (6.27c)$$

If terms  $s_j^{(m)}$  and  $s_j^{(m')}$ , for  $j = 1, 2, \dots, n$ , were totally statistically independent or unrelated for  $m = 1, 2, \dots, p$ , then the average value of the second sum resulted in zero. Note that the second sum is the scalar product of two  $n$ -tuple vectors and if the two vectors are statistically independent (also when orthogonal) their product vanishes. If, however, any of the stored patterns  $s^{(m)}$ , for  $m = 1, 2, \dots, p$ ,

and vector  $\mathbf{s}^{(m')}$  are somewhat overlapping, then the value of the second sum becomes positive. Note that in the limit case the second sum would reach  $n$  for both vectors being identical, understandably so since we have here the scalar product of two identical  $n$ -tuple vectors with entries of value  $\pm 1$ . Thus for the major overlap case, the sign of entry  $s_i^{(m')}$  is expected to be the same as that of  $net_i^{(m')}$ , and we can write

$$net_i^{(m')} \cong s_i^{(m')} n, \quad \text{for } i = 1, 2, \dots, n \quad (6.27d)$$

This indicates that the vector  $\mathbf{s}^{(m')}$  does not produce any updates and is therefore stable.

Assume now that the input vector is a distorted version of the prototype vector  $\mathbf{s}^{(m')}$ , which has been stored in the memory. The distortion is such that only a small percentage of bits differs between the stored memory  $\mathbf{s}^{(m')}$  and the initializing input vector. The discussion that formerly led to the simplification of (6.27c) to (6.27d) still remains valid for this present case with the additional qualification that the multiplier originally equal to  $n$  in (6.27d) may take a somewhat reduced value. The multiplier becomes equal to the number of overlapping bits of  $\mathbf{s}^{(m')}$  and of the input vector.

It thus follows that the impending update of node  $i$  will be in the same direction as the entry  $s_i^{(m')}$ . Negative and positive bits of vector  $\mathbf{s}^{(m')}$  are likely to cause negative and positive transitions, respectively, in the upcoming recurrences. We may say that the majority of memory initializing bits is assumed to be correct and allowed to take a vote for the minority of bits. The minority bits do not prevail, so they are flipped, one by one and thus asynchronously, according to the will of the majority. This shows vividly how bits of the input vector can be updated in the right direction toward the closest prototype stored. The above discussion has assumed large  $n$  values, so it has been more relevant for real-life application networks.

A very interesting case can be observed for the stored orthogonal patterns  $\mathbf{s}^{(m)}$ . The activation vector  $\mathbf{net}$  can be computed as

$$\mathbf{net} = \left( \sum_{m=1}^p \mathbf{s}^{(m)} \mathbf{s}^{(m)t} - p \mathbf{I} \right) \mathbf{s}^{(m')} \quad (6.28a)$$

The orthogonality condition, which is  $\mathbf{s}^{(i)t} \mathbf{s}^{(j)} = 0$ , for  $i \neq j$ , and  $\mathbf{s}^{(i)t} \mathbf{s}^{(j)} = n$ , for  $i = j$ , makes it possible to simplify (6.28a) to the following form

$$\mathbf{net} = (n - p) \mathbf{s}^{(m')} \quad (6.28b)$$

Assuming that under normal operating conditions the inequality  $n > p$  holds, the network will be in equilibrium at state  $\mathbf{s}^{(m')}$ . Indeed, computing the value of the energy function (6.19) for the storage rule (6.20b) we obtain

$$E(\mathbf{v}) = -\frac{1}{2} \mathbf{v}^t \left( \sum_{m=1}^p \mathbf{s}^{(m)} \mathbf{s}^{(m)t} \right) \mathbf{v} + \frac{1}{2} \mathbf{v}^t p \mathbf{I} \mathbf{v} \quad (6.29a)$$

For every stored vector  $\mathbf{s}^{(m')}$  which is orthogonal to all other vectors the energy value (6.29a) reduces to

$$E(\mathbf{s}^{(m')}) = -\frac{1}{2} \sum_{m=1}^p (\mathbf{s}^{(m')\top} \mathbf{s}^{(m)}) (\mathbf{s}^{(m)\top} \mathbf{s}^{(m')}) + \frac{1}{2} \mathbf{s}^{(m')\top} \mathbf{p} \mathbf{I} \mathbf{s}^{(m')} \quad (6.29b)$$

and further to

$$E(\mathbf{s}^{(m')}) = -\frac{1}{2}(n^2 - pn) \quad (6.29c)$$

The memory network is thus in an equilibrium state at every stored prototype vector  $\mathbf{s}^{(m')}$ , and the energy assumes its minimum value expressed in (6.29c).

Considering the simplest autoassociative memory with two neurons and a single stored vector ( $n = 2, p = 1$ ), Equation (6.29c) yields the energy minimum of value  $-1$ . Indeed, the energy function (6.26) for the memory network of Example 6.1 has been evaluated and found to have minima of that value.

For the more general case, however, when stored patterns  $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \dots, \mathbf{s}^{(p)}$  are not mutually orthogonal, the energy function (6.29b) does not necessarily assume a minimum at  $\mathbf{s}^{(m')}$ , nor is the vector  $\mathbf{s}^{(m')}$  always an equilibrium for the memory. To gain better insight into memory performance let us calculate the activation vector **net** in a more general case using expression (6.28a) without an assumption of orthogonality:

$$\mathbf{net} = n\mathbf{s}^{(m')} - p\mathbf{s}^{(m')} + \sum_{m \neq m'}^p (\mathbf{s}^{(m)} \mathbf{s}^{(m)\top}) \mathbf{s}^{(m')} \quad (6.30a)$$

This resulting activation vector can be viewed as consisting of an equilibrium state term  $(n - p)\mathbf{s}^{(m')}$  similar to (6.28b). In this case discussed before, either full statistical independence or orthogonality of the stored vectors was assumed. If none of these assumptions is valid, then the sum term in (6.30a) is also present in addition to the equilibrium term. The sum term can be viewed as a "noise" term vector  $\boldsymbol{\eta}$  which is computed as follows

$$\boldsymbol{\eta} = (\mathbf{W} - \mathbf{s}^{(m')}\mathbf{s}^{(m')\top} + \mathbf{I}) \mathbf{s}^{(m')} \quad (6.30b)$$

Expression (6.30b) allows for comparison of the noise terms relative to the equilibrium term at the input to each neuron. When the magnitude of the  $i$ 'th component of the noise vector is larger than  $(n - p)s_i^{(m')}$  and the term has the opposite sign, then  $s_i^{(m')}$  will not be the network's equilibrium. The noise term obviously increases for an increased number of stored patterns, and also becomes relatively significant when the factor  $(n - p)$  decreases.

As we can see from the preliminary study, the analysis of stable states of memory can become involved. In addition, firm conclusions are hard to derive unless statistical methods of memory evaluation are employed. The next section will focus on the performance analysis of autoassociative recurrent memories.

## 6.4

### PERFORMANCE ANALYSIS OF RECURRENT AUTOASSOCIATIVE MEMORY

In this section relationships will be presented that relate the size of the memory  $n$  to the number of distinct patterns that can be efficiently recovered. These also depend on the degree of similarity that the initializing key vector has to the closest stored vector and on the similarity between the stored patterns. We will look at example performance and capacity, as well as the fixed points of associative memories.

As mentioned, associative memories recall patterns that display a degree of "similarity" to the search argument. To measure this "similarity" precisely, the quantity called the *Hamming distance* (HD) is often used. Strictly speaking, the Hamming distance is proportional to the dissimilarity of vectors. It is defined as an integer equal to the number of bit positions differing between two binary vectors of the same length.

For two  $n$ -tuple bipolar binary vectors  $\mathbf{x}$  and  $\mathbf{y}$ , the Hamming distance is equal:

$$\text{HD}(\mathbf{x}, \mathbf{y}) \triangleq \frac{1}{2} \sum_{i=1}^n |x_i - y_i| \quad (6.31)$$

Obviously, the maximum HD value between any vectors is  $n$  and is the distance between a vector and its complement. Let us also notice that the asynchronous update allows for updating of the output vector by  $\text{HD} = 1$  at a time. The following example depicts some of the typical occurrences within the autoassociative memory and focuses on memory state transitions.

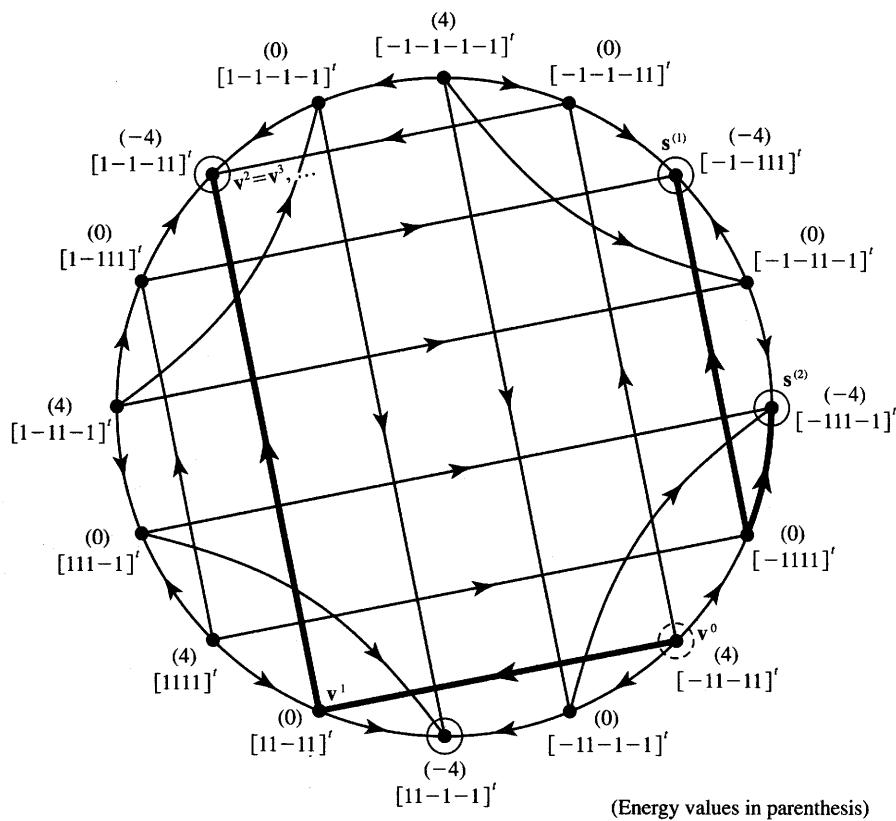
#### EXAMPLE 6.2

Let us look at the design and convergence properties of an autoassociative memory that stores the following two vectors  $\mathbf{s}^{(1)}$ ,  $\mathbf{s}^{(2)}$ :

$$\begin{aligned}\mathbf{s}^{(1)} &= [-1 \quad -1 \quad 1 \quad 1]^t \\ \mathbf{s}^{(2)} &= [-1 \quad 1 \quad 1 \quad -1]^t\end{aligned}$$

The weight matrix computed according to (6.20) results:

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & -2 \\ -2 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \end{bmatrix}$$



**Figure 6.11** Map of transitions for the Example 6.2.

The energy function computed from (6.19) is readily obtainable as

$$E(v) = 2(v_1v_3 + v_2v_4)$$

When the network is allowed to update, it evolves through certain states, which are shown on the state transition diagram in Figure 6.11. States are vertices of the four-dimensional cube. The figure shows all possible asynchronous transitions and their directions. Only transitions between states have been marked on the graph; the self-sustaining or stable state transitions have been omitted for clarity of the picture. Since only asynchronous transitions are allowed for autoassociative recurrent memory, every vertex is connected by an edge only to the neighboring vertex differing by a single bit. Thus, there are only four transition paths connected to every state. The paths indicate directions of possible transitions. As explained earlier, transitions

for the update rule (6.17) are toward lower energy values. The energy values for each state have been marked in parentheses at each vertex.

Let us review several sample transitions. Starting at  $v^0 = [-1 \ 1 \ -1 \ 1]^t$ , and with nodes updating asynchronously in ascending order, we have the state sequence

$$\begin{aligned}v^1 &= [1 \ 1 \ -1 \ 1]^t \\v^2 &= [1 \ -1 \ -1 \ 1]^t \\v^3 = v^4 = \dots &= [1 \ -1 \ -1 \ 1]^t\end{aligned}$$

as it has been marked on the state diagram. The actual convergence is toward a negative image of the stored pattern  $s^{(2)}$ . This result should have been expected. The initial pattern had no particular similarity to any of the stored patterns and apparently did not fall into any closeness category of patterns discussed in the previous section. In fact, the selected initializing pattern  $v^0$  has an HD value of 2 to each of the stored patterns. As such, the HD value is equal to half of the input vector dimension and the memory does not recover any of the stored vectors  $s^{(1)}$  or  $s^{(2)}$ .

Let us look at a regeneration of a key vector  $v^0 = [-1 \ 1 \ 1 \ 1]^t$  which is at HD = 1 to both  $s^{(1)}$  and  $s^{(2)}$ . The transition starting in ascending order leads to  $s^{(1)}$  in single step

$$\begin{aligned}v^1 &= v^0 \\v^2 &= [-1 \ -1 \ 1 \ 1]^t \\v^3 &= v^2 \\v^4 &= v^3, \quad \text{etc.}\end{aligned}$$

The reader can verify that by carrying out the transitions initialized at  $v^0$ , but carried out in descending node order, pattern  $s^{(2)}$  can be recovered. ■

The reader has certainly noticed that the example autoassociative memory just discussed has a number of problems, the most serious of them being uncertain recovery. Indeed, the memory in this example has been heavily overloaded since  $p/n$  is 50%. Overloaded memory by design does not provide error-free or efficient recovery of stored patterns. Another problem encountered in the memory is its unplanned stable states. Evaluation of similar examples of slightly larger memories would make it possible to find out that stable states exist that are not the stored memories. Such purposely not encoded but physically existent states of equilibrium are called spurious memories. *Spurious memories* are caused by the minima of the energy function that are additional to the ones we want. We could also find examples of convergence that are not toward the closest memory as measured with the HD value (see Problem P6.10). These are further drawbacks

of the original model since the memories may not all be fixed points. These issues become particularly important near or above the memory capacity.

### Energy Function Reduction

The energy function (6.19) of the autoassociative memory decreases during the memory recall phase. The dynamic updating process continues until a local energy minimum is found. Similar to continuous-time systems, the energy is minimized along the following gradient vector direction:

$$\nabla_v E(v) = -Wv \quad (6.32a)$$

As we will see below, the gradient (6.32a) is a linear function of the Hamming distance between  $v$  and each of the  $p$  stored memories (Petsche 1988). By substituting (6.20a) into the gradient expression (6.32a), it can be rearranged to the form

$$\nabla_v E(v) = - \sum_{m=1}^p s^{(m)} \left[ n - 2\text{HD}(s^{(m)}, v) \right] + pv \quad (6.32b)$$

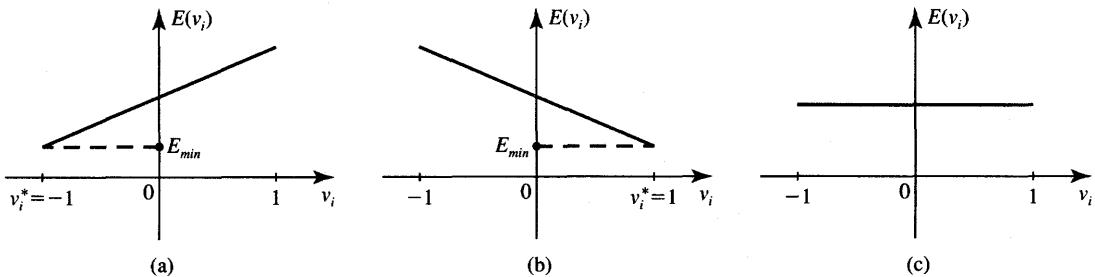
where the scalar product  $s^{(m)t}v$  has been replaced by the expression in brackets (see Appendix). The components of the gradient vector,  $\nabla_{v_i} E(v)$ , can be obtained directly from (6.32b) as

$$\frac{\partial E(v)}{\partial v_i} = -n \sum_{m=1}^p s_i^{(m)} + 2 \sum_{m=1}^p s_i^{(m)} \text{HD}(s^{(m)}, v) + pv_i \quad (6.32c)$$

Expression (6.32c) makes it possible to explain why it is difficult to recover patterns  $v$  at a large Hamming distance from any of the stored patterns  $s^{(m)}$ ,  $m = 1, 2, \dots, p$ .

When bit  $i$  of the output vector,  $v_i$ , is erroneous and equals  $-1$  and needs to be corrected to  $+1$ , the  $i$ 'th component of the energy gradient vector (6.32c) must be negative. This condition enables appropriate bit update while the energy function value would be reduced in this step. From (6.32c) we can notice, however, that any gradient component of the energy function is linearly dependent on  $\text{HD}(s^{(m)}, v)$ , for  $m = 1, 2, \dots, p$ . The larger the HD value, the more difficult it is to ascertain that the gradient component indeed remains negative due to the large potential contribution of the second sum term to the right side of expression (6.32c). Similar arguments against large HD values apply for correct update of bit  $v_i = 1$  toward  $-1$  which requires positive gradient component  $\partial E(v)/\partial v_i$ .

Let us characterize the local energy minimum  $v^*$  using the energy gradient component. For autoassociative memory discussed,  $v^*$  constitutes a local minimum of the energy function if and only if the condition holds that  $v_i^*(\partial E / \partial v_i)|_{v^*} < 0$  for all  $i = 1, 2, \dots, n$ . The energy function as in (6.19) can



**Figure 6.12** Three possible energy function cross sections along the  $v_i$  axis: (a) positive slope, minimum at  $-1$ , (b) negative slope, minimum at  $1$ , and (c) zero slope, no unique minimum.

be expressed as

$$E(\mathbf{v}) = -v_i \sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} v_j - \frac{1}{2} \sum_{k=1}^n \sum_{\substack{j=1 \\ j \neq k, j \neq i}}^n w_{kj} v_k v_j \quad (6.33a)$$

where the first term of (6.33a) is linear in  $v_i$  and the second term is constant. Therefore, the slope of  $E(v_i)$  is a constant that is positive, negative, or zero. This implies that one of the three conditions applies at the minimum  $v^*$

$$\left. \frac{\partial E}{\partial v_i} \right|_{v^*} > 0, \quad v_i^* = -1 \quad (6.33b)$$

$$\left. \frac{\partial E}{\partial v_i} \right|_{v^*} < 0, \quad v_i^* = 1 \quad (6.33c)$$

$$\left. \frac{\partial E}{\partial v_i} \right|_{v^*} = 0, \quad v_i^* = \pm 1 \quad (6.33d)$$

The three possible cases are illustrated in Figure 6.12. The energy function is minimized for  $v_i^* = -1$  (case a) or for  $v_i^* = 1$  (case b). Zero slope of the energy, or gradient component equal to zero (case c), implies no unique minimum at either  $+1$  or  $-1$ .

### Capacity of Autoassociative Recurrent Memory

One of the most important performance parameters of an associative memory is its capacity. Detailed studies of memory capacity have been reported in by McEliece et al. (1987) and Komlos and Paturi (1988). A state vector of the memory is considered to be *stable* if  $\mathbf{v}^{k+1} = \Gamma[\mathbf{W}\mathbf{v}^k]$  provided that  $\mathbf{v}^{k+1} = \mathbf{v}^k$ .

Note that the definition of stability is not affected by synchronous versus asynchronous transition mode; rather, the stability concept is independent from the transition mode.

A useful measure for memory capacity evaluation is the *radius of attraction*  $\rho$ , which is defined in terms of the distance  $\rho n$  from a stable state  $v$  such that every vector within the distance  $\rho n$  eventually reaches the stable state  $v$ . It is understood that the distance  $\rho n$  is convenient if measured as a Hamming distance and therefore is of integer value. For the reasons explained earlier in the chapter the radius of attraction for an autoassociative memory is somewhere between  $1/n$  and  $1/2$ , which corresponds to the distance of attraction between 1 and  $n/2$ .

For the system to function as a memory, we require that every stored memory  $s^{(m)}$  be stable. Somewhat less restrictive is the assumption that there is at least a stable state at a small distance  $\varepsilon n$  from the stored memory where  $\varepsilon$  is a positive number. In such a case it is then still reasonable to expect that the memory has an error correction capability. For example, when recovering the input key vector at a distance  $\rho n$  from stored memory, the stable state will be found at a distance  $\varepsilon n$  from it. Note that this may still be an acceptable output in situations when the system has learned too many vectors and the memory of each single vector is faded. Obviously, when  $\varepsilon = 0$ , the stored memory is stable within a radius of  $\rho$ .

The discussion above indicates that the error correction capability of an autoassociative memory can only be evaluated if stored vectors are not too close to each other. Therefore, each of the  $p$  distinct stored vectors used for a capacity study are usually selected at random. The asymptotic capacity of an autoassociative memory consisting of  $n$  neurons has been estimated in by McEliece et al. (1987) as

$$c = \frac{(1 - 2\rho)^2 n}{4 \ln n} \quad (6.34a)$$

When the number of stored patterns  $p$  is below the capacity  $c$  expressed as in (6.34a), then all of the stored memories, with probability near 1, will be stable. The formula determines the number of key vectors at a radius  $\rho$  from the stored memory that are correctly recallable to one of the stable, stored memories. The simple stability of the stored memories, with probability near 1, is ensured by the upper bound on the number  $p$  given as

$$c = \frac{n}{4 \ln n} \quad (6.34b)$$

For any radius between 0 and  $1/2$  of key vectors to the stored memory, almost all of the  $c$  stored memories are attractive when  $c$  is bounded as in (6.34b). If a small fraction of the stored memories can be tolerated as unrecoverable, and not stable, then the capacity boundary  $c$  can be considered twice as large compared to  $c$  computed from (6.34b). In summary, it is appropriate to state that regardless of the radius of attraction  $0 < \rho < 1/2$  the capacity of the Hopfield memory is

bounded as follows

$$\frac{n}{4 \ln n} < c < \frac{n}{2 \ln n} \quad (6.34c)$$

To offer a numerical example, the boundary values for a 100-neuron network computed from (6.34c) are about 5.4, with 10.8 memory vectors.

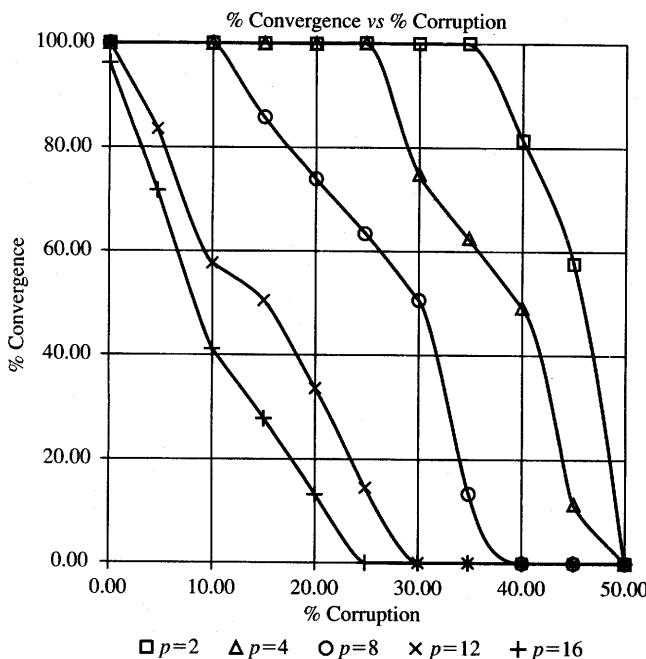
Assume that the number of stored patterns  $p$  is kept at the level  $\alpha n$ , for  $0 < \alpha < 1$ , and  $n$  is large. It has been shown that the memory still functions efficiently at capacity levels exceeding those stated in (6.34c) (Amit, Gutfreund, and Sompolinsky 1985). When  $\alpha \approx 0.14$ , stable states are found that are very close to the stored memories at a distance  $0.03n$ . As  $\alpha$  decreases to zero, this distance decreases as  $\exp(-(1/2)\alpha)$ . Hence, the memory retrieval is mostly accurate for  $p \leq 0.14n$ . A small percentage of error must be tolerated though if the memory operates at these upper capacity levels.

The study by McEliece et al. (1987) also reveals the presence of spurious fixed points, which are not stored memories. They tend to have rather small basins of attraction compared to the stored memories. Therefore, updates terminate in them if they start in their vicinity.

Although the number of distinct pattern vectors that can be stored and perfectly recalled in Hopfield's memory is not large, the network has found a number of practical applications. However, it is somewhat peculiar that the network can recover only  $c$  memories out of the total of  $2^n$  states available in the network as the cube corners of  $n$ -dimensional hypercube.

### Memory Convergence versus Corruption

To supplement the study of the original Hopfield autoassociative memory, it is worthwhile to look at the actual performance of an example memory. Of particular interest are the convergence rates versus memory parameters discussed earlier. Let us inspect the memory performance analysis curves shown in Figure 6.13 (Desai 1990). The memory performance on this figure has been evaluated for a network with  $n = 120$  neurons. As pointed out earlier in this section, the total number of stored patterns, their mutual Hamming distance and their Hamming distance to the key vector determine the success of recovery. Figure 6.13(a) shows the percentage of correct convergence as a function of key vector corruption compared to the stored memories. Computation shown is for a fixed HD between the vectors stored of value 45. It can be seen that the correct convergence rate drops about linearly with the amount of corruption of the key vector. The correct convergence rate also reduces as the number of stored patterns increases for a fixed distortion value of input key vectors. The network performs very well at  $p = 2$  patterns stored but recovers rather poorly distorted vectors at  $p = 16$  patterns stored.



#### Convergence vs. Corruption

Network Type: Hopfield Memory

Network Parameters: Dimension— $n$  (120)

Threshold—Hard Limited

@ Hamming Distance HD = 45

Curve Parameters: Patterns  $P$

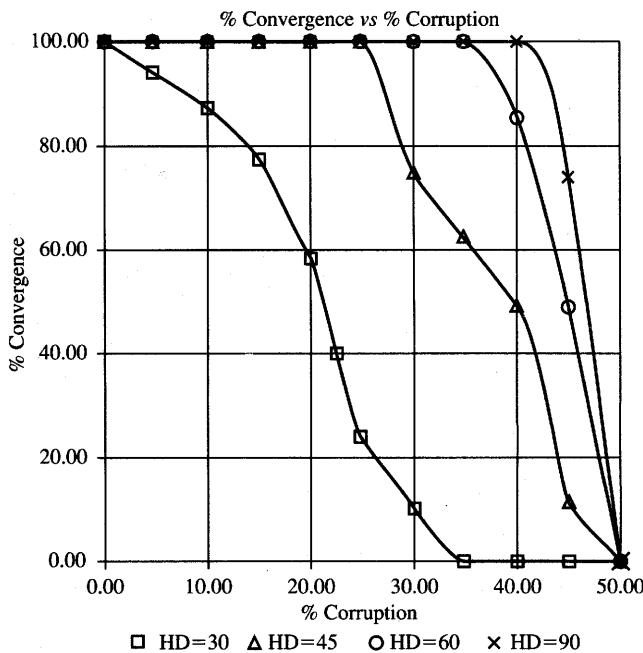
Features: 20 samples per point.

Comments: As can be seen from the curves the performance is of good quality for corruption levels up to 25% with a capacity of  $0.04 \times n$  only. The noise tolerance becomes poor as the number of patterns approaches the capacity of 18.

(a)

**Figure 6.13a** Memory convergence versus corruption of key vector: (a) for a different number of stored vectors, HD = 45.

Figure 6.13(b) shows the percentage of correct convergence events as a function of key vector corruption for a fixed number of stored patterns equal to four. The HD between the stored memories is a parameter for the family of curves shown on the figure. The network exhibits high noise immunity for large and very large Hamming distances between the stored vectors. A gradual



#### *Convergence vs. Corruption*

Network Type: Hopfield Memory

Network Parameters: Dimension— $n$  (120)  
Threshold—Hard Limited  
Patterns  $p = 4$

Curve Parameters: Hamming Distance

Features: 20 samples per point.

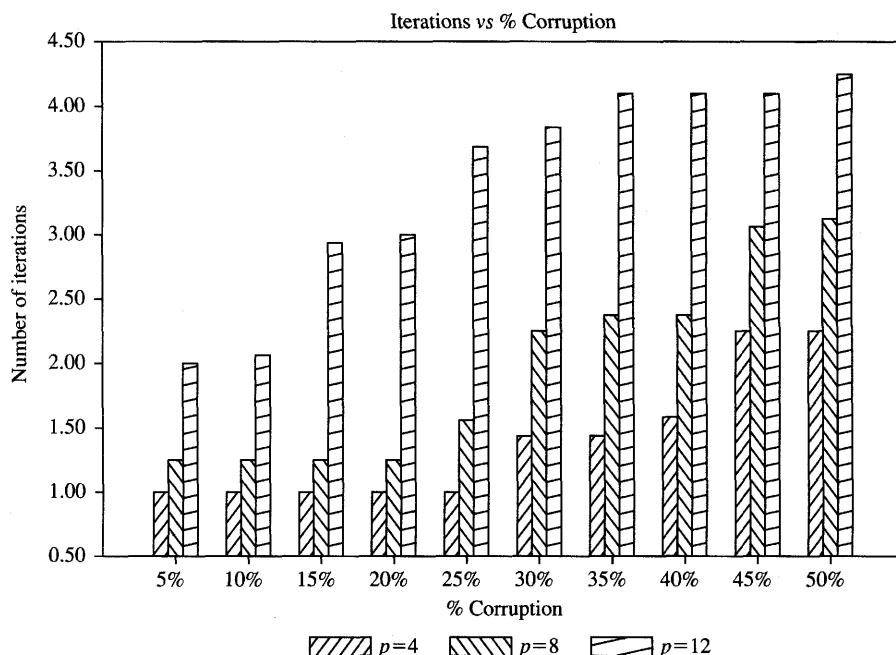
Comments: This network shows excellent performance and is extremely insensitive to noise for corruption levels as high as 35% at a Hamming Distance of 60 between the stored prototypes. An abrupt degradation in performance is observed for prototypes having more than three quarter of their bits in common.

(b)

**Figure 6.13b** Memory convergence versus corruption of key vector (*continued*): (b) for different HD values, four vectors stored.

degradation of initially excellent recovery can be seen as stored vectors become more overlapping. For stored vectors that have 75% of the bits in common, the recovery of correct memories is shown to be rather inefficient.

To determine how long it takes for the memory to suppress errors, the number of update cycles has also been evaluated for example recurrences for



*Iterations vs. Corruption*

Network Type: Hopfield Memory

Network Parameters: Dimension— $n$  (120)

Threshold—Hard Limited

@ Hamming Distance HD = 45

Curve Parameters: Number of Iterations

Features: 20 samples per point.

Comments: The number of iterations during retrieval is fairly low for corruption levels below 20%. It increases roughly in proportion to the number of patterns stored.

(c)

**Figure 6.13c** Memory convergence versus corruption of key vector (continued): (c) the number of sweep cycles for different corruption levels.

the discussed memory example. The update cycle is understood as a full sweep through all of the  $n$  neuron outputs. The average number of measured update cycles has been between 1 and 4 as illustrated in Figure 6.13(c). This number increases roughly linearly with the number of patterns stored and with the percent corruption of the key input vector.

### Fixed Point Concept

We now look in more detail at the convergence of recurrent autoassociative networks. Since the energy function  $E(v)$  of Equation (6.19) is bounded from below, the network evolves under the asynchronous dynamics toward  $E(v^{k+1})$  such that

$$E(v^{k+1}) \leq E(v^k) \quad (6.35a)$$

Since vectors  $v^{k+1}$  and  $v^k$  differ during the memory transitions by at most a single component (bit), the stabilization of  $E(v^{k+1})$  means that

$$E(v^{k+1}) = E(v^k) \quad \text{for } k > k_0 \quad (6.35b)$$

The transitions stop at the energy minimum, which also implies that

$$v^{k+1} = v^k \quad (6.35c)$$

Thus the network reaches its fixed point  $v^k$ , or stable state, at the energy minimum  $E(v^k)$ . We also know from the energy function study that the only attractors of the discussed network are its fixed points. Limit cycles such as  $B$  shown in Figure 6.4(c) are thus excluded in these type of networks.

It is instructive to devote more attention to the fixed-point concept, because it sheds more light on memory performance. Let us begin with a discussion of the one-dimensional recurrent system shown in Figure 6.14. For the output at  $t = k + 1$  we have

$$v^{k+1} = f(w, v^k) \quad (6.36a)$$

The fixed point is defined at  $v^*$  if the following relationship holds:

$$v^* = f(v^*) \quad (6.36b)$$

where it has been assumed that network parameter  $w$  in (6.36a) is constant. In geometrical terms the fixed point is found at the intersection of function  $f(v)$  and  $v$ . Whether or not this point is the solution of recursive Equation (6.36a) remains, however, an open question. In terms of the recursive formula (6.36a), the fixed point is said to be stable if

$$\lim_{k \rightarrow \infty} v^k = v^*, \quad \text{or} \quad (6.37a)$$

$$\lim_{k \rightarrow \infty} e^{k+1} = 0 \quad (6.37b)$$

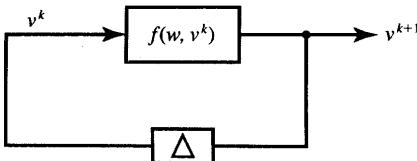


Figure 6.14 One-dimensional recurrent network.

where  $e$  is the recursion error defined as

$$e^{k+1} \triangleq |v^{k+1} - v^*|$$

The recursion error can be expressed using (6.36a) as

$$e^{k+1} = |f(v^k) - v^*| \quad (6.38a)$$

and further rearranged to the form

$$e^{k+1} = \left| \frac{f[v^* + (v^k - v^*)] - f(v^*)}{v^k - v^*} \right| e^k \quad (6.38b)$$

For a differentiable function  $f$  responsible for network's feedforward processing, this further reduces to

$$e^{k+1} \cong |f'(v^*)| e^k \quad (6.38c)$$

Now the condition (6.38c) translates to the form

$$|f'(v)| < 1 \quad (6.39)$$

which is the sufficient condition for the existence of a stable fixed point in the neighborhood where condition (6.39) holds. We have derived the condition for stable fixed-point existence in a simple one-dimensional case.

Let us look at an example of two systems and analyze the existence of stable fixed points. Figure 6.15(a) depicts a curve (Serra and Zanarini 1990) that serves as an example:

$$f(v) = w \left( 1 - 2 \left| \frac{1}{2} - v \right| \right) \quad (6.40)$$

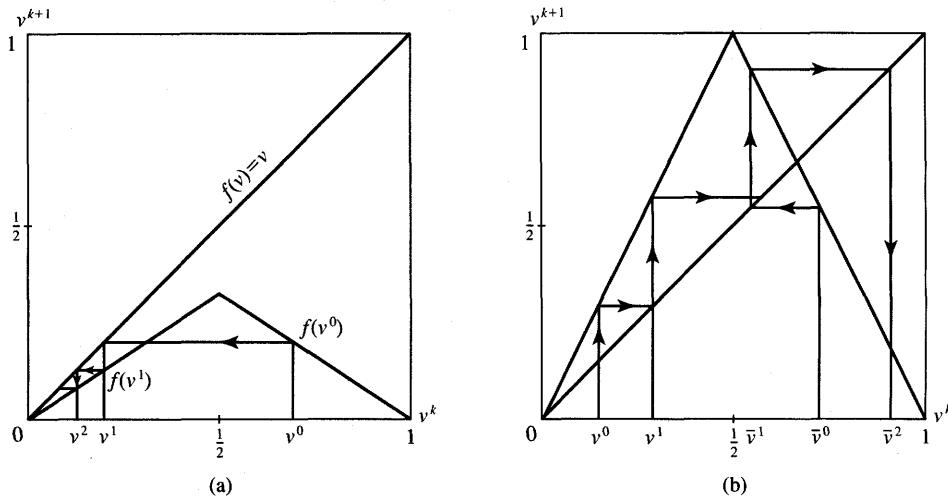
The curve is called a *triangular map* and is made for the parameter  $w = 1/3$ ,  $0 \leq v \leq 1$ . The function has only a single stable fixed point,  $v^* = 0$ . The several successive recursions shown indicate this clearly. Moreover, condition (6.39) is also fulfilled for the entire neighborhood  $0 \leq v \leq 1$ .

Another triangular map made for a different value of the triangular map parameter,  $w = 2$ , is shown in Figure 6.15(b). Two unstable fixed points are found for this case. They are at  $v = 0$  and  $v = 2/3$ . The fixed point instability can be noticed from displayed recursive movements of the solution starting at  $v^0$  and  $\bar{v}^0$ . The reader can verify that performing recursions

$$v^{k+1} = f(v^k) = 2 \left( 1 - 2 \left| \frac{1}{2} - v^k \right| \right)$$

starting either at  $v^0$  or at  $\bar{v}^0$ , the recursive updates are not converging to any of the solutions that clearly exist at  $v = 0$  and  $v = 2/3$ .

As we have seen, even a simple one-dimensional recurrent network exhibits interesting properties. The network from Figure 6.14 may be looked at as a single



**Figure 6.15** Illustration for the example one-dimensional recurrent network performance:  
(a) stable fixed point at 0 and (b) unstable fixed points at 0, and  $2/3$ .

computing node, which illustrates the discussion of the one-dimensional case just concluded. The recurrent progression of outputs in a multidimensional recurrent network is much more complex than shown. There are, however, fixed-point attractors in such memories similar to the ones discussed.

Figure 6.16 shows a single-neuron recurrent memory that has two stable fixed points at  $A$  and  $B$ , and one unstable stationary point at the origin. Although this network does have self-feedback, it operates with a continuous activation function, and thus it does not totally qualify as an associative memory by our criteria. However, it is useful for understanding fixed-point recursion mechanisms, which make stable or unstable neural networks. In limit case of high feedforward gain, the network shown in Figure 6.16 but using a hard-limiting activation function would stabilize at output +1 or -1. Note that the network from Figure 6.16(a) is a disguised bistable flip-flop from Figure 6.8(a). It employs a single non-inverting neuron rather than two inverting ones in cascade. Their operating principle concept, behavior and final outcome are the same.

### Modified Memory Convergent Toward Fixed Points

Recall from the previous section that in several basic cases of memory studies we have been able to show that  $\text{net} = cs^{(m')}$ , where  $c$  is a positive constant and  $s^{(m')}$  is one of the stored patterns. With respect to the discussion of

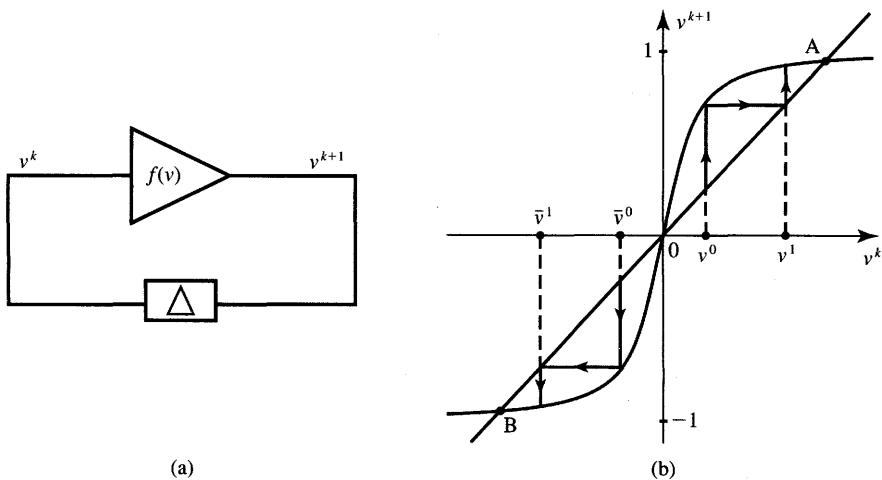


Figure 6.16 One-neuron recurrent network: (a) diagram and (b) fixed points.

stable fixed points, these cases could be understood as stable recurrent solutions for fixed points such that

$$\lim_{k \rightarrow \infty} \mathbf{v}^{k+1} = \mathbf{s}^{(m')} \quad (6.41)$$

Assume that only a single pattern  $\mathbf{s}^{(m')}$  has been stored in an autoassociative memory. The recurrent update of the  $i$ 'th node requires that

$$v_i^{k+1} = \sum_{j=1}^n s_i^{(m')} s_j^{(m')t} s_j^{(m')} - s_i^{(m')} \quad (6.42a)$$

and it reduces to

$$v_i^{k+1} = (n - 1)s_i^{(m')} \quad (6.42b)$$

Thus, the vector  $\mathbf{s}^{(m')}$  can be considered a stable fixed point in the memory.

When the network has been taught more than a single pattern, the patterns stored remain as fixed points, but a new stable state might also appear. In the case for which the number of stored patterns exceeds two, it is not even possible to guarantee that they, in fact, remain as fixed points (Serra and Zanarini 1990). The substantiation of the property is rather tedious, however, and is thus omitted.

The discussion below provides some additional insight into complex recurrent processes within the discussed class of memory networks. Realizing the limitations of the original model for autoassociative memory, the original Hebbian learning rule with the diagonal nullification used throughout Sections 6.3 and 6.4 can be modified. This modification results in a memory with improved properties. A memory recording method is presented below that makes each of the patterns an independent equilibrium point.

Observe that a pattern  $s^{(m')}$  is a fixed point if and only if the following inequality holds:

$$\left( \sum_{j=1}^n w_{ij} s_j^{(m')} \right) s_i^{(m')} \geq 0, \quad \text{for } i = 1, 2, \dots, n \quad (6.43)$$

An alternative design procedure for the memory can be devised based on (6.43). To implement the condition that each of the  $p$  patterns be a fixed point,  $n \times p$  inequalities such as (6.43) must be fulfilled. Note that (6.43) holds when the following condition holds:

$$W s^{(m)} = s^{(m)}, \quad \text{for } m = 1, 2, \dots, p \quad (6.44a)$$

Condition (6.44a) can be written compactly as a single matrix expression

$$W S = S \quad (6.44b)$$

where

$$S \triangleq [s^{(1)} \ s^{(2)} \ \dots \ s^{(p)}]$$

To accomplish the modified memory design, we are interested in finding the matrix  $W$  that fulfills (6.44b). The matrix equation (6.44b) is actually a system of  $np$  linear equations for solving  $n^2$  unknowns which are entries of the  $W$  matrix. Note that for memory that is not overloaded,  $p$  should always be smaller than  $n$ . The linear system (6.44b) is thus underdetermined, so there is a solution for  $W$  that approximates (6.44b). One of the possible solutions to (6.44) has been presented by Personnaz, Guyon, and Dreyfus (1986) and it is reviewed below. The resulting solution weight matrix is

$$W = S S^+ \quad (6.45)$$

where  $S^+$  is the Moore-Penrose pseudoinverse matrix (see Appendix). For linearly independent vectors  $s^m$ , the matrix  $S^+$  can be expressed as

$$S^+ \triangleq (S' S)^{-1} S'$$

It can be seen that since the matrix  $S$  is of size  $n \times p$ , the computation of  $W$  as in (6.45) involves the inversion of a  $p \times p$  matrix. The method presented is called the *projection learning rule* and it departs from the conventional Hebb's learning rule for weight matrix entries (Michel and Farrell 1990). Interestingly, formula (6.45) simplifies to the Hebbian learning rule for orthogonal vectors to be stored. Indeed, using the orthogonality condition we have

$$S' S = I \quad (6.46)$$

which yields from (6.45)

$$W = S S' \quad (6.47)$$

This result for the weight matrix exactly coincides with Hebbian learning rule

(6.14b). In conclusion, the projection learning rule offers an improvement over the Hebbian rule in the sense that each of the stored memories is a stable fixed point. Thus, the method guarantees that the stored vectors correspond to stable solutions. Note that both learning rules lead to symmetric matrices  $\mathbf{W}$ . The disadvantage of the projection learning rule is that the regions of attraction for each pattern become significantly smaller for a larger number of patterns  $p$  stored within a given size network. For  $p$  approaching  $n/2$ , the decrease of memory convergence quality is rather drastic.

### Advantages and Limitations

Theoretical considerations and examples of memory networks discussed in this chapter point out a number of advantages and limitations. As we have seen, recurrent associative memories, whether designed by the Hebbian learning rule or by a modified rule, suffer from substantial capacity limitations. Capacity limitation causes diversified symptoms. It can amount to convergence to spurious memories and difficulties with recovery of stored patterns if they are close to each other in the Hamming distance sense. Overloaded memory may not be able to recover data stored or may recall spurious outputs. Another inherent problem is the memory convergence to stored pattern complements.

In spite of all these deficiencies, the Hopfield network demonstrates the power of recurrent neural processing within a parallel architecture. The recurrences through the thresholding layer of processing neurons tend to eliminate gradually noise superimposed on the initializing input vector. This coerces the incorrect pattern bits toward one of the stored memories. The network's computational ability makes it possible to apply it in speech processing, database retrieval, image processing, pattern classification and other fields. The electronic implementations of Hopfield's associative memories will be discussed in Chapter 9.

## 6.5

### BIDIRECTIONAL ASSOCIATIVE MEMORY

*Bidirectional associative memory* is a heteroassociative, content-addressable memory consisting of two layers. It uses the forward and backward information flow to produce an associative search for stored stimulus-response association (Kosko 1987, 1988). Consider that stored in the memory are  $p$  vector association pairs known as

$$\left\{ \left( \mathbf{a}^{(1)}, \mathbf{b}^{(1)} \right), \left( \mathbf{a}^{(2)}, \mathbf{b}^{(2)} \right), \dots, \left( \mathbf{a}^{(p)}, \mathbf{b}^{(p)} \right) \right\} \quad (6.48)$$

When the memory neurons are activated, the network evolves to a stable state of two-pattern reverberation, each pattern at output of one layer. The stable reverberation corresponds to a local energy minimum. The network's dynamics involves two layers of interaction. Because the memory processes information in time and involves bidirectional data flow, it differs in principle from a linear associator, although both networks are used to store association pairs. It also differs from the recurrent autoassociative memory in its update mode.

### Memory Architecture

The basic diagram of the bidirectional associative memory is shown in Figure 6.17(a). Let us assume that an initializing vector  $\mathbf{b}$  is applied at the input to the layer  $A$  of neurons. The neurons are assumed to be bipolar binary. The input is processed through the linear connection layer and then through the bipolar threshold functions as follows:

$$\mathbf{a}' = \Gamma[\mathbf{W}\mathbf{b}] \quad (6.49a)$$

where  $\Gamma[\cdot]$  is a nonlinear operator defined in (6.5). This pass consists of matrix multiplication and a bipolar thresholding operation so that the  $i$ 'th output is

$$a'_i = \text{sgn} \left( \sum_{j=1}^m w_{ij} b_j \right), \quad \text{for } i = 1, 2, \dots, n \quad (6.49b)$$

Assume that the thresholding as in (6.49a) and (6.49b) is synchronous, and the vector  $\mathbf{a}'$  now feeds the layer  $B$  of neurons. It is now processed in layer  $B$  through similar matrix multiplication and bipolar thresholding but the processing now uses the transposed matrix  $\mathbf{W}^t$  of the layer  $B$ :

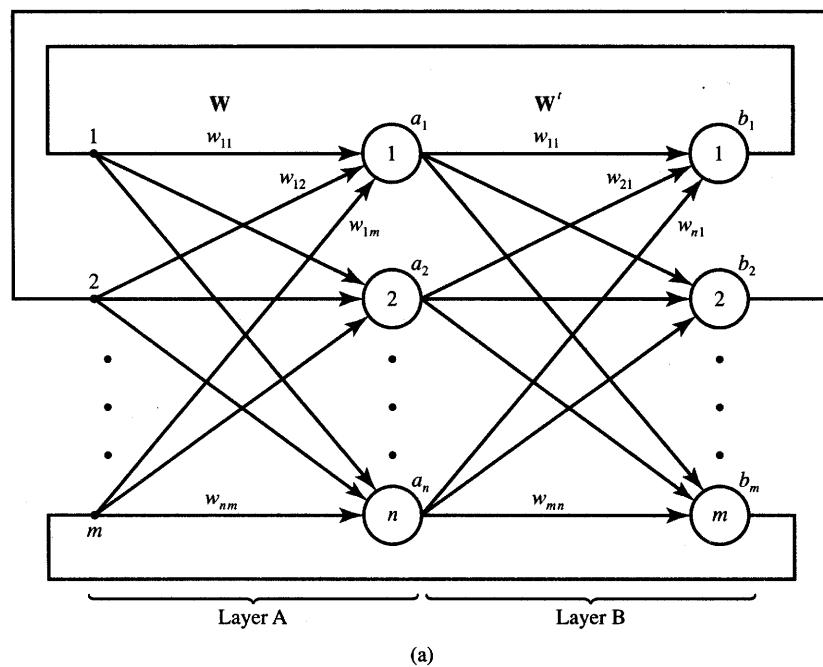
$$\mathbf{b}' = \Gamma[\mathbf{W}^t \mathbf{a}'] \quad (6.49c)$$

or for the  $j$ 'th output we have

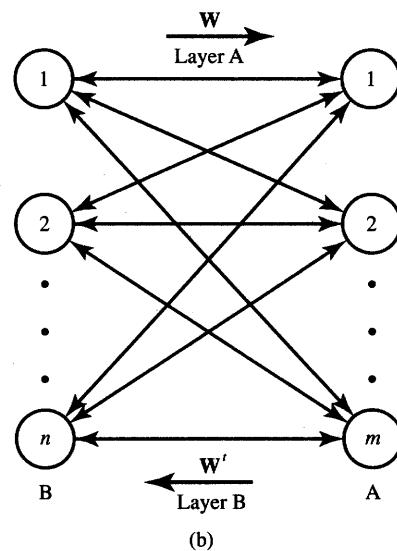
$$b'_j = \text{sgn} \left( \sum_{i=1}^n w_{ij} a'_i \right), \quad \text{for } j = 1, 2, \dots, m \quad (6.49d)$$

From now on the sequence of retrieval repeats as in (6.49a) or (6.49b) to compute  $\mathbf{a}''$ , then as in (6.49c) or (6.49d) to compute  $\mathbf{b}''$ , etc. The process continues until further updates of  $\mathbf{a}$  and  $\mathbf{b}$  stop. It can be seen that in terms of a recursive update mechanism, the retrieval consists of the following steps:

- |                              |  |
|------------------------------|--|
| <i>First Forward Pass:</i>   | $\mathbf{a}^1 = \Gamma[\mathbf{W}\mathbf{b}^0]$        |
| <i>First Backward Pass:</i>  | $\mathbf{b}^2 = \Gamma[\mathbf{W}^t \mathbf{a}^1]$     |
| <i>Second Forward Pass:</i>  | $\mathbf{a}^3 = \Gamma[\mathbf{W}\mathbf{b}^2]$        |
|                              | $\vdots$   |
| <i>k/2'th Backward Pass:</i> | $\mathbf{b}^k = \Gamma[\mathbf{W}^t \mathbf{a}^{k-1}]$ |



(a)



(b)

**Figure 6.17** Bidirectional associative memory: (a) general diagram and (b) simplified diagram.

Ideally, this back-and-forth flow of updated data quickly equilibrates usually in one of the fixed pairs  $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$  from (6.48). Let us consider in more detail the design of the memory that would achieve this aim.

Figure 6.17(b) shows the simplified diagram of the bidirectional associative memory often encountered in the literature. Layers  $A$  and  $B$  operate in an alternate fashion—first transferring the neurons' output signals toward the right by using matrix  $\mathbf{W}$ , and then toward the left by using matrix  $\mathbf{W}'$ , respectively.

The bidirectional associative memory maps bipolar binary vectors  $\mathbf{a} = [a_1 \ a_2 \ \dots \ a_n]^t$ ,  $a_i = \pm 1$ ,  $i = 1, 2, \dots, n$ , into vectors  $\mathbf{b} = [b_1 \ b_2 \ \dots \ b_m]^t$ ,  $b_i = \pm 1$ ,  $i = 1, 2, \dots, m$ , or vice versa. The mapping by the memory can also be performed for unipolar binary vectors. The input-output transformation is highly nonlinear due to the threshold-based state transitions.

For proper memory operation, the assumption needs to be made that no state changes are occurring in neurons of layers  $A$  and  $B$  at the same time. The data between layers must flow in a circular fashion:  $A \rightarrow B \rightarrow A$ , etc. The convergence of memory is proved by showing that either synchronous or asynchronous state changes of a layer decrease the energy. The energy value is reduced during a single update, however, only under the update rule (5.7). Because the energy of the memory is bounded from below, it will gravitate to fixed points. Since the stability of this type of memory is not affected by an asynchronous versus synchronous state update, it seems wise to assume synchronous operation. This will result in larger energy changes and, thus, will produce much faster convergence than asynchronous updates which are serial by nature and thus slow.

Figure 6.18 shows the diagram of discrete-time bidirectional associative memory. It reveals more functional details of the memory such as summing nodes, TLUs, unit delay elements, and it also introduces explicitly the index of recursion  $k$ . The figure also reveals a close relationship between the memory shown and the single-layer autoassociative memory discussed in Section 6.4. If the weight matrix is square and symmetric so that  $\mathbf{W} = \mathbf{W}'$ , then both memories become identical and autoassociative.

### Association Encoding and Decoding

The coding of information (6.48) into the bidirectional associative memory is done using the customary outer product rule, or by adding  $p$  cross-correlation matrices. The formula for the weight matrix is

$$\mathbf{W} = \sum_{i=1}^p \mathbf{a}^{(i)} \mathbf{b}^{(i)t} \quad (6.51a)$$

where  $\mathbf{a}^{(i)}$  and  $\mathbf{b}^{(i)}$  are bipolar binary vectors, which are members of the  $i$ 'th pair. As shown before in (6.8), (6.51a) is equivalent to the Hebbian learning rule

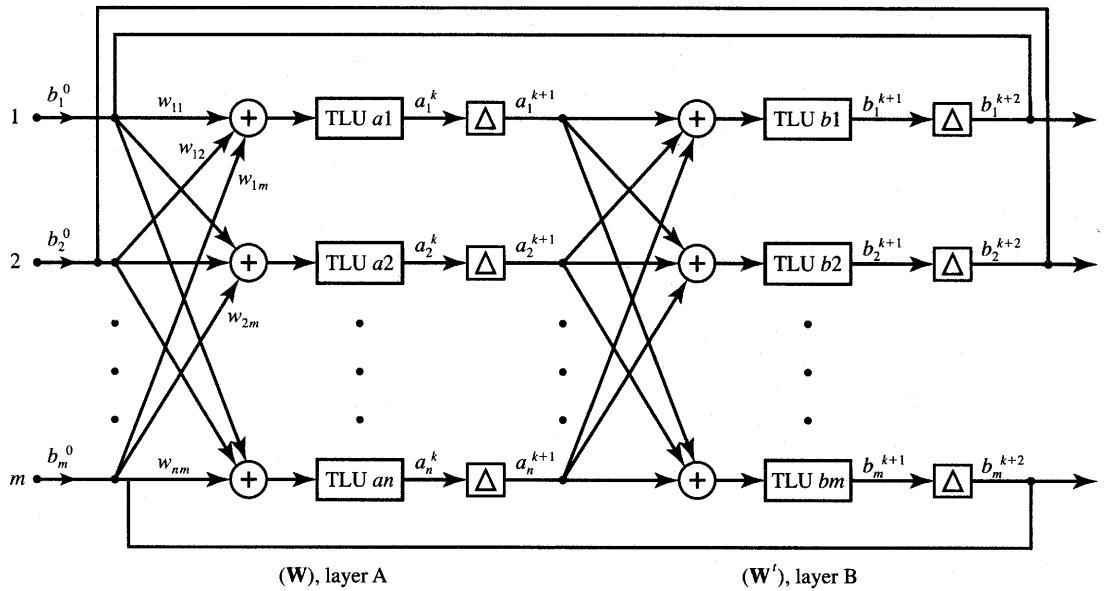


Figure 6.18 Discrete-time bidirectional associative memory expanded diagram.

yielding the following weight values:

$$w_{ij} = \sum_{m=1}^p a_i^{(m)} b_j^{(m)} \quad (6.51b)$$

Suppose one of the stored patterns,  $\mathbf{a}^{(m')}$ , is presented to the memory. The retrieval proceeds as follows from (6.49a)

$$\mathbf{b} = \Gamma \left[ \sum_{m=1}^p (\mathbf{b}^{(m)} \mathbf{a}^{(m)t}) \mathbf{a}^{(m')} \right] \quad (6.52a)$$

which further reduces to

$$\mathbf{b} = \Gamma \left[ n\mathbf{b}^{(m')} + \sum_{m \neq m'}^p \mathbf{b}^{(m)} \mathbf{a}^{(m)t} \mathbf{a}^{(m')} \right] \quad (6.52b)$$

The  $\mathbf{net}_b$  vector inside brackets in Equation (6.52b) contains a signal term  $n\mathbf{b}^{(m')}$  additive with the noise term  $\boldsymbol{\eta}$  of value

$$\boldsymbol{\eta} = \sum_{m \neq m'}^p \mathbf{b}^{(m)} (\mathbf{a}^{(m)t} \mathbf{a}^{(m')}) \quad (6.53)$$

Assuming temporarily the orthogonality of stored patterns  $\mathbf{a}^{(m)}$ , for  $m = 1, 2, \dots, p$ , the noise term  $\boldsymbol{\eta}$  reduces to zero. Therefore, immediate stabilization and exact association  $\mathbf{b} = \mathbf{b}^{(m')}$  occurs within only a single pass through layer B. If the input vector is a distorted version of pattern  $\mathbf{a}^{(m')}$ , the stabilization at  $\mathbf{b}^{(m')}$

is not imminent, however, and depends on many factors such as the HD between the key vector and prototype vectors, as well as on the orthogonality or HD between vectors  $\mathbf{b}^{(i)}$ , for  $i = 1, 2, \dots, p$ .

To gain better insight into the memory performance, let us look at the noise term  $\boldsymbol{\eta}$  as in (6.53) as a function of HD between the stored prototypes  $\mathbf{a}^{(m)}$ , for  $m = 1, 2, \dots, p$ . Note that two vectors containing  $\pm 1$  elements are orthogonal if and only if they differ in exactly  $n/2$  bits. Therefore,  $\text{HD}(\mathbf{a}^{(m)}, \mathbf{a}^{(m')}) = n/2$ , for  $m = 1, 2, \dots, p$ ,  $m \neq m'$ , then  $\boldsymbol{\eta} = 0$  and perfect retrieval in a single pass is guaranteed.

If  $\mathbf{a}^{(m)}$ , for  $m = 1, 2, \dots, p$ , and the input vector  $\mathbf{a}^{(m')}$  are somewhat similar so that  $\text{HD}(\mathbf{a}^{(m)}, \mathbf{a}^{(m')}) < n/2$ , for  $m = 1, 2, \dots, p$ ,  $m \neq m'$ , the scalar products in parentheses in Equation (6.53) tend to be positive, and a positive contribution to the entries of the noise vector  $\boldsymbol{\eta}$  is likely to occur. For this to hold, we need to assume the statistical independence of vectors  $\mathbf{b}^{(m)}$ , for  $m = 1, 2, \dots, p$ . Pattern  $\mathbf{b}^{(m')}$  thus tends to be positively amplified in proportion to the similarity between prototype patterns  $\mathbf{a}^{(m)}$  and  $\mathbf{a}^{(m')}$ . If the patterns are dissimilar rather than similar and the HD value is above  $n/2$ , then the negative contributions in parentheses in Equation (6.53) are negatively amplifying the pattern  $\mathbf{b}^{(m')}$ . Thus, a complement  $-\mathbf{b}^{(m')}$  may result under the conditions described.

### Stability Considerations

Let us look at the stability of updates within the bidirectional associative memory. As the updates in (6.50) continue and the memory comes to its equilibrium at the  $k$ 'th step, we have  $\mathbf{a}^k \rightarrow \mathbf{b}^{k+1} \rightarrow \mathbf{a}^{k+2}$ , and  $\mathbf{a}^{k+2} = \mathbf{a}^k$ . In such a case, the memory is said to be *bidirectionally stable*. This corresponds to the energy function reaching one of its minima after which any further decrease of its value is impossible. Let us propose the energy function for minimization by this system in transition as

$$E(\mathbf{a}, \mathbf{b}) \triangleq -\frac{1}{2}\mathbf{a}'\mathbf{W}\mathbf{b} - \frac{1}{2}\mathbf{b}'\mathbf{W}'\mathbf{a} \quad (6.54a)$$

The reader may easily verify that this expression reduces to

$$E(\mathbf{a}, \mathbf{b}) = -\mathbf{a}'\mathbf{W}\mathbf{b} \quad (6.54b)$$

Let us evaluate the energy changes during a single pattern recall. The summary of thresholding bit updates for the outputs of layer A can be obtained from (6.49b) as

$$\Delta a_i = \begin{cases} 2 & > 0 \\ 0 \text{ for } \sum_{j=1}^m w_{ij}b_j = 0 \\ -2 & < 0 \end{cases} \quad (6.55a)$$

and for the outputs of layer  $B$  they result from (6.49d) as

$$\Delta b_i = \begin{cases} 2 & > 0 \\ 0 & \text{for } \sum_{i=1}^n w_{ij} a_i = 0 \\ -2 & < 0 \end{cases} \quad (6.55b)$$

The gradients of energy (6.54b) with respect to  $\mathbf{a}$  and  $\mathbf{b}$  can be computed, respectively, as

$$\nabla_a E(\mathbf{a}, \mathbf{b}) = -\mathbf{W}\mathbf{b} \quad (6.56a)$$

$$\nabla_b E(\mathbf{a}, \mathbf{b}) = -\mathbf{W}^t \mathbf{a} \quad (6.56b)$$

The bitwise update expressions (6.55) translate into the following energy changes due to the single bit increments  $\Delta a_i$  and  $\Delta b_j$ :

$$\Delta E_{ai}(\mathbf{a}, \mathbf{b}) = - \left( \sum_{j=1}^m w_{ij} b_j \right) \Delta a_i, \quad \text{for } i = 1, 2, \dots, n \quad (6.57a)$$

$$\Delta E_{bj}(\mathbf{a}, \mathbf{b}) = - \left( \sum_{i=1}^n w_{ij} a_i \right) \Delta b_j, \quad \text{for } j = 1, 2, \dots, m \quad (6.57b)$$

Inspecting the right sides of Equations (6.57) and comparing them with the ordinary update rules as in (6.55) lead to the conclusion that  $\Delta E \leq 0$ . As with recurrent autoassociative memory, the energy changes are nonpositive. Since  $E$  is a bounded function from below according to the following inequality:

$$E(\mathbf{a}, \mathbf{b}) \geq - \sum_{i=1}^n \sum_{j=1}^m |w_{ij}| \quad (6.58)$$

then the memory converges to a stable point. The point is a local minimum of the energy function, and the memory is said to be bidirectionally stable. Moreover, no restrictions exist regarding the choice of matrix  $\mathbf{W}$ , so any arbitrary real  $n \times m$  matrix will result in bidirectionally stable memory. Let us also note that this discussion did not assume the asynchronous update for energy function minimization. In fact, the energy is minimized for either asynchronous or synchronous updates.

### Memory Example and Performance Evaluation

#### EXAMPLE 6.3

In this example we demonstrate coding and retrieval of pairs of patterns. The memory to be designed needs to store four pairs of associations. Four 16-pixel bit maps of letter characters need to be associated to 7-bit binary

vectors as shown in Figure 6.19(a). The respective training pairs of vectors  $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$ ,  $i = 1, 2, 3, 4$  are:

$$\begin{aligned}\mathbf{a}^{(1)} &= [1 \ 1 \ 1 \ -1 \ 1 \ -1 \ -1 \ -1 \ 1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1]^t, \\ \mathbf{b}^{(1)} &= [1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1]^t \\ \mathbf{a}^{(2)} &= [1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ 1 \ 1 \ -1 \ 1 \ 1 \ 1 \ -1 \ -1 \ 1]^t, \\ \mathbf{b}^{(2)} &= [1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1]^t \\ \mathbf{a}^{(3)} &= [1 \ 1 \ 1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1]^t, \\ \mathbf{b}^{(3)} &= [1 \ -1 \ 1 \ 1 \ -1 \ 1 \ -1]^t \\ \mathbf{a}^{(4)} &= [-1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ 1]^t, \\ \mathbf{b}^{(4)} &= [-1 \ 1 \ 1 \ -1 \ 1 \ -1 \ 1]^t\end{aligned}\tag{6.59}$$

The weight matrix can be obtained from (6.51) as

$$\mathbf{W} = \begin{bmatrix} 4. & -4. & -2. & 2. & -2. & 4. & -2. \\ 2. & -2. & 0. & 0. & -4. & 2. & 0. \\ 2. & -2. & 0. & 0. & -4. & 2. & 0. \\ 2. & -2. & 0. & 4. & 0. & 2. & -4. \\ 2. & -2. & -4. & 0. & 0. & 2. & 0. \\ 0. & 0. & -2. & 2. & 2. & 0. & -2. \\ 0. & 0. & 2. & 2. & -2. & 0. & -2. \\ -2. & 2. & 0. & 0. & 4. & -2. & 0. \\ 0. & 0. & -2. & -2. & 2. & 0. & 2. \\ -2. & 2. & 4. & 0. & 0. & -2. & 0. \\ -2. & 2. & 0. & 0. & 4. & -2. & 0. \\ -2. & 2. & 0. & 0. & 4. & -2. & 0. \\ 4. & -4. & -2. & 2. & -2. & 4. & -2. \\ 2. & -2. & 0. & 0. & -4. & 2. & 0. \\ 2. & -2. & 0. & 0. & -4. & 2. & 0. \\ 0. & 0. & 2. & 2. & 2. & 0. & -2. \end{bmatrix}\tag{6.60}$$

Assume now that the key vector  $\mathbf{a}^1$  at the memory input is a distorted prototype of  $\mathbf{a}^{(2)}$  as shown in Figure 6.19(b) so that  $HD(\mathbf{a}^{(1)}, \mathbf{a}^1) = 4$ .

$$\mathbf{a}^1 = [-1 \ -1 \ -1 \ 1 \ -1 \ 1 \ -1 \ 1 \ 1 \ -1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1]^t$$

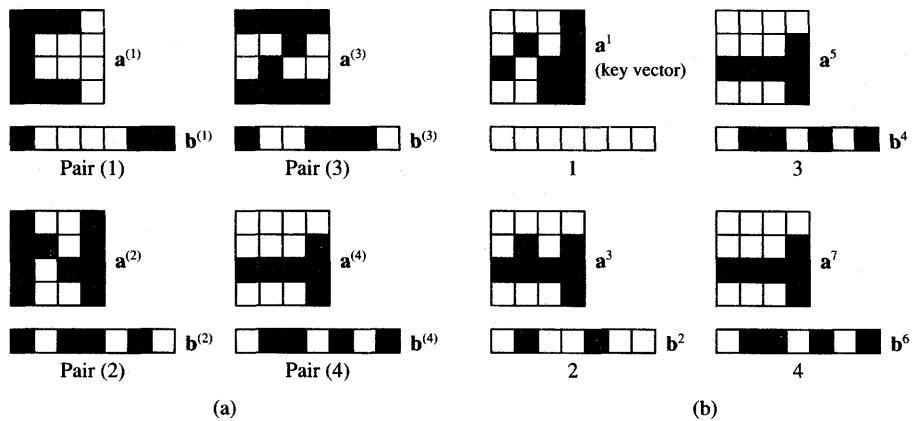
The memory response,  $\mathbf{b}_2$ , is computed from (6.50):

$$\mathbf{b}^2 = \Gamma [-16 \ 16 \ 0 \ 0 \ 32 \ -16 \ 0]^t$$

and the thresholding by the operator  $\Gamma$  yields

$$\mathbf{b}^2 = [-1 \ 1 \ 0 \ 0 \ 1 \ -1 \ 0]^t$$

Continuing the retrieval process until the bidirectional stability is reached



**Figure 6.19** Example of bidirectional associative memory: (a) associations stored and (b) four steps of retrieval of pair 4.

within this memory, the updates are computed of following value:

$$\mathbf{a}^3 = \Gamma [-14 -10 -10 -6 -6 2 -2 10 2 6 10 10 -14 -10 -10 2]^t$$

$$\mathbf{a}^3 = [-1 -1 -1 -1 -1 1 -1 1 1 1 1 1 -1 -1 1]^t$$

$$\mathbf{b}^4 = \Gamma [-28 28 8 -8 40 -28 8]^t$$

$$\mathbf{b}^4 = [-1 1 1 -1 1 -1 1]^t$$

$$\begin{aligned} \mathbf{a}^5 &= \Gamma [-20 -10 -10 -14 -10 -4 -4 \\ &\quad 10 4 10 10 -20 -10 -10 0]^t \end{aligned}$$

$$\mathbf{a}^5 = [-1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1]^t$$

$$\mathbf{b}^6 = \mathbf{b}^4$$

$$\mathbf{a}^7 = \mathbf{a}^5$$

The corresponding sequence of patterns is shown in Figure 6.19(b). The figure shows four first steps of retrieval. Interestingly,  $\mathbf{a}^1$  has converged to the prototype pair ( $\mathbf{a}^{(4)}$ ,  $\mathbf{b}^{(4)}$ ). Let us note, however, that the HD between  $\mathbf{a}^1$  and  $\mathbf{a}^{(4)}$  is

$$\text{HD}(\mathbf{a}^1, \mathbf{a}^{(4)}) = 4$$

We can also notice that other HD values of interest for our case are

$$\text{HD}(\mathbf{a}^1, \mathbf{a}^{(1)}) = 12$$

$$\text{HD}(\mathbf{a}^1, \mathbf{a}^{(2)}) = 4$$

$$\text{HD}(\mathbf{a}^1, \mathbf{a}^{(3)}) = 10$$

We, therefore, can conclude that the memory has converged to one of the two closest prototypes in the Hamming distance sense.

Kosko (1988) has shown that the upper limit on the number  $p$  of pattern pairs which can be stored and successfully retrieved is  $\min(n, m)$ . The substantiation for this estimate is rather heuristic. It can be statistically expected that if  $p > n$ , the noise term  $\eta$  of Equation (6.53) exceeds the signal term. Similarly, for processing of a signal and noise mixture by layer B, it is expected that the dominance of the signal term over the noise component is maintained for  $p < m$ . Hence, a rough and heuristic estimate on memory storage capacity is

$$p \leq \min(m, n) \quad (6.61a)$$

A more conservative heuristic capacity measure (6.61b) is also used in the literature

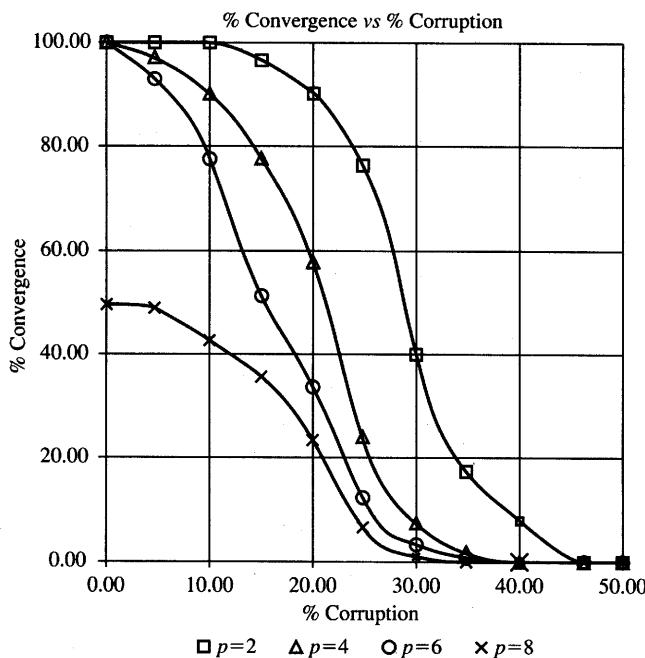
$$p = \sqrt{\min(m, n)} \quad (6.61b)$$

Like the autoassociative recurrent memory, the performance of bidirectional associative memory depends on the properties of the stored patterns, and its performance varies vastly for different prototype pairs. Figure 6.20 shows the simulated convergence versus corruption curves for a bidirectional associative memory with  $n = 25$  and  $m = 9$ . Randomly generated training patterns  $a^{(i)}$  had an average HD = 11 between them. The figure shows four curves for different storage levels. The results of the experiment indicate that the memory performance obviously decreases with the increase of  $p$ , and also with the increase in search argument distortion. But as can be seen from the figure for  $p = 8$ , the memory filled near its capacity routinely fails to identify or recall its true memories stored, even with the association argument undistorted.

In addition, the curves show that a memory of this size and under the conditions described is susceptible to noise in general. Indeed, excellent recall is shown for corruption levels limited to below 10%. This is relatively lower noise immunity than for the autoassociative recurrent memory with good recall properties as shown in Figures 6.12(a) and (b) for levels of distortion up to 25%. Although it is difficult to make comparisons between the two different memory architectures and different network sizes, the input vector noise sensitivity of the bidirectional associative memory makes it rather applicable for low-noise situations.

### Improved Coding of Memories

The generic bidirectional associative memory described in this section suffers from certain limitations of error-free retrieval of stored pairs. Let us see whether the memory performance can be improved by possibly reshaping its energy function. During the decoding, the energy (6.54b) remains stable or



#### Convergence vs. Corruption

Network Type: Bidirectional Associative Memory (BAM)

Network Parameters: Dimension— $n \times m$  (25 × 9)

Threshold—Hard Limited

@ Hamming Distance HD = 45

Curve Parameters: Pattern Pairs  $p$

Features: 20 samples per point.

Comments: The upper limit on the number of patterns that can be stored in the memory is  $\min(n, m)$ . This network is fairly insensitive to noise for corruption levels kept below 10% at half of its upper limit. Serious deterioration is observed at near full capacity where the networks fail to converge even to "true" memories.

**Figure 6.20** Convergence versus corruption in a BAM for a different number of stored association pairs  $p$ .

decreases until it reaches the minimum at  $(\mathbf{a}^f, \mathbf{b}^f)$ , which is of the following value:

$$E(\mathbf{a}, \mathbf{b}) = -\mathbf{a}^T \mathbf{W} \mathbf{b}^f \quad (6.62)$$

The energy  $E$  has a local minimum at  $(\mathbf{a}^f, \mathbf{b}^f)$  when none of the vectors at

$HD = 1$  from  $\mathbf{a}^f$  or  $\mathbf{b}^f$  will yield a lower energy value than (6.62). Consequently, when the energy (6.54b) for one of the trained  $p$  pairs (6.48) does not constitute a local minimum, then the association between a pair of stored vectors cannot be recalled even if the search argument is a member of a stored pair. This would mean in practice that there are unrecoverable stored associations [see Problem P6.20(a) for case studies]. The reason for this deficiency of bidirectional memory is that the coding formula (6.51) does not ensure that stored pairs correspond to local energy minima.

Indeed, inspection of the gradient of the energy function (6.56a) and (6.56b) reveals that at the  $m$ 'th association pair its gradient  $\nabla_a E$  is equal:

$$\nabla_a E \left( \mathbf{a}^{(m)}, \mathbf{b}^{(m)} \right) = - \sum_{i=1}^p \mathbf{a}^{(i)} \mathbf{b}^{(i)t} \mathbf{b}^{(m)} \quad (6.63a)$$

Note that the gradient (6.63a) is computed at the  $m$ 'th association pair, specifically at  $\mathbf{b}^{(m)}$ . This implies that if the  $k$ 'th bit  $a_k^{(m)}$  of the vector  $\mathbf{a}^{(m)}$  changes by  $\Delta a_k^{(m)}$ , the corresponding energy change  $\Delta E_{ak}$  is equal to

$$\Delta E_{ak} \left( \mathbf{a}^{(m)}, \mathbf{b}^{(m)} \right) = - \left( \sum_{i=1}^p \mathbf{a}^{(i)} \mathbf{b}^{(i)t} \mathbf{b}^{(m)} \right) \Delta a_k^{(m)} \quad (6.63b)$$

Assumption of a local minimum  $E(\mathbf{a}^{(m)}, \mathbf{b}^{(m)})$  now requires the proof that the expression in parentheses of (6.63b) and  $\Delta a_k^{(m)}$  be of different signs so that  $\Delta E_{ak}(\mathbf{a}^{(m)}, \mathbf{b}^{(m)})$  is positive. This condition, however, is not guaranteed, since the update value is  $\Delta a_k^{(m)} = \pm 2$ , and the sign of the expression in parenthesis does not alternate with the  $k$ 'th bit changing sign. Therefore, the retrieval of the association  $\mathbf{a}^{(m)}, \mathbf{b}^{(m)}$  is not guaranteed for the simple outer product encoding method as in (6.51).

To ensure that the local minima of energy (6.62) are  $E(\mathbf{a}^{(m)}, \mathbf{b}^{(m)})$ , the *multiple encoding strategy* has been proposed (Wang 1990). For multiple training of order  $q$  for the  $(\mathbf{a}^{(m)}, \mathbf{b}^{(m)})$  pair, one augments the matrix  $\mathbf{W}$  from (6.51) by an additional term

$$\Delta \mathbf{W} = (q - 1) \mathbf{a}^{(m)} \mathbf{b}^{(m)t} \quad (6.64)$$

This indicates that the pair is encoded  $q$  times rather than one time; thus its energy minimum is enhanced. The coding of the  $m$ 'th pair among the  $p$  existing pairs is thus enhanced through amplification of local minima  $E(\mathbf{a}^{(m)}, \mathbf{b}^{(m)})$ .

Using multiple coding (6.64) to produce a distinct energy minimum at  $(\mathbf{a}^{(m)}, \mathbf{b}^{(m)})$  however, may produce another pair, or pairs, of vectors in the training set that becomes nonrecoverable. This situation can be remedied by further multiple training. The multiple training method guarantees that eventually all training pairs will be recalled. Indeed, the weight matrix obtained using  $q$ -tuple training of the  $m$ 'th pair can be written as

$$\mathbf{W} = q \mathbf{a}^{(m)} \mathbf{b}^{(m)t} + \sum_{i \neq m}^p \mathbf{a}^{(i)} \mathbf{b}^{(i)t} \quad (6.65a)$$

For large  $q$ , the first term in (6.65a) dominates  $\mathbf{W}$  so

$$\mathbf{W} \cong q\mathbf{a}^{(m)}\mathbf{b}^{(m)t} \quad (6.65b)$$

Then the recall according to (6.50) proceeds for input  $\mathbf{b}^{(m)}$  as

$$\mathbf{a}^1 \cong q\Gamma \left[ \mathbf{a}^{(m)}(\mathbf{b}^{(m)t}\mathbf{b}^{(m)}) \right] \quad (6.66)$$

Since  $\mathbf{b}^{(m)t}\mathbf{b}^{(m)} = m > 0$ ,  $\mathbf{a}^{(m)}$  is perfectly recalled in this pass. A similar single-step recall would be guaranteed if  $\mathbf{a}^{(m)}$  were used as the search argument in the  $q$ -tuple training case of the  $m$ 'th pair.

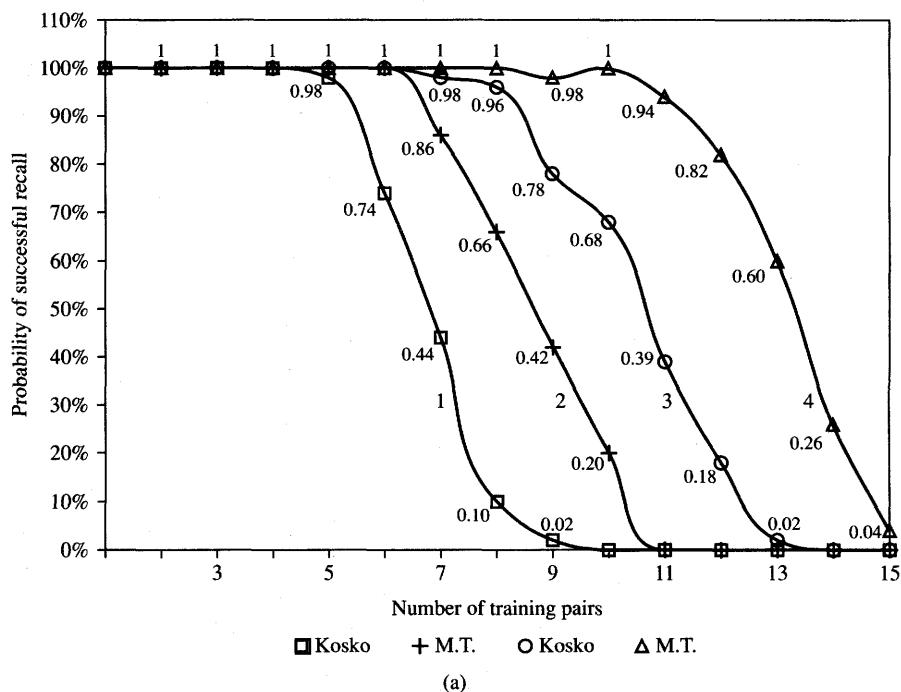
The multiple training method has been tested for randomly generated sizes of memory  $4 \leq n$ , and  $m \leq 256$  with a random  $p$  value such that  $2 \leq p \leq 16$ , and with limiting of the memory load to  $p \leq \sqrt{\min(n, m)}$  (Wang 1990). The simulation of memory performance resulted in 98.5% of correct recalls for the multiple training method versus 90% of correct recalls for the original bipolar associative memory recording algorithm. The multiple training method has also increased network capacity by 20 to 40%. Capacity notion is understood here as an estimate of the number of recallable training pairs  $p$  with a recall success of 90%. The number of training pairs for capacity simulation was selected at four to eight times larger than  $p$  to seek the canceling effects of the noise term.

The probability of correct convergence versus the number of stored pairs is shown in Figure 6.21(a) for  $n = m = 100$ . Curves 1 and 2 correspond to unipolar binary vectors used for training the bidirectional associative memory in the single training and multiple training modes, respectively. Curves 3 and 4 are for the respective training modes using bipolar binary vectors for recording. Although the multiplicity  $q$  of the training has not been explicitly stated in the literature, it is known that it has been kept above the value  $p(2/n)$ .

To guarantee the recall of all training pairs, a method of *dummy augmentation* of training pairs may be used (Wang 1990). The idea of dummy augmentation is to deepen the energy minima at  $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$ , for  $i = 1, 2, \dots, p$ , without adding any risk of false decoding, which normally increases as  $p$  increases. This can be accomplished by generating a noise-free training set obtained through the generation of augmented vectors containing dummy vector elements. The noise-free training set concept can be understood based on the noise expression (6.53) in terms of the Hamming distance between vectors. The pairs  $(\mathbf{a}^{(i)}, \mathbf{a}^{(j)})$  or  $(\mathbf{b}^{(i)}, \mathbf{b}^{(j)})$  are called noise-free if for  $i = 1, 2, \dots, p, j = 1, 2, \dots, p$

$$\begin{aligned} \text{HD} \left( \mathbf{a}^{(i)}, \mathbf{a}^{(j)} \right) &= \frac{n}{2} \\ \text{HD} \left( \mathbf{b}^{(i)}, \mathbf{b}^{(j)} \right) &= \frac{m}{2} \end{aligned} \quad (6.67)$$

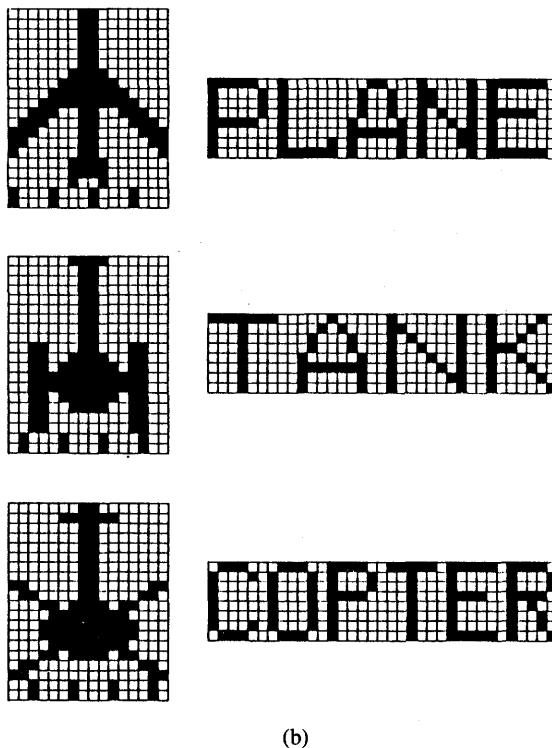
For a noise-free training set used to create the memory weight matrix  $\mathbf{W}$ , the recall of data would be immediate and error free. The training involving the dummy augmentation strategy, however, requires augmenting both vectors  $\mathbf{a}^{(i)}$



**Figure 6.21a** Illustration of improved coding strategies for bipolar heteroassociative memory: (a) convergence versus number of encoded pairs. (Adapted from Wang (1990). © IEEE; with permission.)

and  $\mathbf{b}^{(i)}$ , for  $i = 1, 2, \dots, p$ , with additional data (Wang 1990). An example of a noise-free training set is shown in Figure 6.21(b). Notice the dummy augmentation data present in the two bottom rows of the augmented training patterns  $\mathbf{a}^{(i)}$ ,  $i = 1, 2, 3$ . The bottom rows data do not belong to the original pattern. Their addition to the original training vectors makes the augmented vectors orthogonal according to (6.67). Thus, the dummy augmentation training method corresponds to artificial orthogonalization of stored vectors that were originally nonorthogonal.

The concepts of multiple training and dummy augmentation of training pairs offer significant advantages over the original recording algorithm for discrete-time bidirectional associative memory. They improve the percentage of correct recalls for cases in which the search arguments are both the members of the training set or their distorted counterparts. Dummy augmentation can be guaranteed to produce recall of all training pairs if attaching the dummy data to the training pairs is possible and allowable. One disadvantage though is that the memory size needs to be increased to handle the storage and retrieval of augmented vectors.

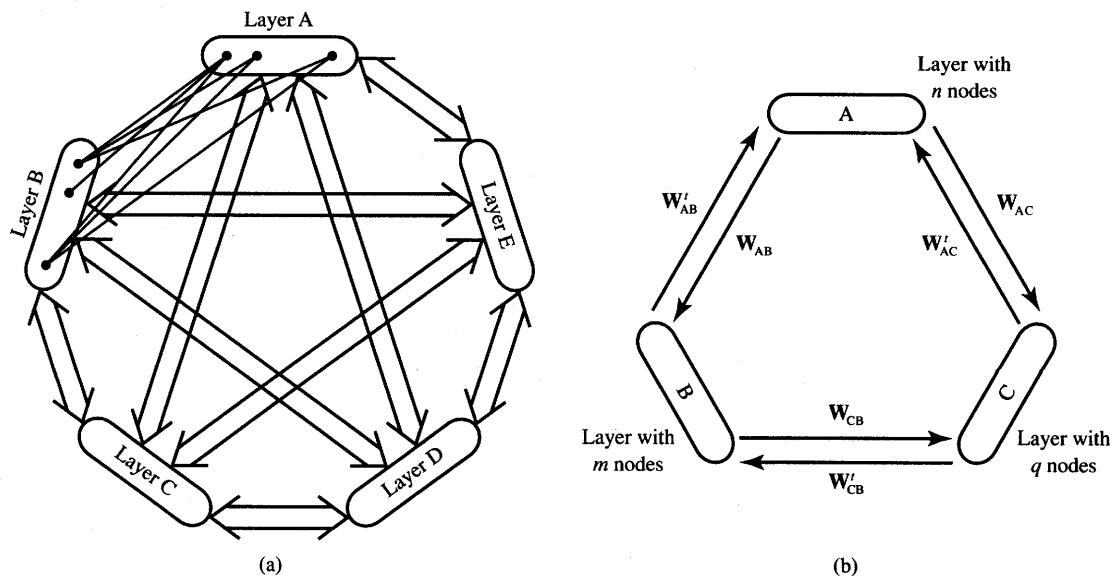


**Figure 6.21b** Illustration of improved coding strategies for bipolar heteroassociative memory (continued): (b) training pairs for dummy augmentation method. (Adapted from Wang (1990). © IEEE; with permission.)

### Multidirectional Associative Memory

Bidirectional associative memory is a two-layer nonlinear recurrent network that accomplishes a two-way associative search for stored stimulus-response associations  $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$ , for  $i = 1, 2, \dots, p$ . The bidirectional model can be generalized to enable multiple associations  $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)}, \mathbf{c}^{(i)}, \dots)$ ,  $i = 1, 2, \dots, p$ . The *multiple association memory* is called *multidirectional* (Hagiwara 1990) and is shown schematically in Figure 6.22(a) for the five-layer case. Layers are interconnected with each other by weights that pass information between them. When one or more layers are activated, the network quickly evolves to a stable state of multi-pattern reverberation. The reverberation which ends in a stable state corresponds to a local energy minimum.

The concept of the multidirectional associative memory will be illustrated with the three-layer network example shown in Figure 6.22(b). Let  $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)}, \mathbf{c}^{(i)})$ ,



**Figure 6.22** Multidirectional associative memory: (a) five-tuple association memory architecture and (b) information flow for triple association memory.

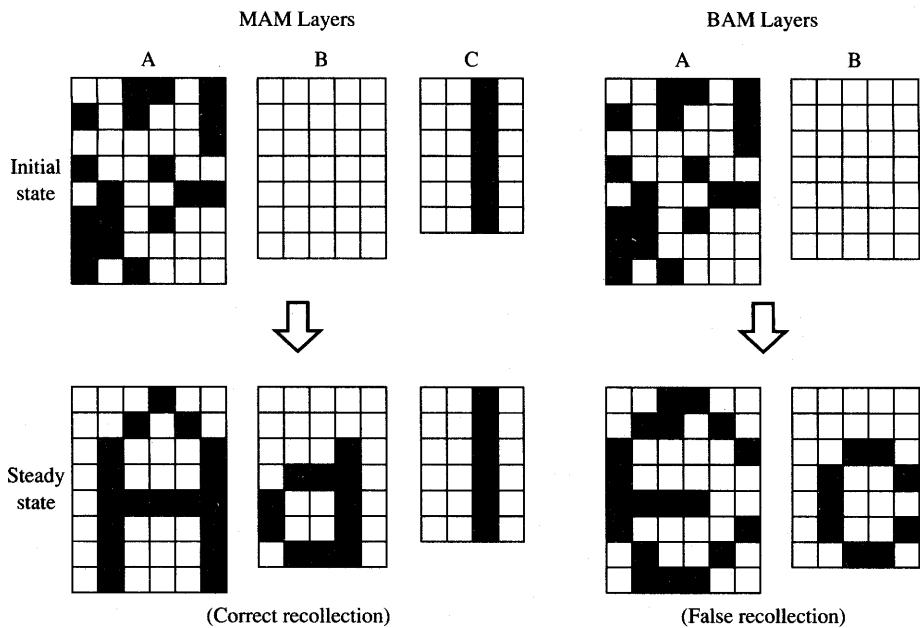
for  $i = 1, 2, \dots, p$ , be the bipolar vectors of associations to be stored. Generalization of formula (6.51a) yields the following weight matrices:

$$\begin{aligned} \mathbf{W}_{AB} &= \sum_{i=1}^p \mathbf{a}^{(i)} \mathbf{b}^{(i)T} \\ \mathbf{W}_{CB} &= \sum_{i=1}^p \mathbf{c}^{(i)} \mathbf{b}^{(i)T} \\ \mathbf{W}_{AC} &= \sum_{i=1}^p \mathbf{a}^{(i)} \mathbf{c}^{(i)T} \end{aligned} \quad (6.68)$$

where the first and second subscript of matrices denote the destination and source layer, respectively. With the associations encoded as in (6.68) in directions  $B \rightarrow A$ ,  $B \rightarrow C$ ,  $C \rightarrow A$ , and reverse direction associations obtained through the respective weight matrix transposition, the recall proceeds as follows: Each neuron independently and synchronously updates its output based on its total input sum from all other layers:

$$\begin{aligned} \mathbf{a}' &= \Gamma [\mathbf{W}_{AB} \mathbf{b} + \mathbf{W}_{AC} \mathbf{c}] \\ \mathbf{b}' &= \Gamma [\mathbf{W}_{CB}^T \mathbf{c} + \mathbf{W}_{AB}^T \mathbf{a}] \\ \mathbf{c}' &= \Gamma [\mathbf{W}_{CB} \mathbf{b} + \mathbf{W}_{AC}^T \mathbf{c}] \end{aligned} \quad (6.69)$$

The neurons' states change synchronously according to (6.69) until a multidirectionally stable state is reached.



**Figure 6.23** Synchronous MAM and BAM example. (Adapted from Hagiwara (1990). © IEEE; with permission.)

Figure 6.23 displays snapshots of the synchronous convergence of three- and two-layer memories. The bit map of the originally stored letter A has been corrupted with a probability of 44% to check the recovery. With the initial input as shown, the two-layer memory does not converge correctly. The three-directional memory using additional input to layer C recalls the character perfectly as a result of a multiple association effect. This happens as a result of the joint interaction of layers A and B onto layer C. Therefore, additional associations enable better noise suppression. In the context of this conclusion, note also that the bidirectional associative memory is a special, two-dimensional case of the multidirectional network.

## 6.6

### ASSOCIATIVE MEMORY OF SPATIO-TEMPORAL PATTERNS

The bidirectional associative memory concept can be used not only for storing  $p$  spatial patterns in the form of equilibria encoded in the weight matrix;

it can also be used for storing sequences of patterns in the form of dynamic state transitions. Such patterns are called *temporal* and they can be represented as an ordered set of vectors or functions. We assume all temporal patterns are bipolar binary vectors given by the ordered set, or sequence,  $S$  containing  $p$  vectors:

$$S \triangleq \{s^{(1)}, s^{(2)}, \dots, s^{(p)}\} \quad (6.70)$$

where column vectors  $s^{(i)}$ , for  $i = 1, 2, \dots, p$ , are  $n$ -dimensional. The neural network is capable of memorizing the sequence  $S$  in its dynamic state transitions such that the recalled sequence is

$$s^{(i+1)} = \Gamma[W s^{(i)}] \quad (6.71)$$

where  $\Gamma$  is the nonlinear operator as in (6.5) and the superscript summation is computed modulo  $p + 1$ . Starting at the initial state of  $x(0)$  in the neighborhood of  $s^{(i)}$ , the sequence  $S$  is recalled as a cycle of state transitions. This model was proposed in Amari (1972) and its behavior was mathematically analyzed. The memory model discussed in this section can be briefly called *temporal associative memory*.

To encode a sequence such that  $s^{(1)}$  is associated with  $s^{(2)}$ ,  $s^{(2)}$  with  $s^{(3)}$ , ..., and  $s^{(p)}$  with  $s^{(1)}$ , encoding can use the cross-correlation matrices  $s^{(i+1)} s^{(i)t}$ . Since the pair of vectors  $s^{(i)}$  and  $s^{(i+1)}$  can be treated as heteroassociative, the bidirectional associative memory can be employed to perform the desired association. The sequence encoding algorithm for temporal associative memory can thus be formulated as a sum of  $p$  outer products as follows

$$W = \sum_{i=1}^{p-1} s^{(i+1)} s^{(i)t} + s^{(1)} s^{(p)t}, \quad \text{or} \quad (6.72a)$$

$$W = \sum_{i=1}^p s^{(i+1)} s^{(i)t} \quad (6.72b)$$

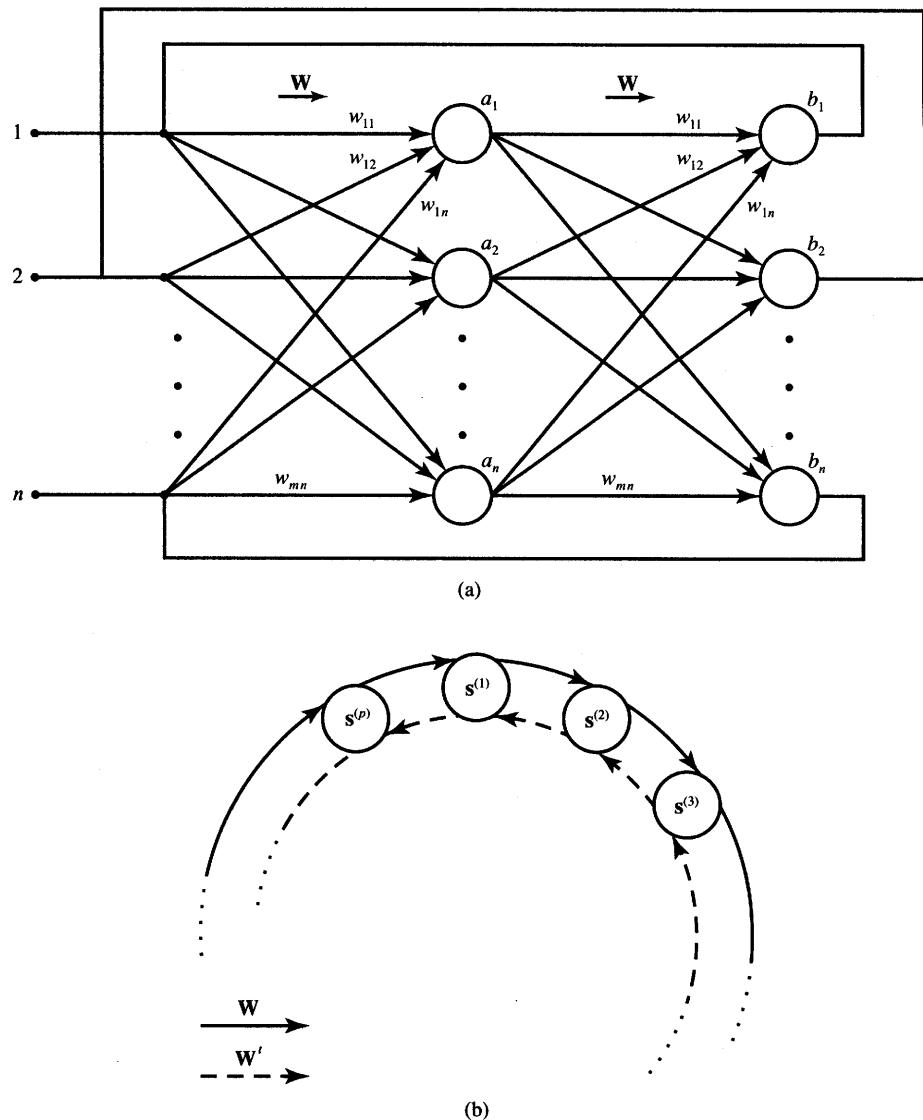
where the superscript summation in (6.72b) is modulo  $p + 1$ . Note that if the unipolar vectors  $s^{(i)}$  are to be encoded, they must first be converted to bipolar binary vectors to create correlation matrices as in (6.72), as has been the case for regular bidirectional memories encoding.

A diagram of the temporal associative memory is shown in Figure 6.24(a). The network is a two-layer bidirectional associative memory modified in such a way that both layers  $A$  and  $B$  are now described by identical weight matrices  $W$ . We thus have recall formulas

$$a = \Gamma[W b] \quad (6.73a)$$

$$b = \Gamma[W a] \quad (6.73b)$$

where it is understood that layers  $A$  and  $B$  update nonsimultaneously and in an alternate circular fashion. To check the proper recall of the stored sequence,



**Figure 6.24** Temporal associative memory: (a) diagram and (b) pattern recall sequences (forward and backward).

vector  $s^{(k)}$ ,  $k = 1, 2, \dots, p$ , is applied to the input of the layer  $A$  as in (6.73a). We thus have

$$\mathbf{a} = \Gamma \left[ \left( s^{(2)} s^{(1)t} + \dots + s^{(k+1)} s^{(k)t} + \dots + s^{(1)} s^{(p)t} \right) s^{(k)} \right] \quad (6.74)$$

The vector  $\text{net}_a$  in brackets of Equation (6.74) contains a signal term  $ns^{(k+1)}$  and the remainder, which is the noise term  $\eta$

$$\eta = \sum_{i \neq k}^p s^{(i+1)} (s^{(i)t} s^{(k)}) \quad (6.75)$$

where the superscript summation is modulo  $p + 1$ . Assuming the orthogonality of the vectors within the sequence  $S$ , the noise term is exactly zero and the thresholding operation on vector  $ns^{(k+1)}$  results in  $s^{(k+1)}$  being the retrieved vector. Therefore, immediate stabilization and exact association of the appropriate member vector of the sequence occurs within a single pass within layer A. Similarly, vector  $s^{(k+1)}$  at the input to layer B will result in recall of  $s^{(k+2)}$ . The reader may verify this using (6.73b) and (6.72). Thus, input of any member of the sequence set  $S$ , say  $s^{(k)}$ , results in the desired circular recalls as follows:  $s^{(k+1)} \rightarrow s^{(k+2)} \rightarrow \dots \rightarrow s^{(p)} \rightarrow s^{(1)} \rightarrow \dots$ . This is illustrated in Figure 6.24(b), which shows the forward recall sequence.

The reader may easily notice that reverse order recall can be implemented using the transposed weight matrices in both layers A and B. Indeed, transposing (6.72b) yields

$$W^t = \sum_{i=1}^p s^{(i)} s^{(i+1)t} \quad (6.76)$$

When the signal term due to the input  $s^{(k)}$  is  $ns^{(k-1)}$ , the recall of  $s^{(k-1)}$  will follow.

Obviously, if the vectors of sequence  $S$  are not mutually orthogonal, the noise term  $\eta$  may not vanish, even after thresholding. Still, for vectors stored at a distance  $HD \ll n$ , the thresholding operation in layer A or B should be expected to result in recall of the correct sequence.

This type of memory will undergo the same limitations and capacity bounds as the bidirectional associative memory. The storage capacity of the temporal associative memory can be estimated using expression (6.61a). Thus, we have the maximum length sequence to be bounded according to the condition  $p < n$ . More generally, the memory can be used to store  $k$  sequences of length  $p_1, p_2, \dots, p_k$ . Together they include:

$$p = \sum_{i=1}^k p_i \quad (6.77)$$

patterns. In such cases, the total number of patterns as in (6.77) should be kept below the  $n$  value.

The temporal associative memory operates in a synchronous serial fashion similar to a single synchronous update step of a bidirectional associative memory. The stability of the memory can be proven by generalizing the theory of stability of the bidirectional associative memory. The temporal memory energy function is defined as

$$E = - \sum_{i=1}^p s^{(i+1)} W s^{(i)} \quad (6.78)$$

Calculation of the energy increment due to changes of  $\mathbf{s}^{(k)}$  produces the following equation:

$$E = -\mathbf{s}^{(k+1)} \mathbf{W} \mathbf{s}^{(k)} - \mathbf{s}^{(k)} \mathbf{W} \mathbf{s}^{(k-1)} - \sum_{\substack{i=k \\ i \neq k-1}}^p \mathbf{s}^{(i+1)} \mathbf{W} \mathbf{s}^{(i)} \quad (6.79)$$

The gradient of energy with respect to  $\mathbf{s}^k$  becomes

$$\nabla_s E = -\mathbf{W}^t \mathbf{s}^{(k+1)} - \mathbf{W} \mathbf{s}^{(k-1)} \quad (6.80)$$

Considering bitwise updates due to increments  $\Delta s_i^{(k)}$  we obtain

$$\Delta_s E_i = - \left( \sum_{j=1}^n w_{ji} s_j^{(k+1)} + \sum_{i=1}^n w_{ij} s_j^{(k-1)} \right) \Delta s_i^{(k)} \quad (6.81)$$

Each of the two sums in parentheses in Equation (6.81) agree in sign with  $\Delta s_i^{(k)}$  under the sgn ( $net_i$ ) update rule. The second sum corresponds to  $net_i$  due to the input  $\mathbf{s}^{(k-1)}$ , which retrieves  $\mathbf{s}^{(k)}$  in the forward direction. The first sum corresponds to  $net_i$  due to the input  $\mathbf{s}^{(k+1)}$ , which again retrieves  $\mathbf{s}^{(k)}$  in the reverse direction. Thus, the energy increments are negative during the temporal sequence retrieval  $\mathbf{s}^{(1)} \rightarrow \mathbf{s}^{(2)} \rightarrow \dots \rightarrow \mathbf{s}^{(p)}$ . As shown by Kosko (1988), the energy increases stepwise, however, at the transition  $\mathbf{s}^{(p)} \rightarrow \mathbf{s}^{(1)}$ , and then it continues to decrease within the complete sequence of  $p - 1$  retrievals to follow.

## EXAMPLE 6.4

In this example temporal associative memory is designed for the vector sequence consisting of vectors  $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \mathbf{s}^{(3)}$  arranged in sequence  $S$ :

$$S = \{[1 \ -1 \ -1 \ 1 \ -1]^t, [1 \ 1 \ -1 \ -1 \ 1]^t, [1 \ -1 \ 1 \ -1 \ 1]^t\} \quad (6.82)$$

The memory weight matrix  $\mathbf{W}$  is obtained from (6.72) as

$$\mathbf{W} = \mathbf{s}^{(2)} \mathbf{s}^{(1)t} + \mathbf{s}^{(3)} \mathbf{s}^{(2)t} + \mathbf{s}^{(1)} \mathbf{s}^{(3)t} \quad (6.83a)$$

Substituting the temporal vectors specified, the weight matrix (6.83) becomes

$$\mathbf{W} = \begin{bmatrix} 3 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & 3 & -3 \\ -1 & 3 & -1 & -1 & 1 \\ -1 & -1 & 3 & -1 & 1 \\ 1 & 1 & -3 & 1 & -1 \end{bmatrix} \quad (6.83b)$$

Now evaluate recall of the encoded sequence  $S$ . With input  $\mathbf{s}^{(1)}$ , the layer response is

$$\Gamma[\mathbf{W} \mathbf{s}^{(1)}] = \Gamma [3 \ 7 \ -5 \ -5 \ 5]^t \quad (6.84)$$

which yields the desired vector  $s^{(2)}$  after thresholding. It is interesting to note from (6.74) and (6.75) that the noise vector  $\eta_1$  for the computed transition is

$$\eta_1 = \begin{bmatrix} 3 \\ 7 \\ -5 \\ -5 \\ 5 \end{bmatrix} - 5s^{(2)} \quad (6.85a)$$

which reduces to

$$\eta_1 = [-2 \ 2 \ 0 \ 0 \ 0]^t \quad (6.85b)$$

The magnitude of each component of  $\eta_1$  is smaller than the magnitude of the corresponding component of  $5s^{(2)}$ ; thus, recall is correct in this step due to the sufficient value of the signal-to-noise ratio. Further updates of the memory yield

$$\Gamma[Ws^{(2)}] = \Gamma[5 \ -7 \ 5 \ -3 \ 3]^t \quad (6.86)$$

and thus recall of  $s^{(3)}$  as required, with the noise term

$$\eta_2 = [0 \ -2 \ 0 \ 2 \ -2]^t \quad (6.87)$$

Finally,  $s^{(1)}$  is recalled due to the thresholding operation

$$\Gamma[Ws^{(3)}] = \Gamma[5 \ -7 \ -3 \ 5 \ -5]^t \quad (6.88)$$

with the noise vector

$$\eta_3 = [0 \ 2 \ 2 \ 0 \ 0]^t \quad (6.89)$$

This terminates the recall of the sequence  $s^{(2)}, s^{(3)}, s^{(1)}$ . ■

## 6.7

### CONCLUDING REMARKS

The associative memories studied in this chapter are a class of artificial neural networks capable of implementing complex associative mapping in a space of vector information. The domain of such mapping is a set of memory vectors, usually having binary values. Recording, or learning, algorithms are used to encode the required mapping between the memory input and output. In addition to coded memories, however, stable false attractors may often result as a side effect of the coding algorithm.

Interesting observations can be drawn from the comparison of the artificial neural memory architectures just studied to the capacity of the human brain. Biological studies have estimated the number of neurons in a human cortex to

be of order of  $10^{11}$  (Amit 1989). According to our conclusions in Section 6.4, a fully connected cortex with  $10^{11}$  synapses could store  $\alpha 10^{11}$  patterns, where  $\alpha$  is the proportionality constant having a lower bound on its estimated value of 0.14. Each of the patterns can be understood at a first glance as a  $10^{11}$ -tuple pattern not correlated with any other pattern in the cortex. This results in a number of  $\alpha 10^{22}$  bits of storage space apparently available in the brain. As tentative as this rough estimate is, just its order of magnitude indicates an information pressure to which any brain model relates.

But since the nerve network of the cortex is not fully connected, the above assumption does not hold. According to neurobiological research estimates, each neuron has only about  $10^4$  synapses. The cortex can thus be viewed as a system of  $10^7$  elementary networks each comprising  $10^4$  fully connected neurons. Such a system can store and retrieve  $\alpha 10^4$  patterns of  $10^4$  bits each. Thus, the total informational capacity is  $\alpha 10^7 \times 10^4 \times 10^4$ , which is equal to  $\alpha 10^{15}$  bits. A similar order of magnitude capacity results if we consider a network of  $10^{11}$  neurons to be a uniformly connected network with a mean of  $10^4$  synapses per neuron (Amit 1989).

The estimate of the number of information bits that can penetrate the central nervous system in a human lifetime through the sensory organs is  $10^{20}$  bits. This estimate, from von Neumann (1958), is based on the neurobiologically determined mean rate of neural activity (14/s), the number of input channels equal to the estimated number of binary neurons in the brain ( $10^{10}$ ), and the mean length of human life ( $10^9$  s).

There is still a five order of magnitude difference between the number of information bits penetrating the natural human memory ( $10^{20}$  bits) and its inherent capacity ( $\alpha 10^{15}$  bits). The explanation of this discrepancy can be examined by considering the correlation between the ensuing information clusters. Once the network organizes, there are no reasons for independent storage of related patterns. Correlated patterns are storable within their basins of attractions. Therefore, any significant correlations between learned patterns reduce the number of neurons required for storage.

The following are the most important quality criteria for design of neural associative memories with an auto- or heteroassociative processing capability (Hassoun and Youssef 1989):

1. a large capacity
2. an immunity to noisy or partial input
3. few false states which are pseudomemories
4. no memory limit cycles which are oscillatory states
5. implementation of the HD criterion or some other similar measure of neighborhood.

A number of recording algorithms and memory architectures have been covered

in this chapter. More approaches have been proposed in the literature. As a result, improved recording techniques have been proposed to optimize the association process. The reader is referred to the specialized literature for further reading on the topic. A high-performance recording technique based on the Ho-Kashyap algorithm is given by Hassoun and Youssef (1989) and Hassoun (1989). Another approach involves a hidden layer of neurons, which increases the orthogonality of the stored patterns so that, similar to a Hopfield memory, the stored patterns are easier to retrieve (Hoffman and Davenport 1990). Higher order learning rules are also introduced by involving higher order correlations between the stored patterns (Chen et al. 1986). Despite applications of various configurations and recording methods, however, none of the associative memories has yet been designed to be perfect and fault-free.

Due to the parallel processing mode, associative memories can be successfully applied if correctly designed. An important design aspect should involve accounting for a safe memory operation margin. This should include the study of existing possible trade-offs between the memory size and the numbers of stored patterns or their pairs. The study should also involve the expected noise tolerance, distances between stored patterns, etc.

Neural associative memories are potentially useful in a wide range of applications such as content-addressable storage, search-and-retrieve tasks, optimization computing, image restoration, pattern recognition, classification, and code correcting (Grant and Sage 1986). Other applications include storage of words and of continuous speech. In optical signal processing, associative memories are used for code filtering, code mapping, code joining, code shifting, and projecting.

## PROBLEMS

Please note that problems highlighted with an asterisk (\*) are typically computationally intensive and the use of programs is advisable to solve them.

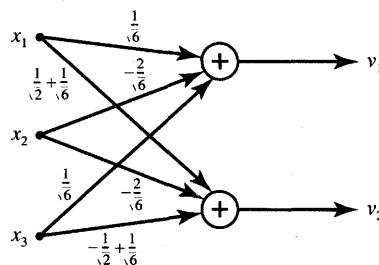
*P6.1* The linear associator has to associate the following pairs of vectors:

$$\mathbf{s}^{(1)} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \end{bmatrix}^t \rightarrow \mathbf{f}^{(1)} = [0 \ 1 \ 0]^t$$

$$\mathbf{s}^{(2)} = \begin{bmatrix} 1 \\ 2 \\ -\frac{5}{6} \\ \frac{1}{6} \end{bmatrix}^t \rightarrow \mathbf{f}^{(2)} = [1 \ 0 \ 1]^t$$

$$\mathbf{s}^{(3)} = \begin{bmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{6} \\ \frac{1}{6} \end{bmatrix}^t \rightarrow \mathbf{f}^{(3)} = [0 \ 0 \ 0]^t$$

- (a) Verify that vectors  $\mathbf{s}^{(1)}$ ,  $\mathbf{s}^{(2)}$ , and  $\mathbf{s}^{(3)}$  are orthonormal.
- (b) Create partial weight matrices for each desired association.
- (c) Compute the total weight matrix.



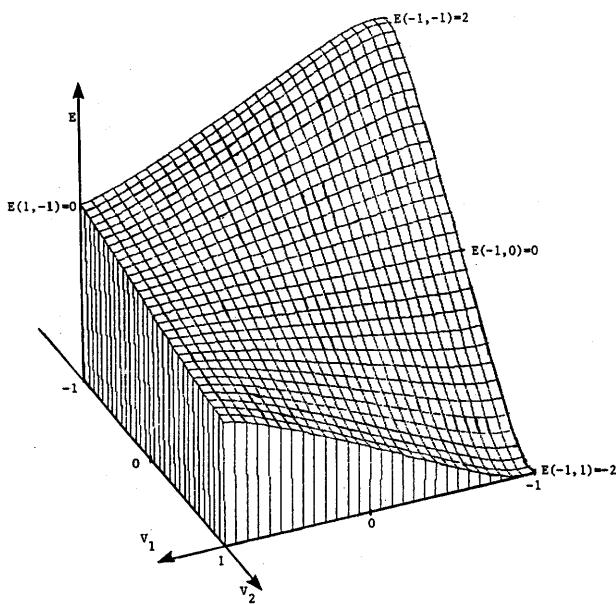
**Figure P6.2** Linear associator network.

- (d) Verify the association performed by the network for one of the specified patterns  $s^{(i)}$ .
- (e) Distort  $s^{(i)}$  by setting one of its components to zero and compute the resulting noise vector.
- P6.2 The linear associator shown in Figure P6.2 has been trained to associate three unipolar binary output vectors to the following input vectors:

$$\begin{aligned} s^{(1)} &= \left[ \frac{1}{\sqrt{3}} \quad \frac{1}{\sqrt{3}} \quad \frac{1}{\sqrt{3}} \right]^t \\ s^{(2)} &= \left[ \frac{1}{\sqrt{2}} \quad 0 \quad -\frac{1}{\sqrt{2}} \right]^t \\ s^{(3)} &= \left[ \frac{1}{\sqrt{6}} \quad -\frac{2}{\sqrt{6}} \quad \frac{1}{\sqrt{6}} \right]^t \end{aligned}$$

Find the weight matrix  $\mathbf{W}$ . Then compute vectors  $\mathbf{f}^{(1)}$ ,  $\mathbf{f}^{(2)}$ , and  $\mathbf{f}^{(3)}$ , which have been encoded in the network.

- P6.3 Assume that a linear associator has been designed using the cross-correlation matrix for heteroassociative association of  $p$  orthonormal patterns. Subsequently, another orthonormal pattern  $s^{(p+1)}$  associated with  $\mathbf{f}^{(p+1)}$  must be stored. An incremental change in the weight matrix needs to be performed using the cross-correlation concept. Prove that the association  $s^{(p+1)} \rightarrow \mathbf{f}^{(p+1)}$  results in no noise term present at the output.
- P6.4 The linear autoassociator needs to associate distorted versions of vectors  $s_1$ ,  $s_2$ , and  $s_3$  with their undistorted prototypes specified in Problem P6.1. Calculate the weight matrix for the linear autoassociator network.
- P6.5 A two-neuron recurrent autoassociative memory has one stable state. Its energy function is of the form  $E = -\frac{1}{2}\mathbf{v}'\mathbf{W}\mathbf{v} - \mathbf{i}'\mathbf{v}$ , so it includes the bias term. The energy function for the circuit is shown in Figure P6.5. Compute elements of  $\mathbf{W}$  and  $\mathbf{i}$  and draw the network diagram.



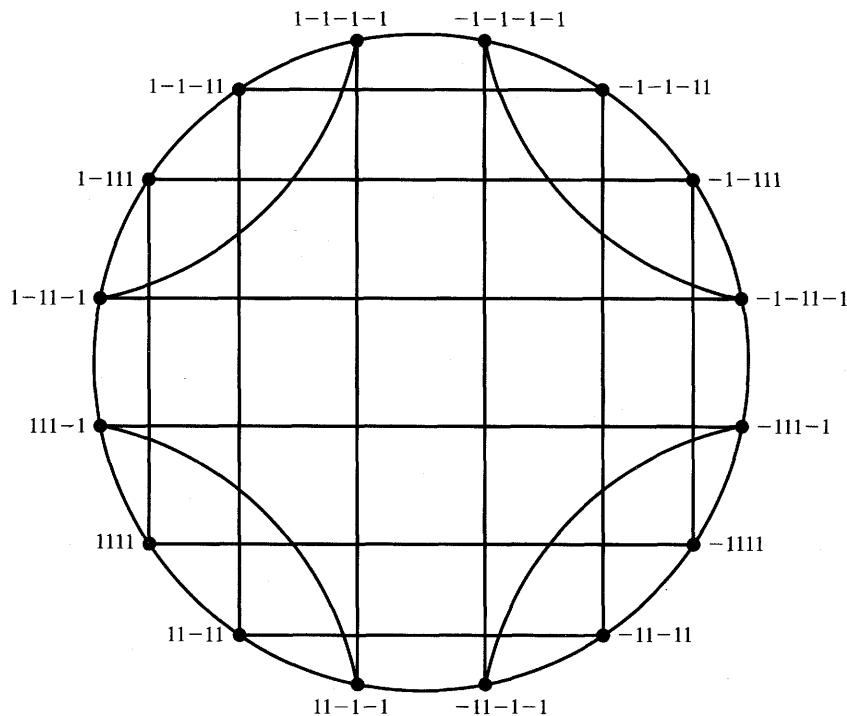
**Figure P6.5** Energy surface for memory network in Problem P6.5.

- P6.6* The memory recording algorithm expressing the Hebbian learning rule with subsequent nullification of the diagonal matrix

$$\mathbf{W} = \sum_{m=1}^p \mathbf{s}^{(m)} \mathbf{s}^{(m)t} - p \mathbf{I}$$

can also be used for unipolar binary vector storage. Prove that this mode of coding is invariant under the complement operation  $\mathbf{s}^{(m)} = \mathbf{s}'^{(m)}$ , i.e., it leads to identical weight matrices in both cases.

- P6.7* A three-bit autoassociative recurrent memory updating under asynchronous rule stores a single vector  $\mathbf{s}^{(1)} = [1 \ 1 \ -1]^t$ . Prepare a state transition map using a three-dimensional cube with the energy values assigned to each vertex, and with the direction of all possible transitions assigned to each edge of the cube. Then evaluate all transitions using each vertex as an initial condition.
- P6.8* A blank state transition map for a recurrent autoassociative memory with asynchronous update is shown in Figure P6.8. Assume that  $\mathbf{s}^{(1)} = [1 \ 1 \ -1 \ 1]^t$  and  $\mathbf{s}^{(2)} = [1 \ -1 \ 1 \ -1]^t$  need to be stored.
- (a) Compute  $E(\mathbf{v})$ .
  - (b) Mark energy values at each node of the map.
  - (c) Label all possible transitions on the state diagram.



**Figure P6.8** Blank state transition map for Problem P6.8.

- (d) Identify whether or not any initial conditions occur that lead to final states that are either spurious memories or that are true memories but are nonoptimal in the Hamming distance sense.
- P6.9** A four-neuron Hopfield autoassociative memory has been designed for  $s^{(1)} = [1 \ -1 \ -1 \ 1]^t$ . Assume that the recall is performed synchronously, i.e. all four memory outputs update at the same time.
- Draw the state transition map for this recall mode assuming that any of the 16 states can be considered to be the initial one.
  - Compute  $E(v)$ .
  - Determine if the energy value decreases in this update mode.
- P6.10** The following vectors need to be stored in a recurrent autoassociative memory:
- $$s^{(1)} = [1 \ 1 \ 1 \ 1 \ 1]^t$$
- $$s^{(2)} = [1 \ -1 \ -1 \ 1 \ -1]^t$$
- $$s^{(3)} = [-1 \ 1 \ 1 \ 1 \ 1]^t$$
- Compute the weight matrix  $W$ .

- (b) Apply the input vector  $\mathbf{v}^0 = [1 \ -1 \ -1 \ 1 \ 1]^t$  and allow for asynchronous convergence in ascending node order starting at node 1.
- (c) With the same input vector as in part (b) allow for asynchronous convergence in descending node order starting at node 5.
- (d) Comment as to whether Hopfield memory provides an optimal solution in terms of the Hamming distance criterion. [The results may be surprising since this is an overloaded memory. Assume  $\text{sgn}(0) = +1$  for update purposes in this problem.]

**P6.11** The following unipolar binary vectors must be stored in the recurrent autoassociative memory using the outer product method with the nullification of the diagonal:

$$\mathbf{s}^{(1)} = [1 \ 0 \ 0 \ 1 \ 0]^t$$

$$\mathbf{s}^{(2)} = [0 \ 1 \ 1 \ 0 \ 1]^t$$

$$\mathbf{s}^{(3)} = [1 \ 1 \ 0 \ 1 \ 0]^t$$

- (a) Compute matrix  $\mathbf{W}$ .
- (b) Find the analytical expression for the energy function that the memory is minimizing.

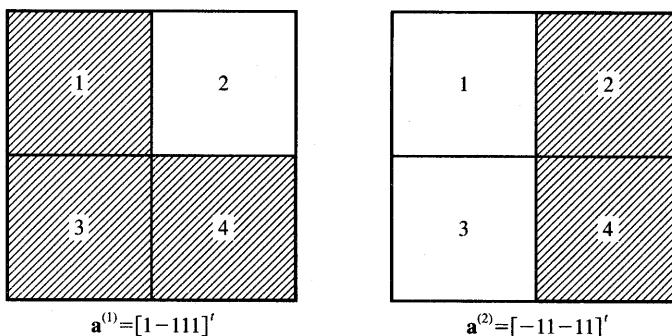
**P6.12** Assume that the recurrent autoassociative memory has been designed for  $p < n$  orthogonal patterns encoded in its weight matrix. Show that presenting the key pattern  $\mathbf{s}^{(p+1)}$ , which is orthogonal to the encoded patterns  $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \dots, \mathbf{s}^{(p)}$ , but has not been stored, results in the initial energy value  $E(\mathbf{s}^{(p+1)}) = +\frac{1}{2}np$ . Then compute the energy minima values for this memory for each of the stored vectors  $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \dots, \mathbf{s}^{(p)}$ .

**P6.13** Compute the energy values for all 16 bipolar binary key vectors for a five-bit autoassociative recurrent memory having the following weight matrix (complement vectors need not be checked):

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & 2 & 0 & -2 \\ 0 & 0 & 0 & -2 & 0 \\ 2 & 0 & 0 & 0 & -2 \\ 0 & -2 & 0 & 0 & 0 \\ -2 & 0 & -2 & 0 & 0 \end{bmatrix}$$

By comparing the energy levels, prepare a hypothesis regarding the two stored vectors. Then verify your hypothesis by comparing the computed weight matrix with the matrix  $\mathbf{W}$  specified above.

**P6.14** Vectors  $\mathbf{a}^{(1)} = [1 \ -1 \ 1 \ 1]^t$  and  $\mathbf{a}^{(2)} = [-1 \ 1 \ -1 \ 1]^t$  represent the bit maps shown in Figure P6.14. They need to be autoassociated for input vectors at  $\text{HD} = 1$  from the prototypes stored. Design the following two autoassociators:



**Figure P6.14** Bit maps for Problem P6.14.

- (a) a linear autoassociator in the form of a linear associative memory with added thresholding element (TLU) at the output, if needed, but no feedback
- (b) a recurrent autoassociative memory with asynchronous updating.

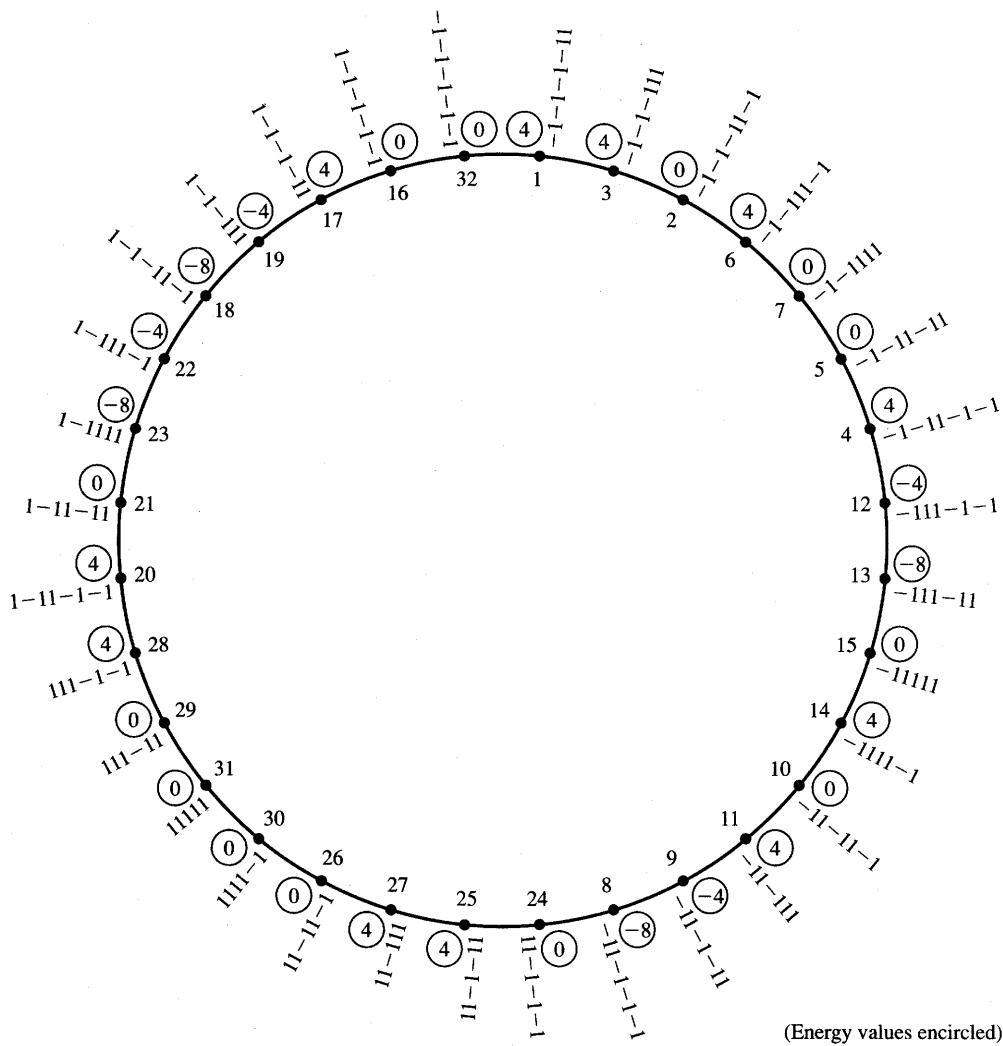
Compare the performance of both networks by evaluating responses to eight input vectors at HD = 1 from  $\mathbf{a}^{(1)}$  and  $\mathbf{a}^{(2)}$ . The nodes of the memory designed in part (b) should be updated in ascending order starting at node 1.

**P6.15** A recurrent autoassociative memory has been designed to store  $\mathbf{s}^{(1)} = [1 \ -1 \ 1 \ 1 \ 1]^T$  and  $\mathbf{s}^{(2)} = [-1 \ 1 \ 1 \ -1 \ 1]^T$ . The energy values have been computed and marked on the map of state transitions for this memory as shown in Figure P6.15. By analyzing the memory state diagram for all transitions at HD = 1, determine if this memory has any false spurious terminal states. If yes, what are they and what are the input vectors that could possibly be associating with the false memories?

**P6.16\*** Design the autoassociative recurrent memory to store the bit maps of the characters *A*, *I*, and *O* as specified in Figure P6.16. List the resulting **W** matrix. Implement the asynchronous retrieval algorithm starting at the upper leftmost pixel of the map, moving left to right during the update. Simulate the asynchronous convergence of the memory toward each of the characters stored in the following cases of key vectors:

- (a) left upper pixel of each character reversed
- (b) four center column pixels of the character field reversed.

Provide the retrieved vectors in each of six cases and note the number of recursion cycles. (Each recursion cycle consists here of a sequence of 16 updates.)



**Figure P6.15** Unfinished state transition diagram for Problem P6.15.

**P6.17** Analyze graphically the recurrent network with one neuron as shown in Figure 6.15 for the two cases of neuron activation functions as shown in Figure P6.17a,b. Determine whether points *A*, *B*, *C*, and *D* are stable or unstable fixed points, or cyclic solutions.

**P6.18** The table here lists three scalar energy functions  $E(x)$  that are often encountered in the studies of associative memory. Their respective expressions for gradient vectors  $\nabla E(x)$  are also listed. Derive the expressions for

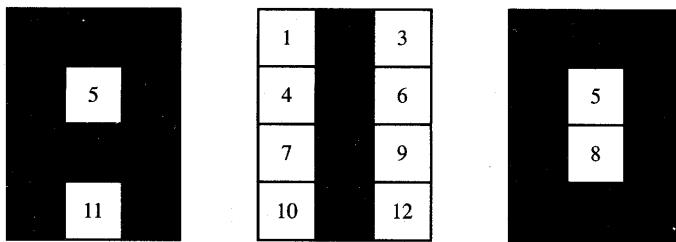


Figure P6.16 Bit maps of characters A,I,O for Problem P6.16.

gradient vectors for the energy functions listed in the left column. Assume  $\mathbf{W}$  is  $(n \times n)$  and  $\mathbf{i}$  and  $\mathbf{x}$  are  $(n \times 1)$ .

Case	$E(\mathbf{x})$	$\nabla E(\mathbf{x})$
1	$\mathbf{i}'\mathbf{x}$	$\mathbf{i}$
2	$\mathbf{x}'\mathbf{W}\mathbf{i}$	$\mathbf{W}\mathbf{i}$
3	$\mathbf{x}'\mathbf{W}\mathbf{x} + \mathbf{i}'\mathbf{x}$	$(\mathbf{W}' + \mathbf{W})\mathbf{x} + \mathbf{i}$

P6.19\* Design the bidirectional associative memory that associates characters  $(A, C)$ ,  $(I, I)$ , and  $(O, T)$  designed using the bit maps of  $A$ ,  $I$ , and  $O$  from Problem P4.16 and of  $C$ ,  $I$ , and  $T$  from Figure 4.19.

(a) Compute  $\mathbf{W}$ .

(b) Check associations within stored pairs using search arguments as the stored characters.

P6.20\* Given are the three following training pairs  $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$ , for  $i = 1, 2, 3$ , for bidirectional associative memory:

$$(\mathbf{a}^{(1)}, \mathbf{b}^{(1)}) = \left( [1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1]^t, [1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \ 1 \ -1]^t \right)$$

$$(\mathbf{a}^{(2)}, \mathbf{b}^{(2)}) = \left( [-1 \ 1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1 \ 1]^t, [1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ -1 \ 1]^t \right)$$

$$(\mathbf{a}^{(3)}, \mathbf{b}^{(3)}) = \left( [1 \ -1 \ 1 \ -1 \ 1 \ 1 \ -1 \ 1 \ 1]^t, [-1 \ 1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1 \ 1]^t \right)$$

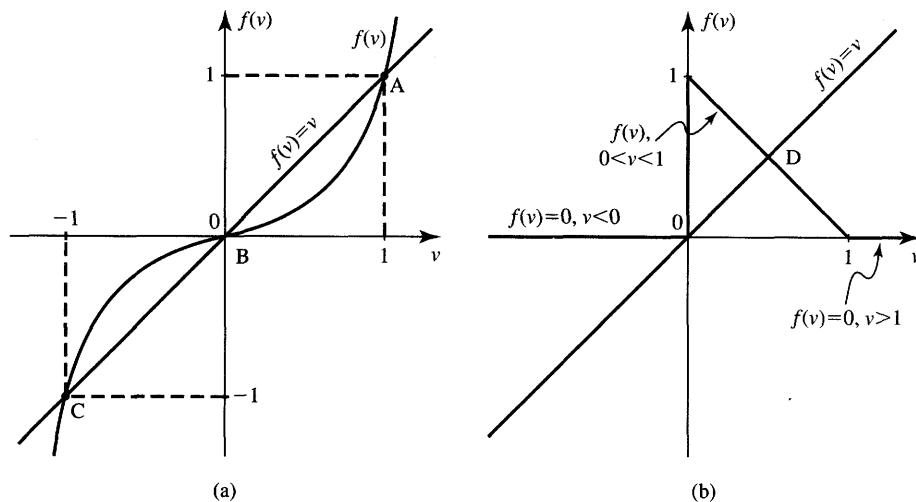


Figure P6.17 Activation functions for neuron in Problem P6.17.

Find the weight matrix  $\mathbf{W}$  for the associative memory. Then:

- Determine that the input  $\mathbf{a}^{(2)}$  does not result in retrieval of  $\mathbf{b}^{(2)}$ ; find  $\mathbf{b}^{(f)}$  retrieved instead of  $\mathbf{b}^{(2)}$ .
- Find  $E(\mathbf{a}^{(f)}, \mathbf{b}^{(f)})$  and  $E(\mathbf{a}^{(2)}, \mathbf{b}^{(2)})$  and compare their values.
- Determine whether or not  $E(\mathbf{a}^{(2)}, \mathbf{b}^{(2)})$  is a local minimum. This can be done by evaluating  $E(\mathbf{a}^{(2)}, \mathbf{b}^{(2)})$  and comparing it with the energy values at points that are at HD = 1 from  $\mathbf{a}^{(2)}$  and then from  $\mathbf{b}^{(2)}$ .

P6.21 Prove the following theorem: If a training pair  $(\mathbf{a}^{(m)}, \mathbf{b}^{(m)})$  is a local minimum of the energy function as in (6.54b), then the recovery is guaranteed since

$$\Gamma[\mathbf{W}\mathbf{b}^{(m)}] = \mathbf{a}^m \quad \text{and} \quad \Gamma[\mathbf{W}\mathbf{a}^{(m)}] = \mathbf{b}^{(m)}$$

[Hint: Examine the condition for local minimum

$$-\mathbf{a}^{(m)}\mathbf{W}\mathbf{b}^{(m)t} < -\mathbf{a}'^{(m)}\mathbf{W}\mathbf{b}^{(m)t}$$

where  $\mathbf{a}'^{(m)}$  is defined as differing by HD = 1 from  $\mathbf{a}^{(m)}$ . Repeat for  $\mathbf{b}'^{(m)}$  differing by HD = 1 from  $\mathbf{b}^{(m)}$ .]

P6.22\* Some of the three training pairs in Problem P6.20 coded according to (6.51) are not recoverable when training of multiplicity 1 is used. However, a multiple training method can be used to guarantee the recovery. Apply the multiple training method to obtain the weight matrix with guaranteed recovery for each member of the training pair. Try multiplicity of 2 or 3 in order to test the encoding of each pair. Devise the lowest multiplicity of training for each pair that would guarantee recovery.

P6.23 A temporal associative memory needs to be designed for recall of the following sequence:

$$\mathbf{s}^{(1)} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ -1 \end{bmatrix}, \quad \mathbf{s}^{(2)} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \quad \mathbf{s}^{(3)} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

Compute the weight matrix  $\mathbf{W}$  and check the recall of patterns in the forward and backward directions. Assume a bipolar binary activation function.

P6.24 The weight matrix of the temporal associative memory is known as

$$\mathbf{W} = \begin{bmatrix} -1 & 3 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 3 \\ -1 & -1 & 3 & -1 & -1 \\ -1 & 3 & -1 & -1 & -1 \\ 3 & -1 & -1 & 3 & 1 \end{bmatrix}$$

Knowing that a vector  $\mathbf{s}^{(1)} = [-1 \ 1 \ -1 \ -1 \ 1]^t$  belongs to a sequence, find the remaining vectors of the sequence. Having found the full sequence, verify that encoding it actually yields the weight matrix  $\mathbf{W}$  as specified in the problem. Calculate the noise term vectors generated at each recall step and determine that they are suppressed during the thresholding operation.

## REFERENCES

- Amari, S. I. 1972. "Learning Patterns and Pattern Sequences by Self-Organizing Nets," *IEEE Trans. Computers* (21), pp. 1197–1206.
- Amit, D. J., G. Gutfreund, and H. Sompolinsky. 1985. "Spin-glass Models of Neural Networks," *Phys. Rev. A* 32: 1007–1018.
- Amit, D. J. 1989. *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge: Cambridge University Press.
- Carpenter, G. A. 1989. "Neural Network Models for Pattern Recognition and Associative Memory," *Neural Networks* 2: 243–257.
- Chen, H. H., Y. C. Lee, G. Z. Sun, and H. Y. Lee. 1986. "Higher Order Correlation Model for Associative Memory," in *Neural Networks for Computing*, ed. J. S. Denker. New York: American Institute of Physics, pp. 86–99.
- Desai, M. S. 1990. "Noisy Pattern Retrieval Using Associative Memories," MSEE thesis, University of Louisville, Kentucky.

- Grant, P. M., and J. P. Sage. 1986. "A Comparison of Neural Network and Matched Filter Processing for Detecting Lines in Images," in *Neural Networks for Computing*, ed. J. S. Denker. New York: American Institute of Physics, pp. 194-199.
- Hagiwara, M. 1990. "Multidimensional Associative Memory," in *Proc. 1990 IEEE Joint Conf. on Neural Networks*, Washington, D.C., January 15-19, 1990. New York: IEEE, Vol. I, pp. 3-6.
- Hassoun, M. H., and A. M. Youssef. 1989. "High Performance Recording Algorithm for Hopfield Model Associative Memories," *Opt. Eng.* 28(1): 46-54.
- Hassoun, M. H. 1989. "Dynamic Heteroassociative Memories," *Neural Networks*, 2: 275-287.
- Hoffman, G. W., and M. R. Davenport. 1990. "A Network that Uses the Outer Product Rule, Hidden Neurons, and Peaks in the Energy Landscape," in *Proc. 1990 IEEE Int. Symp. on Circuits and Systems*, New Orleans, La., May 1-3, 1990. New York: IEEE, pp. 196-199.
- Hopfield, J. J. 1982. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. National Academy of Sciences* 79: 2554-2558.
- Hopfield, J. J. 1984. "Neurons with Graded Response Have Collective Computational Properties Like Those of Two State Neurons," *Proc. National Academy of Sciences* 81: 3088-3092.
- Jong, T. L., and H. M. Tai. 1988. "Associative Memory Based on the Modified Hopfield Neural Net Model," in *Proc. 30th Midwest Symp. on Circuits and Systems*, St. Louis, Mo., August 1988, pp. 748-751.
- Kamp, Y., and M. Hasler. 1990. *Recursive Neural Networks for Associative Memory*. Chichester, U.K.: John Wiley & Sons.
- Kohonen, T. 1977. *Associative Memory: A System-Theoretical Approach*, Berlin: Springer-Verlag.
- Kohonen, T., et al. 1981. "Distributed Associative Memory," in *Parallel Models of Distributed Memory Systems*, ed. G. E. Hinton and J. A. Anderson. Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- Kohonen, T. 1987. "Adaptive, Associative, and Self-Organizing Functions in Neural Computing," *Appl. Opt.* 26(23): 4910-4918.
- Komlos, J., and R. Paturi. 1988. "Convergence Results in an Associative Memory Model," *Neural Networks* 1: 239-250.
- Kosko, B. 1987. "Adaptive Bidirectional Associative Memories," *Appl. Opt.* 26(23): 4947-4959.
- Kosko, B. 1988. "Bidirectional Associative Memories," *IEEE Trans. Systems, Man, and Cybernetics* 18(1): 49-60.

- McEliece, R. J., E. C. Posner, E. R. Rodemich, and S. V. Venkatesh. 1987. "The Capacity of the Hopfield Associative Memory," *IEEE Trans. Information Theory* IT-33(4): 461–482.
- Michel, A. N., and J. A. Farrell. 1990. "Associative Memories via Artificial Neural Networks," *IEEE Control Systems Magazine* (April): 6–17.
- Personnaz, L., I. Guyon, and G. Dreyfus. 1986. "Collective Computational Properties of Neural Networks: New Learning Mechanism," *Phys. Rev. A* 34(5): 4217–4228.
- Petsche, T. 1988. "Topics in Neural Networks," Ph.D. diss., Princeton University, New Jersey.
- Petsche, T., and B. W. Dickinson. 1990. "Trellis Codes, Receptive Fields, and Fault-Tolerant Self-Repairing Neural Networks," *IEEE Trans. Neural Networks* 1(2): 154–166.
- Serra, R., and G. Zanarini. 1990. *Complex Systems and Cognitive Processes*, Berlin: Springer-Verlag.
- von Neumann, J. 1958. *The Computer and the Brain*. New Haven, Conn.: Yale University Press, p. 87.
- Wang, Y. F., J. B. Cruz, and J. H. Mulligan. 1990. "Two Coding Strategies for Bidirectional Associative Memory," *IEEE Trans. Neural Networks* 1(1): 81–92.
- Wang, Y. F., J. B. Cruz, and J. H. Mulligan. 1990. "On Multiple Training for Bidirectional Associative Memory," *IEEE Trans. Neural Networks*, 1(5): 275–276.
- Wang, Y. F., J. B. Cruz, and J. H. Mulligan. 1991. "Guaranteed Recall of All Training Pairs for Bidirectional Associative Memory," *IEEE Trans. Neural Networks*, 2(6): 559–567.

---

# MATCHING AND SELF-ORGANIZING NETWORKS

*New opinions are always suspected,  
and usually opposed, without any  
other reason, but because they are  
not already common.*

J. LOCKE

- 7.1 Hamming Net and MAXNET
- 7.2 Unsupervised Learning of Clusters
- 7.3 Counterpropagation Network
- 7.4 Feature Mapping
- 7.5 Self-Organizing Feature Maps
- 7.6 Cluster Discovery Network (ART1)
- 7.7 Concluding Remarks

The neural networks covered in previous chapters represent a number of distinct network classes. We studied single-layer and multilayer perceptron networks, which are essentially feedforward recall architectures trained with supervision. Continuous- and discrete-time single-layer networks with recurrent recall form another separate architectural group of networks. Associative memories can also be treated as a distinct class of neural networks. Despite somewhat diversified architectures, associative memories exhibit a common learning mode based on auto- and cross-correlation between stored vectors. Although other learning techniques exist, memories are typically designed with fixed weights by using the technique of batch learning, or recording.

The networks we discuss in this chapter represent a combination of the architectures and recall modes studied thus far. One novel aspect is the networks' learning mode. Our purpose in this chapter is to present networks that learn neither by the familiar correlation rule, nor by the perceptron or gradient descent techniques we have studied. In fact, no feedback from the environment will be provided during learning. The network must discover for itself any relationships

of interest that may exist in the input data. Our interest is in designing networks that are able to translate the discovered relationships into outputs. We will see in this chapter that discovery of patterns, features, regularities, or categories can be learned without a teacher.

Networks trained without a teacher usually learn by matching certain explicit familiarity criteria. These networks can produce output that tells us how familiar it is with the present pattern. This is done by comparing the present pattern with typical patterns seen in the past. Consider that the present pattern is evaluated for "similarity" with typical patterns from the past. One important measure of similarity used for learning studied in this chapter is the maximum value of the scalar product of the weights and input vector. Using the scalar product metric, weights can be trained in an unsupervised mode to resemble frequent input patterns, or better, pattern clusters. Another often used measure of similarity is the topological neighborhood, or distance, between the responding neurons arranged in regular geometrical arrays.

Our objective in this chapter is to explore the unsupervised training algorithms of neural networks so that their weights can sensibly adapt during the process called *self-organization*. Networks trained in this mode will not only react to values of inputs but also to their statistical parameters. In this chapter we will assume that both inputs and their probability density will affect the training. Rare inputs will have less impact for learning than those occurring frequently. Sometimes, training inputs will need to be applied in their natural sequence of occurrence.

Unsupervised learning may seem at first impossible to accomplish. Indeed, at the beginning of the learning process, the network's responses may be implausible. The lack of a teacher's input forces the network to learn gradually by itself which features need to be considered in classification or recognition and thus need to be reinforced. At the same time, other less important features can be neglected or suppressed based on the mapping criteria that underlie the unsupervised learning algorithms. The proximity measures allow the computation of important indicators of how to differentiate between more and less important features during the unsupervised learning. What is very peculiar to this learning mode is that the indicators have to be created based on the history of learning experience.

Unsupervised learning can only be implemented with redundant input data. Redundancy provides knowledge about statistical properties of input patterns. In fact, there are close connections between statistical approaches to pattern classification and the neural network techniques discussed here (Duda and Hart 1973).

In the absence of target responses or guidelines provided directly by the teacher, the network can, for example, build classification/recognition/mapping performance during the process devised in Chapter 2 as competitive learning. The winning output node will receive the only reward during learning in the

form of a weight adjustment. In some cases, the extension of rewards for the node's neighborhood will also be allowed. Strengthening the winning weights gradually builds into the network a history of what has been seen in the past. Such networks are termed *networks with history sensitivity*.

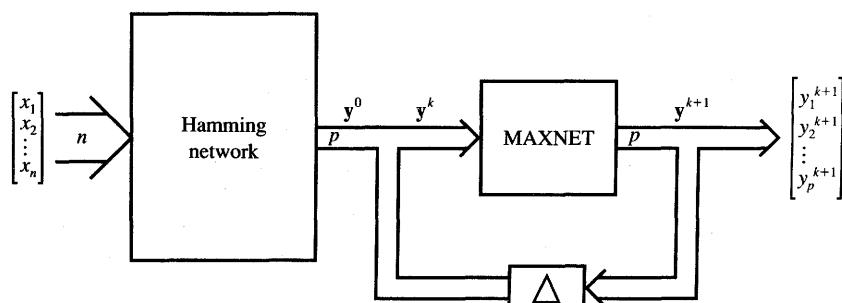
The neural network architectures covered in this chapter include the Hamming network, MAXNET, the clustering Kohonen layer, the Grossberg outstar learning layer, a counterpropagation network, self-organizing feature mapping networks, and an adaptive resonance network for cluster discovery and classification of binary vectors.

## 7.1

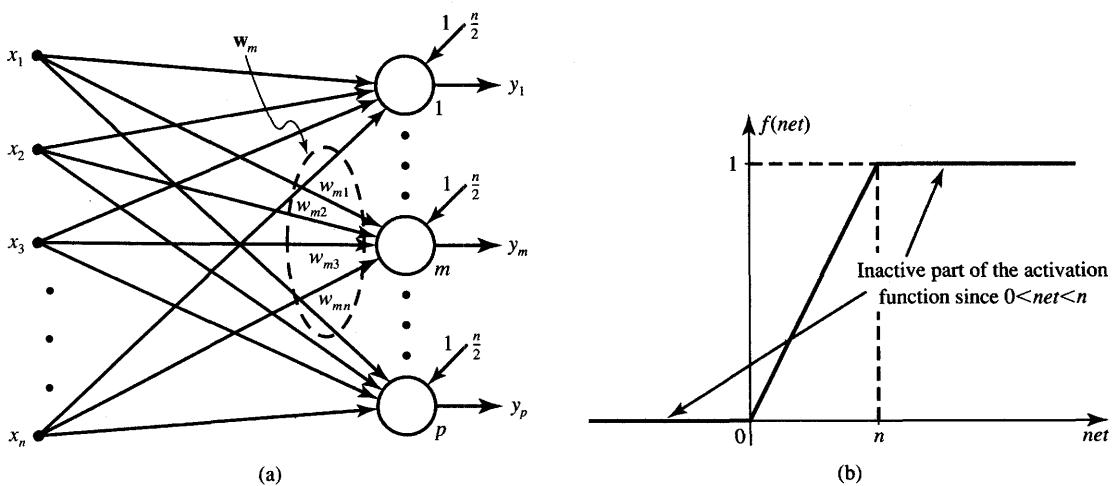
### HAMMING NET AND MAXNET

In this section a two-layer classifier of binary bipolar vectors will be covered. The block diagram of the network is shown in Figure 7.1. It is a minimum Hamming distance classifier, which selects the stored classes that are at a minimum HD value to the noisy or incomplete argument vector presented at the input. This selection is essentially performed solely by the *Hamming network*. The Hamming network is of the feedforward type and constitutes the first layer of the classifier. The  $p$ -class Hamming network has  $p$  output neurons. The strongest response of a neuron is indicative of the minimum HD value between the input and the category this neuron represents. The second layer of the classifier is called *MAXNET* and it operates as a recurrent recall network in an auxiliary mode. Its only function is to suppress values at MAXNET output nodes other than the initially maximum output node of the first layer (Lippmann 1987).

As stated, the proper part of the classifier is the Hamming network responsible for matching of the input vector with stored vectors. The expanded diagram of the Hamming network for classification of bipolar binary  $n$ -tuple input vectors



**Figure 7.1** Block diagram of the minimum HD classifier.



**Figure 7.2** Hamming network for  $n$ -bit bipolar binary vectors representing  $p$  classes: (a) classifier network and (b) neurons' activation function.

is shown in Figure 7.2(a). The purpose of the layer is to compute, in a feed-forward manner, the values of  $(n - \text{HD})$ , where HD is the Hamming distance between the search argument and the encoded class prototype vector. Assume that the  $n$ -tuple prototype vector of the  $m$ 'th class is  $s^{(m)}$ , for  $m = 1, 2, \dots, p$ , and the  $n$ -tuple input vector is  $x$ . Note that the entries of the weight vector  $w_m$  defined as

$$w_m = [w_{m1} \quad w_{m2} \quad \dots \quad w_{mn}]^t, \quad \text{for } m = 1, 2, \dots, p$$

connect inputs to the  $m$ 'th neuron, which performs as the class indicator.

Before we express the value of the Hamming distance by means of the scalar product of  $x$  and  $s^{(m)}$ , let us formalize the metric introduced here. A vector classifier with  $p$  outputs, one for each class, can be conceived such that the  $m$ 'th output is 1 if and only if  $x = s^{(m)}$ . This would require that the weights be  $w_m = s^{(m)}$ . The classifier outputs are  $x^t s^{(1)}, x^t s^{(2)}, \dots, x^t s^{(m)}, \dots, x^t s^{(p)}$ . When  $x = s^{(m)}$ , only the  $m$ 'th output is  $n$ , provided the classes differ from each other, and assuming  $\pm 1$  entries of  $x$ . The scalar product of vectors has been used here as an obvious measure for vector matching.

The scalar product  $x^t s^{(m)}$  of two bipolar binary  $n$ -tuple vectors can be written as the total number of positions in which the two vectors agree minus the number of positions in which they differ. Note that the number of different bit positions is the HD value. Understandably, the number of positions in which two vectors agree is  $n - \text{HD}$ . The equality is written

$$x^t s^{(m)} = (n - \text{HD}(x, s^{(m)})) - \text{HD}(x, s^{(m)}) \quad (7.1a)$$

This is equivalent to

$$\frac{1}{2} \mathbf{x}' \mathbf{s}^{(m)} = \frac{n}{2} - \text{HD}(\mathbf{x}, \mathbf{s}^{(m)}) \quad (7.1b)$$

We can now see that the weight matrix  $\mathbf{W}_H$  of the Hamming network can be created by encoding the class vector prototypes as rows in the form as below

$$\mathbf{W}_H = \frac{1}{2} \begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \dots & s_n^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \dots & s_n^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ s_1^{(p)} & s_2^{(p)} & \dots & s_n^{(p)} \end{bmatrix} \quad (7.2)$$

where the  $1/2$  factor is convenient for scaling purposes. Now the network with input vector  $\mathbf{x}$  yields the value of  $(1/2)\mathbf{x}'\mathbf{s}^{(m)}$  at the input to the node  $m$ , for  $m = 1, 2, \dots, p$ . Adding the fixed bias value of  $n/2$  to the input of each neuron results in the total input  $net_m$

$$net_m = \frac{1}{2} \mathbf{x}' \mathbf{s}^{(m)} + \frac{n}{2}, \quad \text{for } m = 1, 2, \dots, p \quad (7.3a)$$

Using the identity (7.1b),  $net_m$  can be expressed as

$$net_m = n - \text{HD}(\mathbf{x}, \mathbf{s}^{(m)}) \quad (7.3b)$$

Let us apply neurons in the Hamming network with activation functions as in Figure 7.2(b). The neurons need to perform only the linear scaling of (7.3b) such that  $f(net_m) = (1/n)net_m$ , for  $m = 1, 2, \dots, p$ . Since inputs are between 0 and  $n$ , we obtain the outputs of each node scaled down to between 0 and 1. Furthermore, the number of the node with the highest output indeed indicates the class number to which  $\mathbf{x}$  is at the smallest HD. A perfect match of input vector to class  $m$ , which is equivalent to the condition  $\text{HD} = 0$ , is signaled by  $f(net_m) = 1$ . An input vector that is the complement of the prototype of class  $m$  would result in  $f(net_m) = 0$ . The response of the Hamming network essentially terminates the classification in which only the first layer of network from Figure 7.1 computes the relevant matching score values. As seen, the classification by the Hamming network is performed in a feedforward and instantaneous manner.

MAXNET needs to be employed as a second layer only for cases in which an enhancement of the initial dominant response of the  $m$ 'th node is required. As a result of MAXNET recurrent processing, the  $m$ 'th node responds positively, as opposed to all remaining nodes whose responses should have decayed to zero. As shown in Figure 7.3(a), MAXNET is a recurrent network involving both excitatory and inhibitory connections. The excitatory connection within the network is implemented in the form of a single positive self-feedback loop with a weighting coefficient of 1. All the remaining connections of this fully coupled feedback network are inhibitory. They are represented as  $M - 1$  cross-feedback synapses with coefficients  $-e$  from each output. The second layer weight matrix

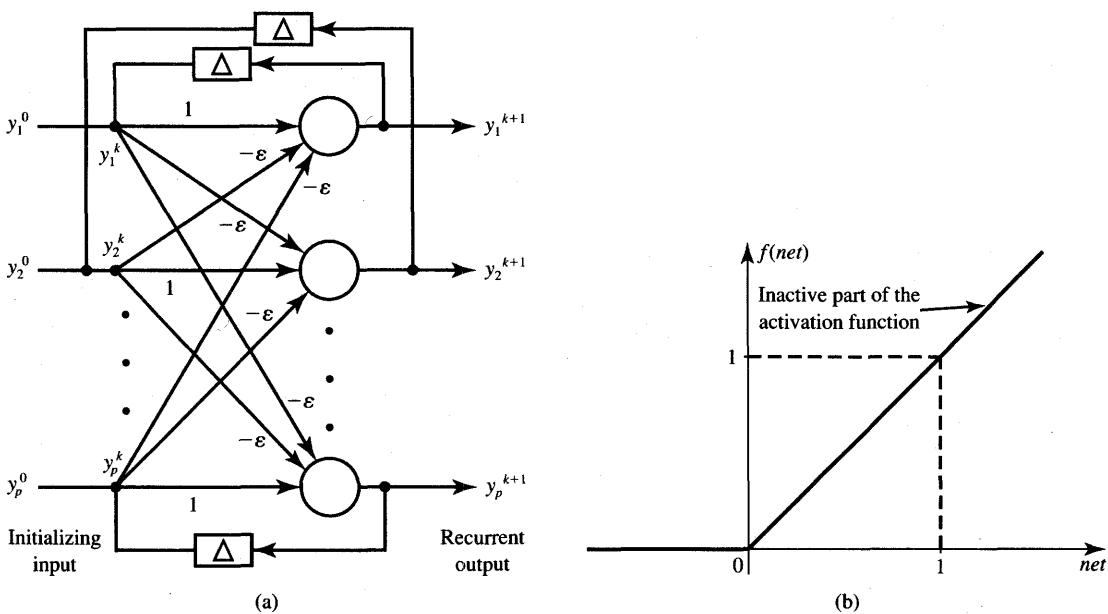


Figure 7.3 MAXNET for  $p$  classes: (a) network architecture and (b) neuron's activation function.

$\mathbf{W}_M$  of size  $p \times p$  is thus of the form (Pao 1989)

$$\mathbf{W}_M = \begin{bmatrix} 1 & -\varepsilon & -\varepsilon & \cdots & -\varepsilon \\ -\varepsilon & 1 & -\varepsilon & \cdots & -\varepsilon \\ -\varepsilon & -\varepsilon & \ddots & & \vdots \\ \vdots & & & & \vdots \\ -\varepsilon & -\varepsilon & -\varepsilon & \cdots & 1 \end{bmatrix} \quad (7.4)$$

where  $\varepsilon$  must be bounded  $0 < \varepsilon < 1/p$ . The quantity  $\varepsilon$  can be called the *lateral interaction coefficient*. With the activation function as shown in Figure 7.3(b) and the initializing inputs fulfilling conditions

$$0 \leq y_i^0 \leq 1, \quad \text{for } i = 1, 2, \dots, p$$

the MAXNET network gradually suppresses all but the largest initial network excitation. When initialized with the input vector  $\mathbf{y}^0$ , the network starts processing it by adding positive self-feedback and negative cross-feedback. As a result of a number of recurrences, the only unsuppressed node will be the one with the largest initializing entry  $y_m^0$ . This means that the only nonzero output response node is the node closest to the input vector argument in HD sense. The recurrent processing by MAXNET leading to this response is

$$\mathbf{y}^{k+1} = \Gamma[\mathbf{W}_M \mathbf{y}^k] \quad (7.5a)$$

where  $\Gamma$  is a nonlinear diagonal matrix operator with entries  $f(\cdot)$  given below:

$$f(\text{net}) = \begin{cases} 0, & \text{net} < 0 \\ \text{net}, & \text{net} \geq 0 \end{cases} \quad (7.5b)$$

Each entry of the updated vector  $\mathbf{y}^{k+1}$  decreases at the  $k$ 'th recursion step of (7.5a) under the MAXNET update algorithm, with the largest entry decreasing slowest. This is due to the conditions on the entries of matrix  $\mathbf{W}_M$  as in (7.4), specifically, due to the condition  $0 < \varepsilon < 1/p$ .

Assume that  $y_m^0 > y_i^0$ ,  $i = 1, 2, \dots, p$  and  $i \neq m$ . During the first recurrence, all entries of  $\mathbf{y}^1$  are computed on the linear portion of  $f(\text{net})$ . The smallest of all  $\mathbf{y}^0$  entries will first reach the level  $f(\text{net}) = 0$ , assumed at the  $k$ 'th step. The clipping of one output entry slows down the decrease of  $y_m^{k+1}$  in all forthcoming steps. Then, the second smallest entry of  $\mathbf{y}^0$  reaches  $f(\text{net}) = 0$ . The process repeats itself until all values except for one, at the output of the  $m$ 'th node, remain at nonzero values. The example below explains the design and operation of the discussed classifier.

### EXAMPLE 7.1

Let us design the minimum Hamming distance classifier for three characters  $C$ ,  $I$ , and  $T$  as shown in Figure 4.19 for which the bipolar binary class prototype vectors are

$$\begin{aligned} \mathbf{s}^{(1)} &= [1 \ 1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1 \ 1]^t \\ \mathbf{s}^{(2)} &= [-1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1]^t \\ \mathbf{s}^{(3)} &= [1 \ 1 \ 1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1]^t \end{aligned} \quad (7.6a)$$

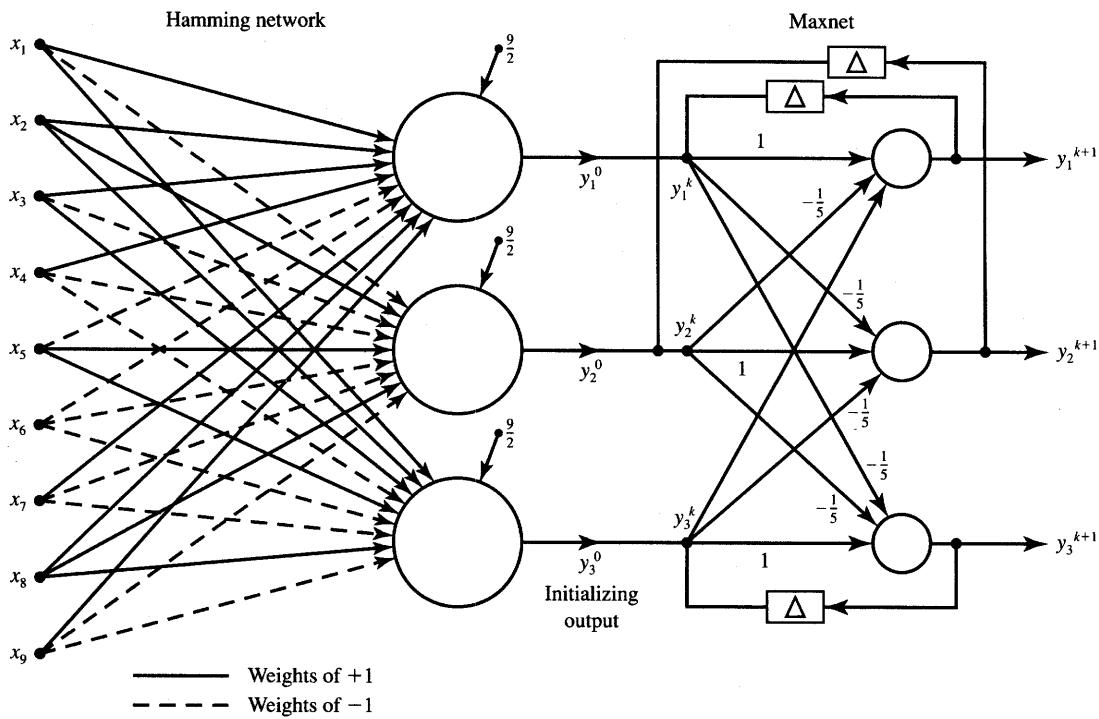
The weight matrix  $\mathbf{W}_H$  of the Hamming network obtained from (7.2) is

$$\mathbf{W}_H = \begin{bmatrix} 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \end{bmatrix} \quad (7.6b)$$

Note that the matrix of Equation (7.6b) determines the bipolar binary values of weights  $w_{ij}$ . Using formula (7.3a), the input vector  $\text{net}$  is expressed as follows:

$$\text{net} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} \frac{9}{2} \\ \frac{9}{2} \\ \frac{9}{2} \\ \frac{9}{2} \end{bmatrix} \quad (7.6c)$$

Selecting  $\varepsilon = 0.2 < 1/3$  terminates the design process of both layers. The



**Figure 7.4** Hamming network and MAXNET for Example 7.1.

designed network is shown in Figure 7.4. Let us look at sample responses of both layers of the network. Assume the test input vector to be

$$\mathbf{x} = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]^T$$

For the Hamming layer neurons' activation and responses, respectively, we obtain from (7.3a) with this input

$$\mathbf{net} = \begin{bmatrix} 7 \\ 3 \\ 5 \end{bmatrix}$$

$$\Gamma[\mathbf{net}] = \begin{bmatrix} \frac{7}{9} \\ \frac{1}{3} \\ \frac{5}{9} \end{bmatrix}$$

Since the computed output,  $\Gamma[\mathbf{net}]$ , is the input vector to MAXNET, we thus have  $\Gamma[\mathbf{net}] = \mathbf{y}^0$ . The MAXNET recurrent formula (7.5a) yields the

following activation:

$$\begin{aligned}\mathbf{net}^k &= \begin{bmatrix} net_1^k \\ net_2^k \\ net_3^k \end{bmatrix} \\ &= \begin{bmatrix} 1 & -\frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & 1 & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & 1 \end{bmatrix} \begin{bmatrix} y_1^k \\ y_2^k \\ y_3^k \end{bmatrix}\end{aligned}$$

and the following response computed from (7.5b)

$$\mathbf{y}^{k+1} = \begin{bmatrix} f(net_1^k) \\ f(net_2^k) \\ f(net_3^k) \end{bmatrix}$$

In the recurrences  $k = 0, 1, 2, 3$ , the network produces the following activations and responses, respectively:

**Step 1:**  $k = 0$ :

$$\begin{aligned}\mathbf{net}^0 &= \begin{bmatrix} 1 & -0.2 & -0.2 \\ -0.2 & 1 & -0.2 \\ -0.2 & -0.2 & 1 \end{bmatrix} \begin{bmatrix} 0.777 \\ 0.333 \\ 0.555 \end{bmatrix} \begin{bmatrix} 0.599 \\ 0.067 \\ 0.333 \end{bmatrix}, \\ \mathbf{y}^1 &= \begin{bmatrix} 0.599 \\ 0.067 \\ 0.333 \end{bmatrix}\end{aligned}$$

**Step 2:**  $k = 1$ :

$$\mathbf{net}^1 = \begin{bmatrix} 0.520 \\ -0.120 \\ 0.120 \end{bmatrix}, \quad \mathbf{y}^2 = \begin{bmatrix} 0.520 \\ 0 \\ 0.120 \end{bmatrix}$$

**Step 3:**  $k = 2$ :

$$\mathbf{net}^2 = \begin{bmatrix} 0.480 \\ -0.14 \\ 0.096 \end{bmatrix}, \quad \mathbf{y}^3 = \begin{bmatrix} 0.480 \\ 0 \\ 0.096 \end{bmatrix}$$

**Step 4:**  $k = 3$ :

$$\mathbf{net}^3 = \begin{bmatrix} 0.461 \\ -0.115 \\ -10^{-7} \end{bmatrix}, \quad \mathbf{y}^4 = \begin{bmatrix} 0.461 \\ 0 \\ 0 \end{bmatrix}$$

This terminates the recursion, since for further recurrences all values of  $y^{k+1} = y^4$ , for  $k > 3$ . Notice that the thresholding began in Step 2 due to the conditions at the second neuron of the MAXNET. The activation value  $net_2^1 = -0.12$  has resulted in the output  $y_2^2 = 0$  in this step. This has obviously slowed down the rate of decrease of other nonzero outputs and, more importantly, of  $y_1$ . The result computed by the network after four recurrences indicates that the vector  $x$  presented at the input for the minimum Hamming distance classification has been at the smallest HD from  $s^{(1)}$  and, therefore, it represents the distorted character  $C$ . ■

Let us summarize the benefits of the discussed neural network architecture. As stated, the Hamming net and MAXNET jointly implement the optimum minimum bit error classification. Let us compare the size of this classifier with its Hopfield autoassociator network counterpart. Assume that a classifier needs to classify a 50-tuple vector belonging to one of the five classes. The Hamming network requires a total of 255 connections to 5 neurons. For comparison, a Hopfield autoassociative memory would involve 2450 connections to 50 neurons. In addition, Hopfield memory is sensitive to the HD parameter within the set of stored patterns. As a result of a random search for the minimum of the energy, Hopfield network may produce spurious responses. Also, the network has rather stringent capacity limits.

This comparison indicates that the Hamming network looks somewhat advantageous as a minimum HD classifier. Moreover, the network is rather straightforward to build and, excluding its MAXNET stage, is of the simple feedforward type. However, the Hamming network retrieves only the closest class index and not the entire prototype vector. The Hamming network is therefore not able to restore any of the key pattern vector entries, if input is a corrupted version of the stored prototype, but can only provide the closest similarity measure of the input to the set of prototypes. We should thus realize that the Hamming network is only a pure classifier and not an autoassociative memory. It provides passive classification and has no built-in data restoration mechanisms. Such mechanisms are naturally embedded, however, in recurrent autoassociative architectures.

The reader has undoubtedly noticed that Hamming network and MAXNET weights are designed by recording rather than through incremental learning, whether supervised or unsupervised. We decided to study Hamming networks in the chapter dealing with unsupervised learning networks, however, because Hamming distance classifiers use the same similarity measure as many of the neural networks that learn in an unsupervised environment. Namely, the network detects the match between the stored prototype and the key vector by computing and comparing their scalar product. This is accomplished by encoding the prototype vector into respective weights of the neurons being the class indicator for this specific prototype.

## 7.2

### UNSUPERVISED LEARNING OF CLUSTERS

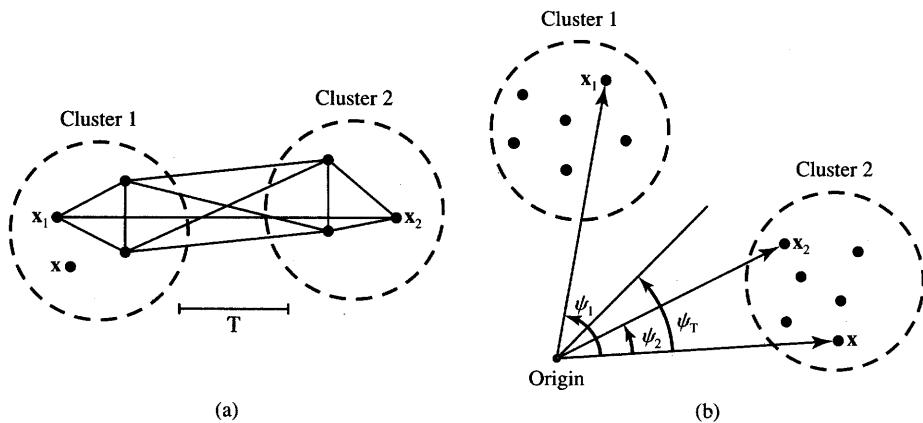
Our discussion thus far has focused on the classification, recognition, and association tasks performed by neural networks that have been previously trained. The training has been either supervised, or network parameters have been explicitly computed based on the design requirements in lieu of training. We have consistently assumed the existence of a training set containing both inputs and required responses. In this section we present unsupervised classification learning. The learning is based on clustering of input data. No *a priori* knowledge is assumed to be available regarding an input's membership in a particular class. Rather, gradually detected characteristics and a history of training will be used to assist the network in defining classes and possible boundaries between them.

Such unsupervised classification, called *clustering*, can possibly be thought of as primary compared to a classification of membership in classes that have already been formed. In fact, evolutionary learning by humans must have originated thousands of years ago in an unsupervised form, since there were no teachers, instructions, templates, or books then. Over the years, all known living things, for example, have first been classified as belonging to certain clusters according to their observable characteristics. The single and sensible criterion used has been the objects' similarities within a cluster. As a result of this grouping, clusters were labeled with appropriate names that best reflect their characteristics. Later on, objects were given names following more detailed classification criteria that were developed within clusters.

The objective of clustering neural networks discussed in this section is to categorize or cluster data. The classes must first be found from the correlations of an input data stream. Since the network actually deals with unlabeled data, the clustering should be followed by labeling clusters with appropriate category names or numbers. This process of providing the category of objects with a label is usually termed as *calibration*.

#### Clustering and Similarity Measures

Clustering is understood to be the grouping of similar objects and separating of dissimilar ones. Suppose we are given a set of patterns without any information as to the number of classes that may be present in the set. The clustering problem in such a case is that of identifying the number of classes according to a certain criterion, and of assigning the membership of the patterns in these classes. The clustering technique presented below assumes that the number of classes is known *a priori*. The pattern set  $\{x_1, x_2, \dots, x_N\}$  is submitted to the



**Figure 7.5** Measures of similarity for clustering data: (a) distance and (b) a normalized scalar product.

input to determine decision functions required to identify possible clusters. Since no information is available from the teacher as far as the desired classifier's responses, we will use the similarity of incoming patterns as the criterion for clustering.

To define a cluster, we need to establish a basis for assigning patterns to the domain of a particular cluster. The most common similarity rule, already used in Chapter 3, is the Euclidean distance between two patterns  $\mathbf{x}$ , for  $\mathbf{x}_i$  defined as

$$\|\mathbf{x} - \mathbf{x}_i\| = \sqrt{(\mathbf{x} - \mathbf{x}_i)'(\mathbf{x} - \mathbf{x}_i)} \quad (7.7)$$

This rule of similarity is simple: the smaller the distance, the closer the patterns. Using (7.7), the distances between all pairs of points are computed. A distance  $T$  can then be chosen to discriminate clusters. The value  $T$  is understood as the maximum distance between patterns within a single cluster. Figure 7.5(a) shows an example of two clusters with a  $T$  value chosen to be greater than the typical within-cluster distance but smaller than the between-cluster distance.

Another similarity rule is the cosine of the angle between  $\mathbf{x}$  and  $\mathbf{x}_i$ :

$$\cos \psi = \frac{\mathbf{x}' \mathbf{x}_i}{\|\mathbf{x}\| \cdot \|\mathbf{x}_i\|} \quad (7.8)$$

This rule is particularly useful when clusters develop along certain principal and different axes as shown in Figure 7.5(b). For  $\cos \psi_2 < \cos \psi_1$ , pattern  $\mathbf{x}$  is more similar to  $\mathbf{x}_2$  than to  $\mathbf{x}_1$ . It would thus be natural to group it with the second of the two apparent clusters. To facilitate this decision, the threshold angle  $\psi_T$  can be chosen to define the minimum angular cluster distance. It should be noted, however, that the measure defined in (7.8) should be used according to certain additional qualifications. If the angular similarity criterion (7.8) is to be efficient, vectors  $\mathbf{x}_1$ ,  $\mathbf{x}_2$ , and  $\mathbf{x}$  should be of comparable, or better, identical lengths.

A number of traditional cluster search algorithms such as maximum-distance,  $K$ -means, and isodata, are used in pattern recognition techniques (Tou and Gonzalez 1974; Duda and Hart 1973). In our presentation, we will focus on techniques that involve weight training and employ connectionist algorithms for building clusters rather than on conventional clustering methods.

### Winner-Take-All Learning

The network discussed in this section classifies input vectors into one of the specified number of  $p$  categories according to the clusters detected in the training set  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ . The training is performed in an unsupervised mode, and the network undergoes the self-organization process. During the training, dissimilar vectors are rejected, and only one, the most similar, is accepted for weight building. As mentioned, it is impossible in this training method to assign network nodes to specific input classes in advance. It is equally impossible to predict which neurons will be activated by members of particular clusters at the beginning of the training. This node to cluster assignment is, however, easily done by calibrating the network after training.

The network to be trained is called the *Kohonen network* (Kohonen 1988) and is shown in Figure 7.6(a). The processing of input data  $\mathbf{x}$  from the training set  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ , which represents  $p$  clusters, follows the customary expression

$$\mathbf{y} = \Gamma[\mathbf{W}\mathbf{x}] \quad (7.9)$$

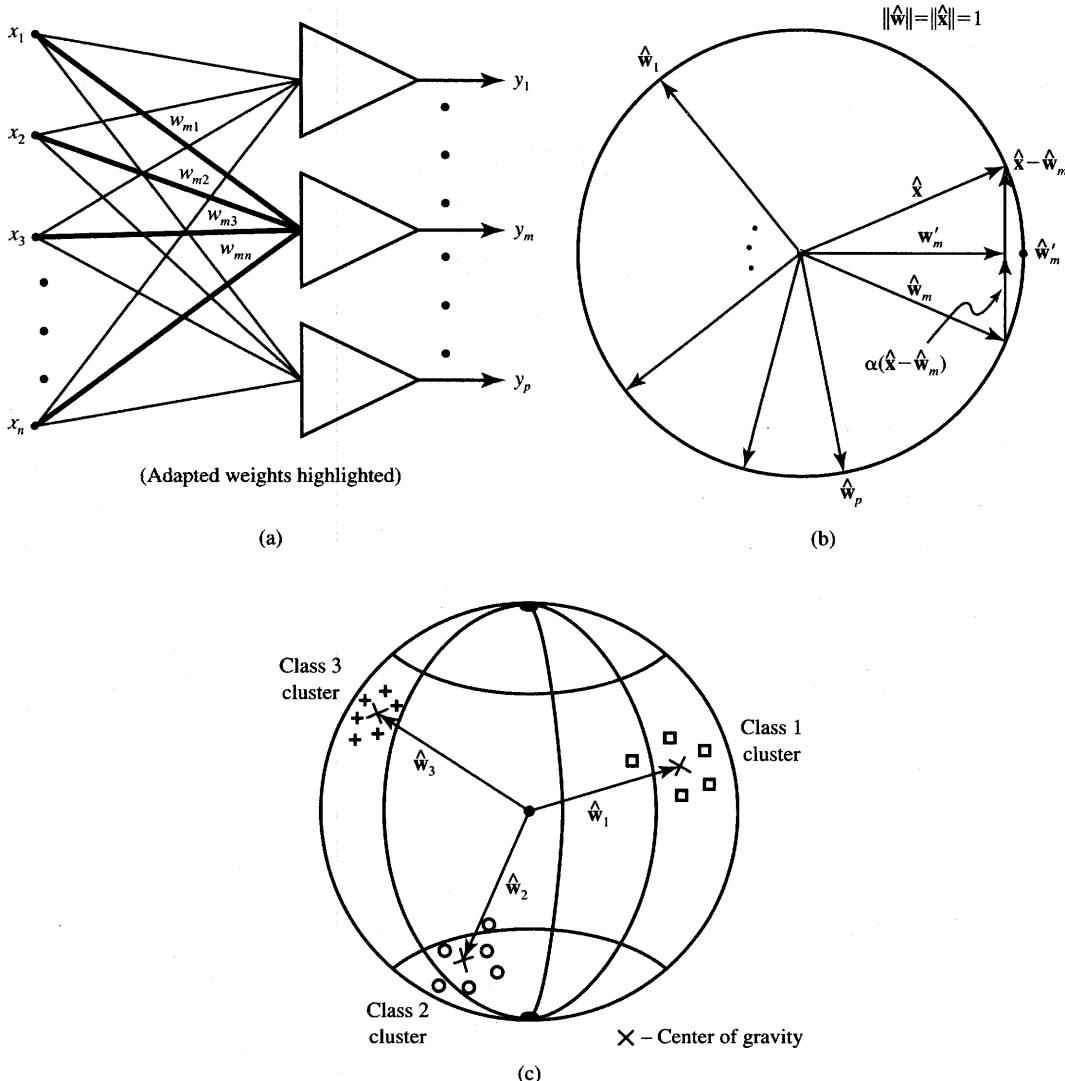
with diagonal elements of the operator  $\Gamma$  being continuous activation functions operating componentwise on entries of vector  $\mathbf{Wx}$ . The processing by the layer of neurons is instantaneous and feedforward. To analyze network performance, we rearrange the matrix  $\mathbf{W}$  to the following form:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^t \\ \mathbf{w}_2^t \\ \vdots \\ \mathbf{w}_p^t \end{bmatrix}$$

where

$$\mathbf{w}_i = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{in} \end{bmatrix}, \quad \text{for } i = 1, 2, \dots, p$$

is the column vector equal to the  $i$ 'th row of the weight matrix  $\mathbf{W}$ . Component weights of  $\mathbf{w}_m$  are highlighted in Figure 7.6(a) showing a winner-take-all learning network. The learning algorithm treats the set of  $p$  weight vectors as variable



**Figure 7.6** Winner-take-all learning rule: (a) learning layer, (b) vector diagram, and (c) weight vectors on a unity sphere for  $p = n = 3$ .

vectors that need to be learned. Prior to the learning, the normalization of all weight vectors is required:

$$\hat{\mathbf{w}}_i \triangleq \frac{\mathbf{w}_i}{\|\mathbf{w}_i\|}, \quad \text{for } i = 1, 2, \dots, p$$

The weight adjustment criterion for this mode of training is the selection of  $\hat{\mathbf{w}}_i$

such that

$$\|\mathbf{x} - \hat{\mathbf{w}}_m\| = \min_{i=1,2,\dots,p} \{\|\mathbf{x} - \hat{\mathbf{w}}_i\|\} \quad (7.10)$$

The index  $m$  denotes the winning neuron number corresponding to the vector  $\hat{\mathbf{w}}_m$ , which is the closest approximation of the current input  $\mathbf{x}$ . Let us see how this learning should proceed in terms of weight adjustments. The left side of Equation (7.10) can be rearranged to the form

$$\|\mathbf{x} - \hat{\mathbf{w}}_m\| = (\mathbf{x}'\mathbf{x} - 2\hat{\mathbf{w}}_m'\mathbf{x} - 1)^{1/2} \quad (7.11)$$

It is obvious from (7.11) that searching for the minimum of  $p$  distances as on the right side of (7.10) corresponds to finding the maximum among the  $p$  scalar products

$$\hat{\mathbf{w}}_m'\mathbf{x} = \max_{i=1,2,\dots,p} (\hat{\mathbf{w}}_i'\mathbf{x}) \quad (7.12)$$

The left side of Equation (7.12) is the activation value of the “winning” neuron which has the largest value  $\text{net}_i$ ,  $i = 1, 2, \dots, p$ . When using the scalar product metric of similarity as in (7.12), the synaptic weight vectors should be modified accordingly so that they become more similar to the current input vector. With the similarity criterion being  $\cos \psi$  as in (7.8), the weight vector lengths should be identical for this training approach. However, their directions should be modified. Intuitively, it is clear that a very long weight vector could lead to a large output of its neuron even if there were a large angle between the weight vector and the pattern. This explains the need for weight normalization.

After the winning neuron has been identified and declared a winner, its weights must be adjusted so that the distance (7.10) is reduced in the current training step. Thus,  $\|\mathbf{x} - \mathbf{w}_m\|$  must be reduced, preferably along the gradient direction in the weight space  $w_{m1}, w_{m2}, \dots, w_{mn}$

$$\nabla_{\mathbf{w}_m} \|\mathbf{x} - \mathbf{w}_m\|^2 = -2(\mathbf{x} - \mathbf{w}_m) \quad (7.13a)$$

Since vectors  $\mathbf{x}$  have a certain probability distribution within a cluster and we are dealing with a single adjustment step due to the single realization of the input, only a fraction of the increment in (7.13a) should be involved in producing the sensible weight adjustments. It seems reasonable to reward the weights of the winning neuron with an increment of weight in the negative gradient direction, thus in the direction  $\mathbf{x} - \mathbf{w}_m$ . We thus have

$$\Delta \hat{\mathbf{w}}'_m = \alpha(\mathbf{x} - \hat{\mathbf{w}}_m) \quad (7.13b)$$

where  $\alpha$  is a small learning constant selected heuristically, usually between 0.1 and 0.7. The remaining weight vectors  $\hat{\mathbf{w}}_i$ ,  $i \neq m$ , are left unaffected. Note that the rule is identical to Equation (2.46) introduced in Chapter 2 in the framework of learning rules.

Using a superscript to index the weight updates and restating the update criterion (7.10), the learning rule (7.13b) in the  $k$ 'th step can be rewritten in a

more formal way as follows:

$$\hat{\mathbf{w}}_m^{k+1} = \hat{\mathbf{w}}_m^k + \alpha^k (\mathbf{x} - \hat{\mathbf{w}}_m^k) \quad (7.13c)$$

$$\hat{\mathbf{w}}_i^{k+1} = \hat{\mathbf{w}}_i^k, \quad \text{for } i \neq m \quad (7.13d)$$

where  $\alpha^k$  is a suitable learning constant and  $m$  is the number of the winning neuron selected based on the scalar product comparison as in (7.12). While learning continues and clusters are developed, the network weights acquire similarity to input data within clusters. To prevent further unconstrained growth of weights,  $\alpha$  is usually reduced monotonically and the learning slows down.

Learning according to Equations (7.12) and (7.13) is called "winner-take-all" learning, and it is a common competitive and unsupervised learning technique. The winning node with the largest  $\text{net}_i$  is rewarded with a weight adjustment, while the weights of the others remain unaffected.

This mode of learning is easy to implement as a computer simulation; one merely searches for the maximum response and rewards the winning weights only. In a real network it is possible to implement a winner-take-all layer by using units with lateral inhibition to the other neurons in the form of inhibitory connections. At the same time, the neuron should possess excitatory connections to itself like in the MAXNET network. Since all the weights must, in addition to providing stable operation of the layer, be modifiable as a result of combined excitatory/inhibitory interactions, such layers may not be easy to develop as a physical neural network capable of meaningful learning.

Let us look at the impact of the learning rule (7.13c) and (7.13d) on the performance of the network. The rule should increase the chances of winning by the  $m$ 'th neuron as in (7.12) for repetition of the same input pattern using the updated weights. If the requirement holds, then inequality (7.14a) should be valid for the new weights  $\hat{\mathbf{w}}'_m$

$$\hat{\mathbf{w}}_m^t \mathbf{x} < (\hat{\mathbf{w}}_m^t + \Delta \hat{\mathbf{w}}_m^t) \mathbf{x} \quad (7.14a)$$

Using (7.13) we obtain

$$\Delta \hat{\mathbf{w}}_m^t \mathbf{x} > 0 \quad (7.14b)$$

or

$$\mathbf{x}^t \mathbf{x} - \hat{\mathbf{w}}_m^t \mathbf{x} > 0 \quad (7.14c)$$

which is equivalent to

$$\|\mathbf{x}^t\| \cdot \|\mathbf{x}\| \cos 0 - \|\hat{\mathbf{w}}_m^t\| \cdot \|\mathbf{x}\| \cos \psi > 0 \quad (7.14d)$$

Assuming normalized vectors  $\hat{\mathbf{x}} = \mathbf{x}$  reduces (7.14d) to

$$1 - \cos \psi > 0 \quad (7.14e)$$

where  $\psi = \angle(\hat{\mathbf{w}}, \hat{\mathbf{x}})$ . Since (7.14e) is always true, the winner-take-all learning rule produces an update of the weight vector in the proper direction.

*(written)*

It is instructive to observe the geometrical interpretation of the rule. Consider that weights are represented as vectors in Figure 7.6(b). Assume that in this step the normalized input vector denoted as  $\hat{x}$  and the vector  $\hat{w}_m$  yield the maximum scalar product  $\hat{w}_i' \hat{x}$ , for  $i = 1, 2, \dots, p$ . Next, a difference vector  $\hat{x} - \hat{w}_m$  is created as shown. To implement the rule of (7.13c) and (7.13d) for  $x = \hat{x}$ , an increment of the weight vector is computed as a fraction of  $\hat{x} - \hat{w}_m$ . The result of weight adjustment in this training step is mainly the rotation of the weight vector  $\hat{w}_m$  toward the input vector without a significant length change. The adjusted weight vector results as  $w'_m$  and is of a length below unity. To begin with the new training step,  $w'_m$  must be renormalized. Let us notice that another input belonging to the  $m$ 'th cluster would make the vector  $w_m$  even more representative of the cluster  $m$ .

In the long term this learning mode leads to the weight vectors that approximate the ensembles of past winning input vectors. However, since the weights are adjusted in proportion to the number of events that end up with weight adjustments, this network reacts to the probability of occurrence of inputs. In this context, the network may be used as a clustering network for the particular probability of training vectors coming from each cluster. After the learning is completed, each  $\hat{w}_i$  represents the centroid of an  $i$ 'th decision region,  $i = 1, 2, \dots, p$ , created in the  $n$ -dimensional space of the pattern data. On the other side, the network possesses an interesting feature sensitivity, which will be discussed later in this chapter in more detail. In summary, vectors  $\hat{w}$  after training will become organized much like the set of example vectors  $\hat{x}$  used for training.

Note that the neurons' activation function is of no relevance for this learning mode. Figure 7.6(c) illustrates an example of three weight vectors of unity length that have acquired the direction of an average pattern cluster. The case shown is for three clusters and three-dimensional input patterns. It can be seen that the normalized weights approximate the centers of gravity of their respective clusters, which are represented each by a point on the surface of a unit sphere and marked by a cross.

Another learning extension is possible for this network when the proper class for some patterns is known *a priori* (Simpson 1990). Although this means that the encoding of data into weights is then becoming supervised, this information accelerates the learning process significantly. Weight adjustments are computed in the supervised mode as in (7.13b) and only for correct classifications. For improper clustering responses of the network, the weight adjustment carries the opposite sign compared to formula (7.13b). We may thus notice that  $\alpha > 0$  for proper node responses, and  $\alpha < 0$  otherwise, in the supervised learning mode for the Kohonen layer.

Another modification of the winner-take-all learning rule is that both the winners' and losers' weights are adjusted in proportion to their level of responses. This may be called *leaky competitive learning* and should provide more subtle learning in the cases for which clusters may be hard to distinguish.

### Recall Mode

The network trained in the winner-take-all mode responds instantaneously during feedforward recall at all  $p$  neuron outputs. The response is computed according to (7.9). Note that the layer now performs as a filter of the input vectors such that the largest output neuron is found as follows

$$y_m = \max(y_1, y_2, \dots, y_p)$$

and the input is identified as belonging to cluster  $m$ . In general, the neurons' activation functions should be continuous in this network. For some applications, however,  $y_m = 1$  and  $y_i = 0$ ,  $i \neq m$ , must be set in the recall mode of the clustering layer. In this way, for example, the weights of the following layer can be fanned out from the activated node of this previous layer while other nodes remain suppressed.

Before a one-to-one vector-to-cluster mapping can be made after the network is trained in the unsupervised mode, it needs to be calibrated in a supervised environment. The calibration involves the teacher applying a sequence of  $p$  best matching class/cluster inputs and labeling the output nodes 1, 2, ...,  $p$ , respectively, according to their observed responses. Obviously, the calibrating labels assigned to the physical neurons of the layer would vary from training to training depending on the sequence of data within the training set, the training parameters, and the initial weights. Once the clustering network is labeled, it can perform as a cluster classifier in a local representation.

### Initialization of Weights

As stated before, preferably random initial weight vectors would be used for this training. This indicates that initial weights should be uniformly distributed on the unity hypersphere in  $n$ -dimensional pattern space. Self-organization of the network using the described training concept suffers from some limitations, however. One obvious deficiency related to a single-layer architecture is that linearly nonseparable patterns cannot be efficiently handled by this network. The second limitation is that network training may not always be successful even for linearly separable patterns. Weights may get stuck in isolated regions without forming adequate clusters. In such cases the training must be reinitialized with new initial weights, or noise superimposed on weight vectors during the training. After the weights have been trained to provide coarse clustering, the learning constant  $\alpha$  should be reduced to produce finer adjustments. This often results in finer weight tuning within each cluster.

One of the weight selection methods developed for training the network is called *convex combination* (Hecht-Nielsen 1987). In this method all weight

vectors are initialized at the value

$$\mathbf{w}_i^0 = \frac{1}{\sqrt{n}} [1 \ 1 \ \dots \ 1]^t, \quad \text{for } i = 1, 2, \dots, p$$

The learning starts at the weights as above and proceeds as in expression (7.13c) and (7.13d) with a very low  $\alpha$  value. This forces the weight vectors at the beginning of learning to be close to the input vectors and to have equal lengths. As learning progresses,  $\alpha$  is slowly increased. This allows for the gradual separation of weights according to the input clusters used for training. This procedure improves the chances for successful training, but does slow down the process.

The simple winner-take-all training algorithm activates only one output neuron for each input vector; it thus provides local representation. The training approach can be modified, however, to a form in which  $K$  winners in the competition are considered and awarded weight increases accordingly. Such learning is called *multiple-winner unsupervised learning*. The winning  $K$  neurons would be the ones best matching the input vector, instead of a single one as in the winner-take-all mode. The outputs of winning neurons can be set so that they sum to unity. This interpolation process usually leads to an increased mapping accuracy in applications using the winner-take-all layer as the first layer. A network using this approach is said to be operating in an interpolative mode (Hecht-Nielsen 1987).

## EXAMPLE 7.2

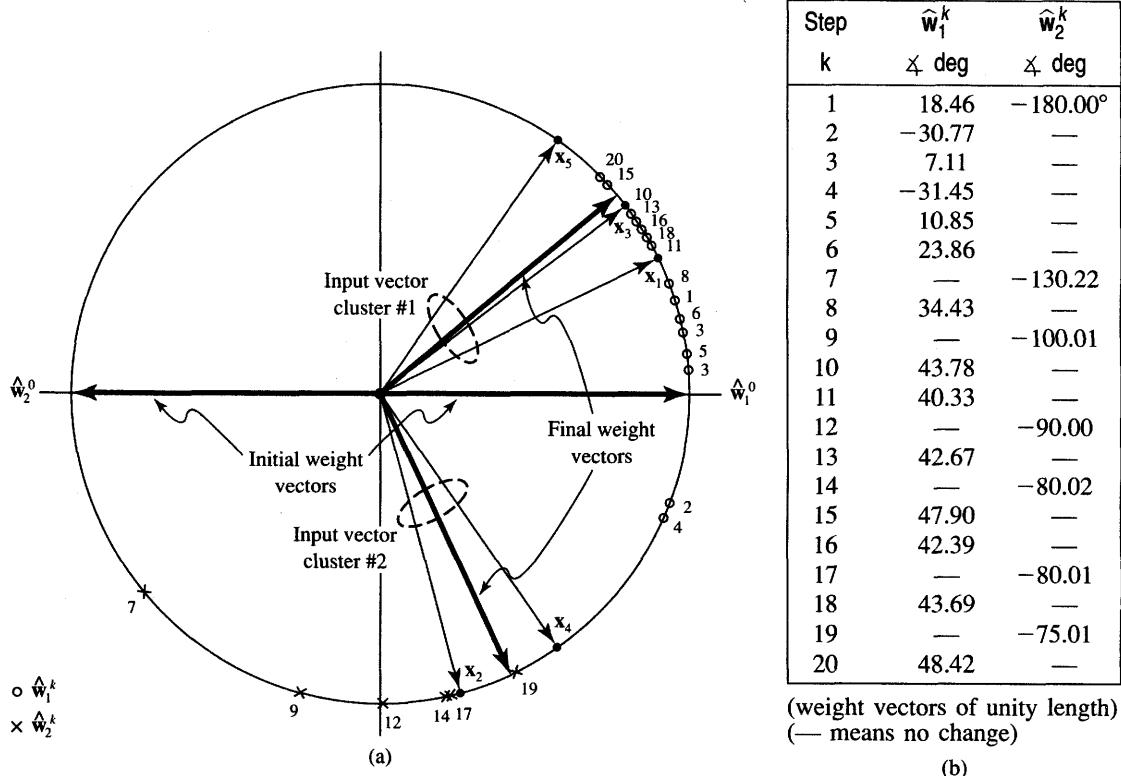
We present here an example of a two-cluster case and provide a simple geometrical interpretation of winner-take-all network training based on the criterion of Equation (7.12). The iterative weight adjustment is unsupervised. The patterns are from a normalized example training set:

$$\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\} = \left\{ \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.1736 \\ -0.9848 \end{bmatrix}, \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}, \begin{bmatrix} 0.342 \\ -0.9397 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.8 \end{bmatrix} \right\} \quad (7.15a)$$

The set of test vectors is shown in Figure 7.7(a). Two output neurons,  $p = 2$ , have been selected to identify two possible clusters. The normalized initial weights selected at random are

$$\mathbf{w}_1^0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{w}_2^0 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

The inputs are submitted in ascending sequence from the set as in (7.15a) and recycled  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_5, \mathbf{x}_1, \mathbf{x}_2, \dots$ . Let us evaluate the adjustments of weight vectors during learning. The reader can verify that for  $\alpha = 1/2$  we



**Figure 7.7** Competitive learning network of Example 7.2: (a) training patterns and weight assignments and (b) weight learning, Steps 1 through 20.

obtain from (7.13b) and (7.13c) after renormalization

$$\hat{w}_1^1 = \begin{bmatrix} 0.948 \\ 0.316 \end{bmatrix}, \quad w_2^1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

This is because the first neuron becomes the winner with  $x_1$  as input. For convenience we convert the training set (7.15a) to the polar form as follows:

$$\{x_1, x_2, x_3, x_4, x_5\} = \{1 \measuredangle 36.87^\circ, 1 \measuredangle -80^\circ, 1 \measuredangle 45^\circ, 1 \measuredangle -70^\circ, 1 \measuredangle 53.13^\circ\} \quad (7.15b)$$

Since  $\hat{w}_1^1 = 1 \measuredangle 18.46^\circ$ , it can be seen that  $\hat{w}_1$  has moved from its initial position  $1 \measuredangle 0^\circ$  toward the potential first quadrant cluster. Note that the weights of the second neuron,  $\hat{w}_2$ , are unaffected in the first training step. The next training step with input  $x_2$  produces  $\hat{w}_1^2 = 1 \measuredangle -30.77^\circ$ , and  $\hat{w}_2^2 = 1 \measuredangle 180^\circ$ , since node 1 is again the winner. The iterative weight adjustments continue. Figure 7.7(b) shows the tabulated results of weight iterations for  $k \leq 20$ .

Analysis of two data clusters produced by the network indicates that  $x_1$ ,  $x_3$ , and  $x_5$  belong to the first quadrant cluster with the center at

$$\frac{x_1 + x_3 + x_5}{3} = 1445^\circ$$

Similarly,  $x_2$  and  $x_4$  form another cluster in the fourth quadrant located at

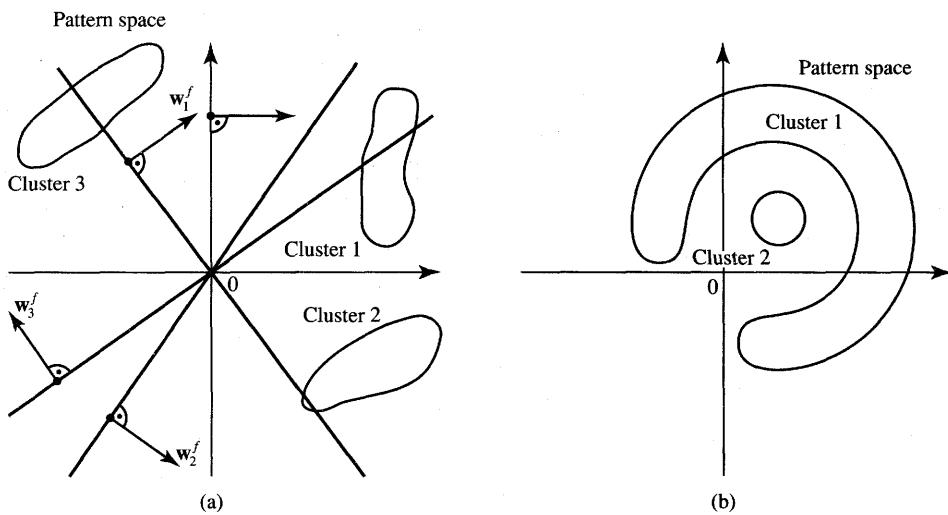
$$\frac{x_2 + x_4}{2} = 14 - 75^\circ$$

For large  $k$ , weight vectors tend to become stabilized as follows: the first neuron weights center around  $1445^\circ$  and the second neuron weights center around  $14 - 75^\circ$ . The geometrical interpretation of weight vector displacement during learning shown on the unity circle of Figure 7.7(b) also indicates that the learning has been successful. Weight vectors  $\hat{w}^1$ ,  $\hat{w}^2$ , ... are marked in the unity circle as points indexed 1, 2, 3, ... 20. We may notice, however, that the first neuron won all of the first six competitions in a row and initially started representing both data clusters at the same time. Despite that initial takeover, during the second training cycle, the cluster containing  $x_2$  and  $x_4$  has been detected and identified by the second neuron. Its respective weights were later reinforced during cycles 3 and 4. Finally, the weight values of the second neuron were organized by acquiring the values of second cluster inputs.

### Separability Limitations

We now see that winner-take-all learning can successfully produce single-layer clustering/classifying networks. However, the networks will only be trainable using the criterion of Equation (7.12) if classes/clusters of patterns are linearly separable from other classes by hyperplanes passing through origin. This is illustrated in Figure 7.8(a), which shows an example of three-class classification in the pattern space as generated by the trained network.

In the case shown, three distinct clusters can be identified and classified. The weights responsible for clustering are marked  $w_1^f$ ,  $w_2^f$ , and  $w_3^f$ . The scalar product metric-based clustering is impossible, however, for patterns distributed as shown in Figure 7.8(b). Since only a single neuron has the strongest response, the winner-take-all rule precludes that the same neuron would at the same time respond if patterns are on the negative side of the partitioning hyperplane. Adding trainable thresholds to neurons' inputs may improve separability conditions. Even then, however, the two neuron network will not be able to learn the solution of an XOR problem, or similar linearly nonseparable tasks. An excessive number of neurons created in the winner-take-all layer could certainly also be of significance when learning difficult clustering.



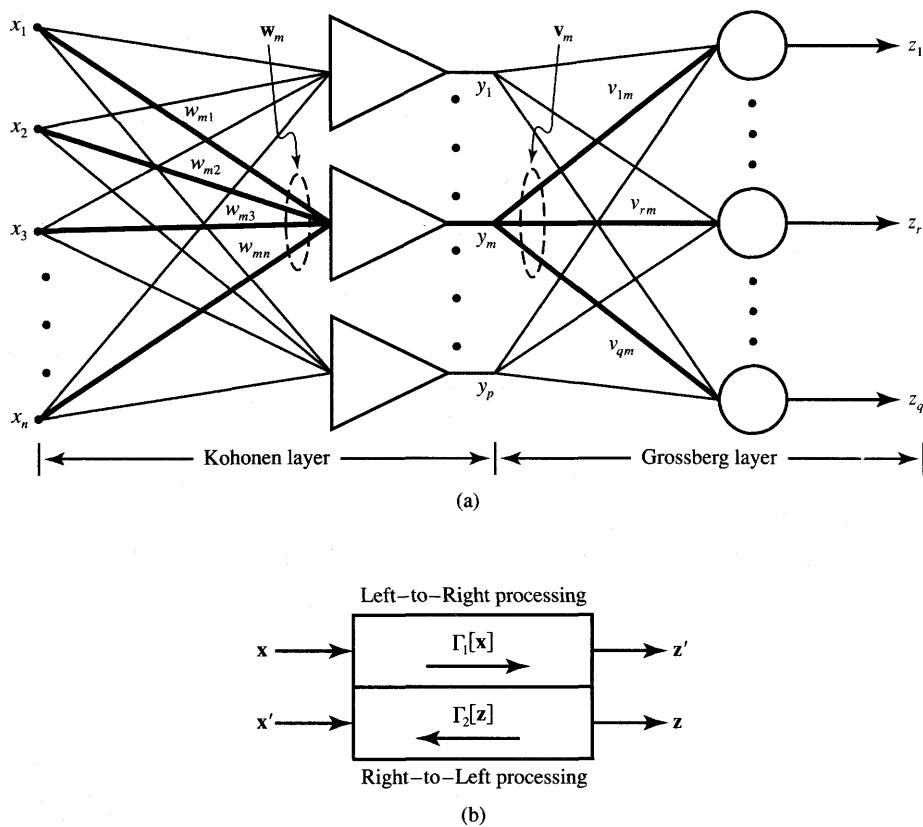
**Figure 7.8** Separability illustration for a winner-take-all learning network ( $w_f$  denotes the final weight vectors): (a) possible classification and (b) impossible classification.

To ensure the separability of clusters with *a priori* unknown numbers of training clusters, the unsupervised training can be performed with an excessive number of neurons, which provides a certain separability safety margin. During the training, some neurons are likely not to develop their weights, and if their weights change chaotically, they will not be considered as indicative of clusters. Therefore, such weights can be omitted during the recall phase, since their outputs do not provide any essential clustering information. The weights of the remaining neurons should settle at values that are indicative of clusters.

## 7.3

### COUNTERPROPAGATION NETWORK

The counterpropagation network is a two-layer network consisting of two feedforward layers. The network was introduced by Hecht-Nielsen (1987, 1988). In its simplest version, it is able to perform vector-to-vector mapping similar to heteroassociative memory networks. Compared to bidirectional associative memory, there is no feedback and delay activated during the recall operation mode. The advantage of the counterpropagation network is that it can be trained to perform associative mappings much faster than a typical two-layer network. The counterpropagation network is useful in pattern mapping and associations, data compression, and classification.



**Figure 7.9** Counterpropagation network: (a) feedforward part and (b) full counterpropagation network.

The network is essentially a partially self-organizing look-up table that maps  $\mathbb{R}^n$  into  $\mathbb{R}^q$  and is taught in response to a set of training examples. The objective of the counterpropagation network is to map input data vectors  $x_i$  into bipolar binary responses  $z_i$ , for  $i = 1, 2, \dots, p$ . We assume that data vectors can be arranged into  $p$  clusters, and that the training data are noisy versions of vectors  $x_i$ . The essential part of the counterpropagation network structure is shown in Figure 7.9. It resembles, at first glance, the class of layered feedforward networks covered in Chapter 4. However, counterpropagation combines two different, novel learning strategies, and neither of them is the gradient descent technique. The network's recall operation is also different than for any previously seen architecture.

The first layer of the network is the Kohonen layer, which is trained in the unsupervised winner-take-all mode described in the previous section. Each of the Kohonen layer neurons represents an input cluster, or pattern class, so if the layer works in local representation, this particular neuron's input and response are the

largest. Similar input vectors belonging to the same cluster activate the same  $m$ 'th neuron of the Kohonen layer among all  $p$  neurons available in this layer. Note that first-layer neurons are assumed with continuous activation function during learning. However, during recall they respond with the binary unipolar values 0 and 1. Specifically, when recalling with input representing a cluster, for example,  $m$ , the output vector  $\mathbf{y}$  of the Kohonen layer becomes

$$\begin{bmatrix} y_1 & y_2 & \cdots & y_m & \cdots & y_p \end{bmatrix} = [0 \ 0 \ \cdots \ 1 \ \cdots \ 0] \quad (7.16)$$

Such a response can be generated as a result of lateral inhibitions within the layer which would need to be activated during recall in a physical system. The second layer is called the *Grossberg layer* due to its outstar learning mode (Grossberg 1974, 1982) as discussed in Chapter 2. The Grossberg layer, with weights  $v_{ij}$ , functions in a familiar manner

$$\mathbf{z} = \Gamma[\mathbf{V}\mathbf{y}] \quad (7.17)$$

with diagonal elements of the operator  $\Gamma$  being a  $\text{sgn}(\cdot)$  function operating componentwise on entries of the vector  $\mathbf{V}\mathbf{y}$ . Let us denote the column vectors of the weight matrix  $\mathbf{V}$  as  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m, \dots, \mathbf{v}_p$ . Now, each weight vector  $\mathbf{v}_m$  for  $i = 1, 2, \dots, p$ , contains entries that are weights fanning out from the  $m$ 'th neuron of the Kohonen layer. Substituting (7.16) into (7.17) results in

$$\mathbf{z} = \Gamma[\mathbf{v}_m] \quad (7.18)$$

$$\text{where } \mathbf{v}_m = \begin{bmatrix} v_{1m} & v_{2m} & \cdots & v_{qm} \end{bmatrix}^t$$

As can be seen, the operation of this layer with bipolar binary neurons is simply to output  $z_i = 1$  if  $v_{im} > 0$ , and  $z_i = -1$  if  $v_{im} < 0$ , for  $i = 1, 2, \dots, q$ . By assigning just any positive and negative values for weights  $v_{im}$  highlighted in Figure 7.9, a desired vector-to-vector mapping  $\mathbf{x} \rightarrow \mathbf{y} \rightarrow \mathbf{z}$  can be implemented by this architecture. This is done under the assumption that the Kohonen layer responds as expressed in (7.16). The target vector  $\mathbf{z}$  for each cluster must be available for learning so that the weight components of  $\mathbf{v}_m$  can be appropriately made equal to +1 or -1 according to

$$\mathbf{v}_m = \mathbf{z} \quad (7.19)$$

However, this is a somewhat oversimplified weight learning rule for this layer. This rule, which is of the batch type rather than incremental, would be appropriate if no statistical relationship existed between input and output vectors within the training pairs  $(\mathbf{x}, \mathbf{z})$ . In practice, such relationships often exist and they also would need to be established in the network during training.

As discussed in the previous section, the training rule for the Kohonen layer has involved the adjustment of weight vectors in proportion to the probability of the occurrence and distribution of winning events. Using the outstar learning

rule of Equation (7.17) incrementally and not binarily as in (7.19) permits us to treat a stationary additive noise in output  $\mathbf{z}$  in a manner similar to the way we considered distributed clusters during the training of the Kohonen layer with "noisy" inputs. The outstar learning rule makes use of the fact that the learning of vector pairs, denoted by the set of mappings  $\{(\mathbf{x}_1, \mathbf{z}_1), \dots, (\mathbf{x}_p, \mathbf{z}_p)\}$ , will be done gradually and thus involve eventual statistical balancing within the weight matrix  $\mathbf{V}$ . The supervised learning rule for this layer in such a case becomes incremental and takes the form of the outstar learning rule defined in (2.48):

$$\Delta \mathbf{v}_m = \beta(\mathbf{z} - \mathbf{v}_m) \quad (7.20)$$

where  $\beta$  is set to approximately 0.1 at the beginning of learning and is gradually reduced during the training process (Wasserman 1989). Index  $m$  denotes the number of the winning neuron in the Kohonen layer. Vectors  $\mathbf{z}_i$ ,  $i = 1, 2, \dots, p$ , used for training are stationary random process vectors with statistical properties that make the training plausible.

Note that the supervised outstar rule learning according to (7.20) starts after completion of the unsupervised training of the first layer. Also, as indicated, the weight of the Grossberg layer is adjusted if and only if it fans out from a winning neuron of the Kohonen layer. As training progresses, the weights of the second layer tend to converge to the average value of the desired outputs. Let us also note that the unsupervised training of the first layer produces active outputs at indeterminate positions. The second layer introduces ordering in the mapping so that the network becomes a desirable look-up memory table. During the normal recall mode, the Grossberg layer outputs weight values  $\mathbf{z} = \mathbf{v}_m$  connecting each output node to the winning first layer neuron. No processing, except for addition and  $\text{sgn}(\text{net})$  computation, is performed by the output layer neurons if outputs are binary bipolar vectors.

The network discussed and shown in Figure 7.9(a) is simply feedforward and does not refer to the counterflow of signals for which the original network was named. The full version of the counterpropagation network makes use of bidirectional signal flow. The entire network consists of the doubled network from Figure 7.9(a). It can be simultaneously both trained and operated in the recall mode in an arrangement such as that shown in Figure 7.9(b). This makes it possible to use it as an autoassociator according to the formula

$$\begin{bmatrix} \mathbf{z}' \\ \mathbf{x}' \end{bmatrix} = \begin{bmatrix} \Gamma_1[\mathbf{x}] \\ \Gamma_2[\mathbf{z}] \end{bmatrix} \quad (7.21)$$

Input signals generated by vector  $\mathbf{x}$ , input, and by vector  $\mathbf{z}$ , desired output, propagate through the bidirectional network in opposite directions. Vectors  $\mathbf{x}'$  and  $\mathbf{z}'$  are respective outputs that are intended to be approximations, or autoassociations, of  $\mathbf{x}$  and  $\mathbf{z}$ , respectively.

Let us summarize the main features of this architecture in its simple feed-forward version. The counterpropagation network functions in the recall mode

as a nearest match look-up table. The input vector  $\mathbf{x}$  finds the weight vector  $\mathbf{w}_m$  that is its closest match among  $p$  vectors available in the first layer. Then the weights that are entries of vector  $\mathbf{v}_m$ , and are fanning out from the winning  $m$ 'th Kohonen's neuron, after  $\text{sgn}(\cdot)$  computation, become binary outputs. Due to the specific training of the counterpropagation network, it outputs the statistical averages of vector  $\mathbf{z}$  associated with input  $\mathbf{x}$ . In practice, the network performs as well as a look-up table can do to approximate vector matching.

Counterpropagation can also be used as a continuous function approximator. Assume that the training pairs are  $(\mathbf{x}_i, \mathbf{z}_i)$  and  $\mathbf{z}_i = g(\mathbf{x}_i)$ , where  $g$  is a continuous function on the set of input vectors  $\{\mathbf{x}\}$ . The mean square error of approximation can be made as small as desired by choosing a sufficiently large number  $p$  of Kohonen's layer neurons. However, for the continuous function approximation, the network is not as efficient as error back-propagation trained networks, since it requires many more neurons for comparable accuracy (Hecht-Nielsen 1990). Counterpropagation networks can be used for rapid prototyping of a mapping and to speed up system development, since they typically require orders of magnitude fewer training cycles than is usually needed in error back-propagation training.

The counterpropagation can use a modified competitive training condition for the Kohonen layer. Thus far we have assumed that the winning neuron, for which weights are adjusted as in (7.13c) and (7.13d), is the one fulfilling condition (7.12) of yielding the maximum scalar product of the weights and the training pattern vector. Another alternative for training is to choose the winning neuron of the Kohonen layer such that instead of (7.12), the minimum distance criterion (7.10) is used directly according to the formula

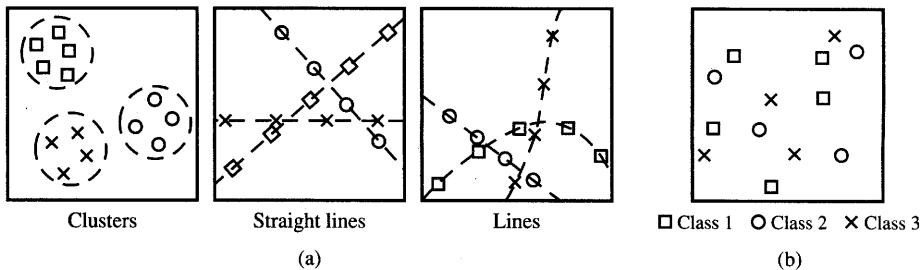
$$\|\mathbf{x} - \mathbf{w}_m\| = \min_{i=1,2,\dots,p} \{\|\mathbf{x} - \mathbf{w}_i\|\} \quad (7.22)$$

The remaining aspects of weight adaptation and of the training and recall mode would remain as described in the previous section for criterion (7.12). The only difference is that the weights do not have to be renormalized after each step in this training procedure.

## 7.4

### FEATURE MAPPING

In this section we focus on the feature mapping capabilities of neural networks. Special attention is devoted to mappings that can be arranged in geometrically regular self-organizing arrays. As discussed in Section 3.1, feature extraction is an important task both for classification or recognition and is often necessary as a preprocessing stage of data. In this way data can be transformed from high-dimensional pattern space to low-dimensional feature space. Our goal in this section is to identify neural architectures that can learn feature mapping



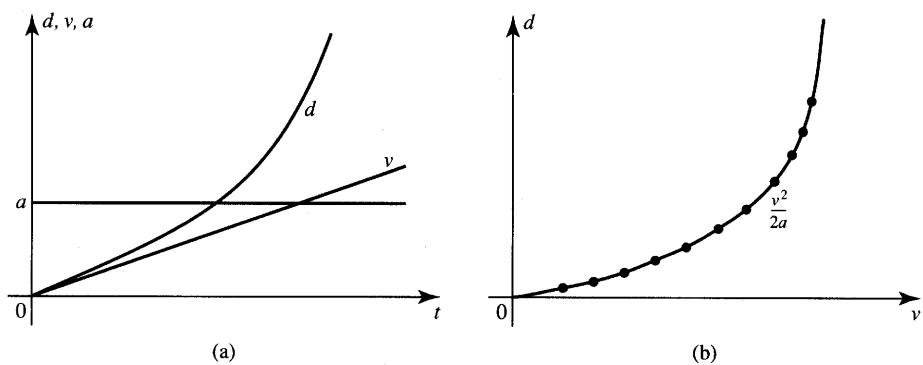
**Figure 7.10** Pattern structure: (a) natural similarity and (b) no natural similarity.

without supervision. Before introducing such architectures we will first reexamine the concept of features and their extraction.

The patterns in the pattern space can be distributed in two different structures. Although it is hard to define these precisely, we can say that the patterns in Figure 7.10(a) have a certain “natural” structure while the patterns in Figure 7.10(b) do not. The natural structure is often related to pattern clustering or their mutual location in the original pattern space. The natural pattern structure apparently makes the perception of them by a human observer easier. In contrast to the harmonious perception of such classes that show common features, a chaotic mixture of patterns is much more difficult to separate, memorize or classify in any way. The basic difference in identifying “natural” versus “unnatural” patterns structure is that the pattern space may be similar to our human perception space, or they may be different. The mismatch between the pattern space and our perception space is causing the apparent lack of pattern similarity or lack of their natural structure.

In our discussion we are interested in two aspects of mapping features. It is important to reduce the dimensionality of vectors in pattern space when representing these vectors in feature space. To facilitate perception, it is equally important to provide as natural a structure of features as possible. We are interested in easy perception of low-dimension patterns no matter how “unnatural” their distribution may be in the original pattern space.

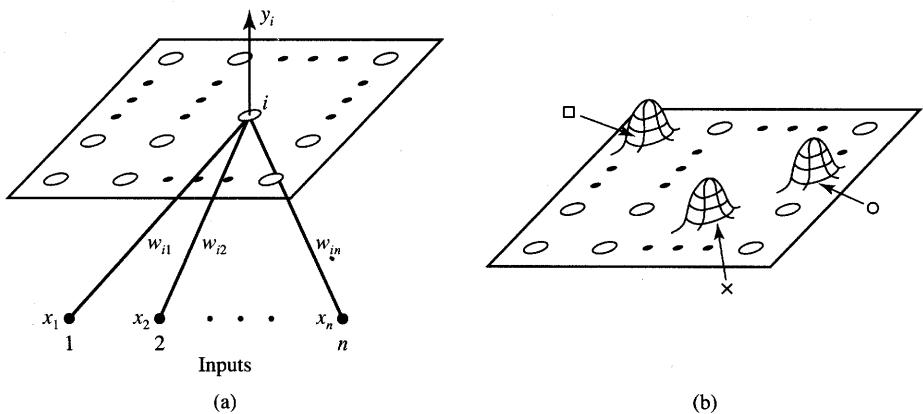
Some patterns exhibit inherently smaller dimensionality that can be perceived by superficial inspection. The inherent dimensionality reduction results from the explicit relationship that may exist between variables or the correlation between data. The number of degrees of freedom in such relationships can be lower than perceived initially. Figure 7.11 shows an example of constrained freedom pattern data. It depicts the longitudinal displacement  $d$  of a point moving with constant acceleration  $a$ . Assume that the pattern space is defined as speed  $v$  and distance  $d$  from the origin. Since the movement parameters are related as  $d = v^2 / (2a)$ , the  $d(v)$ , motion equation, can be considered one-dimensional in pattern space  $v, d$ . This limitation of dimensionality is due to the physical restriction of this motion on the number of degrees of freedom. Most often we



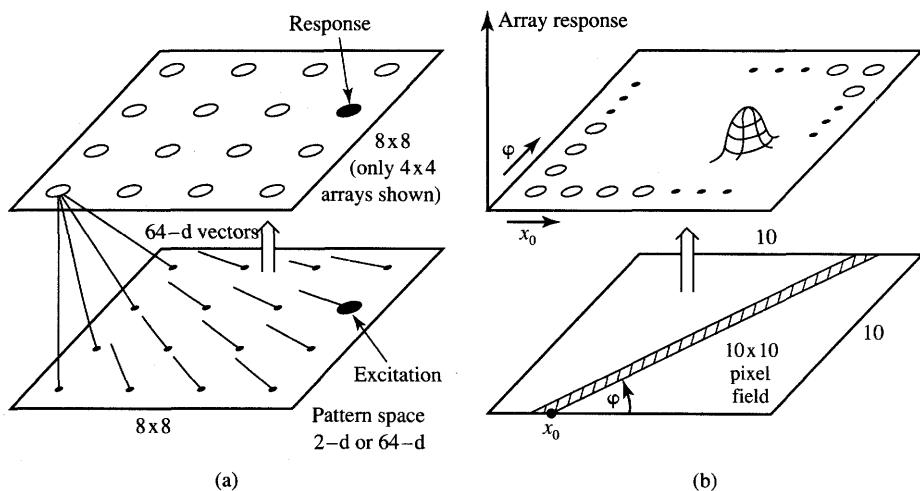
**Figure 7.11** Reduction of degree of freedom: (a) movement parameters versus time and (b) pattern space.

are interested in reducing the dimensionality of patterns in feature space in more involved cases when no analytical relationships such as the one documented in Figure 7.11 can be established.

Patterns in multidimensional space, which often can be clustered in one-, two- or three-dimensional feature space, may have a much more complicated structure in the original pattern space. Our goal is to find a self-organizing neural array that consists of neurons arranged in a virtual one-dimensional segment or in a two-dimensional array. Such an array is typically a rectangle or triangle, or a spatial lattice-type connection of neurons within a three-dimensional cuboid. The neurons of such an array are connected to the input vector as shown in Figure 7.12(a) for an example two-dimensional array. Note that each component in the input vector  $x$  is connected to each of the nodes, and the weight  $w_{ij}$  transmits the input  $x_j$  toward the  $i$ 'th node of the feature array.



**Figure 7.12** Mapping features of input  $x$  into a rectangular array of neurons: (a) general diagram and (b) desirable response peaks for Figure 7.10(b).



**Figure 7.13** Planar pattern to planar feature space mapping: (a) 64-dimensional excitation in pattern space and (b) translation of intercept and angle to planar feature array. [Adapted from Tattershall (1989). © Chapman & Hall; with permission.]

The output  $y_i$  of a neuron within the array is a function of similarity  $S$  between the input vector  $\mathbf{x}$  and  $\mathbf{w}_i$  defined as

$$y_i = f(S(\mathbf{x}, \mathbf{w}_i)) \quad (7.23)$$

We studied in Section 7.2 a simplified version of the unsupervised training of clusters using as the scalar product a similarity metric. Before we exploit similarity metrics any further, let us look at certain interesting aspects of dimensionality reduction.

It seems desirable to filter the input data onto feature space so that the patterns in the image space, as shown in Figure 7.10(b), are mapped into one of the easy to perceive clustering formats. An acceptable arrangement would be that of the leftmost square of Figure 7.10(a). Using the extraction of features architecture of Figure 7.12(a), the planar layer of neurons, called the *feature array*, should possibly display distinct peaks when excited with original patterns. Thus, inputs from classes 1, 2, and 3 should preferably result in perceptually separated excitation peaks as shown in Figure 7.12(b). In the example case illustrated, we map a two-dimensional pattern space into a two-dimensional feature space.

A question arises about the dimensionality of the original planar data. Assume that a number of pattern points are given on the plane or in two-dimensional pattern space. A computational experiment has been designed in which a two-dimensional top array of neurons has been trained using planar input data from the bottom plane (Tattershall 1989). As shown in Figure 7.13(a), the input patterns' dimensionality has been made equal to the number of actual planar inputs.

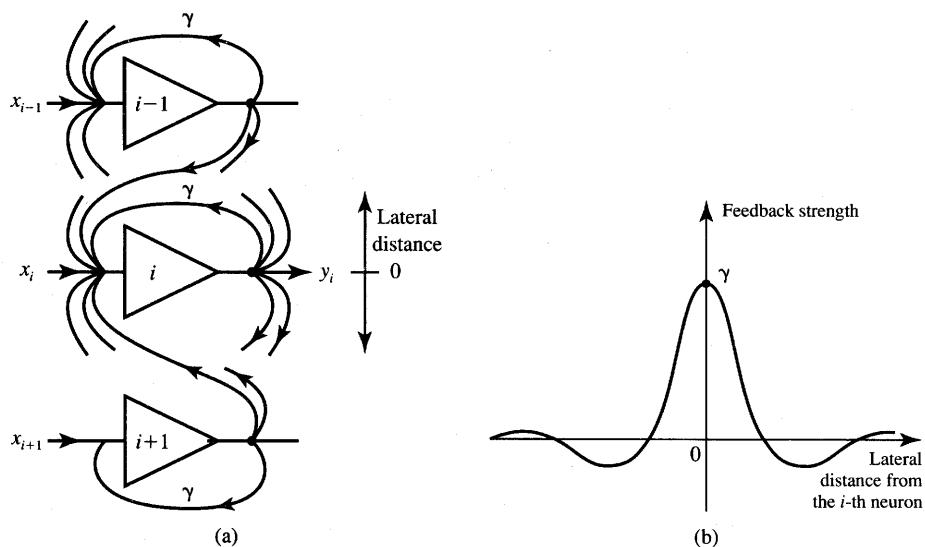
In the experiment reported, planar data coded as 64-dimensional input vectors from the bottom plane have been used for training of the  $8 \times 8$  top planar neuron layer shown.

After training, the single excitations of point inputs highlighted on the figure have been successfully mapped into single peaks of neuron responses at positions directly above the excitations. Thus, the one-to-one geometrical mapping of peak responses generated just above excitations indicates that the dimensionality of any pattern data taken from a two-dimensional space is actually only two. The dimensionality of the pattern representation has no significance. Therefore, the described projection of planar data always results in two-dimensional images.

In a related experiment (Tattershall 1989) illustrated in Figure 7.13(b), a straight line within a square  $10 \times 10$  pixel field has been mapped into a 100-neuron feature array arranged in a square. Since the line has 2 degrees of freedom, two line parameters, which are the intercept with axis and slope, have been produced by the top network trained with 100-dimensional input vectors. The application of a large number of training lines encoded as 100-tuple input vectors causes the feature array to self-organize. Subsequently, an input line forces the output to peak at the corresponding coordinates of intercept and angle. Obviously, calibration of line parameters on the top neural array would also be required after training. The parameter readings can then be made for the performed mapping.

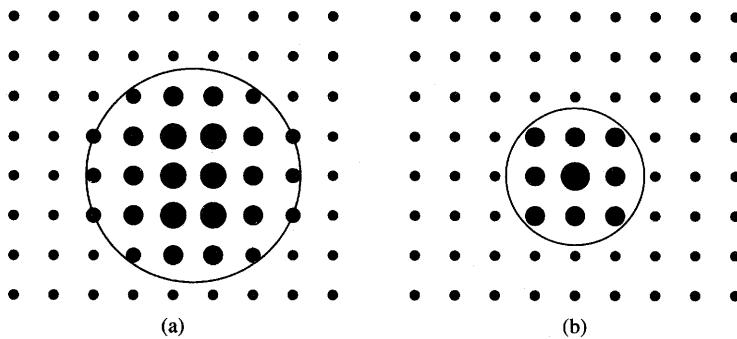
In the following discussion we will look at the details of how regularly shaped neuron layers can arrange and regularize the data from the original pattern space based on a similarity metric. Mappings will be discussed that are at the same time able to preserve pattern space relationships and to reduce the dimensionality of the representation space. The key concept here is the self-organization within a regular array of neurons. This self-organization is demonstrated below based on the lateral feedback concept.

Let us begin by looking for clues about self-organization that may exist in living biological systems. Many biological neural networks in the brain are found to be essentially two-dimensional layers of processing neurons densely interconnected by lateral feedback synapses as shown in Figure 7.14(a). The figure shows a linear array, which is a cross section of a planar array cutting through the neuron  $i$ . The biological neurons of the layer have lateral connection strengths that depend on the distance as illustrated in Figure 7.14(b). The immediate neighborhood of the 50- to  $100\text{-}\mu\text{m}$  radius is characterized by short-range lateral excitatory connections. The self-feedback produced by each biological neuron connecting to itself is positive. It is expressed by the feedback coefficient value  $\gamma > 0$  as shown in the figure. The excitatory area is surrounded by a ring of weaker inhibitory connections of 200- to  $500\text{-}\mu\text{m}$  radius (Kohonen 1984). The feedback connections' weights are shaped like a Mexican hat. It is therefore often mentioned in the literature that the layer Figure 7.14(a) displays the "Mexican hat" type of lateral interaction.



**Figure 7.14** Lateral connections for the clustering demonstration: (a) interconnections of neurons and (b) lateral- and self-feedback strength.

It is interesting to observe the response of a planar array of neurons with lateral connections. A two-dimensional layer of neurons responds with an “activity bubble” produced at a location where the input is maximum. The center of the “activity bubble” is the center of the excitation, too. The typical planar array response pattern is shown in Figure 7.15 displaying circular arrays of activated neurons. Reducing positive lateral feedback of individual neurons results



**Figure 7.15** Planar activity formation for various strengths of lateral interaction: (a) strong positive feedback and (b) weak positive feedback. [Adapted from Kohonen (1984). © Springer Verlag; with permission.]

in a smaller radius for an activated array. This can be seen from comparison of Figures 7.15(a) and (b). These phenomena are due to the self-organization process. The example below explains and illustrates the observed phenomena in more detail.

### EXAMPLE 7.3

Let us look at the self-organization of a linear array of  $n$  neurons arranged as shown in Figure 7.16(a). The linear array, or row, of neurons can be thought of as representing an arbitrary cross section through the center of an activity bubble of the planar array, examples of which are presented in Figure 7.15.

The response of the  $i$ 'th neuron in a linear array can be expressed as

$$y_i(t+1) = f \left( x_i(t+1) + \sum_{k=-k_o}^{k_o} y_{i+k}(t) \gamma_k \right) \quad (7.24)$$

The recursive form of Equation (7.24) is due to the small delay in feeding signals  $y(t)$  back to the input; the delay value has been normalized here to unity for computational convenience. The feedback coefficients  $\gamma_k$  are shown in Figure 7.16(b) as a function of interneuronal distance. The coefficients represent a discretized feedback strength function that can be thought of as an envelope for  $\gamma_k$ . They are represented as a function of discrete variable  $k$  denoting the linear distance rather than as a lateral feedback function defined versus continuous lateral distance. Notice that the neuron's index refers now to the neuron's location in a row. The excitation function for the experiment has been arbitrarily selected with maxima for the fifth and sixth neurons and is defined below in (7.25a). This choice of excitation produces peak local activity formation for the spatial neighborhoods of neurons 5 and 6. Conditions for the simulation experiments are chosen as

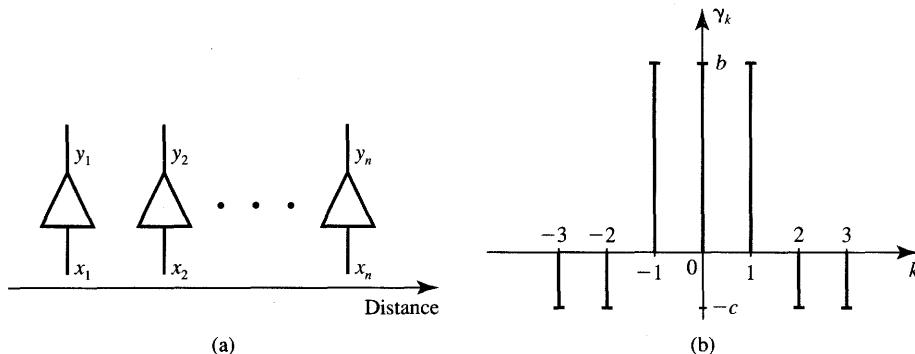
$$x_i(t) = 0.5 \sin^3 \left[ \frac{\pi(i+3)}{17} \right], \quad \text{for } i = 1, 2, \dots, 10 \quad (7.25a)$$

The neuron's activation function and positive and negative feedback coefficients are, respectively,

$$f(\text{net}) \triangleq \begin{cases} 0, & \text{net} \leq 0 \\ \text{net}, & 0 < \text{net} < 2 \\ 2, & \text{net} \geq 2 \end{cases} \quad (7.25b)$$

$$b = 0.4, c = 0.2$$

Fifteen recursive computational steps (7.24) have been performed for the row of 10 neurons. The results are listed in Figure 7.17(a). The first row



**Figure 7.16** Formation of activity in cross section: (a) one-dimensional array of neurons and (b) example lateral feedback.

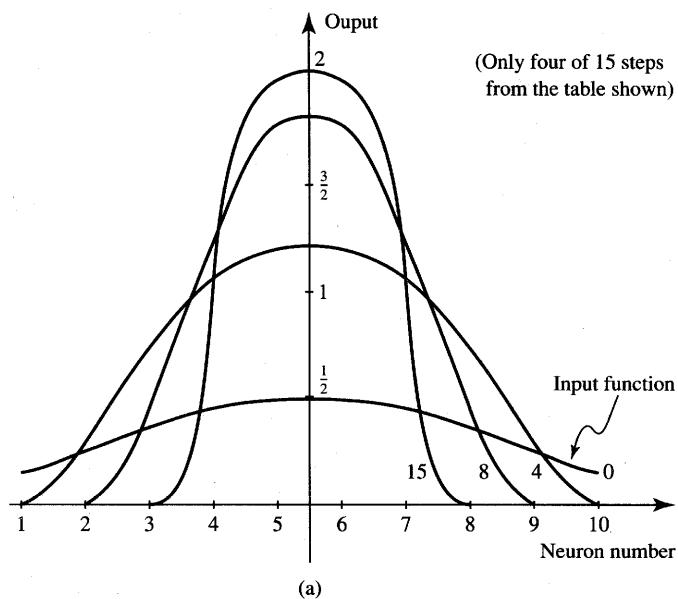
of the table contains excitation values as in (7.25a). It can be seen that neurons 5 and 6 formed peak and saturated responses after 15 recursions. They attained a constant level of 2 which is the maximum response of the node. The responses of neurons 1, 2, 3, 8, 9, and 10 have been laterally attenuated to zero for all cases studied. The response formation took place gradually over the iterations, as can be seen by comparing rows of the table and by inspecting the four curves shown.

The impact of the lateral feedback coefficients  $b$  and  $c$  on the response of the linear array is shown in Figure 7.17(b). It can be seen that increasing the positive feedback coefficients from 0.4 to 0.5 substantially widens the radius of maximum response. Increase of negative feedback coefficients from 0.2 to 0.25 narrows the transition activity regions between the maximum and normal response. Values of  $c$  that are too small can also increase the region of maximum responses. ■

We have observed how activity clusters within contiguous regions are generated in a linear array of neurons. The observations and conclusions from this computational experiment can be helpful in formalizing an algorithm that would implement meaningful unsupervised learning within the self-organizing array. Since the center of activity is at the center of excitation, the weight adaptation algorithm should find the point of maximum activity and activate its respective spatial neighborhood. In this way adaptive weight vectors can be tuned to the input variable. Although we only considered an example of the linear array, a simple generalization provides the formation of localized activity to more dimensions.

$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$	$y_8$	$y_9$	$y_{10}$	Step
0.15	0.25	0.36	0.44	0.49	0.49	0.45	0.36	0.25	0.15	0
0.15	0.37	0.55	0.69	0.78	0.78	0.69	0.55	0.37	0.16	1
0.11	0.39	0.66	0.86	0.96	0.96	0.86	0.66	0.39	0.11	2
0.05	0.36	0.71	0.97	1.09	1.09	0.97	0.72	0.36	0.05	3
0.00	0.29	0.73	1.06	1.20	1.20	1.06	0.73	0.29	0.00	4
0.00	0.21	0.71	1.13	1.32	1.32	1.13	0.71	0.21	0.00	5
0.00	0.13	0.65	1.18	1.45	1.45	1.18	0.65	0.13	0.00	6
0.00	0.04	0.56	1.20	1.60	1.60	1.21	0.56	0.04	0.00	7
0.00	0.00	0.44	1.22	1.78	1.78	1.22	0.44	0.00	0.00	8
0.00	0.00	0.31	1.22	1.99	1.99	1.22	0.31	0.00	0.00	9
0.00	0.00	0.18	1.21	2.00	2.00	1.21	0.18	0.00	0.00	10
0.00	0.00	0.11	1.16	2.00	2.00	1.16	0.12	0.00	0.00	11
0.00	0.00	0.07	1.12	2.00	2.00	1.12	0.07	0.00	0.00	12
0.00	0.00	0.03	1.09	2.00	2.00	1.10	0.04	0.00	0.00	13
0.00	0.00	0.01	1.08	2.00	2.00	1.08	0.01	0.00	0.00	14
0.00	0.00	0.00	1.06	2.00	2.00	1.07	0.00	0.00	0.00	15

Positive feedback coefficient  $b = .4$   
Negative feedback coefficient  $c = .2$



**Figure 7.17a** Clustering of activity within a cross section: (a) neuron outputs for Steps 0 through 15.

$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$	$y_8$	$y_9$	$y_{10}$	Step
0.00	0.00	0.00	0.88	2.00	2.00	0.88	0.00	0.00	0.00	15
	Positive feedback coefficient $b = .4$									
	Negative feedback coefficient $c = .25$									
0.00	0.38	1.49	2.00	2.00	2.00	2.00	1.49	0.39	0.00	15
	Positive feedback coefficient $b = .5$									
	Negative feedback coefficient $c = .2$									
0.00	0.00	0.16	1.39	2.00	2.00	1.41	0.17	0.00	0.00	15
	Positive feedback coefficient $b = .5$									
	Negative feedback coefficient $c = .25$									
0.00	0.00	0.00	1.06	2.00	2.00	1.06	0.00	0.00	0.00	15
	Positive feedback coefficient $b = .5$									
	Negative feedback coefficient $c = .3$									
0.00	0.00	0.00	0.88	2.00	2.00	0.88	0.00	0.00	0.00	15
	Positive feedback coefficient $b = .5$									
	Negative feedback coefficient $c = .35$									

(b)

**Figure 7.17b** Clustering of activity within a cross section (continued): (b) steady-state activity for different feedback coefficients.

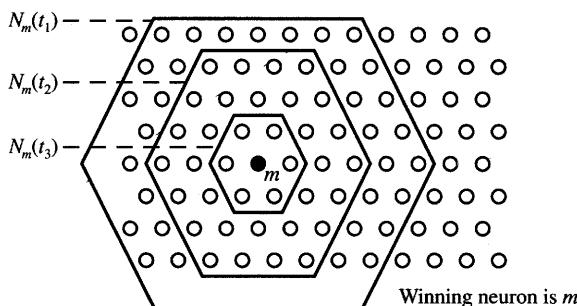
## 7.5

### SELF-ORGANIZING FEATURE MAPS

The feature mapping algorithm is supposed to convert patterns of arbitrary dimensionality into the responses of one- or two-dimensional arrays of neurons. The time-domain recursion illustrated in Example 7.3 needs to be replaced by a self-organizing weight learning rule.

First we discuss simple one-dimensional mapping. Assume that a set of input patterns  $\{x_i, i = 1, 2, \dots\}$  has been arranged in an ordering relation with respect to a single feature within the set. One of the simplest systems thought to produce localized responses is a linear array of neurons that receive the same set of input signals in parallel. The linear array system is defined by Kohonen (1982) as the one producing one-dimensional topology-preserving mapping for  $i_1 > i_2 > i_3 \dots$  when

$$\begin{aligned}
 y_{i1} &= \max_i \{y_i(x_1), \quad i = 1, 2, \dots, n\} \\
 y_{i2} &= \max_i \{y_i(x_2), \quad i = 1, 2, \dots, n\} \\
 y_{i3} &= \max_i \{y_i(x_3), \quad i = 1, 2, \dots, n\} \\
 &\vdots
 \end{aligned} \tag{7.26}$$



**Figure 7.18** Topological neighborhood definition,  $t_1 < t_2 < t_3 \dots$

This definition can be generalized for two or three dimensions. The topological ordering of output is understood in any mapping case as the one preserving the neighborhood. The input ordering need not be specific and can follow arbitrary metrics and orders. Also, the dimensionality of the input space vectors is not restricted. The basic network on which most of the discussion in this section is based is shown in Figure 7.12(a) with feature mapping from an original input space into a two-dimensional rectangular neuron array.

Learning within self-organizing feature maps results in finding the best matching neuron cells which also activate their spatial neighbors to react to the same input. Such collective and cooperative learning tunes the network in an orderly fashion by defining some feature coordinates over the trained network. After learning, each input causes a localized response having a position on the neurons' array that reflects the dominant feature characteristics of the input.

The feature mapping can be thought of as a nonlinear projection of the input pattern space on the neurons' array that represents features. The projection makes the topological neighborhood relationship geometrically explicit in low-dimensional feature space. Following the formation of self-organized internal representation for existing implicit relationships between original data, the same implicit input space relationships become explicit on the spatial neuron map.

The self-organizing feature map algorithm outlined below has evolved as a result of a long series of simulations. The rigorous mathematical analysis of the dynamics of the self-organization algorithm and the algorithm's proof remain yet to be discovered (Kohonen 1990). Thus, our focus in this section is on formulation of the algorithm, its intuitive interpretation, and sample applications.

The algorithm is explained below for a planar array of neurons with hexagonal neighborhoods (Kohonen 1990) as shown in Figure 7.18. As illustrated in Figure 7.12(a), input  $x$  is applied simultaneously to all nodes. Instead of using the scalar product metric of similarity, the spatial neighborhood  $N_m$  is used here as a more adequate measure  $S$  of similarity between  $x$  and  $w_i$ . The weights affecting the currently winning neighborhood,  $N_m$ , undergo adaptation at the current learning step, other weights remain unaffected. The neighborhood  $N_m$  is found

around the best matching node  $m$  selected such that

$$\|\mathbf{x} - \mathbf{w}_m\| = \min_i \{\|\mathbf{x} - \mathbf{w}_i\|\} \quad (7.27)$$

The radius of  $N_m$  should be decreasing as the training progresses,  $N_m(t_1) > N_m(t_2) > N_m(t_3) \dots$ , where  $t_1 < t_2 < t_3 \dots$ . The radius can be very large as learning starts, since it may be needed for initial global ordering of weights. The local ordering within the global order would gradually follow thereafter. Toward the end of training, the neighborhood may involve no cells other than the central winning one. As we realize, this learning neighborhood eventually becomes identical to the neighborhood of the simple competitive learning rule of Kohonen's layer used for cluster learning covered in Section 7.2.

The reader can notice that no explicit lateral interactions of the Mexican hat type are present in the weight adjustment algorithm. However, the mechanism of lateral interaction is embedded in the definition of localized neighborhoods and in the subsequent weight adaptation within the neighborhood.

The weight updating rule for self-organizing feature maps is defined as

$$\Delta \mathbf{w}_i(t) = \alpha [\mathbf{x}(t) - \mathbf{w}_i(t)] \quad \text{for } i \in N_m(t) \quad (7.28a)$$

where  $N_m(t)$  denotes the current spatial neighborhood. Since learning constant  $\alpha$  depends both on training time and the size of the neighborhood, (7.28a) can be rewritten in a more detailed expression as follows:

$$\Delta \mathbf{w}_i(t) = \alpha(N_i, t) [\mathbf{x}(t) - \mathbf{w}_i(t)] \quad \text{for } i \in N_m(t) \quad (7.28b)$$

where  $\alpha$  is a positive-valued learning function,  $0 < \alpha(N_i, t) < 1$ . Because  $\alpha$  needs to decrease as learning progresses, it is often convenient to express it as a decreasing function of time. In addition,  $\alpha$  can be expressed as a function of the neighborhood radius. An example of a function yielding good practical results in a series of simulations (Kohonen 1990) is

$$\alpha(N_i, t) = \alpha(t) \exp \left[ -\|\mathbf{r}_i - \mathbf{r}_m\| / \sigma^2(t) \right] \quad (7.29)$$

where  $\mathbf{r}_m$  and  $\mathbf{r}_i$  are the position vectors of the winning cell and of the winning neighborhood nodes, respectively, and  $\alpha(t)$  and  $\sigma(t)$  are suitably decreasing functions of learning time  $t$ .

It seems that the main conditions for the self-organization of the Kohonen's feature map are as follows: (1) The neurons are exposed to a sufficient number of inputs; (2) only the weights leading to an excited neighborhood of the map are affected; and (3) the adjustment is in proportion to the activation received by each neuron within the neighborhood. As a result, the weight adaptation rule tends to enhance the same responses to a sufficiently similar subsequent input.

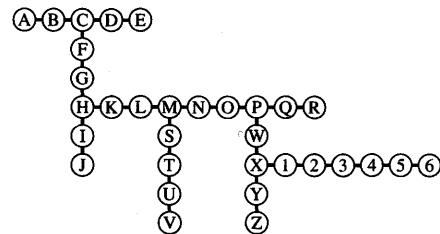
As a result of weight adjustment algorithm implementation, a planar neuron map is obtained with weights coding the stationary probability density function  $p(x)$  of the pattern vectors used for training. The ordered image of patterns forms the weights  $w_{ij}$ .

	Item																															
Attribute	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6
$x_1$	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3		
$x_2$	0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3		
$x_3$	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	6	6	6	6	6	6	6	6	6	6		
$x_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2		
$x_5$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	

(a)

B	C	D	E	*	Q	R	*	Y	Z
A	*	*	*	*	P	*	*	X	*
*	F	*	N	O	*	W	*	*	1
*	G	*	M	*	*	*	*	2	*
*	H	K	L	*	T	U	*	3	*
*	I	*	*	*	*	*	*	4	*
*	J	*	S	*	*	V	*	5	6

(b)



(c)

**Figure 7.19** Self-organizing feature mapping example: (a) list of patterns, (b) feature map produced after training, and (c) minimum spanning tree. [from Kohonen (1984). © Springer Verlag; reprinted with permission.]

An interesting example of the mapping of five-dimensional data vectors is shown in Figure 7.19 (Kohonen 1984). Figure 7.19(a) lists 32 different five-dimensional input vectors labeled  $A$  through  $6$ . The rectangular array of features consists of 70 neurons, each connected by five weights with pattern components  $x_1, x_2, x_3, x_4$ , and  $x_5$ . The array has been trained using vectors  $x_A \dots x_6$  selected at random from the training set. After 10,000 training steps the weights stabilized and then the obtained network was calibrated. Calibration was performed by supervised labeling of array neurons in response to a specific known vector from the training set. Thus, when vector  $B$  was input, the upper left corner neuron from Figure 7.19(b) produced the strongest response in the entire array. It was therefore labeled  $B$ , etc. As seen from the figure, 32 neurons are provided with labels and 38 neurons remained uncommitted. Let us analyze the meaning of the obtained map.

Inspection of the similarity of the training vectors reveals that vectors  $x_1 \dots x_6$  can be arranged in a so-called *minimum spanning tree* as illustrated in Figure 7.19(c). The minimum spanning tree is a technique for investigating clustering or neighborhoods of data (Andrews 1972). The approach results from a graph theoretical analysis of an arbitrary data set. The minimum spanning tree

is defined as a planar tree connecting all points in the data set to their closest neighbors such that the total tree length is minimized. Each vertex of the graph corresponds to only one element from the data set. If we assign a number to each graph edge that is equal to the distance between the vertices that the edge connects, then the minimum spanning tree of a set is that spanning tree which has the sum of the edges of the tree of minimum value.

By inspecting the data  $x_A \dots x_6$ , the distance between  $x_A$  and  $x_B$  is determined to be

$$\|x_A - x_B\| = 1 < \|x_A - x_i\|, \quad \text{for } i = C, \dots, 6$$

Thus the edge connecting  $A$  and  $B$  should probably belong to the minimum spanning tree. Further, since we have

$$\|x_B - x_C\| = 1 < \|x_B - x_i\|, \quad \text{for } i = D, \dots, 6$$

then the edge connecting  $B$  and  $C$  will probably be part of the tree. Since the distances both between  $x_C$  and  $x_D$ , and  $x_C$  and  $x_F$  are both of unity value, the tree would be likely to include the edges  $CD$  and  $CF$ . The reader can continue generating the minimum spanning tree by inspecting distances between the data and growing branch after branch of the planted tree.

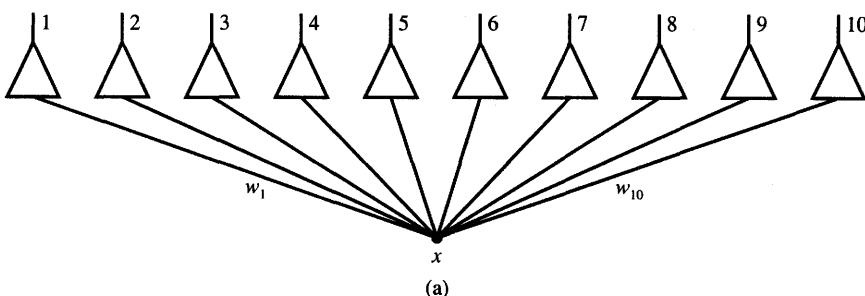
The feature map produced through self-organization has the same structure as the minimum spanning tree. Each piece of data corresponds to a single neuron cell, namely to the best matching one. Thus, the neuron represents its image on the feature array. The data at minimum distances in original five-dimensional space are arranged into topological neighborhoods in two-dimensional feature space. The distances between the vectors  $x_\alpha$  and  $x_\beta$ , defined in the original pattern space as the norm

$$\|x_\alpha - x_\beta\| = \sqrt{\sum_{i=1}^5 (x_{\alpha,i} - x_{\beta,i})^2}$$

are preserved in the feature space where they become planar distances. Planar distances are perceptually very pervasive when compared with five-dimensional space distances. As can be seen from the map, distances within multidimensional space between 32 data items are now flattened on the plane. By connecting the topological neighbors on the plane, we obtain the minimum spanning tree. The obtained map of features is somewhat squeezed into the minimum space, but the topological relations are preserved on the map with some parts of the minimum spanning tree lightly bent on the map.

#### EXAMPLE 7.4

In this example, a linear array of 10 neurons is trained using one-dimensional data  $x$  distributed uniformly between 0 and 1. The data with equal probability



Weights $w_1 - w_{10}$ (unordered) after 1000 training steps									
0.859	0.961	0.753	0.052	0.181	0.297	0.541	0.398	0.469	0.637
0.041	0.124	0.429	0.315	0.223	0.646	0.736	0.544	0.845	0.952
0.528	0.778	0.707	0.623	0.323	0.872	0.963	0.433	0.069	0.210
0.169	0.064	0.690	0.481	0.879	0.960	0.790	0.275	0.385	0.583
0.049	0.152	0.590	0.651	0.721	0.954	0.833	0.498	0.257	0.372
0.061	0.182	0.265	0.707	0.521	0.596	0.357	0.450	0.814	0.937
0.073	0.202	0.436	0.933	0.798	0.669	0.497	0.552	0.378	0.302
0.920	0.975	0.837	0.610	0.724	0.501	0.315	0.402	0.202	0.067
0.052	0.165	0.684	0.471	0.366	0.274	0.752	0.942	0.835	0.583
0.061	0.265	0.163	0.782	0.853	0.949	0.617	0.707	0.497	0.387

(b)

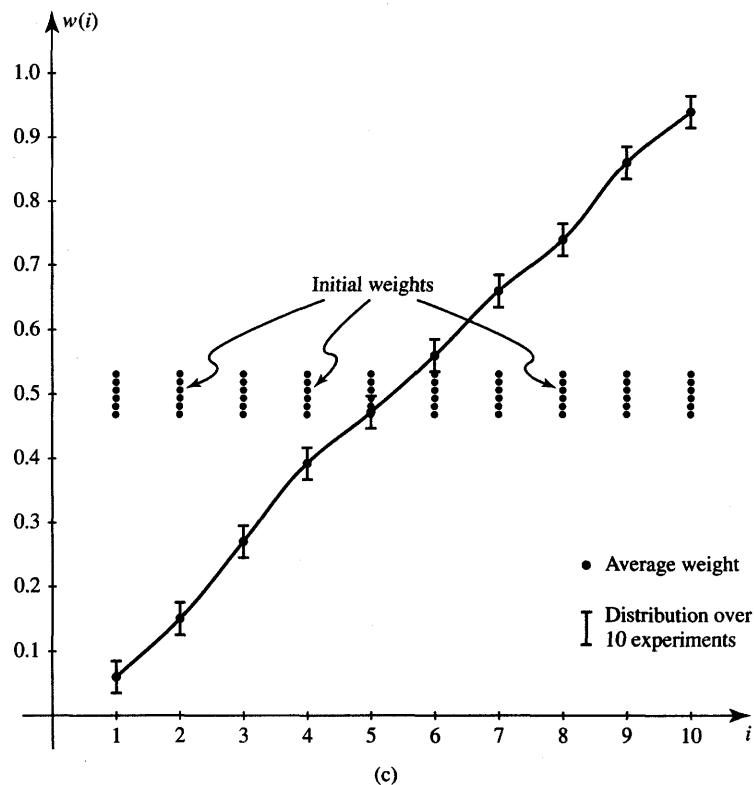
**Figure 7.20a,b** Linear array for Example 7.4: (a) network diagram, (b) weights after training, 10 experiments.

of occurrence are thus mapped from one-dimensional pattern space to ten-neuron, one-dimensional linear feature space.

The network arrangement is shown in Figure 7.20(a). Initial weights are random and centered around 0.5 with 0.05 radius. The initial neighborhood radius of one node has been reduced to zero nodes after 300 training steps. The results of 1000 iterations are tabulated in Figure 7.20(b). Each of the 10 rows lists final weights. Each of the 10 computational training experiments started at different initial values and was performed with data submitted in different random sequences. As can be seen, the initially unordered weights can be ordered to become approximately linear. This can be done by calibrating the trained network through applying known inputs between 0 and 1 that are equally spaced in this range. The average value for a weight matrix obtained in the series of 10 simulations can be computed as

$$\mathbf{W} \cong \frac{1}{100} [ 6 \ 17 \ 28 \ 39 \ 47 \ 57 \ 66 \ 75 \ 85 \ 95 ]'$$

The weights obtained in the experiment are close to the empirical cumulative

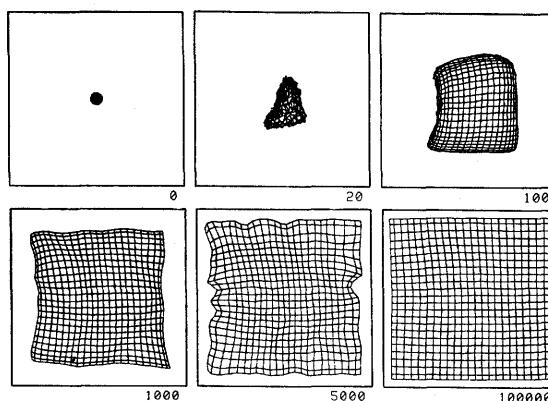


**Figure 7.20c** Linear array for Example 7.4 (continued): (c) weight values for an ordered linear array.

distribution function of a uniform sample of numbers from the range  $[0, 1]$ . Figure 7.20(c) shows that weights become similarly organized to the training patterns.

When applying the self-organizing feature mapping algorithm, formulas (7.27) and (7.28) alternate. Input  $x$ , even if deterministic and defined, is treated as having a probability density of occurrence. The weight adjustments under Kohonen's algorithm reflect this probability of occurrence. In fact, this learning algorithm summarizes the history of the input data along with their topological relationships in the original pattern space. Moreover, in real-world training situations, input  $x$  should involve patterns in their natural sequence of occurrence.

Numerous technical reports have been written about successful applications of the feature map algorithm (Kohonen 1984). The algorithm has been applied in sensory mapping, robot control, vector quantization, and speech recognition. One explanatory example of feature mapping is shown in Figure 7.21. The figure

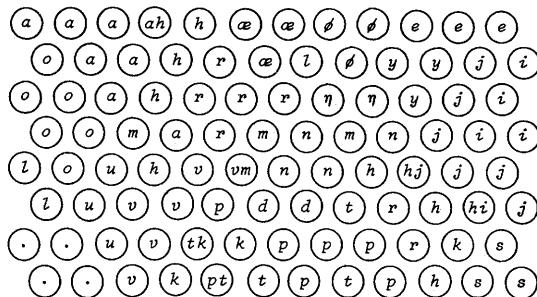


**Figure 7.21** Self-organizing feature map for two input nodes,  $25 \times 25$  array of neurons, and uniformly distributed excitation [from Kohonen (1990). © IEEE; reprinted with permission.]

shows the initial and intermediate phases of self-organization of a square feature map for a two-dimensional input vector. The training has been performed on random independent inputs uniformly distributed over the unity square shown in each of the six boxes. In fact, each of the boxes displays the distribution of weights during training. The initial weights are selected to be random and close to the values of 0.5 and 0.5. The final weights,  $w_{i1}$  and  $w_{i2}$ , of the square array are shown after 100 000 training iterations in the last of the six boxes. The horizontal and vertical coordinates of the plots are the resulting values  $w_{i1}$  and  $w_{i2}$ , respectively, for an array of  $25 \times 25$  neurons. Line intersections of the maps specify weight values for a single  $i$ 'th neuron. Lines between the nodes on the graph merely connect weight points for neurons that are topological nearest neighbors.

In practical applications of self-organizing feature maps, input vectors are often high dimensional. In speech recognition, for example, there are typically 15-tuple input pattern vectors. During the initial processing of the speech signal, the microphone signal is first converted into a 15-channel fast Fourier transform (FFT) spectral signal covering the range of frequencies from 200 hertz to 5 kilohertz. Each phoneme of speech tends to occupy one segment of the 15-dimensional spectral signal space. Although spectra of different phonemes partially overlap and 100 percent accurate discrimination is not completely achievable by any method of phoneme classification, an array of properly trained neurons is able to extract most phoneme features and project them on the plane as described below.

The planar neural array has been trained on spectral signals generated every 9.83 ms during the continuous speech (Kohonen 1988). The spectral FFT samples have been used for the weight adaptation algorithm (7.27) and (7.28) in their natural order of occurrence. During training, various neurons become sensitized



**Figure 7.22** Speech phoneme map after training. [from Kohonen (1990). © IEEE; reprinted with permission.]

to spectra of different phonemes. As a result, the trained feature array shown in Figure 7.22 has been generated. The figure displays a phoneme map of natural Finnish speech. The initially unlabeled map was subsequently calibrated using standard reference phoneme spectra. As can be seen, most cells that learned a phoneme have acquired a unique phoneme characterization. Some cells indicate responses to two different phonemes. Discrimination of overlapping phonemes *k*, *p*, and *t* has not worked too reliably and would need an additional phoneme map discrimination.

The map facilitates the planar visualization of complex multidimensional waveforms of speech. The visualization of speech phonemes may be useful for speech therapy or training, but the ultimate goal of speech phoneme processing is automatic speech recognition. Let us note that Figure 7.22, showing the phoneme map, can be thought of as the keyboard of a “phonetic” typewriter (Kohonen 1988). The reader may also refer to Figure 1.11 showing visualization of continuous speech signals on phoneme maps in the form of word trajectories. Generation of word trajectories points out another very interesting application of Kohonen’s feature map.

It is often reported in the technical literature that the self-organizing map’s successful training depends on the initial weights and the selection of training parameters. Several practical conditions are of importance for the user attempting the implementation of the algorithm. They are discussed below (Kohonen 1990).

The number of weight adjustment steps should be expected to be reasonably large since the learning here is a statistical process. A rule of thumb is that, for good statistical accuracy, the number of steps should be at least 500 times larger than the number of array neurons. Note that although 100 000 training steps are not uncommon, the algorithm is rather simple computationally. Also, intermediate results may be recycled, in particular when the set of training samples is not large.

For approximately 1000 initial steps, the learning function  $\alpha(t)$  in (7.29) should be kept close to unity, then it should start decreasing at  $t = t_p$ . Linear or

exponential decreases are exemplified below. The learning function  $\alpha(t)$  can be specified for the discussed choices as follows:

$$\alpha(t) = \alpha_0, \quad \text{for } t < t_p \quad (7.30a)$$

and for  $t > t_p$  we have linear decrease

$$\alpha(t) = \alpha_0 \left( 1 - \frac{t - t_p}{t_q} \right) \quad (7.30b)$$

or, alternatively, exponential decrease

$$\alpha(t) = \alpha_0 \exp \left[ (-t - t_p) / t_q \right] \quad (7.30c)$$

where  $t_q$  is the time length responsible for the rate of  $\alpha(t)$  decay. Its value should be suitably chosen by the user. After the initial map ordering for large  $\alpha$  values,  $\alpha(t)$  should be kept to a small positive value in the range of 0.01 over an extended period of training. This would suggest that a decelerated decay of  $\alpha$  originally chosen as in (7.30b) and (7.30c) may be suitable in many cases.

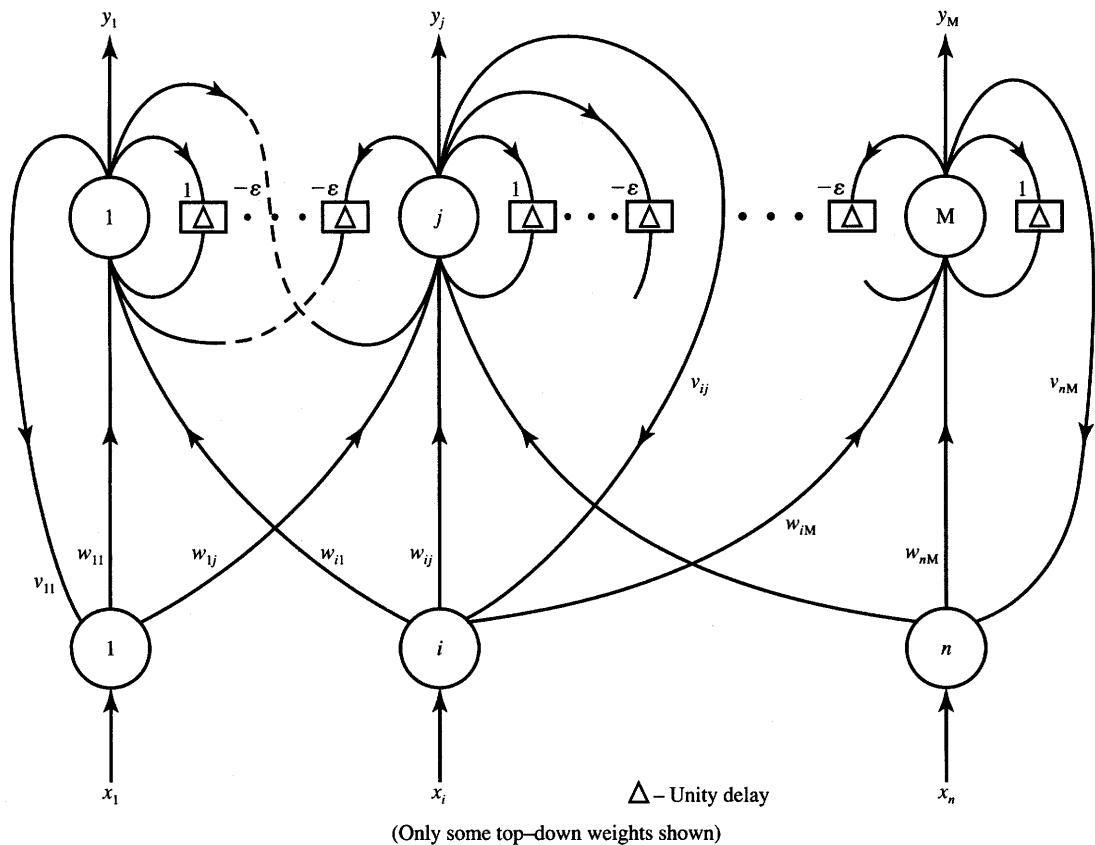
The criteria for selection of the controlled neighborhood radius was introduced in (7.29). At the beginning of training, the radius of the neighborhood can cover the entire network. After the gradual reduction of the neighborhood radius, the network should be continuously trained in the fine weight adjustment mode with nearest neighbors only. Then, the neighborhood should shrink to zero and only include the neuron itself.

## 7.6

### CLUSTER DISCOVERY NETWORK (ART1)

The network covered in this section was developed by Carpenter and Grossberg (1987, 1988) and is called an *adaptive resonance theory I* (ART1) network. It serves the purpose of cluster discovery. Similar to networks using a single Kohonen layer with competitive learning neurons covered in Section 7.2, this network learns clusters in an *unsupervised* mode. The novel property of the ART1 network is the controlled discovery of clusters. In addition, the ART1 network can accommodate new clusters without affecting the storage or recall capabilities for clusters already learned.

The network produces clusters by itself, if such clusters are identified in input data, and stores the clustering information about patterns or features without *a priori* information about the possible number and type of clusters. Essentially the network “follows the leader” after it originates the first cluster with the first input pattern received. It then creates the second cluster if the distance of the second pattern exceeds a certain threshold, otherwise the pattern is clustered with the first cluster. This process of pattern inspection followed by either new cluster



**Figure 7.23** Network for discovering clusters (elements computing norms for the vigilance test and elements performing the vigilance test and disabling  $y_j$  are not shown).

origination or acceptance of the pattern to the old cluster is the main step of ART1 network production. A more detailed description of the training and its algorithm is given below.

The central part of the ART1 network computes the matching score reflecting the degree of similarity of the present input to the previously encoded clusters. This is done by the topmost layer of the network in Figure 7.23 performing bottom-to-top processing. This part of the network is functionally identical to the Hamming network and MAXNET from Figure 7.1(a). The initializing input to the  $m$ 'th node of MAXNET is the familiar scalar product similarity measure between the input  $\mathbf{x}$  and the vector  $\mathbf{w}_m$ . We thus have the initial matching scores of values

$$y_m^0 = \mathbf{w}_m^T \mathbf{x}, \quad \text{for } m = 1, 2, \dots, M \quad (7.31)$$

where  $\mathbf{w}_m = [w_{1m} \ w_{2m} \ \dots \ w_{nm}]^T$ . Note that the double subscript convention

for weights  $w_{ij}$  in this section (and local for only this section of the text) is not followed. The first weight index denotes input node number "from," the second index denotes node number "to." This is to conform with common notation used in the technical literature for this network. The activation function  $f(\text{net})$  for the MAXNET neuron is shown in Figure 7.3(b) and given by (7.5b). It is also assumed that a unity delay element stores each MAXNET neuron output signal during the unity time  $\Delta$  during recursions, before it arrives back at the top-layer node input. The input of the topmost layer is initialized with vector  $\mathbf{y}^0$ , entries of which are computed as matching scores (7.31), and thereafter the layer undergoes recurrent updates as expressed in (7.5a). We thus have for this portion of the network

$$\mathbf{y}^{k+1} = \Gamma[\mathbf{W}_M \mathbf{y}^k] \quad (7.32)$$

where the weight matrix  $\mathbf{W}_M$  is defined as in (7.4) and fulfills all related conditions of the MAXNET weight matrix as stated in Section 7.1. The initializing matching scores vector  $\mathbf{y}^0 = \text{net}^0$  for (7.32) and for the top-layer recurrences is given by the simple feedforward mapping

$$\mathbf{y}^0 = \mathbf{W}\mathbf{x} \quad (7.33)$$

where  $\mathbf{W}$  is the bottom-to-top processing weight matrix containing entries  $w_{ij}$  as follows

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{n1} \\ w_{12} & w_{22} & \cdots & w_{n2} \\ \vdots & \vdots & \cdots & \vdots \\ w_{1M} & w_{2M} & \cdots & w_{nM} \end{bmatrix}$$

As for the MAXNET network, the single nonzero output for a large enough recursion index  $k$ ,  $y_j^{k+1}$ , is produced by the  $j$ 'th top-layer neuron. For this winning neuron we have

$$y_j^0 = \sum_{i=1}^n w_{ij} x_i = \max_{m=1,2,\dots,M} \left( \sum_{i=1}^n w_{im} x_i \right) \quad (7.34)$$

The above discussion is pertinent to the matching score computation of the cluster discovery algorithm. Formulas (7.32), (7.33), and (7.34) are used for calculation of the subscript associated with the largest value  $y_m^0$ ,  $m = 1, 2, \dots, M$ . This is equivalent to the search for the winning node of the top layer. After recursions in the top layer have stabilized, the number of the node with the nonzero output, say  $j$ , is the potential cluster number. As we have realized by now, this part of computation also provides the identification of the weight vector  $\mathbf{w}_j$  that best matches the input vector  $\mathbf{x}$  of all vectors  $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_M$ , in the scalar product metric.

The top-down part of the network checks the similarity of the candidate cluster with the stored cluster reference data and performs the vigilance test on normalized  $\sum_{i=1}^n v_{ij} x_i$  entries as described below. The vigilance threshold  $\delta$ ,

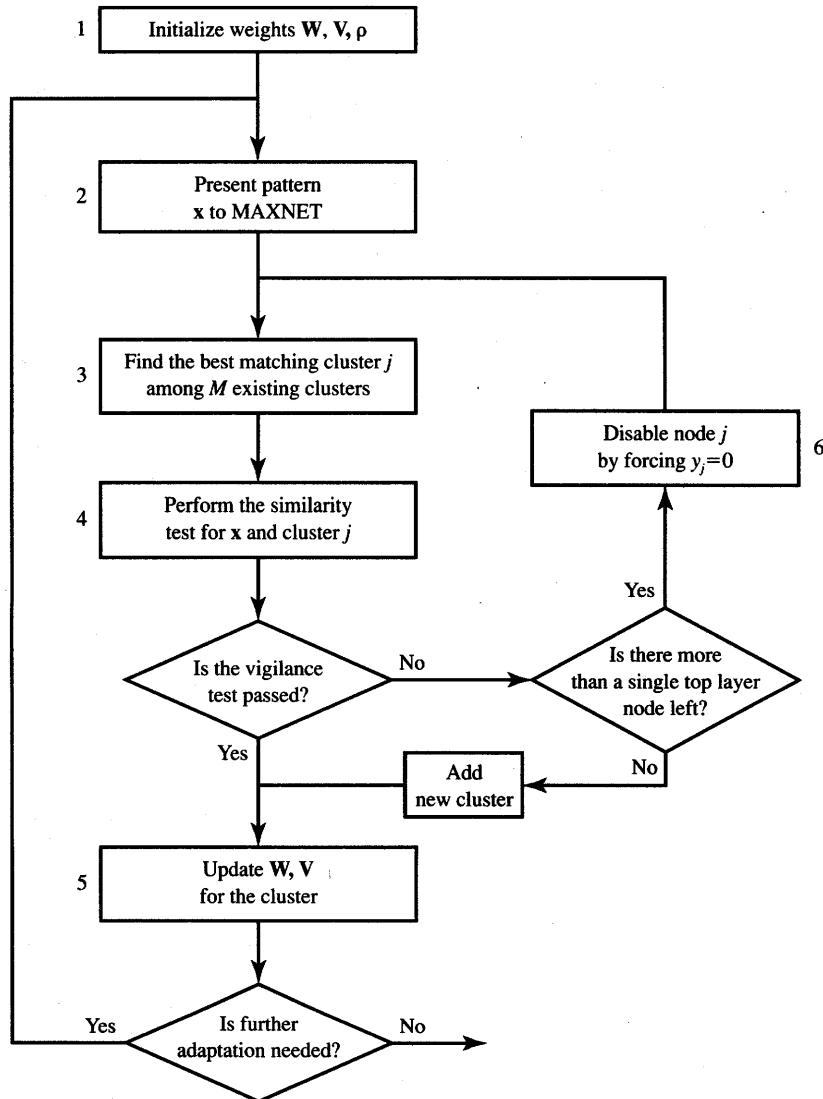


Figure 7.24 Flowchart of the ART1 encoding algorithm for unipolar binary inputs.

$0 < \delta < 1$ , sets the degree of required similarity, or “match,” between a cluster or pattern already stored in the ART1 network and the current input in order for this new pattern to resonate with the encoded one. If the match between the input vector and  $j$ 'th cluster is found, the network is said to be in resonance. Let us review the steps of the learning algorithm for cluster discovery (Lippmann 1987; Pao 1989). The cluster encoding algorithm flowchart that we follow below is shown in Figure 7.24.

■ *Summary of the Adaptive Resonance Theory 1 Network Algorithm (ART 1)*

**Step 1:** The vigilance threshold  $\rho$  is set, and for the  $n$ -tuple input vectors and  $M$  top-layer neurons the weights are initialized. The matrices  $\mathbf{W}$ ,  $\mathbf{V}$  are  $(M \times n)$  and each is initialized with identical entries:

$$\mathbf{W} = \left[ \frac{1}{1+n} \right] \quad (7.35a)$$

$$\mathbf{V} = [1] \quad (7.35b)$$

$$0 < \rho < 1 \quad (7.35c)$$

**Step 2:** Binary unipolar input vector  $\mathbf{x}$  is presented at input nodes,  $x_i = 0, 1$ , for  $i = 1, 2, \dots, n$ .

**Step 3:** All matching scores are computed from (7.31) or (7.33) as follows:

$$y_m^0 = \sum_{i=1}^n w_{im} x_i, \quad \text{for } m = 1, 2, \dots, M \quad (7.36)$$

In this step, selection of the best matching existing cluster,  $j$ , is performed according to the maximum criterium (7.34) as follows:

$$y_j^0 = \max_{m=1,2,\dots,M} (y_m^0) \quad (7.37)$$

[A practical note is that the recurrences of (7.32) done by MAXNET may be skipped in a discrete-time learning simulation since their only consequence of importance is suppressing to zero all upper node outputs but the  $j$ 'th one.]

**Step 4:** The similarity test for the winning neuron  $j$  is performed as follows:

$$\frac{1}{\|\mathbf{x}\|} \sum_{i=1}^n v_{ij} x_i > \rho \quad (7.38)$$

where  $\rho$  is the vigilance parameter and the norm  $\|\mathbf{x}\|$  is defined for the purpose of this algorithm as follows:

$$\|\mathbf{x}\| \triangleq \sum_{i=1}^n |x_i| \quad (7.39)$$

If the test (7.38) is passed, the algorithm goes to Step 5.

If the test has failed, the algorithm goes to Step 6 only if the top layer has more than a single active node left. Otherwise, the algorithm goes to Step 5.

**Step 5:** Entries of the weight matrices are updated for index  $j$  passing the test of Step 4. The updates are only for entries  $(i,j)$ ,

where  $i = 1, 2, \dots, M$ , and are computed as follows:

$$w_{ij}(t+1) = \frac{v_{ij}(t)x_i}{0.5 + \sum_{i=1}^n v_{ij}(t)x_i} \quad (7.40a)$$

$$v_{ij}(t+1) = x_i v_{ij}(t) \quad (7.40b)$$

This updates the weights of the  $j$ 'th cluster (newly created or the existing one). The algorithm returns to Step 2.

**Step 6:** The node  $j$  is deactivated by setting  $y_j$  to 0. Thus, this mode does not participate in the current cluster search. The algorithm goes back to Step 3 and it will attempt to establish a new cluster different than  $j$  for the pattern under test.

In Step 4, the vigilance test is performed on the input vector by comparing the scalar product of the best matching cluster weights  $v_j$  with the input vector divided by the number of bits equal to 1 of the input vector. If the ratio is less than the vigilance threshold, the input is considered to be different from previously stored cluster exemplars. If, at the same time, there are no cluster nodes left for disabling, a new cluster is added. Addition of a new cluster exemplar requires adding one new node and  $2n$  new connections.

The vigilance threshold  $\rho$  is typically a fraction indicating how close an input must be to a stored cluster prototype to provide a desirable match. A value close to 1 indicates that a close match is required; smaller values indicate that a poorer match is acceptable. Thus, for lower values of  $\rho$ , the same set of input patterns may result in a smaller number of clusters being discovered by the ART1 network.

The cluster  $j$  is stored in the form of weight vector  $w_j$  as defined in (7.31). These encoded vectors constitute so-called *long-term network memory*. In comparison, weights  $v_{ij}$  used for verification of cluster exemplar proximity can be considered to be the short-term memory of the network. In summary, the ART1 network learning algorithm described performs an off-line search through the encoded cluster exemplars and is trying to find a sufficiently close match. If no match is found, a new class is created.

The recall of stored patterns by this network can be interpreted as identical to learning; however, the vigilance test is always passed and no weight updates are performed. A complete description of discrete-time recall in the ART1 network involves complex difference equations, since the system involves numerous feedback loops of which only autonomous competitive top-layer recurrences were taken into account in this discussion. The name of the system comes from the resonance that the system finds in Step 4 when the pattern possibly resonates with one of the patterns, say of class  $j$ , and reinforces the storage for this class.

**EXAMPLE 7.5**

This example illustrates stages of discrete-time learning of an example ART1 network with four category neurons indicating the potential for four clusters and a 25-dimensional input vector with entries 0, 1 (Stork 1989). We assume here that the  $\mathbf{W}$  and  $\mathbf{V}$  matrices are of size  $(25 \times 4)$ . Although we are not sure initially whether or not four top-layer cluster neurons are all needed, we consider the choice  $M = 4$  as the only reasonable network initialization. If the number of clusters results after training as smaller than the number of input vectors, the actual number of cluster neurons needed in the top layer would reduce accordingly. Simulations of learning with two different vigilance values are shown in Figure 7.25. The vectors representing the bit maps of input patterns are arranged in column vectors in the sequence in which they are presented, left to right.

Consider first the high-vigilance case shown in Figure 7.25(a) of  $\rho = 0.7$ . Weights are initialized as

$$w_{ij} = \frac{1}{26}$$

$$v_{ij} = 1, \quad i = 1, \dots, 25, \quad j = 1, 2, 3, 4$$

**Step 1:** When pattern  $A$  is presented, one of the four top-layer neurons has the largest output. It is arbitrarily denoted as neuron number 1. The vigilance test is unconditionally passed since the left side of (7.38) is of unity value in the first pass. This results in unconditional definition of the first cluster. The result of the weight adjustment computed from (7.40a) is that all weights  $w_{i1}$  connecting inputs having  $x_i = 1$  with the top-level node 1 are increased to  $2/11$ . Also, top-down weights  $v_{i1}$  adjusted according to (7.40b) remain at unity value as initialized if  $x_i = 1$ ; otherwise they are set to 0.

Figure 7.26 shows the ART1 network after completion of the first training step. Note that from among  $25 \times 4 = 100$  weights that process raw input from the bottom to the top,  $w_{ij}$ , only 5 have changed. From among 100 top-to-bottom weights,  $v_{ij}$ , 5 are set to the unity value and 19 are zeroed. All the remaining weights remain as initialized and are not shown in the figure.

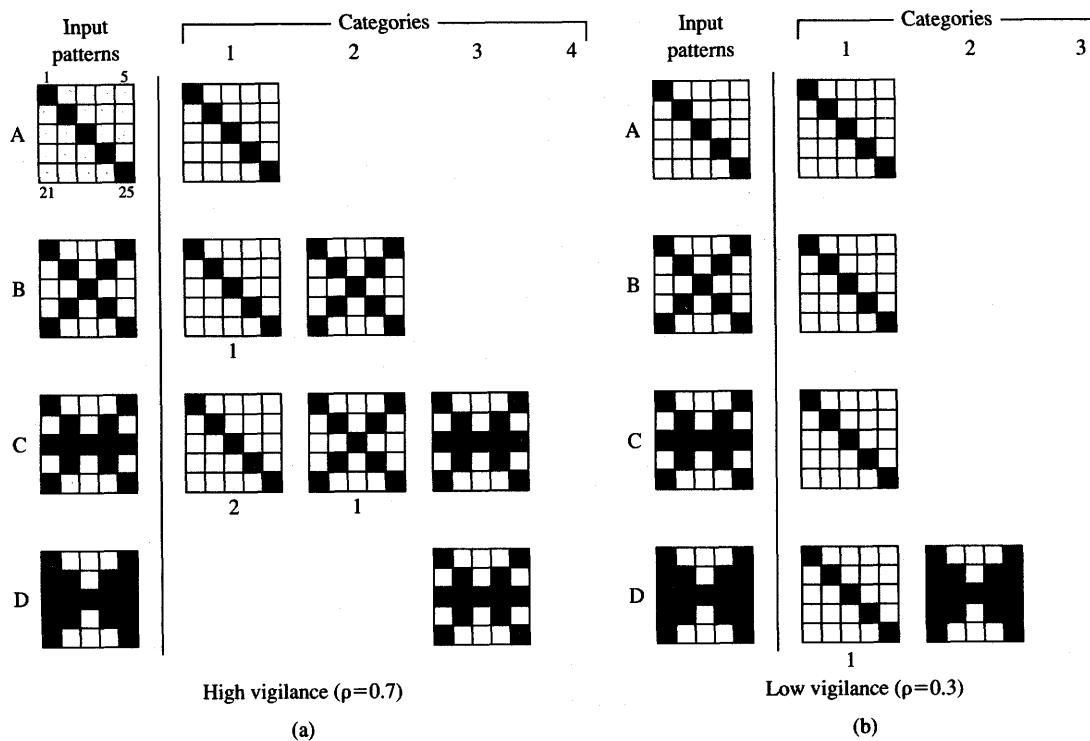
*Summary of first pass of algorithm, pattern A is input:*

$$w_{1,1} = w_{7,1} = w_{13,1} = w_{19,1} = w_{25,1} = \frac{2}{11}$$

The remaining weights  $w_{i1} = 1/26$  as initialized.

$$v_{1,1} = v_{7,1} = v_{13,1} = v_{19,1} = v_{25,1} = 1 \text{ as initialized.}$$

The remaining weights are recomputed as  $v_{i1} = 0$ .



**Figure 7.25** Discrete-time learning simulation of an example ART1 network. [from Carpenter and Grossberg (1987). © Academic Press; reprinted with permission.]

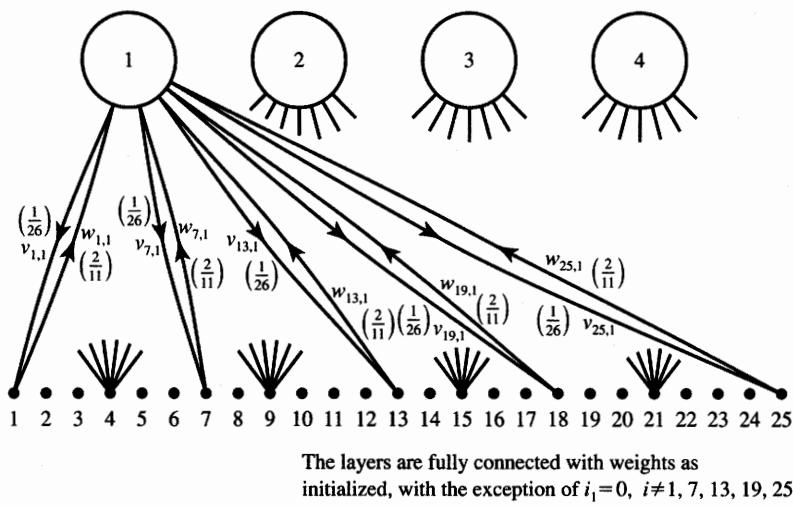
**Step 2:** During the presentation of input pattern *B* there is no top-layer node competing for clustering, since the only active node is 1. The vigilance test results in

$$\frac{1}{\|\mathbf{x}\|} \sum_{i=1}^{25} v_{i1} x_i = 5 \left( \frac{1}{9} \right) < 0.7$$

Due to the failure of the vigilance test and the absence of other nodes for further evaluation and for potential disabling, pattern *B* is treated as a new cluster. The cluster is therefore represented by another neuron, arbitrarily numbered 2. Its connecting weights are recomputed as  $w_{i2}$  and  $v_{j2}$ .

*Summary of second pass of algorithm, pattern B is input:*

$$\begin{aligned} w_{1,2} &= w_{5,2} = w_{7,2} = w_{9,2} = w_{13,2} = w_{17,2} \\ &= w_{19,2} = w_{21,2} = w_{25,2} = \frac{2}{19} \end{aligned}$$



**Figure 7.26** The ART1 network from Example 7.5 after the first training step.

The remaining weights  $w_{i2} = 1/26$ .

$$v_{1,2} = v_{5,2} = v_{7,2} = v_{9,2} = v_{13,2} = v_{17,2} = v_{19,2} = v_{21,2} = v_{25,2} = 1$$

The remaining weights  $v_{i2} = 0$ .

**Step 3:** During the presentation of pattern  $C$  we have the following matching scores computed from (7.34):

$$y_1^0 = 5 \left( \frac{2}{11} \right) + 8 \left( \frac{1}{26} \right) = 1.217$$

$$y_2^0 = 9 \left( \frac{2}{19} \right) + 4 \left( \frac{1}{26} \right) = 1.101$$

Thus, neuron 1 results as a winner. However, the vigilance test is not passed since

$$\frac{1}{\|\mathbf{x}\|} \sum_{i=1}^{25} v_{i1} x_i = \frac{5}{13} < 0.7$$

Node 1 is then disabled and, consequently, node 2 emerges as a winner because of the absence of a competitor node. The vigilance test yields

$$\frac{1}{\|\mathbf{x}\|} \sum_{i=1}^{25} v_{i2} x_i = \frac{9}{13} < 0.7$$

and it fails again, indicating therefore inadequate similarity between the input  $C$  and any of the two existing clusters.

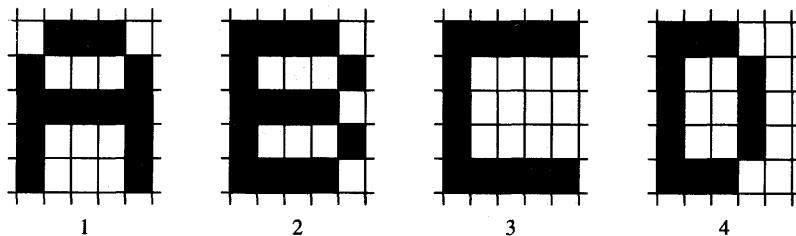
Thus, top-layer neuron numbered 3 is added as a new category and weights  $w_{i3}$  and  $v_{i3}$  are updated accordingly. In contrast to the recent rejection of existing clusters, pattern  $D$  is classified to category 3 because of the similarity with pattern  $C$ . This is due to the fact that the vigilance test was passed and to the resonance of pattern  $D$  within the existing category 3. Notice that the weights computed originally within the third pass of the algorithm and connecting to and from top-layer neuron 3 have been affected in the fourth pass. After patterns  $A$ ,  $B$ ,  $C$ , and  $D$  have been categorized and recorded, when subsequently presented to the network, each of them will access the appropriate category neuron of the top layer and resonate with it. Noticeably, raising the vigilance level to 0.9 or 0.95 will entirely prevent the network from recoding within existing clusters. Therefore, using the training algorithm with this value of the vigilance level enables pattern recall.

The results of ART1 network learning with low vigilance set at 0.3 are shown in Figure 7.25(b). Just two categories are formed in this case. If the vigilance is set at 0.8, however, four categories are produced, one for each pattern. Lowering it to 0.2 would cause only one diffuse category to be formed based on the pattern set  $A$ ,  $B$ ,  $C$ , and  $D$ . All of the four patterns would be categorized as of the same category.

The ART1 network discussed here is a simplified version of the network working only with unipolar binary inputs. The network performs very well with perfect input patterns, but even a small amount of noise superimposed on training patterns can cause problems (Lippmann 1987). With no noise, the vigilance threshold can be set such that two patterns that are somewhat similar but do differ are considered to be different by the network. However, for training using noisy patterns, they can become categorized under noisy conditions. Under no noise conditions, the algorithm shows good performance in both learning and recall. It is reported to be stable and reliable for clustering and classifying unipolar binary input vectors.

### EXAMPLE 7.6

This example demonstrates the storage of patterns under noisy conditions. The ART1 network is developed using the ANS program from the Appendix. Assume that there are four noise-free unipolar binary pattern vectors representing bit maps of the characters  $A$ ,  $B$ ,  $C$ , and  $D$ , which are shown in Figure 7.27(a). The bit maps are of size  $5 \times 5$ , like those in the previous example.



(a)

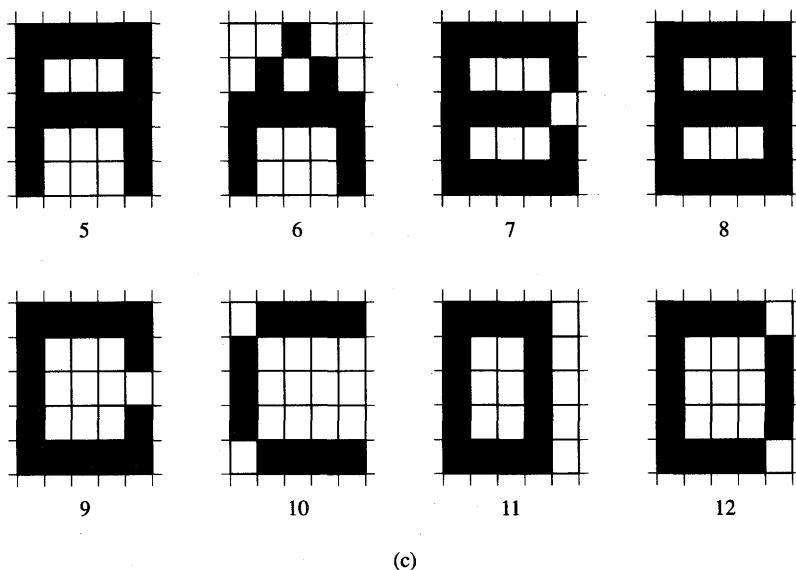
Input pattern number	Clustering			Desired resonance cluster	
	Vigilance level				
	0.95	0.90	0.85		
1	1	1	1	1	
2	2	2	2	2	
3	3	3	3	3	
4	4	4	4	4	
5	5	5	1	1	
6	6	6	5	1	
7	7	7	2	2	
8	8	7	6	2	
9	7	7	3	3	
10	3	3	3	3	
11	9	8	4	4	
12	8	7	4	4	

(b)

Figure 7.27a,b Clustering of patterns in Example 7.6: (a) noise-free prototypes, (b) cluster generation by the ART1 network.

Training of the network is performed with patterns 1 through 4 first. As a result, four clusters are expected to be built and encoded in the network, one cluster for each respective character. As can be seen from Figure 7.27(b), for each of the three vigilance levels selected for this experiment equal to 0.95, 0.9, and 0.85, four clusters are appropriately generated.

Consider that a distorted version of character A shown in Figure 7.27(c) is submitted as input to the network as the fifth pattern. From the fifth row of the table of Figure 7.27(b) it can be seen that the input fails to cluster it with the previously encoded character A unless the vigilance level is lowered to 0.85. When the sixth pattern representing another distorted version of character A is input, however, the network fails to cluster it with node 1 under any of the three vigilance levels used for clustering of the fifth



**Figure 7.27c** Clustering of patterns in Example 7.6 (continued): (c) noisy patterns.

character. It creates therefore another category that is either number 5 or 6, dependent on the vigilance level previously used for the fifth character's processing.

The seventh and eighth input characters represent distorted versions of character *B* associated with the cluster neuron 2. They both define the new clusters for the highest vigilance level. For the lowest of the vigilance levels, the seventh input character resonates properly with the second node, however, another new cluster is built for the eighth input. The reader can see the tabulated results of clustering for the remaining patterns numbered 9 through 12 from the figure. Note that the network is unable to cluster all noisy versions of patterns under the appropriate category number, but it can do this for some of them. If, however, the vigilance level is lowered below 0.85, then original patterns 1 through 4 used to develop the network do not all form separate categories in the first four training steps. This example illustrates what limitations can be encountered by the user when training the ART1 with noisy versions of the prototype vectors.

The reader has probably noticed that most of the neural networks studied thus far have the property that learning a new pattern, or association, influences the recall of existing or previously stored information in the network. When a fully trained network must learn a new mapping in a supervised mode, it may be even required to relearn from the beginning. Similarly, when a new pattern

is added to the associative or autoassociative memory, this may affect the recall of all previously stored data.

The ART1 network explores a radically different configuration. Note that the network works with a finite supply of output units,  $M$ . They are not used, however, until needed to indicate a cluster, and can be considered as waiting for a potential cluster to be defined. The network, therefore, exhibits a degree of plasticity when acquiring new cluster data. At the same time, the recall of the data already learned is not affected, and the weights for clusters already stored are not modifiable. Although the formal mathematical description and convergence proofs of the model have been omitted in our presentation, the basic concepts of learning and recall provided should enable both the understanding and practical use of this important neurocomputing algorithm.

## 7.7

### CONCLUDING REMARKS

Our focus in this chapter has been on networks that learn in an unsupervised mode based on the specified proximity criterion. Except for Hamming network, all networks learned either the similarity between inputs or their input/output mapping without supervision. The proximity, or similarity measures, are embedded in an unsupervised learning algorithm to enable differentiation between similar and dissimilar inputs. In addition, the differentiation capability has been gradually developed within the network based on the learning experience.

We discussed for the most part self-organization, which uses the scalar product similarity measure for developing clusters. We also covered self-organization of patterns into such images in geometrically regular neuron arrays that it reflects the input patterns' mutual structure. Self-organizing arrays have been able to translate clusters in  $n$ -dimensional input pattern space into clustering in low-dimensional feature, or image, space. Such self-organizing has preserved and enhanced the topological relationships present in the original input space.

The response of a self-organizing neural network can be used to encode input in fewer dimensions while keeping as much of the vital input information as possible. We have seen that a drastic dimensionality reduction of input data can take place in situations of simultaneous clustering and feature mapping. Indeed, a space with a high dimensionality of input data is generally expected to be mostly empty, and the nonempty parts of the space can be identified and projected into regular array of neurons responding with images that are easy to perceive.

As we have seen, the competitive learning rule enables the unsupervised training process to work. Winner-take-all learning and an adjustable neighborhood definition have been used for weight formation. In addition, extensive lateral inhibition equivalent to mixed positive and negative feedback has provided the mechanism for selecting the winning neuron in regularly shaped feature arrays.

Most networks in this chapter operate through discrimination of the weight to pattern similarity. The input/output mapping is typically implemented by activating only those neurons whose weights exhibit learned similarity to the present input. Based on this detected similarity, the network output is generated, sometimes directly by the receiving layer, but often by the following layer.

The networks covered in this chapter have found many applications. They include minimum-distance vector classification, vector quantization, auto- and heteroassociative memory, classification of noisy vectors into clusters, and cluster synthesis. The class of unsupervised learning neural networks is particularly well suited to applications that require data quantization. Examples where the capability of data quantization is useful include statistical analysis, codebook communication, and transmission and data compression. The idea of data quantization is to categorize a given set of random input vectors through a finite number of classes. The index of the class is found as a number of the output neuron that fires. We then can transmit or store the index of the class rather than complete data. This, however, requires a list of codes, or codebook, that has been agreed upon. Such a codebook must be available at the receiving end of the information transmission system, since the data encoded by the network at the transmitter side must now be decoded.

Finally, self-organizing feature maps can be applied for extracting the hidden features of multi-dimensional input patterns and their probability of occurrence. The inputs are mapped to the line or plane of output units so that neighborhood relations are transferred from the input space to the neuron output space. This is especially useful for detection of hidden relationships that are difficult to group, organize or categorize in a sensible manner. One of the interesting applications of the map is for human language study and is presented in Section 8.6.

---

## PROBLEMS

Please note that problems highlighted with an asterisk (\*) are typically computationally intensive and the use of programs is advisable to solve them.

**P7.1** Design the Hamming network for the three following class prototype vectors:

$$\mathbf{s}^{(1)} = [1 \ -1 \ 1 \ -1 \ 1 \ -1]^t,$$

$$\mathbf{s}^{(2)} = [-1 \ -1 \ -1 \ -1 \ -1 \ 1]^t,$$

$$\mathbf{s}^{(3)} = [1 \ 1 \ 1 \ -1 \ -1 \ 1]^t$$

- (a) Compute the weight matrix.
- (b) Find  $net_m$ , for  $m = 1, 2, 3$ , for the input vector  $\mathbf{x} = [1 \ 1 \ 1 \ 1 \ 1 \ 1]$  and verify that the HD parameter computed by the network agrees with the actual HD.

- (c) Find the normalized responses  $f(\text{net}_m)$  at the output of each node of the Hamming network.

*P7.2* MAXNET with four input nodes has been designed for  $\varepsilon = 0.1$ . After the two initial recurrences completed, the following network output resulted:

$$\mathbf{y}^2 = [0.186 \quad 0.307 \quad 0.428 \quad 0.549]^t$$

Based on these data:

- (a) Find the initializing vector  $\mathbf{y}^0$  that has been applied initially, prior to recurrences.
- (b) Knowing that  $\mathbf{y}^0$  is the output vector of the Hamming network, determine the HD value between the closest 10-bit prototype and the Hamming network input vector originally applied to yield the computed  $\mathbf{y}^0$ .

*P7.3* The MAXNET with four output nodes,  $p = 4$ , receives the input vector

$$\mathbf{y}^0 = [0.5 \quad 0.6 \quad 0.7 \quad 0.8]^t$$

- (a) Find the  $\varepsilon$  value that would be required to suppress the output of the weakest node exactly to the zero value after the first cycle.
- (b) Find subsequent responses of the network,  $\mathbf{y}^1$  and  $\mathbf{y}^2$ , for the computed value of  $\varepsilon$ .

*P7.4\** Design the Hamming network and the associated MAXNET network to provide optimum classification of bipolar binary vectors representing characters *A*, *I*, and *O* in Problem P4.16 in terms of their minimum Hamming distance.

*P7.5* Assume that the following condition holds for the vector initializing a MAXNET network with  $p$  nodes:

$$0 \leq y_1^0 < y_2^0 < \dots < y_{p-1}^0 < y_p^0 \leq 1$$

Prove that the following increment condition holds during recurrences in the linear mode:

$$|y_p^{k+1} - y_p^k| < |y_{p-1}^{k+1} - y_{p-1}^k| < \dots < |y_1^{k+1} - y_1^k|$$

*P7.6* Given are three cluster centers as

$$\mathbf{x}_1 = \begin{bmatrix} 10 \\ 2 \\ 5 \end{bmatrix}^t, \quad \mathbf{x}_2 = \begin{bmatrix} 5 \\ 1 \\ -2 \end{bmatrix}^t, \quad \mathbf{x}_3 = \begin{bmatrix} -3 \\ -4 \\ 0 \end{bmatrix}^t$$

Compute the weights of the network that has three cluster detecting neurons. Perform a computation based on the requirement that weights must match such patterns. Assume that the normalized weight values of the

three neurons have initially been tuned to the normalized values  $\mathbf{x}_1$ ,  $\mathbf{x}_2$ , and  $\mathbf{x}_3$ , and then denormalized in inverse proportion to each pattern's distance from the origin. Calculate the responses of each neuron in the recall mode for  $f(\text{net}) = \frac{2}{1+\exp(-\text{net})} - 1$  with inputs  $\mathbf{x}_1$ ,  $\mathbf{x}_2$ , and  $\mathbf{x}_3$ .

- P7.7* Perform the first learning cycle using the following normalized pattern set:  $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\} = \{1445^\circ, 14-135^\circ, 1490^\circ, 14-180^\circ\}$  and  $\alpha = 0.5$ , using the winner-take-all training rule for two cluster neurons. Draw the resulting separating hyperplanes. Initial weights are to be assumed  $\hat{\mathbf{w}}_1^0 = [1 \ 0]'$  and  $\hat{\mathbf{w}}_2^0 = [0 \ -1]'$ .
- P7.8* Solve Problem P7.7 graphically using vector subtraction/addition, and renormalization, if needed. Then reduce  $\alpha$  to 0.25 and analyze the impact of reduced learning constant  $\alpha$  on the initial phase of training.
- P7.9\** Implement the winner-take-all learning algorithm of the Kohonen layer for clustering noisy characters *C*, *I*, and *T* shown in Figure 4.19 into each of the respective three classes. Assume that the noisy training characters are at distance  $HD = 1$  from the noise-free prototypes. Compute the final weight vectors  $\mathbf{w}_C$ ,  $\mathbf{w}_I$ , and  $\mathbf{w}_T$ . During the recall operation, compute the responses  $f(\text{net}_C)$ ,  $f(\text{net}_I)$ , and  $f(\text{net}_T)$  of each neuron with continuous unipolar activation function. Use inputs that are respective bipolar binary vectors as specified in Section 4.5. Assume  $\lambda = 1$ .
- P7.10* Analyze the following counterpropagation network for which the feed-forward part is only operational. The network is shown in Figure P7.10. Assume the bipolar binary input and output vectors. Provide results for three- and two-dimensional cubes indicating cluster centers, which the network is supposed to detect and subsequently map into output vectors.
- P7.11* The feedforward part of a counterpropagation network is shown in Figure P7.11(a). It is designed to map two bit maps represented by vectors  $\mathbf{x}_1$  and  $\mathbf{x}_2$  shown in Figure P7.11(b) into two three-tuple vectors,  $\mathbf{z}_1$  and  $\mathbf{z}_2$ . Identify vectors  $\mathbf{z}_1$  and  $\mathbf{z}_2$  by analyzing the network. Also, complete the design of the Kohonen layer.
- P7.12* The objective in this problem is to design a so-called Gray code converter of bipolar binary numbers using the feedforward part processing of the counterpropagation network. The Gray code converter has the property that when  $\mathbf{x}_i$  indices increase as 0, 1, 2, 3, only a single output bit of  $\mathbf{z}_i$  changes.
- (a) Design the first layer of the converter as a Kohonen layer, which converts four two-bit input vectors

$$\mathbf{x}_0 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \mathbf{x}_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

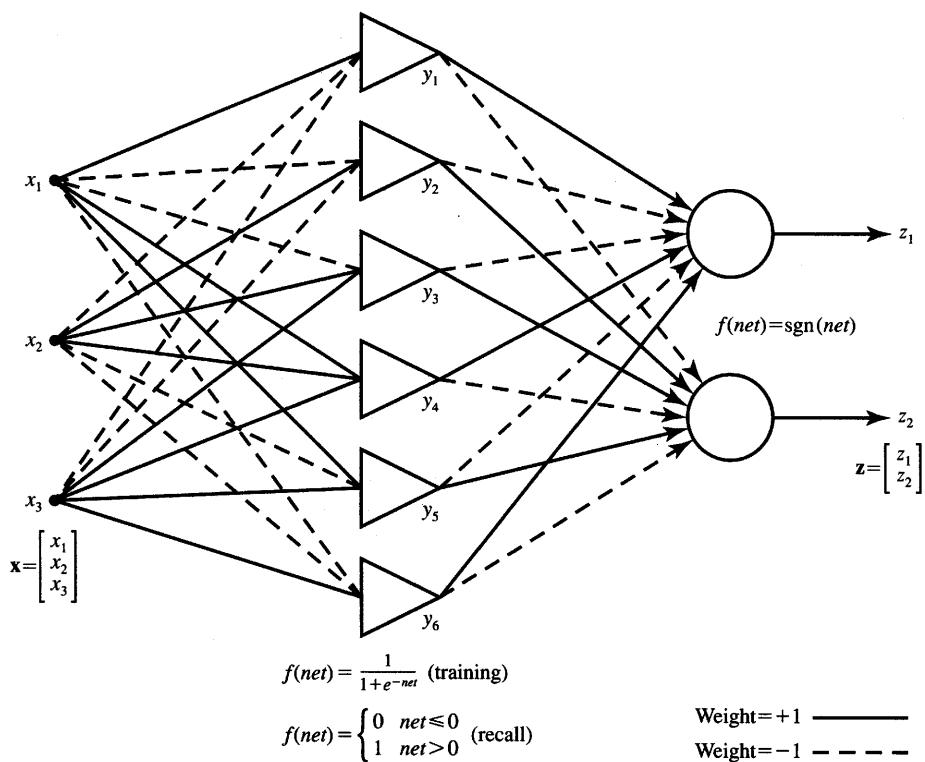


Figure P7.10 Feedforward network for analysis in Problem P7.10.

into, respectively,

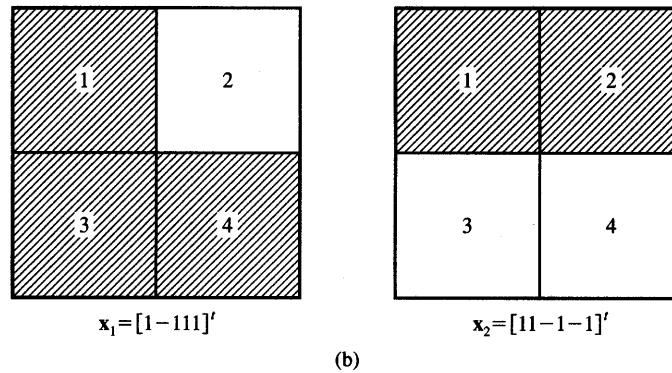
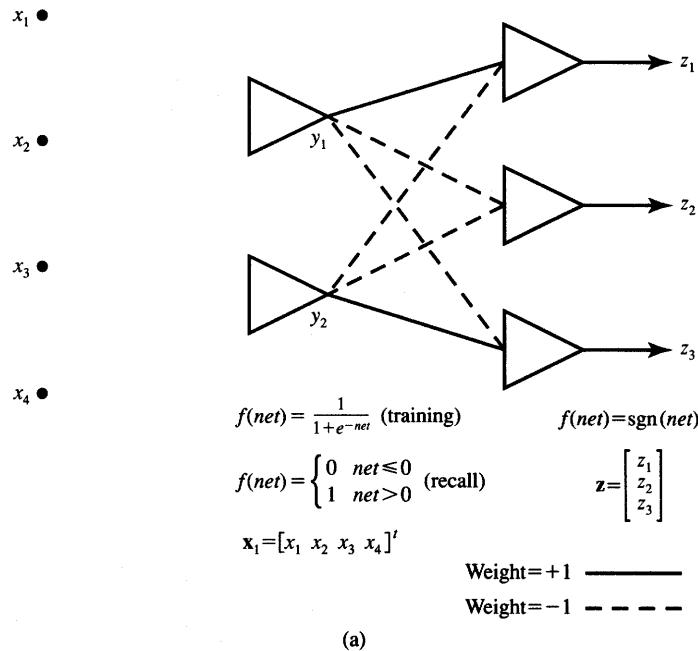
$$\mathbf{y}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \mathbf{y}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \mathbf{y}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{y}_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- (b) Design the second layer of the converter as a Grossberg layer of the feedforward counterpropagation network. Find  $\mathbf{V}$  such that the corresponding converter outputs are

$$\mathbf{z}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{z}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \mathbf{z}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mathbf{z}_3 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

Use the explicit design equation (instead of training on noisy input data) to implement mapping and to obtain  $\mathbf{W}$  and  $\mathbf{V}$ .

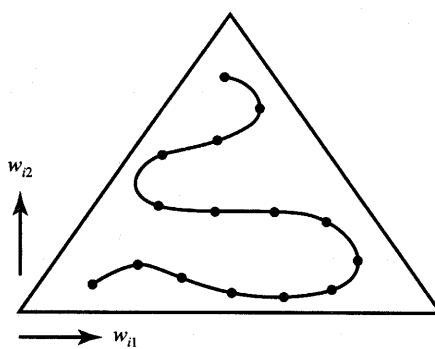
- P7.13 Design the feedforward part of the counterpropagation network that performs the operation  $XOR (x_1, x_2, x_3)$  such that  $\mathbf{z} = x_1 \oplus x_2 \oplus x_3$  as shown in Figure 3.9(b). Use the explicit design equation instead of training the network on noisy data to obtain  $\mathbf{W}$  and  $\mathbf{V}$ . Assume  $x_i = \pm 1$  and  $z_i = \pm 1$ .



**Figure P7.11** Feedforward part of the counterpropagation network: (a) diagram and (b) input vector bit maps.

**P7.14** Design the feedforward part of the counterpropagation network that associates the binary bipolar vectors representing bit maps of character pairs  $(A, C)$ ,  $(I, I)$ ,  $(O, T)$  as specified in Problem P6.19. Use the explicit design formulas to find  $\mathbf{W}$  and  $\mathbf{V}$  instead of training the network on noisy data.

**P7.15\*** Solve Problem P7.14 by using noisy training data for all six characters specified. Assume that the noisy vectors are at the distance  $\text{HD} = 1$  from the undistorted prototypes. Train the network in the three following modes:



**Figure P7.16** Feature map for Problem P7.16.

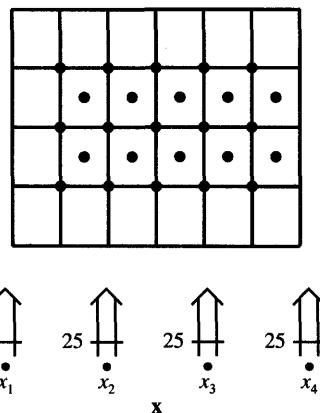
- (a) noisy input vectors only at HD = 1 from input prototypes, outputs are undistorted
- (b) noisy output vectors only at HD = 1 from output prototypes, inputs are undistorted
- (c) both input and output distorted so that each is at HD = 1 from respective prototypes (random bits distorted, no input/output correlation)

**P7.16** The Figure P7.16 shows the final weights of a self-organizing feature map that has been trained for the following conditions (Kohonen 1984): two-dimensional input vectors have been uniformly distributed over the triangular area and the self-organizing array consisted of 15 neurons. Lines between points connect topological neighbors only; points indicate weight coordinates. Sketch the diagram of the network that, in your opinion, has undergone self-organization. Label inputs, neurons, and weights for the computational experiment, results of which are reported on the figure. Suggest the initial weight values that may have been used.

**P7.17** A set of five four-dimensional vectors is given

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \mathbf{x}_4 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{x}_5 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Implement the self-organizing feature map algorithm to map the given vectors onto a  $5 \times 5$  array of neurons. Hexagonal neighborhoods should be formed for the array training as shown in Figure P7.17. Decrease  $\alpha$  linearly from 0.5 to 0.04 for the first 1000 steps, then decrease it for steps 1000 to 10 000 from 0.04 to 0. Decrease the neighborhood radius from covering the entire array initially (radius of 2 units) to the radius of zero value after 1000 steps. Then calibrate the computed map and draw the produced minimum spanning tree for inputs  $\mathbf{x}_1, \dots, \mathbf{x}_5$ . Compare the



**Figure P7.17** Hexagonal map and inputs for Problem P7.17.

computed results produced by the network with the predicted minimum spanning tree.

**P7.18** Perform three steps of the discrete-time learning algorithm of the ART1 network for  $n = 9$  and  $M = 3$  assuming that the training vectors are:

$$\mathbf{x}_1 = [1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1]^t$$

$$\mathbf{x}_2 = [1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1]^t$$

$$\mathbf{x}_3 = [1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1]^t$$

Select an appropriate vigilance threshold  $\rho$  so that all three patterns are clustered separately when the sequence  $\mathbf{x}_1$ ,  $\mathbf{x}_2$ , and  $\mathbf{x}_3$  is presented. Compute the final weights  $\mathbf{W}$ , and  $\mathbf{V}$  of the trained network.

**P7.19** Figure 7.25(a) shows the simulation steps of discrete-time learning in the ART1 network for  $n = 25$  and  $M = 4$ . The bits of the training vectors are 1 and 0 for black and white bit-map pixels, respectively. Find the following values of the learning network:

- (a) the lowest value of  $\rho$  that enables definition of a new cluster when pattern  $B$  is at the input
- (b) the matching scores  $y_1^0$ ,  $y_2^0$ , and  $y_3^0$  of neurons when pattern  $D$  is at the input
- (c) the lowest value of  $\rho$  that enables definition of a new cluster when pattern  $D$  is at the input.

**P7.20** Figure 7.25(b) shows the simulation results of discrete-time learning in the ART1 network for  $n = 25$  and  $M = 4$ . The bits of the training vectors

are 1 and 0 for black and white pixels, respectively. Find the vigilance threshold preventing the occurrence of the second cluster when pattern  $D$  is at the input. Assume that patterns  $A$ ,  $B$ , and  $C$  have already been clustered together as shown.

P7.21\* Implement the learning and recall algorithm of the adaptive resonance theory network ART1 for the following conditions:

$$M = 4, n = 25, \text{ and user-specified } \rho.$$

Perform the algorithm and program tests by learning the cluster prototypes as shown in Example 7.5. Test the learning and recall phases for the different vigilance levels.

---

## REFERENCES

- Andrews, H. C. 1972. *Introduction to Mathematical Techniques in Pattern Recognition*. New York: Wiley Interscience.
- Carpenter, G. A., and S. Grossberg. 1988. "Neural Dynamics of Category Learning and Recognition: Attention, Memory Consolidation and Amnesia," in *Brain Structure, Learning and Memory*, ed. J. Davis, R. Newburgh, I. Wegman. AAAS Symp. Series. Westview Press, Boulder, Colo.
- Carpenter, G. A., and S. Grossberg. 1987. "A Massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics, and Image Proc.* 37: 54–115.
- Duda, R. O., and P. E. Hart. 1973. *Pattern Classification and Scene Analysis*, New York: John Wiley and Sons.
- Grossberg, S. 1977. *Classical and Instrumental Learning by Neural Networks. Progress in Theoretical Biology*, vol 3. New York: Academic Press, pp. 51–141.
- Grossberg, S. 1982. *Studies of Mind and Brain: Neural Principles of Learning Perception, Development, Cognition, and Motor Control*. Boston: Reidell Press.
- Hecht-Nielsen, R. 1987. "Counterpropagation Networks," *Appl. Opt.* 26(23): 4979–4984.
- Hecht-Nielsen, R. 1988. "Applications of Counterpropagation Networks," *Neural Networks* 1: 131–139.
- Hecht-Nielsen, R. 1990. *Neurocomputing*. Reading, Mass.: Addison-Wesley Publishing Co.
- Kohonen, T. 1982. "A Simple Paradigm for the Self-Organized Formation of Structured Feature Maps," in *Competition and Cooperation in Neural*

- Nets*, Lecture Notes in Biomathematics, ed. S. Amari, M. Arbib. Berlin: Springer-Verlag.
- Kohonen, T. 1984. *Self-Organization and Associative Memory*. Berlin: Springer-Verlag.
- Kohonen, T. 1988. "The 'Neural' Phonetic Typewriter," *IEEE Computer* 27(3): 11-22.
- Kohonen, T. 1990. "The Self-organizing Map," *Proc. IEEE* 78(9): 1464-1480.
- Lippmann, R. P. 1987. "An Introduction to Computing with Neural Nets," *IEEE Magazine on Acoustics, Signal and Speech Processing* (April): 4-22.
- Pao, Y. H. 1989. *Adaptive Pattern Recognition and Neural Networks*. Reading, Mass.: Addison-Wesley Publishing Co.
- Simpson, P. I. 1990. *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementation*. New York: Pergamon Press.
- Stork, D. G. 1989. "Self-organization, Pattern Recognition, and Adaptive Resonance Networks," *J. Neural Network Computing* (Summer): 26-42.
- Tattershall, G. 1989. "Neural Map Applications," in *Neural Network Architectures*, ed. I. Alexander. Cambridge: The MIT Press.
- Tou, J. T., and R. C. Gonzalez. 1974. *Pattern Recognition Principles*. Reading, Mass.: Addison-Wesley Publishing Co.
- Wasserman, P. D. 1989. *Neural Computing Theory and Practice*. New York: Van Nostrand Reinhold.



---

# APPLICATIONS OF NEURAL ALGORITHMS AND SYSTEMS

*Nihil simul inventum est et perfectum. (Nothing is invented and perfected at the same time.)*

LATIN PROVERB

- 8.1 Linear Programming Modeling Network
- 8.2 Character Recognition Networks
- 8.3 Neural Networks Control Applications
- 8.4 Networks for Robot Kinematics
- 8.5 Connectionist Expert Systems for Medical Diagnosis
- 8.6 Self-Organizing Semantic Maps
- 8.7 Concluding Remarks

---

**O**ur study thus far has focused on the theoretical framework and mathematical foundations of artificial neural systems. Having understood the computational principles of massively parallel interconnected simple "neural" processors, we may now put them to good use in the design of practical systems. Numerous clues and guidelines for applications of neural algorithms and systems were developed in preceding chapters. However, most such applications have been presented to illustrate the operating principles of neural networks rather than to reflect a practical focus. As we shall see in this chapter, neurocomputing architectures are successfully applicable to many real-life problems. Moreover, their application potential seems to be enormous but not yet fully realized.

Examples of diverse applications are tied together in this chapter. The published technical references on applications of neural networks number well over a thousand papers and perhaps several hundred more are printed every year. Instead of merely enumerating many successful project titles of that list in an encyclopaedic manner, the approach taken in this chapter is to study about a dozen applications in detail. The applications selected for coverage are believed to be representative of the field, and the depth of their coverage should prepare

the readers to pursue their own application project development. However, since the area of neural networks is expanding very rapidly, new problems and solution approaches may emerge in the near future beyond those presented in this chapter.

First we discuss a single-layer fully coupled feedback network for solving classical optimization problems. The network used for this purpose is a gradient-type network and it can be designed so that it minimizes the linear cost function in the presence of linear constraints. Character recognition networks are discussed for both printed and written character evaluation. We shall see that although ordinary multilayer feedforward networks consisting of continuous perceptrons can be successfully used for this purpose, unconventional architectures and approaches result in even higher character recognition rates.

Selected approaches to building neurocontrollers are presented with explanations of how they make it possible to identify the plants, and how they can be used as controllers. Adaptive control issues as well as neurocontrol of the dynamic plant are examined and illustrated with application examples. Subsequently, techniques closely related to neurocontrol systems and applied to the control of robot kinematics are covered. Feedforward networks capable of performing kinematics transformations, trajectory control, and robot hand target point learning are examined.

Next, three connectionist expert systems are presented. The real-life examples of medical expert systems for diagnosis of skin diseases, low back pain, and heart diseases are examined. Their performance is reviewed and compared with the success rate of human experts. Finally, the applications for self-organizing semantic feature maps are covered. Their practical use can be envisaged for sentence processing and extraction of symbolic meaning and contextual information implicit in human language.

In this chapter an attempt is made to review a number of successful implementations of massively parallel networks, and to apply them to different tasks. We will try to determine the best architectures to accomplish certain functions, how the networks should be trained, whether or not they can be useful during the training phase, how well they perform in real-life training examples, and how they are able to generalize to unknown tasks from the already known ones. Most of these questions cannot be answered independently and some of them cannot be answered conclusively. By the end of the chapter, however, we will have an understanding of how to select and design networks to fulfill a number of particular goals.

## 8.1

### LINEAR PROGRAMMING MODELING NETWORK

The interconnected networks of analog processors can be used for the solution of constrained optimization problems, including the linear programming

problem. As studied in Chapter 5, properly designed single-layer feedback networks are capable of acquiring the properties of gradient-type systems. Such systems can be used to solve optimization problems since they minimize, in time, their energy function value. The crucial task in designing a system seeking the minimum of its energy function is the translation of the optimization problem into a suitable definition of the energy function. The gradient-type network of analog processors that can find the solution of the linear programming problem is discussed in this section.

The *linear programming problem* may be stated as follows: Find the unknowns  $v_1, v_2, \dots, v_n$ , so as to minimize the cost function

$$c \triangleq a_1 v_1 + a_2 v_2 + \dots + a_n v_n \quad (8.1a)$$

where  $a_1, a_2, \dots, a_n$  are called price coefficients. The minimization is subject to the following conditions, which are linear constraints:

$$\begin{aligned} w_{11}v_1 + w_{12}v_2 + \dots + w_{1n}v_n &\geq b_1 \\ w_{21}v_1 + w_{22}v_2 + \dots + w_{2n}v_n &\geq b_2 \\ &\vdots \\ w_{m1}v_1 + w_{m2}v_2 + \dots + w_{mn}v_n &\geq b_m \end{aligned} \quad (8.1b)$$

where coefficients  $b_j$ , for  $j = 1, 2, \dots, m$ , are called bounds or requirements.

The cost function (8.1a) may be equivalently expressed in matrix form using the price vector  $\mathbf{a}$  and the variable vector  $\mathbf{v}$

$$c = \mathbf{a}'\mathbf{v} \quad (8.2a)$$

where

$$\begin{aligned} \mathbf{a} &\triangleq [a_1 \ a_2 \ \dots \ a_n]^t \\ \mathbf{v} &\triangleq [v_1 \ v_2 \ \dots \ v_n]^t \end{aligned}$$

The linear constraint condition (8.1b) now becomes

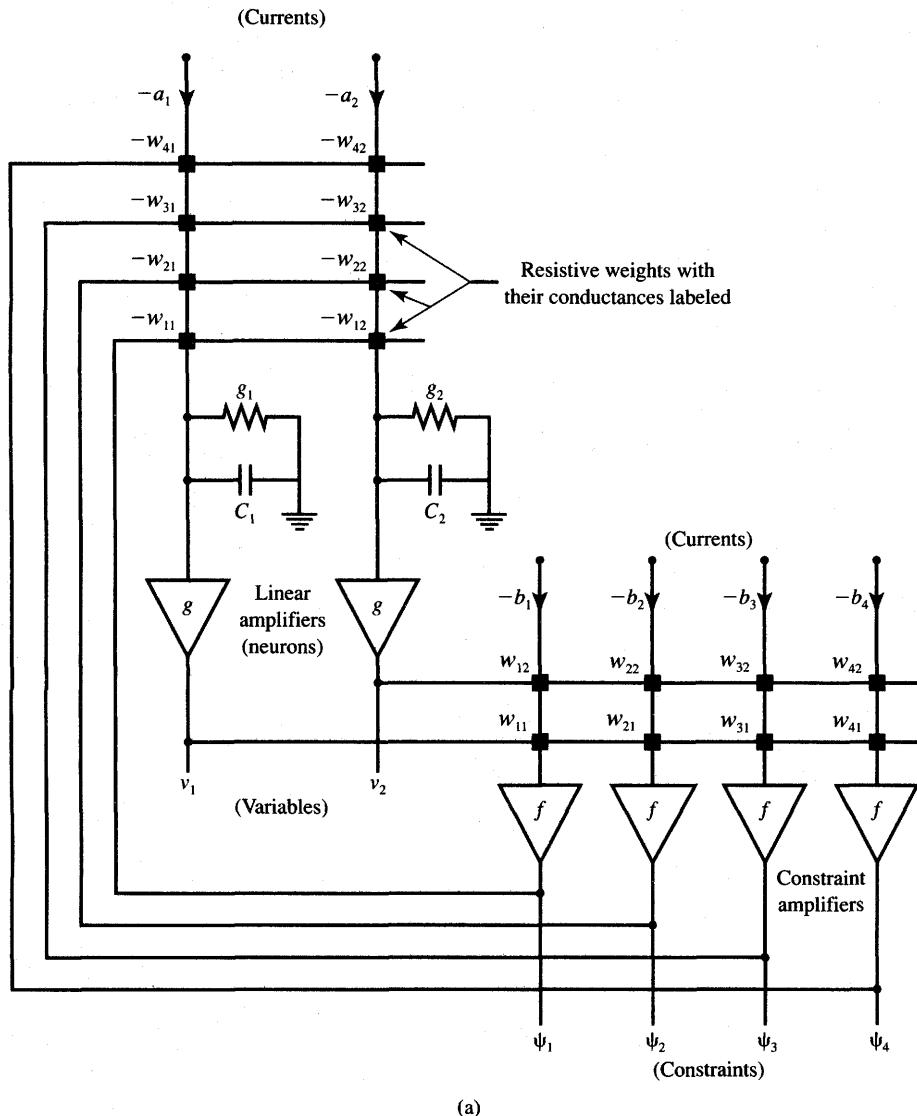
$$\mathbf{w}_j' \mathbf{v} \geq b_j, \quad \text{for } j = 1, 2, \dots, m \quad (8.2b)$$

where

$$\mathbf{w}_j \triangleq [w_{j1} \ w_{j2} \ \dots \ w_{jn}]^t$$

contains the  $n$  variable weight coefficients  $w_{ji}$  of each  $j$ 'th linear constraint inequality (8.1b).

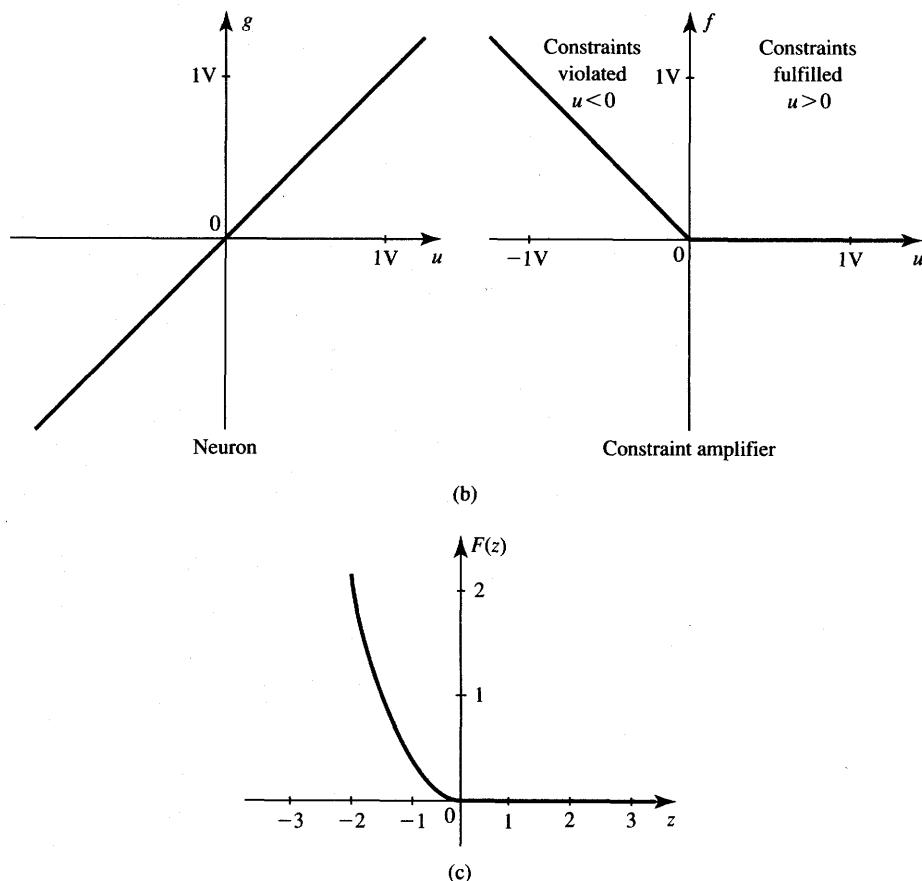
A gradient-type network based on the energy minimization concept can be used to compute the solution of problem (8.1) or (8.2). A variation of the mathematical analysis used in Chapter 5 leads to the network shown in Figure 8.1(a) for the case  $n = 2$  and  $m = 4$ . The description below follows the approach and the network design example outlined in Tank and Hopfield (1986).



(a)

**Figure 8.1a** The architecture of a single-layer feedback network that solves a two-variable four-constraint linear programming problem: (a) network. [©IEEE; reprinted from Tank and Hopfield (1986) with permission.]

The network modeling the outlined minimization task consists of  $n$  neurons with output voltages  $v_i$ , for  $i = 1, 2, \dots, n$ , which are variables in the linear programming problem. The input-output relationship of neurons is denoted as  $v_i = g(u_i)$ . Since the neurons are described here by a linear activation function, only  $n$  simple linear amplifiers of constant gain need to be used. The remaining



**Figure 8.1b,c** The architecture of a single-layer feedback network that solves a two-variable four-constraint linear programming problem (*continued*): (b) neuron and constraint amplifier characteristics, and (c) penalty function. [©IEEE; reprinted from Tank and Hopfield (1986) with permission.]

$m$  amplifiers in the network are called constraint amplifiers and are characterized by the nonlinear relationship  $\psi_j = f(u_j)$ :

$$f(u_j) \triangleq \begin{cases} 0, & \text{for } u_j \geq 0 \\ -u_j, & \text{for } u_j < 0 \end{cases} \quad (8.3a)$$

where

$$u_j = \mathbf{w}_j^T \mathbf{v} - b_j, \quad \text{for } j = 1, 2, \dots, m \quad (8.3b)$$

The voltage transfer characteristics of the neuron and of the constraint amplifier are shown in Figure 8.1(b). The nonlinear function definition (8.3a) provides for the output of the constraint amplifier to yield a large positive output value

when the corresponding constraint inequality it represents is violated, i.e., when  $u_j$  is negative. Each of the constraint amplifiers receives constraint input current proportional to the bound value  $-b_j$ , and its additional input is expressed as the linear combination of neurons' outputs in the form  $\sum_i w_{ji}v_i$ .

The output voltage  $\psi_j$  of the  $j$ 'th constraint amplifier produces the input currents to all neurons that are linear amplifiers. The input current of the  $i$ 'th neuron is proportional to  $-w_{ji}$ , as well as to the price coefficient  $-a_i$ . As with the neurons of the Hopfield model network of Figure 5.4, the  $g(u_i)$  amplifiers have input conductance and capacitance denoted as  $g_i$  and  $C_i$ , respectively. Although these components are not explicitly designed parts of the linear programming modeling network, they both represent the amplifiers' parasitic input impedance and are responsible for the appropriate time-domain behavior of the entire network. At the same time, we assume here that the response time of the  $\psi(u_j)$  constraint amplifiers is negligibly small compared to that of the  $g(u_i)$  amplifiers.

Under the assumptions stated, the KCL equation for the neuron input nodes of the network shown in Figure 8.1(a) can be expressed as it was in (5.11). It takes the form of the following state equation:

$$C_i \frac{du_i}{dt} = -a_i - u_i G_i - \sum_{j=1}^m w_{ji} \psi_j, \quad \text{for } i = 1, 2, \dots, n \quad (8.4a)$$

where  $G_i$  denotes the sum of all conductances connected to the input of the  $i$ 'th neuron and is equal to

$$G_i \triangleq g_i - \sum_{j=1}^m w_{ji} \quad (8.4b)$$

Using (8.3), the above formula can be expressed as follows:

$$C_i \frac{du_i}{dt} = -a_i - u_i G_i - \sum_{j=1}^m w_{ji} f(\mathbf{w}_j' \mathbf{v} - b_j), \quad \text{for } i = 1, 2, \dots, n \quad (8.5)$$

The discussed network for solving the linear programming task needs to minimize a meaningfully selected energy function. Assume now that the following Liapunov's function is an adequate choice for the energy function in this problem:

$$E(\mathbf{v}) \triangleq \mathbf{a}' \mathbf{v} + \sum_{j=1}^m F(\mathbf{w}_j' \mathbf{v} - b_j) + \sum_{i=1}^n G_i \int_0^{v_i} g^{-1}(v) dv \quad (8.6a)$$

where  $F(z)$  is defined for dummy variable  $z$  as follows:

$$f(z) = \frac{dF(z)}{dz} \quad (8.6b)$$

The components of the gradient vector of the assumed energy function (8.6a) can be expressed by finding its derivatives as follows

$$\frac{\partial E(\mathbf{v})}{\partial v_i} = a_i + u_i G_i + \sum_{j=1}^m w_{ji} f(\mathbf{w}_j' \mathbf{v} - b_j) \quad (8.7)$$

The time derivative of the energy function of (5.13) can now be expressed using (8.7):

$$\frac{dE}{dt} = \sum_{i=1}^n \frac{dv_i}{dt} \left( a_i + u_i G_i + \sum_{j=1}^m w_{ji} f(\mathbf{w}_j' \mathbf{v} - b_j) \right) \quad (8.8)$$

Since the bracketed expression in (8.8) is the right side of the network state equation (8.4a), the time derivative of the energy function can be rewritten in the form similar to (5.17):

$$\frac{dE}{dt} = - \sum_{i=1}^n C_i \frac{dv_i}{dt} \cdot \frac{du_i}{dt} \quad (8.9a)$$

Using (5.18), the above expression can be rearranged as follows:

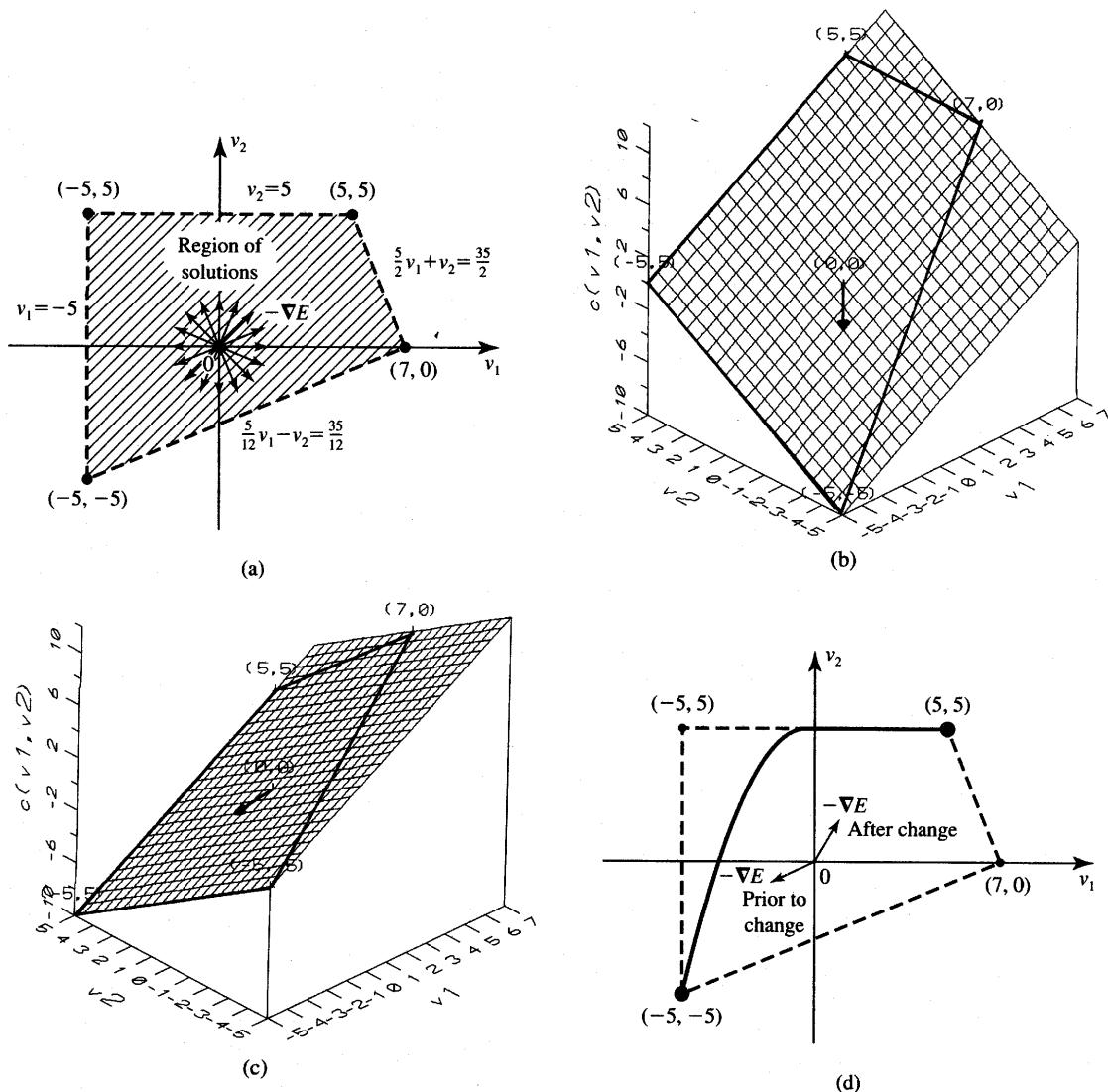
$$\frac{dE}{dt} = - \sum_{i=1}^n C_i g^{-1}(v_i) \left( \frac{dv_i}{dt} \right)^2 \quad (8.9b)$$

Since  $C_i > 0$ , and  $g^{-1}(v_i)$  is the monotone increasing function, the sum on the right side of (8.9) is nonnegative, and therefore we have  $dE/dt \leq 0$ , unless  $dv_i/dt = 0$ , in which case  $dE/dt = 0$ . Thus, the network shown in Figure 8.1 and described with the state equations (8.4) is a gradient-type network. During the transient period, the network seeks the minima of its energy function (8.6a) and the motion of the system stops there. It will seek the minima in space  $v_1, v_2, \dots, v_n$  in the direction which has the positive projection on the negative gradient vector,  $-\nabla E(\mathbf{v})$ , as explained in more detail in Section 5.3.

In fact, the cost term  $\mathbf{a}'\mathbf{v}$  is only minimized during the transient state, while the second term of (8.6a) is needed to prevent the violation of the constraint relationship (8.2b) and to push the network equilibrium point toward the edge of the allowed region. This can be seen from Figure 8.1(c) which shows the nonzero penalty function  $F(z)$  for inequality constraints that are not fulfilled. There is no penalty for constraints met when  $z > 0$ . However, a steep penalty applies if  $F(z) = (1/2)z^2$  for  $z < 0$ . This results from integrating the function  $f(u)$  introduced in (8.3a) and using the definition of  $F(z)$  from (8.6b).

A small computational network has been developed using an electronic model as shown in Figure 8.1(a) (Tank and Hopfield, 1986) to solve a two-variable linear programming problem with four constraints. The constraint amplifiers with voltage transfer functions (8.3a) are operational amplifiers with diode circuitry to provide both clamping response ( $\psi = 0$ ), and linear response ( $\psi = -u$ ) circuits. The following constraint inequalities representing inequalities (8.1b) chosen arbitrarily for the experiment are

$$\begin{aligned} v_2 &\leq 5 \\ -v_1 &\leq 5 \\ \frac{5}{12}v_1 - v_2 &\leq \frac{35}{12} \\ \frac{5}{2}v_1 + v_2 &\leq \frac{35}{2} \end{aligned} \quad (8.10a)$$



**Figure 8.2** Two-variable linear programming example: (a) constraints and solution region, (b) cost plane  $c = v_1 + v_2$ , (c) cost plane  $c = v_1 - v_2$ , and (d) trajectory when gradient vectors are switched. [Parts (a) and (d) ©IEEE; adapted from Tank and Hopfield (1986) with permission.]

By comparing coefficients of inequalities (8.10a) with those of (8.1b), weight coefficients (conductances  $w_{ij}$ ) and input biases (currents  $b_j$ ) can be computed. Figure 8.2(a) shows constraint lines (8.10a) depicted with dashed lines on the plane  $v_1$ ,  $v_2$ , which is the solution domain. The region of penalty-free solutions

is within the shaded quadrangle and includes its edges and vertices. Indeed, the solution point minimizing the cost function (8.1a) is one of the four vertices of the quadrangle shown.

The choice of the solution point by the system modeling the problem is dependent on the orientation of the cost function. Let us look at the geometrical interpretation of the cost minimization issue. The cost function itself can be depicted as a plane in the three-dimensional space  $v_1, v_2, c$ , for  $n = 2$ . Moreover, the cost plane described by Equation (8.1a) always intersects the origin. The direction of the negative gradient vector of the linear two-variable cost function  $c$  can be represented as a vector from the origin pointing in one of the directions as shown in Figure 8.2(a). The negative gradient vector of the cost function  $c$  is given as

$$-\nabla c(v) = \begin{bmatrix} -a_1 \\ -a_2 \end{bmatrix} \quad (8.10b)$$

As the cost function is changed, the cost plane tilts in a new direction, and the optimum solution of the linear programming problem may also change by displacement from one point to another. In such a case, the solution would dislocate from one vertex to another. Figures 8.2(b) and (c), show two sample cost planes defined by the cost function (8.1a) with price coefficients  $a_1 = a_2 = 1$ , and  $a_1 = 1, a_2 = -1$ , respectively. The corresponding vectors indicating the direction of the steepest descent of cost functions can be expressed as

$$-\nabla c = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \quad (8.10c)$$

and

$$-\nabla c = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad (8.10d)$$

The negative gradient vectors, shown for each of the planes in Figures 8.2(b) and (c) as coming out of the origin, indicate the directions of the transient solution of the problem for known constraints. As can be seen from Figure 8.2(a), the first vector would guarantee convergence of the network to the vertex  $(-5, -5)$ , the second would result in the solution at the vertex  $(-5, 5)$ .

For the series of experiments reported in Tank and Hopfield (1986), the output voltages  $v_1$  and  $v_2$  of the modeling network have consistently stabilized in one of the four vertices shown, dependent on the price coefficients chosen. The cost plane gradient vector was swept in a circle to check the convergence to the optimal solution in each case. Each time, the circuit stabilized at the minimum cost points corresponding to the correct constrained solution for a chosen gradient direction. In another set of experiments reported (Tank and Hopfield, 1986), the gradient vectors rapidly switched direction after the network had found the solution. This is illustrated in Figure 8.2(d). The stable solution of the problem found by the network moved accordingly from the vertex  $(-5, -5)$  to the vertex  $(5, 5)$  along the trajectory as shown. Since the solution area is always convex

for linear programming problems (Simmons, 1972), the network is guaranteed to converge to the optimum solution.

## 8.2

### CHARACTER RECOGNITION NETWORKS

As already discussed in Chapter 4, multilayer feedforward networks can be trained to solve a variety of data classification and recognition problems. Some of the examples of introductory classification applications were presented in Section 4.6. This section covers in more detail character recognition networks of both printed and handwritten characters. Initial emphasis is placed on the comparison of different learning techniques for feedforward architectures. We also demonstrate that handwritten character recognition tasks can be successfully performed by unusual and unconventional neural network designs. Specifically, successfully tested digit recognizers have employed hand-crafted multiple neural template windows, or transformations, and have involved specialized multilayer feature detectors with a limited number of degrees of freedom.

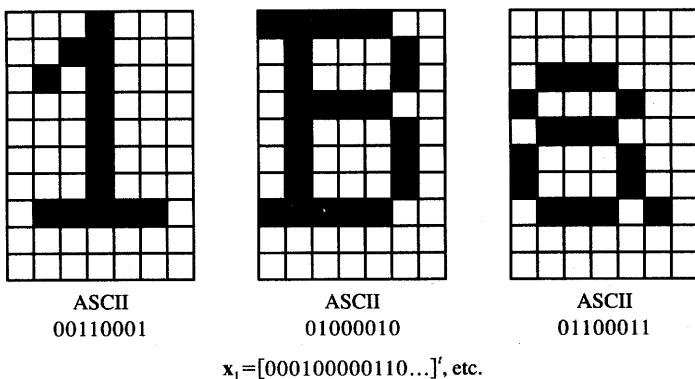
---

#### Multilayer Feedforward Network for Printed Character Classification

This section provides a comprehensive performance evaluation of the limitations and capabilities of multilayer feedforward networks for printed character recognition. Network learning parameters, architectures, and learning profiles are investigated (Zigoris 1989). Based on the comparison of different network architectures, the best size and training parameters will be devised for printed character classifiers (Zurada et al. 1991).

An introductory study of a multilayer continuous perceptron network for simple printed character recognition has been reported by Chandra and Sudhakar (1988). The performance of various classifiers was discussed and compared for recognition of digits "0" through "9" encoded as 20-tuple binary-valued input vectors representing black and white pixels. An error back-propagation network for classification of these data was evaluated. The network is of a 20–10–10 size denoting the number of input nodes, hidden-layer neurons, and output neurons, respectively. This convention for denoting network size using symbols  $I-J-K$  is consistent with Figure 4.7 and with the notation in earlier chapters. At least 1000 training cycles were used to develop the network, and reasonable classification results were achieved and reported by Chandra and Sudhakar (1988).

A similar study was done on pixel-image classification of amoebae nuclei images (Dayhoff and Dayhoff 1988). This application, which is similar in



**Figure 8.3** Bit maps for sample characters “1,” “B,” and “a,” for recognition.

principle to printed character recognition, involved a gray-scale pixel matrix with geometric patterns that need to be recognized. The  $7 \times 9$  pixel-grid signal was used and fed through 16 hidden-layer neurons to a  $7 \times 9$  output grid. The study was based on four training images; 3000 cycles were required to train the network successfully.

Let us review the performance of multilayer feedforward architectures for classification of a full set of printed lowercase and uppercase letters, digits, and punctuation characters. In this application, standard dot-matrix printer characters need to be recognized robustly and correctly from a set of 95 characters. Each character is a 7-column  $\times$  10-row pixel matrix defined by Apple Computer (1986). The example bit maps for three characters from the set are shown in Figure 8.3. The training set consisting of 95 characters has also been used to test the performance of the trained network, and it has provided a relatively large number of input/output pattern pairs. Depending on the requirements for the classifying network, the output pattern has been chosen either as an 8-bit ASCII code of the character (distributed representation) or a single output bit that has been set to 1 for each character pattern to be recognized (local representation). The total of either 8 or 95 output neurons has been required, respectively, for the output layer trained in each of the output representations mentioned.

Continuous unipolar neurons are employed throughout this experiment for training and recall. The binary input vector representing the pixel pattern of each character has been fed directly into the network. No effort has been made to perform either preprocessing or feature extraction of any kind in this experiment. The outputs have been thresholded at 0.5 to obtain a binary output value in the test mode. The three basic network properties investigated for this application were the learning constant  $\eta$ , the activation function steepness coefficient  $\lambda$ , and the network architecture. The topologies of the network under study have been limited out of necessity since the number of different architectures potentially relevant for this classification task is enormous.

The number of input nodes  $I$  has been kept at a constant value of 70, equal to the number of pixels. Single and double hidden-layer networks with different layer sizes denoted as  $J$  have been evaluated for both 8 and 95 output neurons. The number of output neurons is denoted as  $K$ . Ideally, the network should be able to generate 8-bit ASCII encoded output. Such an 8-bit output architecture using eight output neurons is of particular importance because it combines two steps. Both recognition and encoding of the character pattern are performed by the network in a single processing step. In addition, this architecture yields more concise classification results compared to the local representation which requires 95 output nodes.

The simulation experiments described later in this section indicate that the number of neurons in each of the hidden layers can drastically affect both the accuracy and the speed of training. Some patterns require decision regions that cannot be realized by a low number of hidden layer neurons. Too many neurons in the hidden layer, however, can prevent effective training from occurring since the initial decision regions formed become too complex due to the excessive number of neurons for the application. The end result is often that the network weights get readily trapped in a local minimum during training and no further learning occurs.

The performance of the network has been measured in terms of its response errors with respect to correct classification results. Four different error measures have been monitored during the training and recall phases. They are  $E_{\max}$ ,  $E_{\text{rms}}$ ,  $N_{\text{err}}$ , and  $E_d$ .

The maximum error denoted as  $E_{\max}$  expresses the largest of the single neuron errors encountered during the full training cycle consisting of  $P = 95$  steps, where  $P$  denotes the size of the training set. A value of 1 implies that at least 1 bit is totally incorrect. Since the character recognition application requires binary output, the network implements a postprocessing thresholding function to yield binary output decisions. The error  $E_{\text{rms}}$  is defined as in (4.35) and it serves as an ordinary root-mean-square normalized quadratic error measure, before thresholding. The absolute number of bit errors  $N_{\text{err}}$  at thresholded outputs during the single complete training cycle is also used as a local measure of comparison of learning quality and speed. The decision error  $E_d$  defined in (4.36) is used as a normalized error value computed over the total number of output bit errors  $N_{\text{err}}$  resulting at  $K$  thresholded outputs over the complete training cycle consisting of presentation of  $P$  patterns.

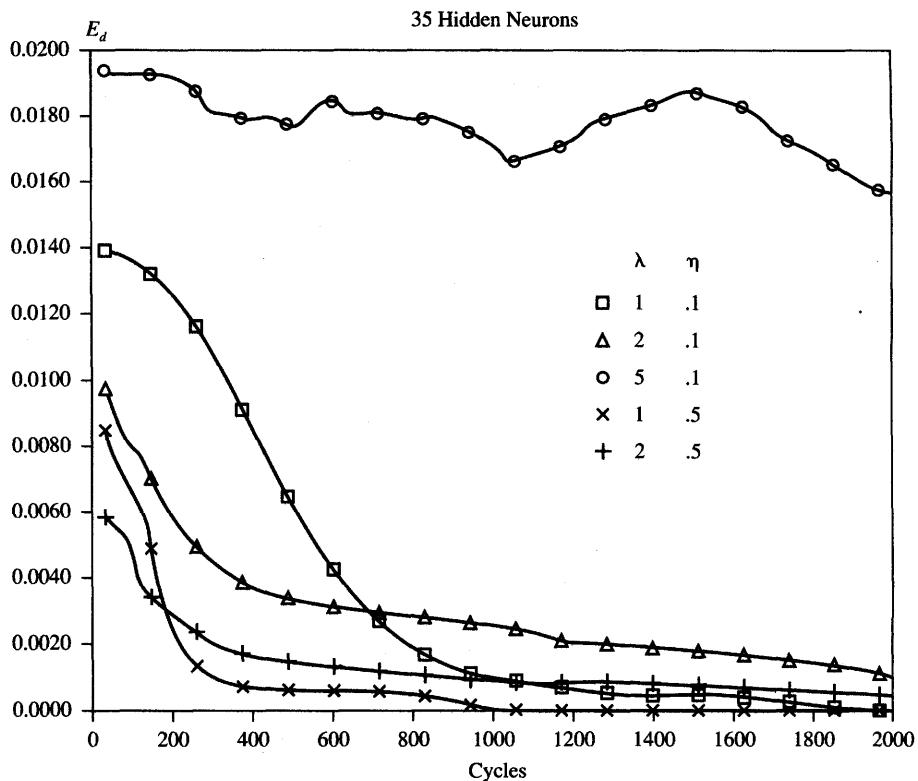
The simulated error back-propagation training runs were performed in two phases (Zurada et al. 1991). The first phase consisted of a large number of preliminary runs with various combinations of parameters and architectures. The runs were performed to determine an overall training effectiveness for various and often intentionally diversified learning conditions. This can be termed as a coarse search for suitable architectures. During the second phase of the runs, optimal parameters for network learning were explored. This second phase emphasized a fine heuristic search for optimum training conditions.

**TABLE 8.1**

Evaluation of 70-J-95 architecture, single hidden layer,  $J = 35$  or  $70$  (printed character recognition network).

Run	Steepness Factor $\lambda$	Learning Constant $\eta$	Hidden Layer $J$	Training Length (cycles)	Max Error $E_{\max}$	$E_{\text{rms}}$	Bit Errors $N_{\text{err}}$	Decision Error $E_d$
1	1	0.1	70	3600	0.9991	$10^{-6}$	1	0.0001
2		0.5	70	575	0.0209	$10^{-6}$	0	0
3	1	0.1	35	1900	0.1279	$10^{-6}$	0	0
				3975	0.0033	$10^{-6}$	0	0
4		0.5	35	1000	0.0627	$10^{-6}$	0	0
				3975	0.0004	0	0	0
5	1	0.1	35	3500	1	$2 \cdot 10^{-6}$	4	0.0004
6		0.5	35	3975		$2 \cdot 10^{-6}$	2	0.0004
7	2	0.1	70	3450	1	$4 \cdot 10^{-6}$	12	0.0013
				3975		$4 \cdot 10^{-6}$	12	0.0013
8		0.5	70	2400		$2 \cdot 10^{-6}$	5	0.0006
				3975		$2 \cdot 10^{-6}$	5	0.0006
9		0.1	35	3975		$8 \cdot 10^{-6}$	51	0.0057
10	5	0.1	70	2200	1	$9 \cdot 10^{-6}$	68	0.0075
				3975		$9 \cdot 10^{-6}$	68	0.0075
11		5	70	250		$1.9 \cdot 10^{-5}$	281	0.0311
				3975		$1.9 \cdot 10^{-5}$	280	0.0310

Table 8.1 shows results of the initial series of runs for a single hidden layer network of size 70-J-95 with  $J = 35$  or  $70$ . The training length for each run was different and involved up to several thousand cycles as listed in the fifth column of the table. The most important result of the preliminary training runs has been that for  $\lambda = 5$  the network did not train properly (runs 9–11). The lowest number of single bit errors was 51 for 35 hidden units when  $\eta = 0.1$ . The number of errors got worse as the number of hidden units was increased. In practical simulation the error tends to oscillate. Since the activation function around zero has been very steep for this particular series of runs, this caused overcorrections and the weights did not settle at their proper value but instead

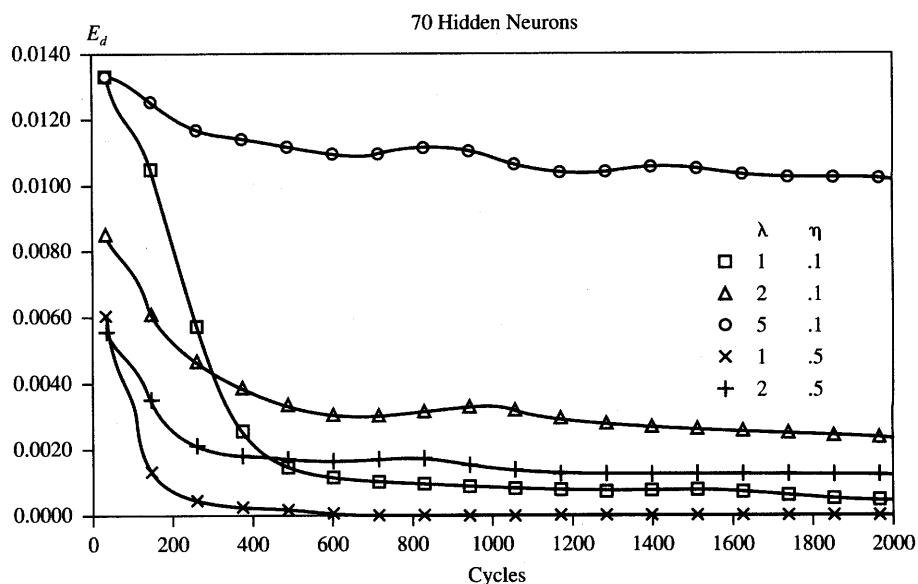


**Figure 8.4** Learning profiles for a network with 35 hidden neurons; local representation.

swung from one extreme to another. Further increases of  $\eta$  tend to aggravate the problem by further overcorrections and have led to increased error values.

The problem of excessive errors is reduced with  $\lambda = 2$ , which ensures slower learning, but at least several bit errors have consistently occurred (runs 5–8). Finally, runs with  $\lambda = 1$  seem to yield the best results (runs 1–4). Very small error values have been obtained for both 35 and 70 hidden units and for learning constants having a value of either 0.1 or 0.5. The training for 0 bit errors required fewer cycles with 70 hidden units (run 2); however, a 70–35–95 network with  $\lambda = 1$  initially turned out to be the best for the task evaluated (runs 3 and 4). Simulations for 140 hidden units resulted in bit errors that were unacceptably high (637 and 1635), oscillatory, and involved prolonged training sessions.

Figure 8.4 illustrates the decision error of the recall phase versus the training length for the 70–35–95 network. It has been observed from a series of simulations that most training runs for the same network configuration and learning

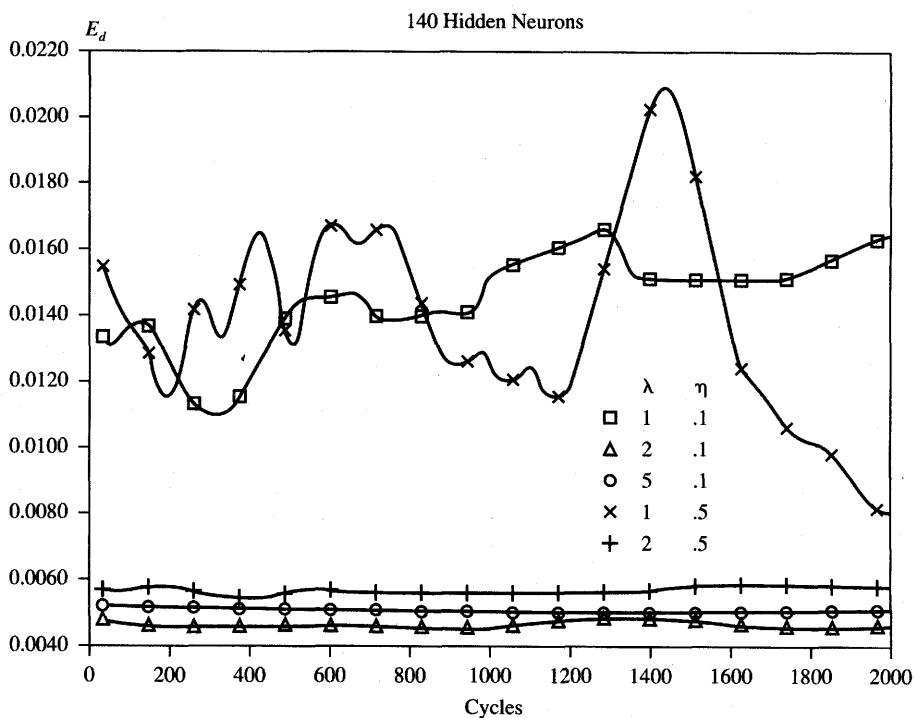


**Figure 8.5** Learning profiles for a network with 70 hidden neurons; local representation.

parameters yield only a slightly different learning profile. This is due to the fact that weights are initialized at random values. Thus, the learning profiles shown in the figure can be considered typical, or average, for the selected network size and training parameters. The best learning profile and the least final error is achieved for  $\lambda = 1$  and  $\eta = 0.5$  (run 2). The decision error  $E_d$  typically reduces to zero after several hundred training cycles. The error curves on Figure 8.4 for  $\lambda = 1$  and 2 also show reasonable training speed and accuracy.

In contrast, the decision error  $E_d$  for  $\lambda = 5$ ,  $\eta = 0.1$  does not converge to a reasonable value. Somewhat similar learning profiles have resulted for 70 hidden-unit networks. Training results for this network configuration are depicted in Figure 8.5. Figure 8.6 shows the results of a 140 hidden-neuron network training process. Notice that the error axis now starts at  $E_d = 0.04$ . The decision error does not undergo substantial reduction in spite of continued training. This configuration was therefore found to be unacceptable and was not pursued beyond the preliminary phase of training runs.

Table 8.2 shows the simulation results for double hidden-layer networks with the output layer of eight neurons. For this configuration the characters need to be encoded by the classifier in the ASCII format. Various architectures that have been tested employ the first hidden layer of size  $J_1 = 35$  and 70, and the second hidden layer of size  $J_2 = 8$  and 70. The results for double hidden layers are, in general, better than for a single hidden layer, and the final errors



**Figure 8.6** Learning profiles for a network with 140 hidden neurons; local representation.

are smaller. Three specific architectural hidden layer sizes ( $J_1, J_2$ ) that have been simulated are: (70, 70), (70, 8), and (35, 8) (Zurada et al. 1991).

Simulations 1 through 6 correspond to  $\lambda = 1$ . Virtually all runs show reasonable results and a small number of bit errors. Runs 2 and 4 have the most bit errors, whereas all remaining runs converge to the single bit error. These particular runs have  $\eta = 0.5$  in common. In addition to one bit error, runs 3, 5, and 6 also display the fastest convergence. Runs 7 through 12 are performed for  $\lambda = 2$ . These results are more varied and consist of some excellent results along with some extremely bad ones. The notably bad results are runs 10 and 12. Not only has training been excessively long, but run 10 resulted in an oscillatory bit error with a peak value of 103, and run 12 resulted in a large error of 150 incorrect decision bits. Runs 7 and 8 exhibited superb results since they both converged to zero bit errors. In addition, convergence of training to zero bit errors was extremely quick, 90 cycles for run 7, and only 40 cycles for run 8. Both these runs had  $J_1 = 70$  and  $J_2 = 70$  in common. Given this large number of hidden-layer units, the redundancy in feature recognition for this network is rather large. This seems to be the most likely reason for this particular network architecture to have produced no bit errors.

**TABLE 8.2**

Evaluation of 70-J1-J2-8 architecture, double hidden layer (printed character recognition network).

Run	Steepness Factor $\lambda$	Learning Constant $\eta$	Hidden Layers		Training Length (cycles)	Max Error $E_{\max}$	$E_{\text{rms}}$	Bit Errors $N_{\text{err}}$	Decision Error $E_d$
			J1	J2					
1	1	0.1	70	70	2190	1	$58 \cdot 10^{-6}$	1	0.0013
2		0.5	70	70	310		$83 \cdot 10^{-6}$	3	0.0039
3	1	0.1	70	8	390	1	$98 \cdot 10^{-6}$	1	0.0013
4		0.5	70	8	980		$137 \cdot 10^{-6}$	8	0.0105
5	1	0.1	35	8	620	1	$80 \cdot 10^{-6}$	1	0.0013
6		0.5	35	8	870		$87 \cdot 10^{-6}$	1	0.0039
7		0.1	70	70	90	0.2034	$47 \cdot 10^{-6}$	0	0
	2				2520	0.0001	$3.6 \cdot 10^{-6}$	0	0
					4990	0.0001	$3.6 \cdot 10^{-6}$	0	0
8		0.5	70	70	40	0.9994	$58 \cdot 10^{-6}$	0	0
					210	0.0041	$11 \cdot 10^{-6}$	0	0
					830	0.0001	$3.6 \cdot 10^{-6}$	0	0
					3300	0.0000	0	0	0
9	2	0.1	70	8	2160	1	$51 \cdot 10^{-6}$	1	0.0013
10		0.5	70	8	4990	1	$435 \cdot 10^{-6}$	103	0.1355
11	2	0.1	35	8	110	1	$98 \cdot 10^{-6}$	1	0.0026
12		0.5	35	8	4990	1	$551 \cdot 10^{-6}$	150	0.1973

Table 8.3 summarizes the results of two runs with  $J_1 = J_2 = 70$ ,  $K = 8$ ,  $\lambda = 2$ ,  $\eta = 0.1$ , and  $\eta = 0.5$ . Since this configuration with this choice of learning parameters has been so successful, additional simulations have been performed for this case to ensure consistency and to allow possibly for further optimization of the training. Both runs in Table 8.3 show excellent results, with run 1 converging to zero decision errors after 190 cycles, and run 2 converging to zero decision errors after only 60 cycles.

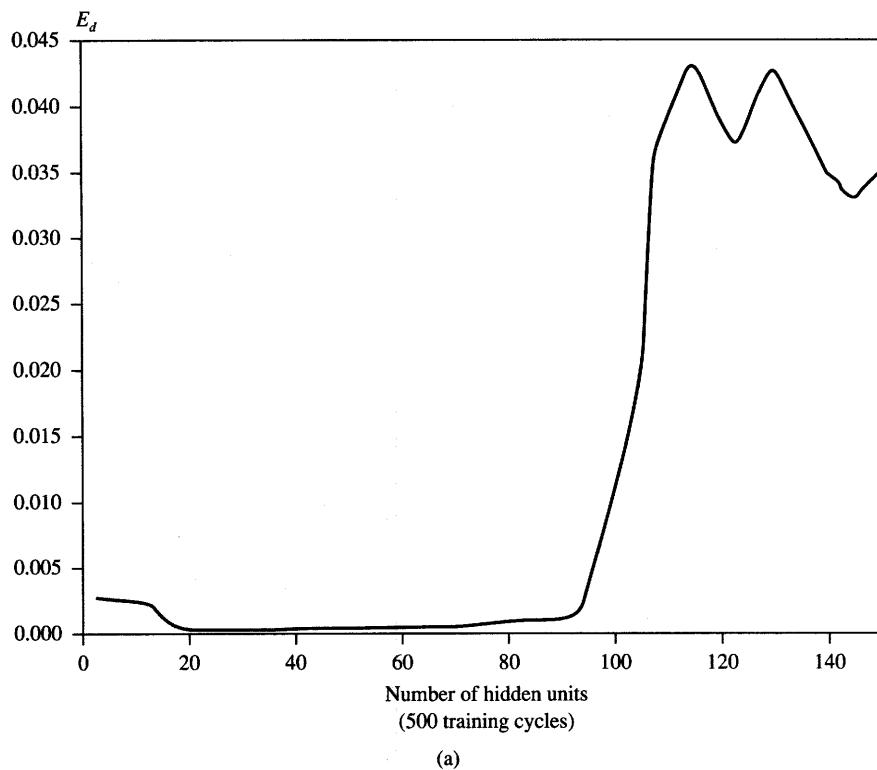
The discussion presented thus far indicates that the number of hidden layers and their sizes are of crucial importance for a particular application. Apparently a range of values exists for the number of hidden layers and their size that yields especially good results. Figure 8.7(a) shows the final decision error  $E_d$  for  $\lambda = 1$

**TABLE 8.3**

Evaluation of 70-70-70-8 architecture (printed character recognition network).

Run	Steepness Factor $\lambda$	Learning Constant $\eta$	Hidden Layers		Training Length (cycles)	Max Error $E_{\max}$	$E_{\text{rms}}$	Bit Errors $N_{\text{err}}$	Decision Error $E_d$
			J1	J2					
1	2	0.1	70	70	1	0.999	$798 \cdot 10^{-6}$	374	0.492
					10	0.999	$529 \cdot 10^{-6}$	160	0.210
					20	0.999	$428 \cdot 10^{-6}$	106	0.139
					30	0.998	$319 \cdot 10^{-6}$	50	0.066
					40	0.997	$214 \cdot 10^{-6}$	23	0.030
					50	0.997	$120 \cdot 10^{-6}$	4	0.005
					60	0.996	$98 \cdot 10^{-6}$	3	0.004
					70	0.996	$91 \cdot 10^{-6}$	3	0.004
					80	0.996	$76 \cdot 10^{-6}$	2	0.003
					90	0.993	$76 \cdot 10^{-6}$	2	0.003
					190	0.042	$29 \cdot 10^{-6}$	0	0
					200	0.020	$22 \cdot 10^{-6}$	0	0
					1500	0.0003	$3.6 \cdot 10^{-6}$	0	0
					2000	0.0002	$3.6 \cdot 10^{-6}$	0	0
2	2	0.5	70	70	1	0.997	$789 \cdot 10^{-6}$	329	0.433
					10	0.998	$660 \cdot 10^{-6}$	244	0.321
					20	0.999	$428 \cdot 10^{-6}$	105	0.138
					30	0.996	$192 \cdot 10^{-6}$	18	0.024
					40	0.910	$54 \cdot 10^{-6}$	1	0.001
					50	0.756	$51 \cdot 10^{-6}$	1	0.001
					60	0.005	$18 \cdot 10^{-6}$	0	0
					70	0.003	$14 \cdot 10^{-6}$	0	0
					420	0.002	$3.6 \cdot 10^{-6}$	0	0
					1990	0.000	$3.6 \cdot 10^{-6}$	0	0

and  $\eta = 0.1$  versus the number of hidden units of the single hidden layer. The error has been computed after 500 training cycles for the number of hidden layer neurons covering the range from 5 to 150. The number of output units is kept equal to 8 thus ensuring an ASCII format for the output. The results indicate that low error values have been achieved within the range from 20 to 90 hidden units. The network with less than 20 hidden neurons has been incapable of learning all patterns, presumably, because the weight space has been too complicated. The error level has also become intolerable for more than 90 hidden units. The

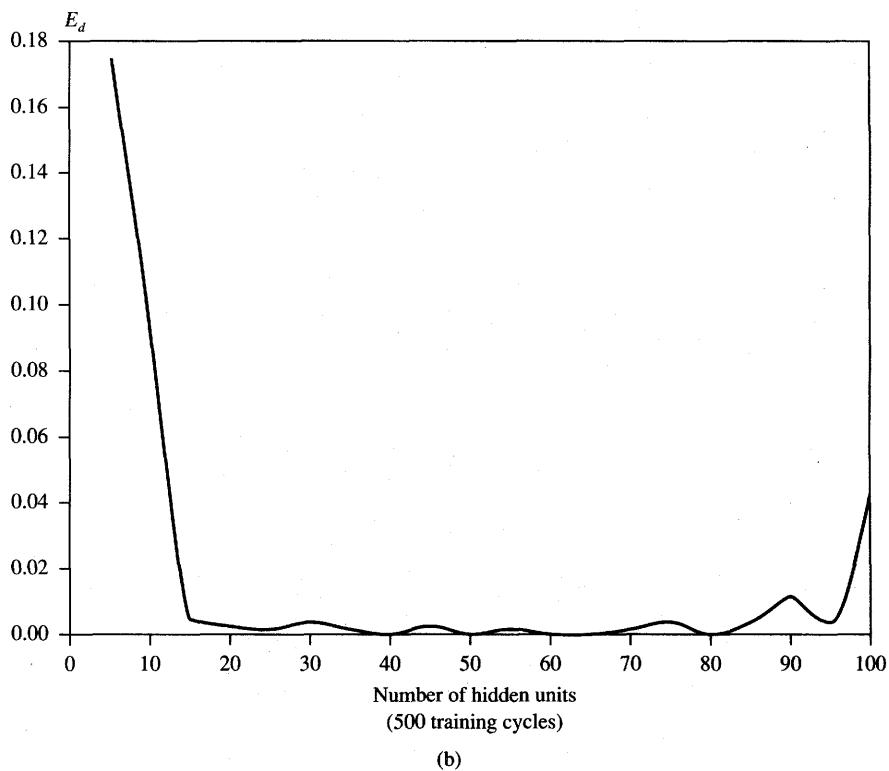


**Figure 8.7a** Decision error versus the number of hidden units,  $\lambda = 1$  and  $\eta = 0.1$ : (a) single hidden-layer network.

system seems to be seriously overdetermined because of the large number of hidden units.

The training of a network with two hidden layers yielded superior results. Figure 8.7(b) shows the final decision error  $E_d$  versus the number of hidden units. In this training simulation, the number of hidden units is assumed to be variable but identical for both layers. The resulting optimum architecture is somewhat similar to that of the single hidden-layer network, ranging from 20 to 85 units. The primary difference was that the two hidden-layer network version tends to learn more reliably and quickly.

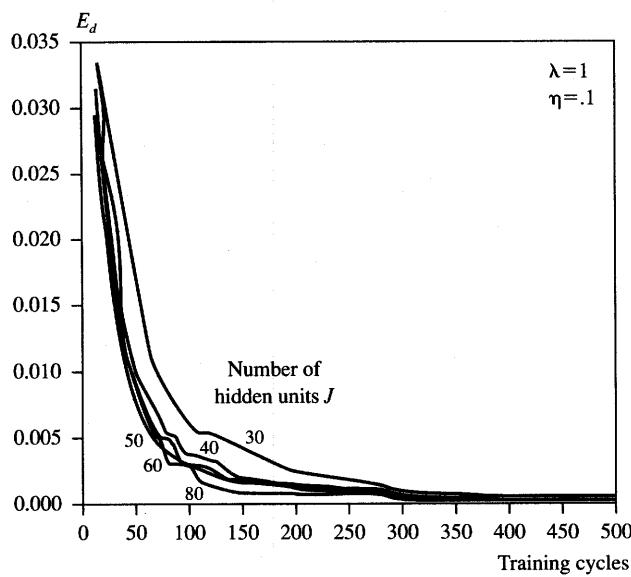
Figure 8.8 shows selected learning profiles for the training of single-layer networks. Several hidden-layer sizes covering the range from 20 to 80 units have been tested to find an optimum architecture. The number of training cycles needed to achieve a fixed decision error value seems to remain largely invariant with respect to the number of hidden-layer units. Although it is impossible to express analytically the error profiles as a function of training length, the simulations



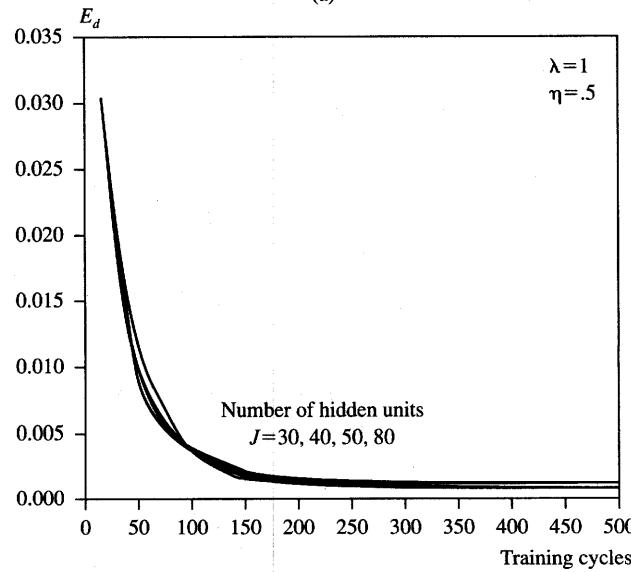
**Figure 8.7b** Decision error versus the number of hidden units,  $\lambda = 1$  and  $\eta = 0.1$  (continued): (b) double hidden-layer network of identical sizes ( $J_1 = J_2$ ).

seem to indicate that the decision error is approximately inversely proportional to the number of training cycles. With continued training, the error somewhat saturates at a low level. These conclusions seem to hold only for the case of feasible architectures, i.e., architectures that can learn the problem under study.

For this nontrivial application of multilayer feedforward networks for character classification, a number of reasonable solutions emerge. Let us summarize the results of the classification case study discussed in this section. The computationally most efficient value of activation steepness,  $\lambda$ , seems to be of unity value throughout each learning session. The learning constant of 0.5 works best with the number of hidden units ranging from 25 to 80. This size of hidden layer seems to be the most advantageous for both one and two hidden-layer systems. The best classification results are obtained with a two hidden-layer network employing about 70 hidden units per layer. Learning within this architecture has been quick and reliable; this size of layers presumably provides excellent redundancy. Remarkably, it has also been found that this size could be reduced



(a)



(b)

**Figure 8.8** Learning profiles for a single hidden-layer network,  $K = 95$ : (a)  $\eta = 0.1$  and (b)  $\eta = 0.5$ .

to approximately 25 without any major deterioration of classification. Training of networks with both too many or too few hidden units has resulted in slow learning with a somewhat excessive number of decision errors.

Finally, the configuration with 8 output neurons turns out to be generally superior compared to the local representation of classification involving 95 output neurons. Although both these architectures ensure a zero final decision error and thus both achieve perfect character classification, the configuration with 8 output neurons trains faster. It also results in a smaller network.

---

### **Handwritten Digit Recognition: Problem Statement**

The ultimate success of neural networks depends on their effectiveness in solving a variety of real-life classification, recognition, or association problems that are more demanding and more difficult than any of the problems studied thus far. Let us look at recognition of objects that are drastically less structured than standard printed characters. In this group of problems, character recognition of handwritten digits represents an important example of a realistic, yet difficult, benchmark recognition task. This task has a clearly defined commercial importance and a level of difficulty that makes it challenging but still not so large and complex so as to be completely intractable. Recent technical reports have demonstrated that neural networks can perform handwritten digit recognition with state-of-the-art accuracy (Jackel et al. 1988; LeCun et al. 1989; Howard et al. 1990). In a number of respects, neural network solutions surpass even those obtained using other approaches. The basic neurocomputing techniques used for handwritten character classification are reviewed below.

Analysis shows that many difficult pattern recognition problems can be formulated as multidimensional curve fitting using the methods of approximation learning described in Section 2.4. In character recognition, however, the general rules for distinguishing between characters are neither known, nor have they been formulated. The best current process is to examine a large cross section of the character population with the goal of finding a function that adequately generalizes the exemplars from the training set. This would possibly allow for recognition of the test characters not contained in the original training set. Let us look at the neural network learning algorithms and approaches that first preprocess and then map handwritten digit character images to one of the ten categories. As we will see, neural network classifiers often yield comparable or better accuracy and, more importantly, require far less development time compared to conventional classifiers.

The example database used to train and test the network consists of 9298 isolated numerals digitized from handwritten zip codes that appear on U.S. mail envelopes (LeCun et al. 1989). Typical digit examples used for this project are

80322-4129 80206

40004 14310

37872 05753

~~35502~~ 75216

35460 44209

1011913485726803224414186  
 6359720299299722510046701  
 3084111591010615406103631  
 1064111030475262009979966  
 8912086708557131427955460  
 2018730187112993089970984  
 0109707597331972015519066  
 1075318255182-814358090943  
 17875416554603546055  
 18255108503047520439401

**Figure 8.9** Examples of handwritten zip codes (top) and normalized digits from the training/test database (bottom). [© MIT Press; reprinted from LeCun et al. (1989) with permission.]

shown in Figure 8.9. The digits are written by many different people using a great variety of sizes, writing styles, instruments, and with a widely varying amount of care. It can be noted that many of the digits are poorly formed and are hard to classify, even for a human. Of the 7291 sample digits used for training, 2007 have been used for test purposes. The captured digits are first normalized to fill an area consisting of  $40 \times 60$  black and white pixels. The resulting normalized patterns are presented to a neural network after being reduced to  $16 \times 16$  pixel images. Further processing is performed by a mix of custom hardware and specialized software for image processing.

The general, common strategy for handwritten character processing is to extract features from the images and then perform the classification based on the resultant feature map (Graf, Jackel, and Hubbard 1988). As the first step of the recognition sequence, a digital computer equipped with frame-grab hardware captures a video image of a handwritten character. The computer thresholds the gray-level image into black and white pixels. This stage can be termed "image capturing." Next, the image is scaled both horizontally and vertically to fill a  $16 \times 16$  pixel block. The  $16 \times 16$  scaled character format appears to be of adequate resolution for the set of 10 digits, but would probably be too small to deal with letters or more complex characters.

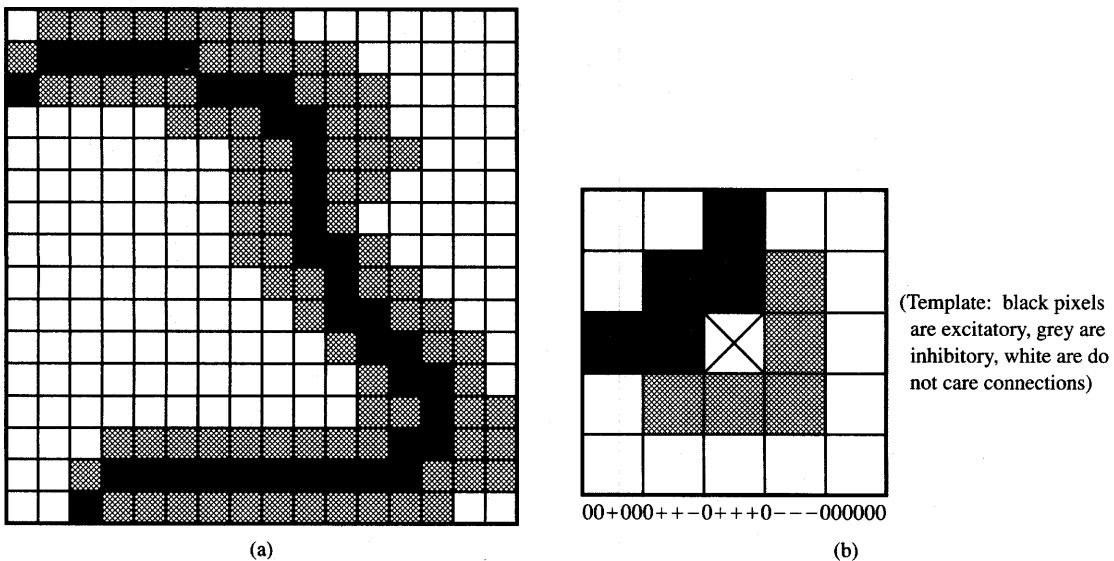
---

### Recognition Based on Handwritten Character Skeletonization

This section reports on one of the earlier attempts at building neural network-based handwritten character recognizers (Jackel et al. 1988; Graf, Jackel, and Hubbard 1988). The image capturing and scaling as described in the preceding paragraph is followed by the "skeletonization." This is a process that makes the lines one dimensional by removing meaningless linewidth variations. The rationale for line-thinning preprocessing is that a smaller number of features is sufficient to analyze the skeletonized images. Since the linewidth does not carry much information about the character itself, the skeletonization removes pixels of the image until only a backbone of the character remains, and the broad strokes are reduced to thin lines. Skeletonization is illustrated in Figure 8.10(a) where the gray area represents the original character, the digit 3, and the black area is its skeletonized image.

The skeletonization is implemented through scanning of the  $5 \times 5$  pixel window template across the entire image. Twenty different 25-bit window templates are stored on the neural network chip. These templates are responsible for performing the skeletonization of the images. One of the templates is shown in Figure 8.10(b). During scanning each template tests for a particular condition that allows the deletion of the middle pixel in the window. If the match of the image pixels to any of the templates exceeds a preset threshold value, the center pixel of the character bit map is deleted.

In this manner, the entire  $16 \times 16$  pixel image is scanned, and a decision is made for each pixel in the image whether that pixel just makes the line fat and can be deleted, or whether its presence is crucial to keep the connectivity of the character intact (Graf, Jackel, and Hubbard 1988). The 20 different templates for scanning images were crafted by network designers in a systematic but *ad hoc* manner. Examples of a single scanning pass through the image using four selected window templates are shown in Figure 8.11. The black pixels deleted

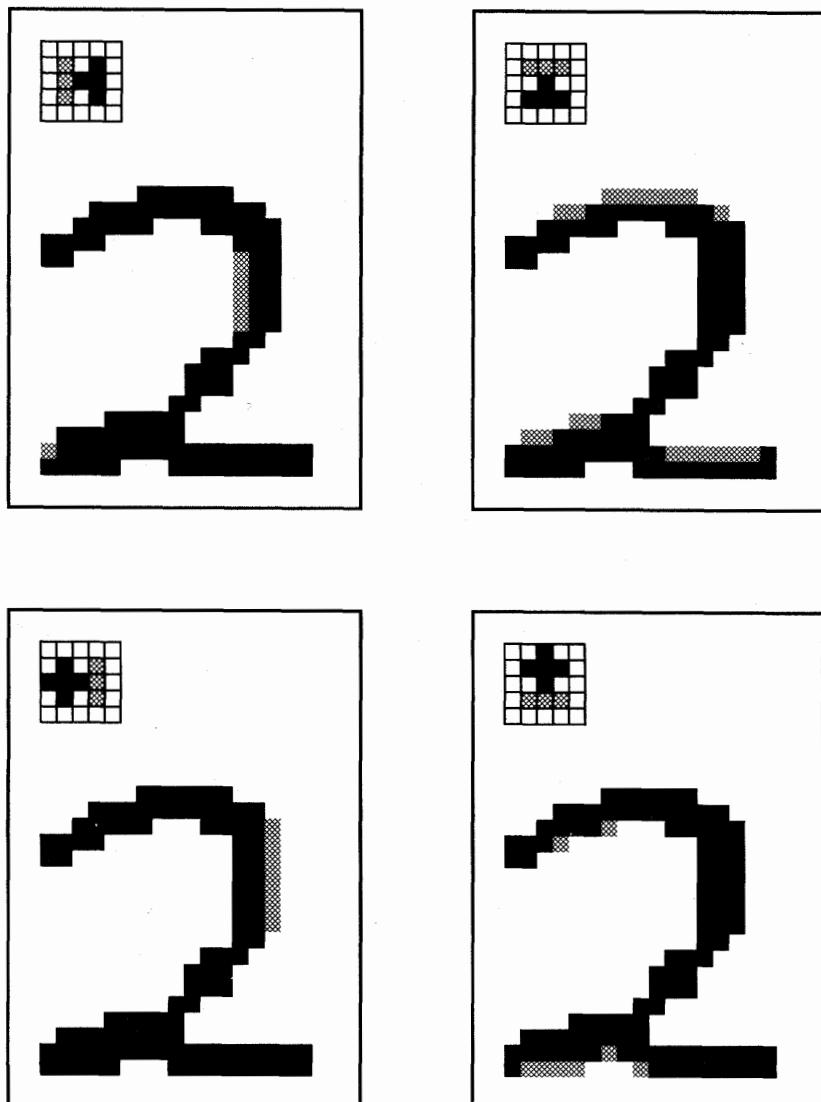


**Figure 8.10** Skeletonization of handwritten characters: (a) pixel image of digit 3 and (b) example of one of the window templates used for line thinning. [©IEE; reprinted from Graf, Jackel, and Hubbard (1988) with permission]

in the images shown as a result of each template pass are marked in gray on the four images in each box.

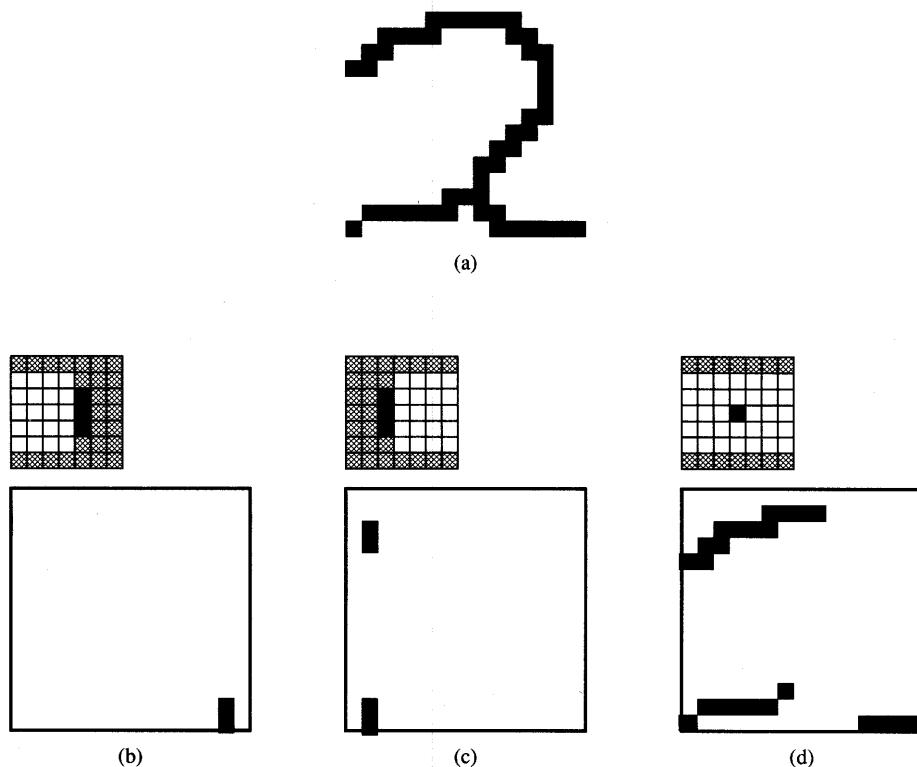
The skeletonization process is followed by the feature extraction process for each character. In the feature extraction stage, the thinned image is presented to a number of  $7 \times 7$  feature extracting pixel window templates. The neural network chip stores 20 different feature extracting window templates, again chosen *ad hoc*, but inspired by results from experimental neurobiology. Feature extraction processing by means of window templates is illustrated in Figure 8.12. The templates check for the presence of oriented lines, oriented line ends, and arcs in the image of a skeletonized character, for example, like the one displayed in Figure 8.12(a). Examples of the extracted features of line endstops and horizontal lines are shown in Figures 8.12(b), (c), and (d). Whenever a feature template match exceeds the preset threshold, a 1 is set in a blank feature map for that corresponding feature. In this way, a feature map is produced for every template. The maps of such features for each evaluated character skeleton are then subjected to an "OR" function after the scan is completed (Jackel et al. 1988).

The feature extraction process terminates by mapping the skeletonized input pattern of  $16 \times 16$  size image into 20 different  $16 \times 16$  feature images. To compress the data, each feature map is then coarse-blocked into a  $3 \times 3$  array. As a final result of processing for feature extraction, 20 different  $3 \times 3$  feature



**Figure 8.11** The skeletonization process, four window templates and four passes shown. (Templates: black pixels are excitatory, gray are inhibitory, white are do-not-care connections; digits: gray pixels are deleted by each template.) [©IEEE; adapted from Jackel et al. (1988) with permission.]

vectors, consisting jointly of 180 feature entries, are produced. These entries are then classified into one of the 10 categories signifying each of the 10 digits. Several conventional classification algorithms have been used to determine the class membership of the resulting vector. The overall results of this method



**Figure 8.12** Examples of feature extraction: (a) skeletonized image for future extraction, (b) and (c) features of line endstops, and (d) features of horizontal lines. (Templates: black pixels are excitatory, gray are inhibitory, white are do-not-care connections.) [©IEEE; reprinted from Jackel et al. (1988) with permission.]

for handwritten character recognition have varied between 95% accuracy for hastily written digits and 99% accuracy for carefully written digits (Jackel et al. 1988). The tests were performed on digits taken again from the U.S. post office databases.

The approach to handwritten digit recognition described in this section is based on skeletonization and feature extraction using *ad hoc* experimentation and hand-chosen templates. This concept uses specially developed window templates and also a carefully organized scanning process. Through the scanning of suitable window templates across the character bit map, the original pattern of a character is decomposed into a single large-size feature vector. This is then followed by image compression. The sequence of processing is implemented by a neural network chip designed specifically for this purpose. The final classification of the feature vector obtained from each character is obtained using conventional computation.

The reader may have noticed that the approach presented in this section involves neither neural learning techniques nor the known recall algorithms described in the preceding chapters. Rather, the approach is based on analyzing the matching of the pattern with the number of movable window templates. The matching involves processing of the pattern segment through suitably chosen windows that represent weights for the pixels overlapping with the evaluated image. As we see, this approach, in which neural network learning has been replaced by using intuitively produced weight matrices associated with each window template, can be termed rather unconventional.

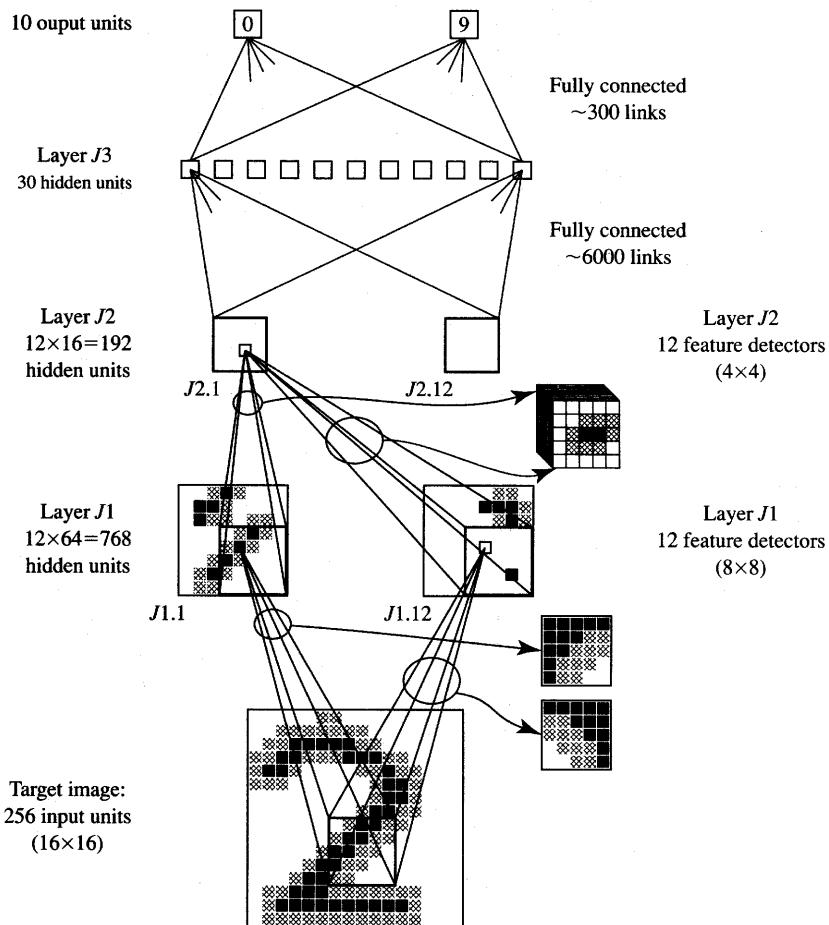
---

### Recognition of Handwritten Characters Based on Error Back-propagation Training

Unlike the network described in the preceding section, the multilayer feed-forward neural network for handwritten character recognition can be directly fed with images obtained as slightly preprocessed original patterns containing low-level information. First, the  $40 \times 60$  pixel binary input patterns are transformed to fit into a  $16 \times 16$  pixel frame using a linear transformation. This frame contains the target image. The transformation preserves the aspect ratio of the character, but the resulting target image consisting of 256 entries is not binary. It has multiple gray levels because a variable number of pixels in the original input pattern can fall into a given pixel of the target image. The gray levels of each resulting target image are then scaled and translated to fall within the bipolar neuron's output value range,  $-1$  to  $+1$  (LeCun et al. 1989). This step is needed to utilize fully the mapping property of the activation function of the first layer of neurons.

The remainder of the recognition processing is performed by a rather elaborate multilayer feedforward network trained using the error back-propagation technique. The architecture of the network is shown in Figure 8.13. The network has three hidden layers,  $J_1$ ,  $J_2$ , and  $J_3$ , and an output layer consisting of 10 neurons. The hidden layers perform as trainable feature detectors.

The first hidden layer denoted as  $J_1$  consists of 12 groups of 64 units per group, each arranged in an  $8 \times 8$  square. Each unit in  $J_1$  has connections only from the  $5 \times 5$  contiguous square of pixels of the original input. The location of the  $5 \times 5$  square shifts by two input pixels between neighbors in the hidden layer. The motivation for this is that high resolution may be needed to detect a feature while the exact position of it is of secondary importance. All 64 units within a group have the same 25 weight values so they all detect the same feature. Thus, a group in layer  $J_1$  serves as a detector of a specific feature across the entire input field. However, the units do not share their thresholds. The weight sharing and limitations to processing of  $5 \times 5$  receptive input fields has resulted in a reduced number of connections equal to 19 968 or  $(12 \times 64)$  by  $(25 + 1)$ , from



**Figure 8.13** Architecture of multilayer feedforward network for handwritten character recognition. [© MIT Press; adapted from LeCun et al. (1989) and reprinted with permission.]

the  $\approx 200\,000$  that would be needed for a fully connected first hidden layer. Due to the weight sharing, only 768 biases and 300 weights remain as free parameters of this layer during training.

The second hidden layer uses a similar set of trainable feature detectors consisting of 12 groups of 16 units, again taking input from  $5 \times 5$  receptive fields. A 50% reduction of image size and weight sharing take place in this layer. The connection scheme is quite similar to the first input layer; however, one slight complication is due to the fact that J1 consists of multiple two-dimensional maps. Inputs to J2 are taken from 8 of the 12 groups in J1. This makes a total of  $8 \times 25$  shared weights per single group in J2. Thus, every unit in J2 has 200 weights and a single bias. Since this J2 layer consists of 192 units, it employs

38 592 ( $192 \times 201$ ) connections. All these connections are controlled by only 2592 free trainable parameters. Twelve feature maps of this layer involve 200 distinct weights each, plus the total of 192 thresholds.

Layer  $J_3$  has 30 units and is fully connected to the outputs of all units in layer  $J_2$ . The 10 output units are also fully connected to the outputs of layer  $J_3$ . Thus, layer  $J_3$  contains 5790 distinct weights consisting of  $30 \times 192$  units plus 30 thresholds. The output layer adds another 310 weights, of which 10 are thresholds of output neurons.

The discussed network has been trained using the error back-propagation algorithm. The training has involved only 23 full cycles. Thus, each character has been presented about 16 769 times to the network under the assumption that each digit occurs with the same probability in the training set of 7291 zip code digits. The weights were initialized with random values so that the total input to each neuron was within the operating range of its sigmoidal activation function. The target values for the output layer were chosen within the range of the sigmoid rather than at extreme  $\pm 1$  values, as is usually done for classifier training. This prevented the weights from growing indefinitely during the training. The weights were updated after the presentation of each single pattern.

Let us summarize the performance of the trained network of Figure 8.13 (LeCun et al. 1989). After completion of the training, only 10 digits from the entire training set had been misclassified (0.14%), thus yielding the decision error  $E_d$  as in (4.36) with a value  $0.137 \times 10^{-6}$ . However, 102 mistakes (5.0%) were reported on the test set and the decision error rose to  $E_d = 0.0517$ . The convergence to the correct response was found to be rather quick during the simulation of the test performance.

In a more realistic performance evaluation test, the rejection of marginal characters has been implemented that are very difficult to classify even for a human. In this approach, marginal characters are excluded from testing. The character rejection from the test set has been based on the criterion that the difference between the two most active output levels should exceed a certain minimum value to ensure reasonably reliable classification. To lower the error of 5% on the full test set to 1% using the rejection concept, the scribbled 12.1% of test patterns had to be removed because of their similarity to the classification of other digits.

Other networks with fewer feature mapping levels were also evaluated by LeCun et al. (1989) but they produced inferior results. For example, a fully connected network of 40 hidden units in a single layer was not able to generalize as well, compared to the architecture described. Misclassifications measured 1.6 and 8.1% on the training set and test set, respectively. In addition, 19.4% of the test set characters had to be removed to achieve the error of 1% when using the marginal character rejection concept.

It can be seen that the error back-propagation algorithm yielded the results that appear to be the state-of-the-art situation for handwritten digit recognition.

The trained network has many actual connections but only relatively few free parameters are selected during training. Also, distinct feature extraction constraints were imposed on the network. The training was implemented on commercial digital signal processing hardware that efficiently performed multiply/accumulate operations. The overall throughput rate of the complete, trained digit recognizer, including the image acquisition time, is reported to be 10 to 12 digit classifications per second.

## 8.3

### NEURAL NETWORKS CONTROL APPLICATIONS

The use of neural networks in control applications—including process control, robotics, industrial manufacturing and aerospace applications, among others—has recently experienced rapid growth. The basic objective of control is to provide the appropriate input signal to a given physical process to yield its desired response. In control systems theory, the physical process to be controlled is often referred to as the plant. The plant input signals, called actuating signals, are typically needed to control motors, switches, or other actuating devices. If the actuating signal at the input to the plant shown in Figure 8.14 is generated by the neural network-based controller, the case can be termed *neural control*, or, briefly, *neurocontrol*. This section covers the basic concepts and examples of neural control techniques. Those neurocontrol aspects that deal specifically with robotics applications are discussed in the subsequent section.

#### Overview of Control Systems Concepts

This section presents an overview of the elementary concepts and terminology of control systems. The scope of presentation is limited only to issues that are later related in the context of neural controllers. Specifically, the concepts of transfer characteristics, transfer functions, and feedforward and feedback control are reviewed below. Readers familiar with such introductory material may proceed directly to the next section.

Solution of the control problem depends directly on the precision with which characteristics of the process are known. In a single-channel control problem

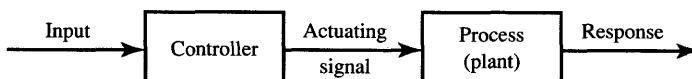
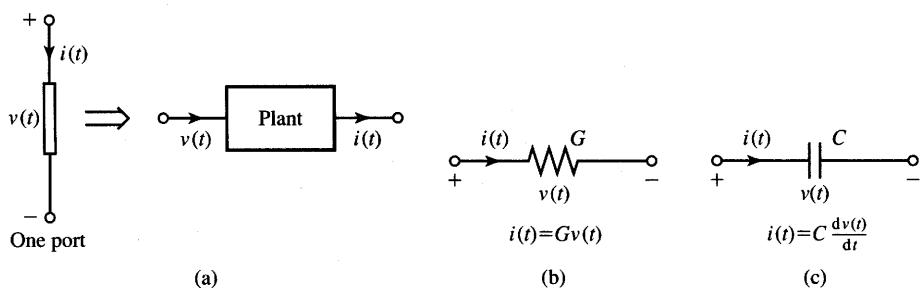


Figure 8.14 Illustration of control problem.



**Figure 8.15** Illustration of static and dynamical plant transfer function: (a) electric one-port element as a plant, (b) resistor, and (c) capacitor.

when the plant input is a scalar variable, for example, the process, if known, can be described by a relationship between the plant actuating signal and the plant response. In general, this relationship is expressed as  $P$ , where  $P$  is the process *transfer function* in the form  $P(s)$ , and  $s$  is the complex variable. We consider only time-invariant plant characteristics in this section. In such cases,  $P(s)$  includes the dynamic properties that relate the input and output signals of the plant. For plants that can be described in the time domain without a time variable,  $P(s)$  is independent of  $s$  and reduces to the simple functional relationship between the actuating signal and the response. Such plants are called nondynamic, memoryless, or simply static. For static plants the transfer function  $P(s)$  becomes  $P$  and is called the *transfer characteristics*.

Let us consider a simple example of static and dynamical plants. Assume that the current through a one-port electric element is the plant output, which needs to be controlled. The voltage across this one-port element is considered to be the plant's input, or actuating signal. The plant is shown in Figure 8.15(a). The figure also illustrates the equivalence between a specific electric one-port element and the commonly used general block diagram representation for the plant.

Since the plant transfer characteristics is defined as the ratio of its response to the excitation, we see that the description of a nondynamic plant yields a simple functional relationship. Now assume that the one-port plant is simply a resistor as shown in Figure 8.15(b). The plant transfer characteristic reduces to the following constant

$$P = \frac{i(t)}{v(t)} = G \quad (8.11a)$$

Thus, a static plant can be described by the transfer characteristics  $P$ . In the case of a static and linear process chosen here as an example (the resistor), the transfer characteristics is simply equal to the resistor's conductance,  $G$ .

Dynamical plants such as the capacitor shown in Figure 8.15(c) cannot be characterized by constants, not even by complex time-invariant functions because

their responses, being solutions of differential or integrodifferential equations, are functions of time. The transfer function for such plants is defined as the ratio of the integral transforms of the output to the input signal. The Laplace transform, denoted here as  $\mathcal{L}\{\cdot\}$  is the most commonly used transform to express the input and output signals of linear plants. Applying Laplace transform theory leads to the following transfer function for the capacitance plant of Figure 8.15(c):

$$P(s) = \frac{I(s)}{V(s)} = sC \quad (8.11b)$$

where

$$I(s) \triangleq \mathcal{L}\{i(t)\}$$

$$V(s) \triangleq \mathcal{L}\{v(t)\}$$

denote transforms of current through the capacitor and voltage across it, respectively. Thus, the transfer function as in (8.11b) can be understood to be a ratio of the Laplace transform of current to the Laplace transform of voltage.

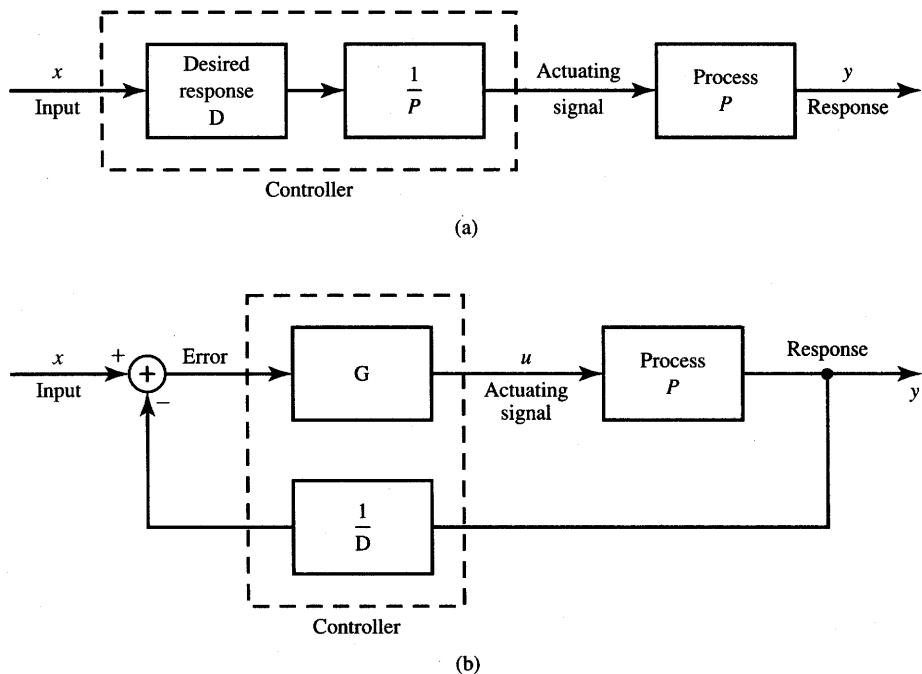
The reader may have noticed that the ratio of output response to the input signal yields the transfer characteristics of a static plant. Similarly, the ratios of the transforms of output to input yield the transfer function of a dynamical plant. We also should realize that the concept of a transfer characteristics is of no use for dynamical plants.

The significance of the transfer characteristics in system and control theory is based on the fact that response of a static system to an arbitrary excitation can be computed as a product of its transfer characteristics and of the excitation. Similarly, the transform of the response of a dynamical system can be expressed as a product of the transfer function and transform of the excitation.

For a plant that is completely known, any desired relationship between input and response can be realized by a simple *open-loop control configuration* as shown in Figure 8.16(a) (Mishkin and Braun 1961). The feedforward controller of Figure 8.16(a) consists of two parts connected in cascade. The second one cancels the process transfer function because its own transfer function has been chosen as the inverse of the known process transfer function. In addition, the front part of the controller introduces the desired overall transfer function of the controller and the plant equal to  $D$ . Thus, the controller's transfer function should be equal to  $D/P$ . This approach is known as *inverse control* since the most essential part of the controller should provide the inverse of the plant's transfer function. The transfer function of the inverse feedforward control system from Figure 8.16(a) computed between its input and output becomes

$$\frac{Y(s)}{X(s)} = D(s) \quad (8.12a)$$

In this approach involving the cascade connection of the controller driving the plant and involving no feedback, the neural controller that acts as the plant inverse needs to be designed.



**Figure 8.16** Control of process  $P$  to achieve the desired response: (a) inverse feedforward control and (b) feedback control.

Realize, however, that the inverse control method fails in any of the two important circumstances: (1) when the transfer function  $P(s)$  or the transfer characteristics  $P$  varies during the course of normal operation of the plant, and (2) when  $P(s)$  or  $P$  is not completely known in advance of design. We customarily turn to the feedback control systems to circumvent any or both of the two limitations mentioned. One of the most common feedback control arrangements is shown in Figure 8.16(b). That such a configuration is directly useful if  $P(s)$  or  $P$  varies or is unknown is clear from simple analytical considerations. The feedback control system of Figure 8.16(b) is described with the transfer function between input and output

$$\frac{Y(s)}{X(s)} = \frac{G(s)P(s)}{1 + G(s)P(s)/D(s)} \quad (8.12b)$$

Note that the system designer can choose  $G(s)$  and  $D(s)$ . If  $G(s)/D(s)$  is made sufficiently large so that a quantity known as the loop-gain and denoted  $LG$  satisfies the condition

$$LG \triangleq \left| \frac{G(s)P(s)}{D(s)} \right|_{s=j\omega} \gg 1 \quad (8.13a)$$

for all frequencies  $\omega$  of interest, then the transfer function (8.12b) reduces to the form approximately equal to (8.12a). Using the condition (8.13a), we easily obtain from (8.12b)

$$\frac{Y(s)}{X(s)} \cong D(s) \quad (8.13b)$$

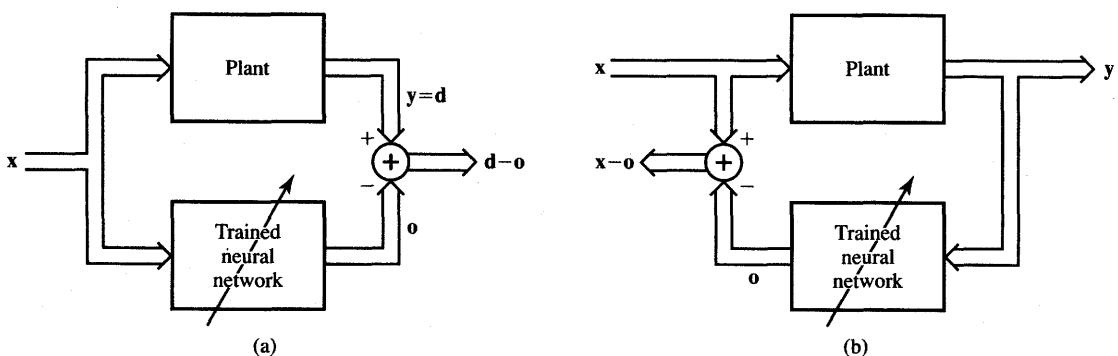
This result indicates that for high and fixed loop-gain values, the output now becomes independent of the plant transfer function  $P(s)$ , as desired. Therefore, the use of feedback provides a useful means for overcoming both the parameter variation of the plant and the ignorance of the designer as to the exact nature of the plant and its transfer function  $P(s)$ . These observations also apply to nondynamic plants. However, as mentioned earlier, for nondynamic plants, transfer functions  $G(s)$ ,  $P(s)$ , and  $D(s)$  in (8.12) and (8.13) become independent of complex variable  $s$ .

In this section we will approach the design of the controllers using the estimated information about the plant. We will assume that such information is gradually acquired during plant operation. Such control systems may be called *learning control systems*. Neural networks fit quite well into such a framework of control systems study. Plant identification techniques that make use of neural network techniques are reviewed first. Neural network applications for control of a nondynamic plant are then discussed. An example of a perceptron-based controller for a low-order dynamical plant will also be designed. The input signal to the perceptron controlling the dynamical plant, however, contains not only the present output of the plant but also its most recent output preceding the time the control decision was made. Finally, the discrete-time system control of the dynamical plant using the neurocontrol concept is studied. The concept involves adaptive control similar to the cerebellar model articulation controller (CMAC).

### Process Identification

The introductory discussion above points out that plant identification can be distinctly helpful in achieving the desired output signal of the plant. In general, identification of the plant should result in usable terms such as coefficients of a plant differential equation or coefficients of its transfer function. The issue of identification is perhaps of even greater importance in the field of adaptive control systems. Since the plant in an adaptive control system varies in operation with time, the adaptive control must be adjusted to account for the plant variations. Neural networks used for identification purposes in this section typically have multilayer feedforward architectures and are trained using the error back-propagation technique.

The basic configuration for *forward plant identification* is shown in Figure 8.17(a). A nonlinear neural network receives the same input  $x$  as the plant, and

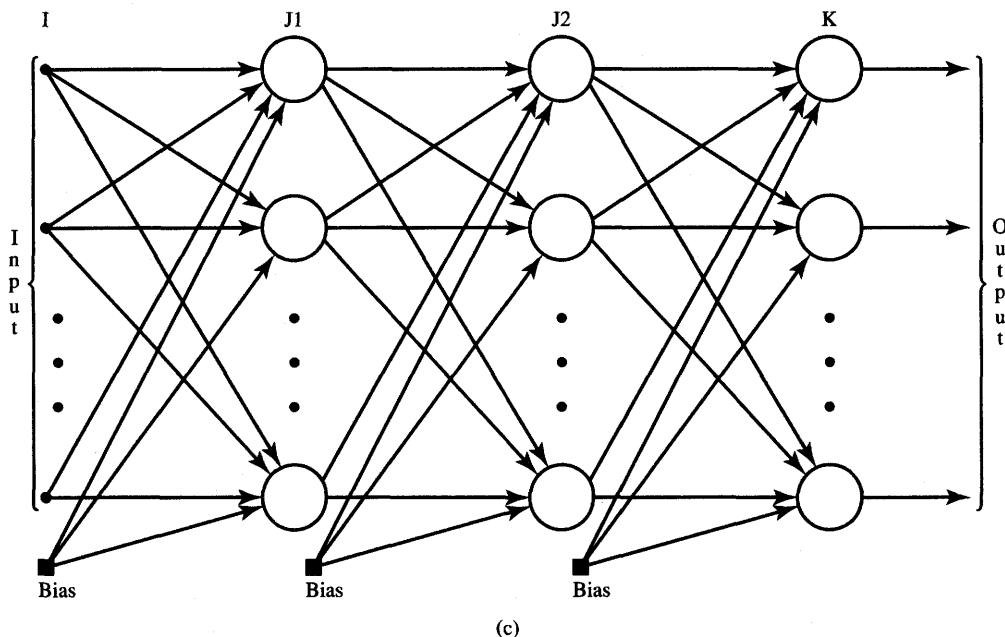


**Figure 8.17a,b** Neural network configuration for plant identification: (a) forward plant identification and (b) plant inverse identification.

the plant output provides the desired response  $\mathbf{d}$  during training. The purpose of the identification is to find the neural network with response  $\mathbf{o}$  that matches the response  $y$  of the plant for a given set of inputs  $x$ . During the identification, the norm of the error vector,  $\|\mathbf{d} - \mathbf{o}\|$ , is minimized through a number of weight adjustments. The minimization is achieved using learning techniques discussed in previous chapters. Figure 8.17(a) shows the case for which the network attempts to model the mapping of plant input to output, with both input and output measured at the same time. This is understandable since the trained neural network is feedforward and instantaneous; thus, we have  $\mathbf{o}(t) = \mathbf{x}(t)$ . However, more complex types of models for plant identification can also be employed. To account for plant dynamics, the input to the neural network may consist of various past inputs to the plant (Barto 1990). These can be produced by inserting the delay elements at the plant input.

In earlier discussions, we learned that the identification of the plant inverse may offer another viable alternative for designing the control system. In contrast to forward plant characteristics identification, now the plant output  $y$  is used as neural network input, as shown in Figure 8.17(b). The error vector for network training is computed as a  $\mathbf{x} - \mathbf{o}$ , where  $\mathbf{x}$  is the plant input. The norm of the error vector to be minimized through learning is therefore  $\|\mathbf{x} - \mathbf{o}\|$ . The neural network trained this way will implement the *mapping of the plant inverse*. Once the network has been successfully trained to mimic the plant inverse, it can be used directly for inverse feedforward control as illustrated in Figure 8.16(a). The properly trained neural network acts as controller in this configuration, and the appropriate actuating signal for the plant is directly available at the network's output.

In both identification cases described, the neural network is trained to copy either the forward or inverse characteristics of the plant. Typically, multilayer feedforward networks are used for this purpose. A trained network as shown in

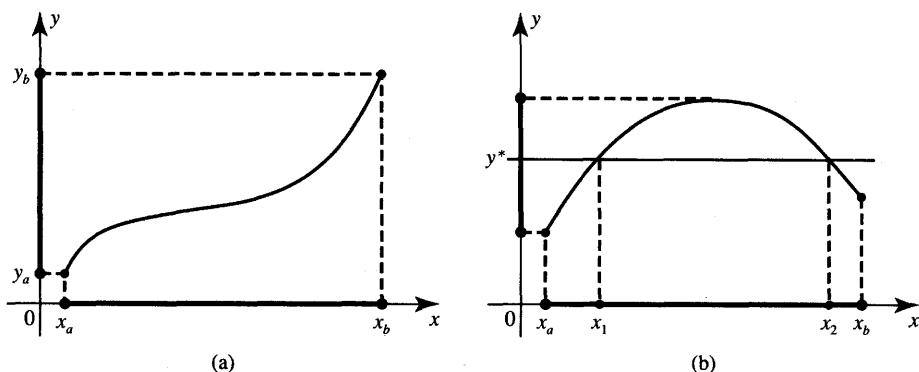


**Figure 8.17c** Neural network configuration for plant identification (*continued*): (c) trained neural network architecture.

Figure 8.17(c) can model the behavior of the actual plant in the forward or inverse mode of operation. Each of the identification modes has its advantages and disadvantages. Forward plant identification is always feasible; however, it does not immediately allow for construction of the plant controller. In contrast, plant inverse identification facilitates simple plant control; however, the identification itself is not always feasible.

A major problem with plant inverse identification arises when the plant inverse is not uniquely defined. This occurs for a plant when, as in Figure 8.18, *more* than one value of  $x$  exists that corresponds to one value of  $y$ . This particular condition coincides with the nonexistence of unique inverse mapping of  $y$  into  $x$ . Figure 8.18 illustrates this limitation of the plant inverse identification for the one-dimensional case. In the discussed case, the neural network modeling the plant inverse attempts to map a single input  $y^*$  to one of the two target responses  $x_1$  or  $x_2$ . It may be that the eventual mapping learned would somewhat tend to average the two desired  $x$  values, but in no case can such identification of the plant inverse be considered adequate.

As an example of plant inverse identification, consider a static plant that converts polar  $(r, \theta)$  to Cartesian  $(x, y)$  coordinates during its forward operation. The trained neural network should convert Cartesian to polar coordinates. A

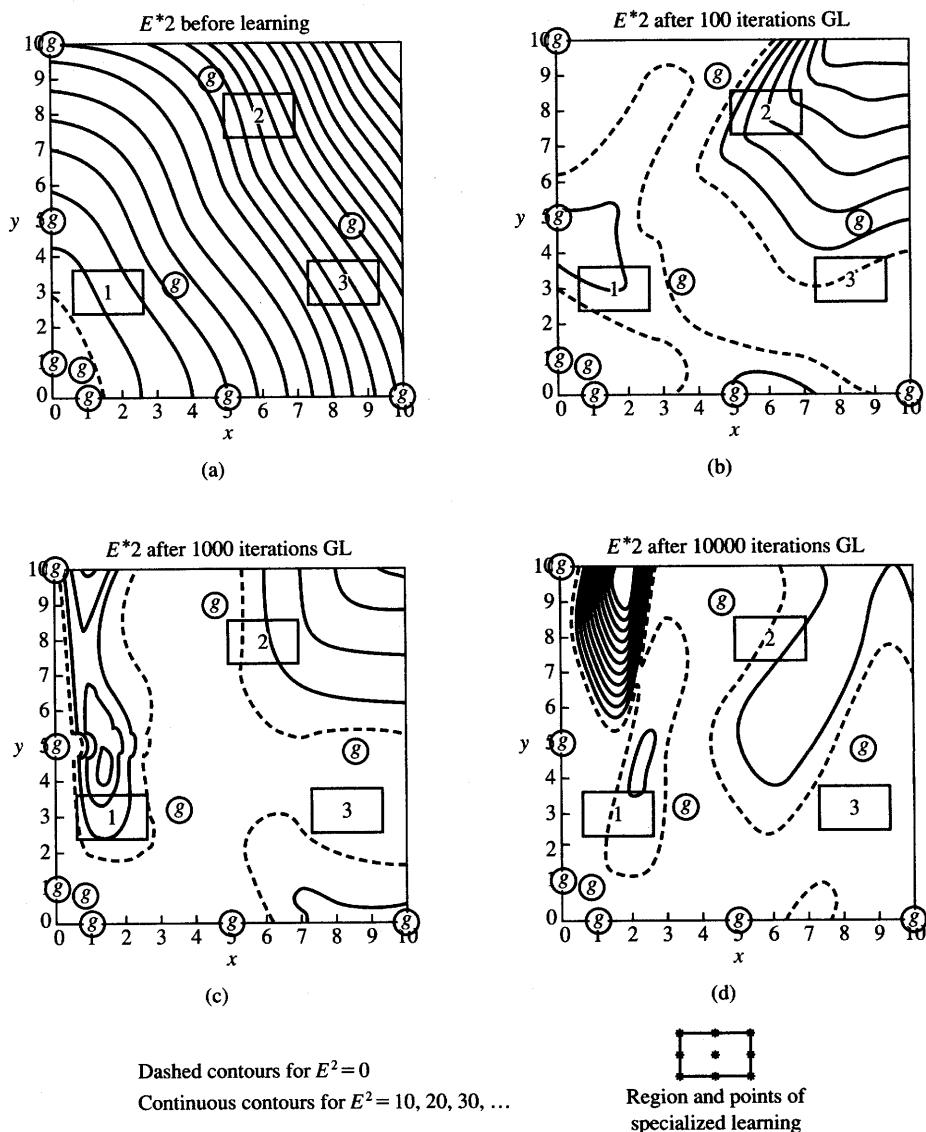


**Figure 8.18** Plant inverse identification example: (a) plant inverse characteristics exists and (b) plant inverse characteristics does not exist.

two-layer feedforward architecture is chosen with two active inputs plus single bias input, 10 hidden-layer neurons, plus a single hidden biasing neuron, and two output neurons (Psaltis, Sideris, and Yamamura 1987, 1988). Hidden-layer neurons with unipolar activation function are selected with  $\lambda = 1$ . Two output neurons are chosen with the linear activation function  $f(\text{net}) = \text{net}$  of unity slope so that they provide an unlimited range of output values. Initial weight values are chosen randomly at  $\pm 0.5$ , and the learning constant used for training is 0.1.

The simulated learning experiment of plant inverse identification involves 10 points spread within the first quadrant such that  $r \in [0, 10]$ , and  $\theta \in [0, \pi/2]$  (Psaltis, Sideris, and Yamamura 1987). The training set points are marked by circled  $g$  letters on Figure 8.19(a). The diagram shows the contour map of mapping error  $\|\mathbf{x} - \mathbf{o}\|^2$ , in Cartesian coordinates, before the training. Figure 8.19(b) shows the contour map of the error after 100 learning cycles. It can be seen that the conversion error has been reduced at and around the training points. Figure 8.19(c) indicates that the error after 1000 cycles has been totally suppressed at the training points. Continued training, however, gives rise to a problem shown in Figure 8.19(d). After 10 000 training cycles high-error pockets were observed outside the training areas. This phenomenon points out that the generalization ability of the network has actually decreased due to excessive training. Poor generalization also indicates that either the number of the training points is too low, or that the chosen architecture is not adequate for the problem requirements.

Let us note that the described training experiment is very similar to the one covered in Example 4.3. Indeed, the plant identification problem discussed here corresponds to a two-variable function approximation learning. The accurate values of the function to be approximated are outputs produced by the plant to be identified. These outputs are desired responses of the multilayer feedforward network, which undergoes the training.



**Figure 8.19** Error contour maps for inverse plant identification: (a) before training, (b) after 100 learning cycles, (c) after 1000 learning cycles, and (d) after 10 000 learning cycles. [©IEEE; adapted from Psaltis, Sideris, and Yamamura (1987) with permission.]

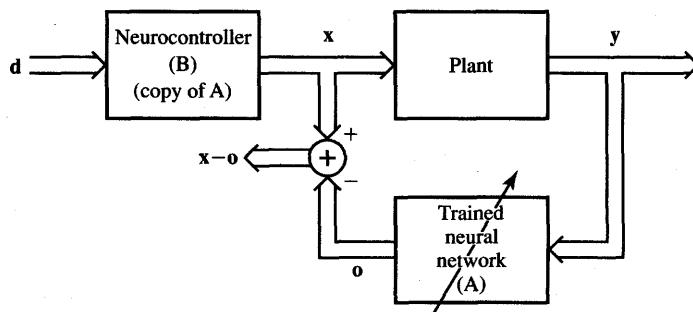
This description of identification procedures and example of plant identification overlook many details of real and complex identification problems. In particular, it does not completely cover the identification of the dynamical systems. It presents, however, the basic guidelines for plant modeling, which

has many uses in signal processing and control. One usually assumes here that the model has a general mathematical form, and then its unknown parameters are estimated. Forward and inverse plant identification configurations seem to be somewhat natural applications for multilayer feedforward neural network models. The plant parameter estimation is implemented using the pairwise arranged examples of plant input/output values. The role of the teacher in plant identification is played by the system being modeled.

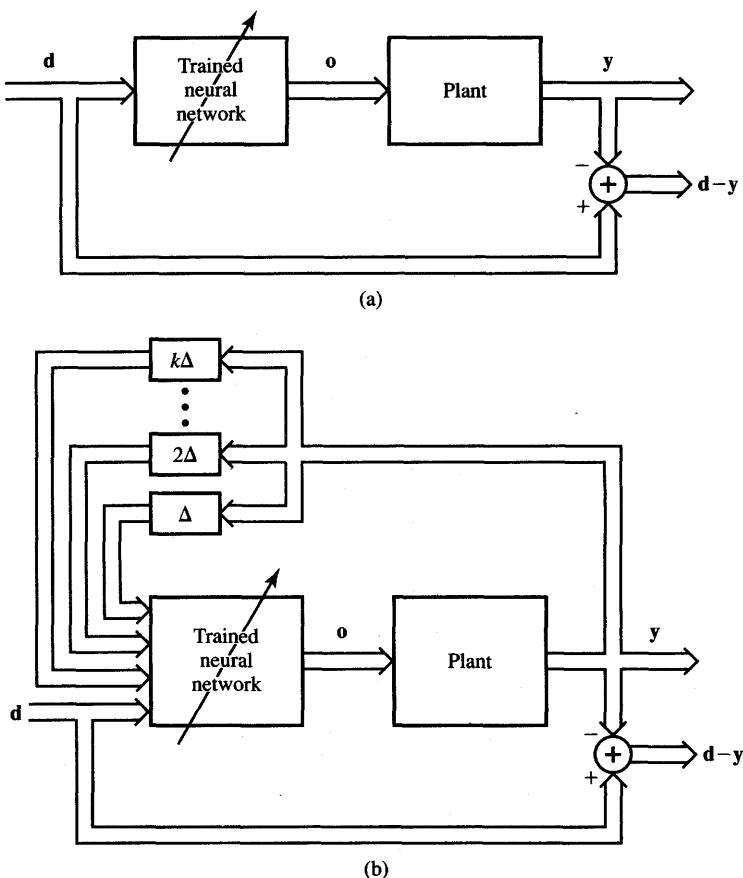
### Basic Nondynamic Learning Control Architectures

Let us review a specific method of control in which multilayer neural networks are used as controllers of nondynamic plants (Psaltis, Sideris, and Yamamura 1987, 1988). The following discussion is closely related to the plant identification techniques discussed in the preceding section. The training of the neural network-based controller is also discussed in order to demonstrate the feasibility of the proposed architecture.

Figure 8.20 shows the feedforward controller implemented using a neural network. Neurocontroller *B* is an exact copy of neural network *A*, which undergoes training. Network *A* is connected in such a way that it gradually learns to perform as the unknown plant inverse. Thus, this configuration is implementing the inverse feedforward control similar to Figure 8.16(a). The input of the controller is *d*, which is the desired response of the plant. The plant's actual response is *y* due to its input *x* produced by the neurocontroller *B*. The error used for training the neural network is the difference between the output signals of networks *A* and *B*. Although the network *B* tracks *A* after each training step, *y* must exactly match *d* for this error to reduce to zero. We may note that this configuration becomes useless if the plant inverse is not uniquely defined, as illustrated by an example in Figure 8.18.



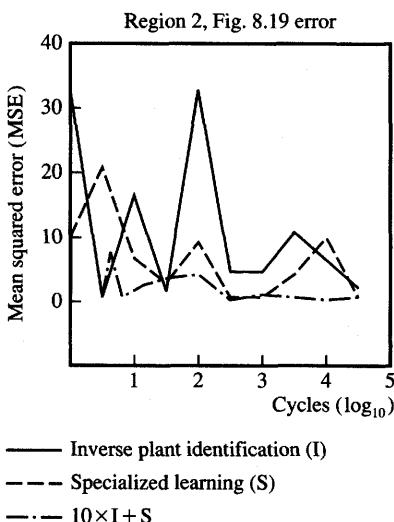
**Figure 8.20** Feedforward control with plant inverse learning.



**Figure 8.21** Specialized on-line learning control architecture: (a) static plant and (b) dynamical plant.

This control architecture is also called an *indirect learning architecture*. The distinct advantages of this configuration are obvious. The controller network can be trained on-line since it undergoes training while its copy performs the useful control function work. Thus, separate training sessions are no longer needed. Moreover, the inputs to the neurocontroller are the desired outputs of the plant. Thus, the controller's training can easily be performed in the region of interest of the output vector domain. Such training in the region of interest can be called specialized training. In addition, the neural network learns continually and is therefore adaptive.

A closely related control architecture for control and simultaneous specialized learning of the output domain of the static plant is shown in Figure 8.21(a). This control mode possesses all advantages of the control mode displayed in



**Figure 8.22** Learning profiles for learning architectures from Figures 8.17 and 8.21.  
[©IEEE; adapted from Psaltis, Sideris, and Yamamura (1988) with permission.]

Figure 8.20, but it uses only a single copy of the neural network instead of the two required by the control system of Figure 8.20.

The control objective of the specialized learning control architecture from Figure 8.21(a) is to obtain the desired output  $d$  of the plant under the condition that the plant input causing this output is unknown. For a dynamical plant, the trained neural network needs to be provided the sequence of recent output plant vectors in addition to present plant output. This would allow for reconstruction of the current state of the plant based on its most recent output history. Figure 8.21(b) shows a specialized learning control architecture of the dynamical plant of order  $k$ . In this case, the last  $k$  output vectors of the controlled plant are augmenting the present input of the trained neural network. It should be understood that the delay units  $\Delta$  placed in the feedback loops denote the time elapsed between the samples of inputs used for training, and that the plant is controlled in real time. Under such conditions,  $k$  can be referred to as the order of the dynamical plant (Saerens and Soquet 1991).

Figure 8.19 illustrates three rectangular regions of specialized learning selected for the static neural network trained using the configuration shown in Figure 8.21(a). Ten points have been used within each rectangular learning specialization region as shown at the bottom of Figure 8.19. The learning profiles based on error  $\|d - y\|$  are shown in Figure 8.22 (Psaltis, Sideris, and Yamamura 1988). The error analyzed and shown in the figure has been averaged for 10 points in region 2. The figure shows that errors for the plant inverse identification method (continuous line) and specialized learning (dashed line) are comparable.

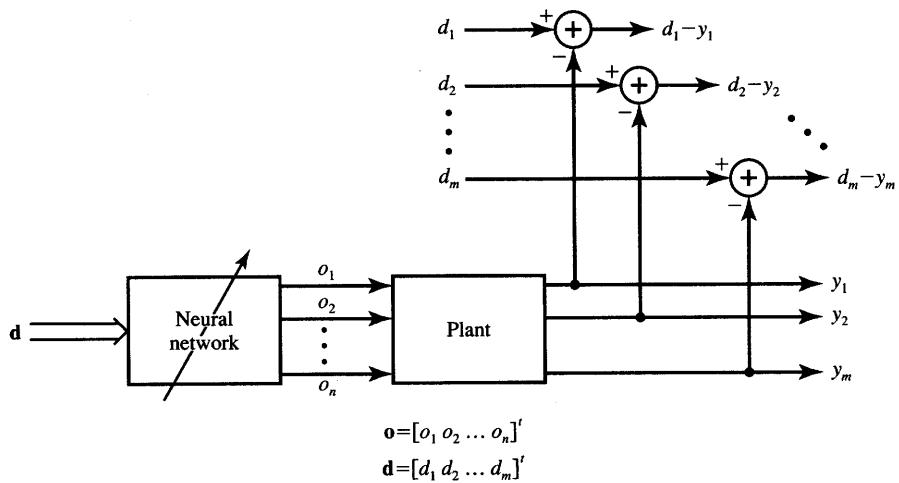


Figure 8.23 Back-propagating error information through the plant.

In addition, the learning experiment consisting of 10 cycles of plant inverse identification preceding the specialized learning has been carried out. The results have shown that this hybrid learning method may have a distinct advantage over each of the two learning methods applied separately. Combining the two learning approaches has resulted in the lowest error compared to other learning architectures. However, even with a series of extensive simulated experiments, it has not been possible to determine conditions under which the two learning methods have to be combined to achieve the best learning results.

The specialized on-line learning control architecture of Figure 8.21 suffers from one major weakness, which may result in slower training or other undesirable characteristics if the plant is known only approximately. We cannot apply the error back-propagation training for this configuration directly because only the error between the accessible plant output  $y$  and its desired output  $d$  is known to the user. The error  $d - y$  must be reduced to the input of the plant first.

The neural network to be trained using the error available at the plant output as shown in Figure 8.21 has been drawn in more detail in Figure 8.23. The customary error back-propagation rule (4.5a) for weight adjustment within the neural network can be rewritten in the form involving neural network outputs  $o_i$  as follows:

$$dw_{ij} = -\eta \frac{\partial E}{\partial o_i} \cdot \frac{\partial o_i}{\partial w_{ij}}, \quad \text{for } i = 1, 2, \dots, n, \quad j = 1, 2, \dots, m \quad (8.14)$$

where the plant output error  $E$  explicitly available from the teacher can be expressed as

$$E = \frac{1}{2} \sum_{i=1}^m (d_i - y_i)^2 \quad (8.15)$$

To compute the error  $E$  reflected at the plant's input we assume that the plant is described with the following algebraic equations:

$$y_i = y_i(\mathbf{o}), \quad \text{for } i = 1, 2, \dots, m \quad (8.16)$$

The output error  $E$  propagates back to the input of the plant and  $\partial E / \partial o_i$ ,  $i = 1, 2, \dots, n$ , can be computed from (8.14), (8.15), and (8.16) as follows:

$$\begin{aligned} -\frac{\partial E}{\partial o_1} &= (d_1 - y_1) \frac{\partial y_1}{\partial o_1} + (d_2 - y_2) \frac{\partial y_2}{\partial o_1} + \dots + (d_n - y_n) \frac{\partial y_n}{\partial o_1} \\ -\frac{\partial E}{\partial o_2} &= (d_1 - y_1) \frac{\partial y_1}{\partial o_2} + (d_2 - y_2) \frac{\partial y_2}{\partial o_2} + \dots + (d_n - y_n) \frac{\partial y_n}{\partial o_2} \\ &\vdots \\ -\frac{\partial E}{\partial o_n} &= (d_1 - y_1) \frac{\partial y_1}{\partial o_n} + (d_2 - y_2) \frac{\partial y_2}{\partial o_n} + \dots + (d_n - y_n) \frac{\partial y_n}{\partial o_n} \end{aligned} \quad (8.17)$$

The expressions (8.17) can be rewritten in matrix form as below.

$$-\nabla_o E = \left( \frac{\partial \mathbf{y}}{\partial \mathbf{o}} \right)^t (\mathbf{d} - \mathbf{y}) \quad (8.18)$$

where

$$\nabla_o E \triangleq \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \frac{\partial E}{\partial o_2} \\ \vdots \\ \frac{\partial E}{\partial o_n} \end{bmatrix}$$

and

$$\left( \frac{\partial \mathbf{y}}{\partial \mathbf{o}} \right) \triangleq \begin{bmatrix} \frac{\partial y_i}{\partial o_j} \end{bmatrix}, \quad \text{for } i = 1, 2, \dots, n, \quad j = 1, 2, \dots, m$$

is the  $m \times n$  Jacobian matrix of the plant. It can be seen from (8.18) that the plant output error  $\mathbf{d} - \mathbf{y}$  propagates back through the plant according to the transpose of the plant's Jacobian matrix. This matrix describes the performance of the linearized plant (8.16) taken at the plant's operating point.

If the plant equations (8.16) are known, the Jacobian matrix can be easily produced in an analytical form at the plant's operating point. If the plant is not known, the array of partial derivatives  $\partial y_i / \partial o_j$  can be determined by deviating the plant's input  $j$  by the amount  $\Delta o_j$  from its operating point and then measuring resulting changes  $\Delta y_1, \Delta y_2, \dots, \Delta y_m$ . This approach is called the *perturbation method* and allows for measurement or simulation of approximate values of the

Jacobian matrix entries. The perturbation method yields approximate values only of  $\partial y_i / \partial o_j$  since it involves finite differences  $\Delta y_i$  and  $\Delta o_j$  rather than infinitesimal differentials. However, the method is always applicable. Also, the evaluation of Jacobian matrix entries can be performed using the perturbation method either computationally or experimentally.

### Inverted Pendulum Neurocontroller

In this section we will review the capability of a simple neural network to learn how to balance an inverted pendulum. The inverted pendulum (also called "broomstick balancing") problem is a classical control problem that has been extensively studied by many researchers and seems to be well understood. Also, the problem is representative of many typical dynamical plant control problems. Thus, designing the controller for this inherently unstable system will allow us to approach and solve a number of other, similar control tasks. The complexity of the task of inverted pendulum balancing is significant enough to make the problem interesting while still being simple enough to make it computationally tractable (Tolat and Widrow 1988).

A number of control techniques using neural networks to balance an inverted pendulum have been investigated (Barto, Sutton, and Anderson 1983; Anderson 1989). In the approach presented below we will use the simple and synergistic approach, which examines the merger of visual image acquisition, training, and neurocontrol (Tolat and Widrow 1988). Since the plant is inherently dynamical, the control must involve the plant's present and past state information.

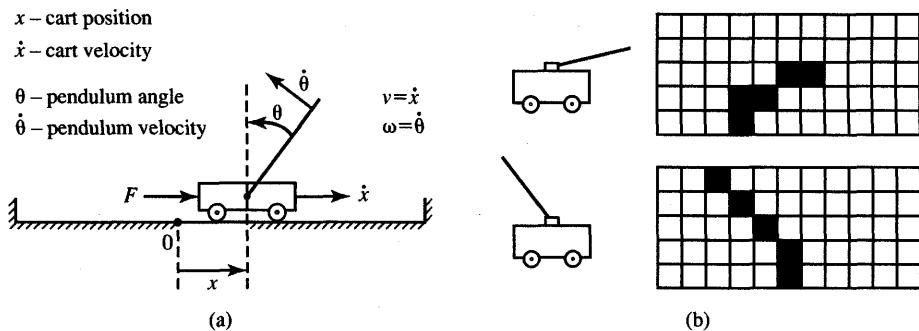
The goal of the inverted pendulum task is to apply a sequence of right and left forces of fixed magnitude such that the pendulum is balanced, and the cart does not hit the edge of the track. The cart-pendulum physical system is shown in Figure 8.24(a). The cart and pendulum are constrained to move only within the vertical plane. Four time variables describe the state of the system: the horizontal position and velocity of the cart ( $x, v$ ), the angle between the pendulum and vertical, and the angular velocity of the falling pendulum ( $\theta, \omega$ ). The force required to stabilize the system is a function of four state variables  $x, v, \theta$ , and  $\omega$  as follows:

$$F(t) = ax(t) + b\dot{x}(t) + c\theta(t) + d\dot{\theta}(t), \quad (8.19)$$

$$\text{where } \dot{x}(t) = v(t), \quad \dot{\theta}(t) = \omega(t)$$

and  $a, b, c$ , and  $d$  are constant coefficients, which need to be found for each system since they are dependent on its physical characteristics.

Equation (8.19) approximates the motion of the system and assumes that it can be characterized by linear differential equations. Knowing the coefficients  $a$ ,



**Figure 8.24** The inverted pendulum illustration: (a) physical system to be controlled and (b) two examples of computer-generated system images: left—real image; right—quantized image. [©IEEE; adapted from Tolat and Widrow (1988) with permission.]

$b$ ,  $c$ , and  $d$  and solving the system's modeling equation (8.19) would allow the user to compute the force that balances the plant. It is implied here, however, that neither Eq. (8.19) nor its coefficients are known both for the controller and for the observer. Let us consider that the only information available is about the order of equation and its general form. Since the equation is of first order, it can be expected that providing the present and most recent plant output value would enable us to train the controller adequately. As a result, the controller would need to generate the positive or negative force that would possibly stabilize the pendulum at its vertical position.

Since the balancing force can be assumed to be constant and of fixed magnitude for each push-pull action, and it differs only by direction, it may be postulated that the following force will stabilize the solution of the differential equation (8.19):

$$F(t) \cong \alpha \operatorname{sgn} \left[ ax(t) + b\dot{x}(t) + c\theta(t) + d\dot{\theta}(t) \right] \quad (8.20)$$

where  $\alpha$  is a positive constant representing the force magnitude. Though the cart and pendulum velocities are involved in (8.20), they are more difficult to measure than linear distances and angles. To circumvent this difficulty, Equation (8.20) can be rewritten in the form of a difference equation:

$$F(t) \cong \alpha \operatorname{sgn} \left[ (a + b)x(t) - bx(t - 1) + (c + d)\theta(t) - d\theta(t - 1) \right] \quad (8.21)$$

This equation approximates the value of the balancing force  $F(t)$  at the present time. It is assumed here that  $x(t - 1)$  and  $\theta(t - 1)$  are cart positions and pendulum angles at an earlier time preceding the evaluation of the plant and the computation of force.

The dynamics of the system remains largely unknown during the process of neurocontroller design. The knowledge of the system's dynamics is limited to the vague form of the difference equation (8.21) and it does not include its coefficients. In addition, the human observer who would measure cart positions and angles of pendulum at every instant is replaced by the input of the visual image of the cart and pendulum. Since it is essential to provide the controller with the present and most recent position and angle data, the present and most recent images arranged in pairs have been used to train the network (Tolat and Widrow 1988).

The neural network training was implemented by software simulation with the input of binary images providing the required state information  $x(t)$ ,  $x(t - 1)$ ,  $\theta(t)$ , and  $\theta(t - 1)$ . Examples of the quantized binary images are shown in Figure 8.24(b). The images were generated on a MacIntosh computer in the form of  $5 \times 11$  pixel maps of the cart and pendulum. Images of finer granularity than  $5 \times 11$  pixels were not used since they required extra computational effort and resulted in no improvement in controller performance. Pictures of smaller size, in contrast, did not provide an adequate resolution and input representation for the solution of this problem.

For the image resolution used and shown in Figure 8.24(b), there were 65 different images involving all possible combinations of 5 distinct cart positions with 13 different pendulum angles. Since the training objective was to produce  $F(t)$  as in (8.21), both the present image and the most recent one representing the cart with pendulum at an earlier time,  $t - 1$ , were used. The time elapsed between two consecutive state recordings, equal to the time between two successive force applications was 100 ms. This time and graphics resolution were found to be sufficient to balance the actual mechanical system investigated. With the two images completely combined, there were  $65^2 = 4225$  training vectors in the training set. Each input vector used for training consisted of  $2 \times 5 \times 11 = 110$  binary entries, not including the bias input.

Let us note how this method of controller design differs from more traditional control approaches. First, the plant's modeling equation (8.19) would have to be known to design a conventional controller. Specifically, the task would require finding initially unknown coefficients  $a$ ,  $b$ ,  $c$ , and  $d$  in (8.19) or (8.21). Analytical computation of force  $F(t)$  would further require solution of the differential equation (8.19). Alternatively, the difference equation (8.21) obtained from (8.19) could be solved by the controller to approximate the value of  $F(t)$  needed to provide the balance of plant.

The solution of (8.21) in the described experiment has been obtained without finding coefficients  $a$ ,  $b$ ,  $c$ , and  $d$ , and even without solving any equations, or measuring position and velocity of the cart, or even measuring the angle or angular velocity of the pendulum. Experiments have shown that a single discrete perceptron with 110 weights and one threshold weight can be trained to find

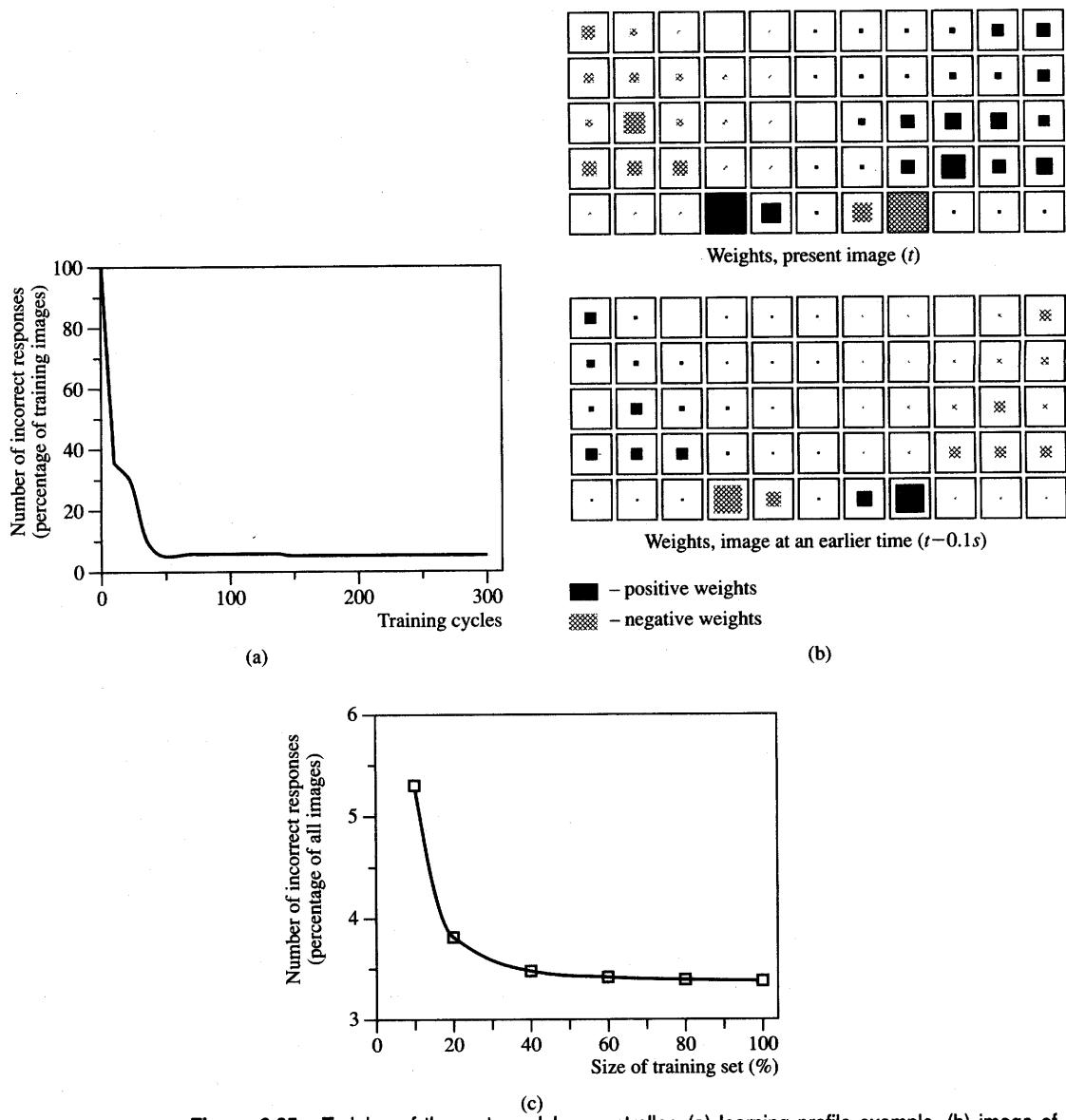
the solution of (8.21). To train the perceptron, 4225 examples of image pairs were submitted along with the teacher's decision as to the desired  $F(t)$  value for the specific set of observations  $x(t)$ ,  $x(t - 1)$ ,  $\theta(t)$ , and  $\theta(t - 1)$ . The teacher's input determines the sign of force ( $F = 1$  or  $F = -1$ ) needed to prevent the pendulum from falling. This input is based on the study of the present and most recent image. Note that both the present and most recent images on  $x$  and  $\theta$  are presented as input patterns from the training set. Supervised training has replaced solving the system of 4225 linear equations with 111 unknowns. This has eliminated finding the best fit of unknowns in this overdetermined system of equations.

In the described control design experiment, the Widrow-Hoff training algorithm has been used to train the weights of the single perceptron controller consisting of 111 weights, one summing node, and one TLU. In addition, the network has proven capable of extracting necessary state information from crude visual images arranged in a time sequence without the need for measurements of motion coordinates or physical parameters of the system.

The learning profile for computation of the applied force in the experiment is shown in Figure 8.25(a). The error of force sign drops dramatically after several passes through the entire training set; then it flattens out. The reported error after 1000 training cycles was 3.4%. The example of final weight values coded into pixel map shade and intensity is shown in Figure 8.25(b). The sign of force  $F$  is found by combining the pixel images of the system with weights and creating an appropriate scalar product of the two 110-tuple vectors plus the threshold value.

In the next series of simulation experiments with inverted pendulum balancing, the training focused on analyzing the generalization capabilities of the trained controller. Only a fraction of the total of 4225 images was selected at random for training purposes. The training subsets were of different sizes and varied from 10 to 100%. Results were computed for four different training sets and then averaged. The final error after 300 training cycles is plotted in Figure 8.25(c) for six different training set sizes. The final error values indicate that the trained network is able to generalize fairly uniformly and well for training sets consisting of more than 40% of the patterns of the complete set.

The neurocontroller designed for keeping the pendulum from falling was able to perform successfully despite occasional failures. A single discrete perceptron was proven to be adequate for this task. More complicated control problems, however, will require larger neural networks with appropriate inputs (Fu 1986). The network used in this case has not made use of explicit decision, or control, rules. The only rules used were common sense teacher guidelines as to how to keep the pendulum vertical. The example remarkably demonstrates that a teacher-machine interaction can be gradually replaced by observing the environment and providing responses suitable for the observations. After training, the controller should be able to perform the control task autonomously by responding correctly to most situations, even those that were not specifically used for training.



**Figure 8.25** Training of the cart-pendulum controller: (a) learning profile example, (b) image of controller's final weights, 4225 images used, and (c) analysis of generalization capability of the trained controller. [©IEEE; adapted from Tolat and Widrow (1988) with permission.]

### Cerebellar Model Articulation Controller

This section discusses the cerebellar model articulation controller (CMAC). Both the concept and the application case study for low-order CMAC control problems are reviewed here based on work by Kraft and Campagna (1990a, 1990b). The basic idea of the CMAC approach is to learn an approximation of the system characteristics and then to use it to generate the appropriate control signal. The approximation of the system characteristics is understood as gradual learning based on the observations of the plant input-output data in real time. The learning of a CMAC controller proceeds somewhat like the winner-take-all competitive learning discussed in Sections 2.5 and 7.2, however, without the competition enforcing the weight adjustment.

Let us consider the case of a first-order plant that needs to be controlled to yield a known, desired output signal. A sample discrete-time first-order plant is described with the difference equation

$$y(k+1) = ay(k) + bx(k) \quad (8.22a)$$

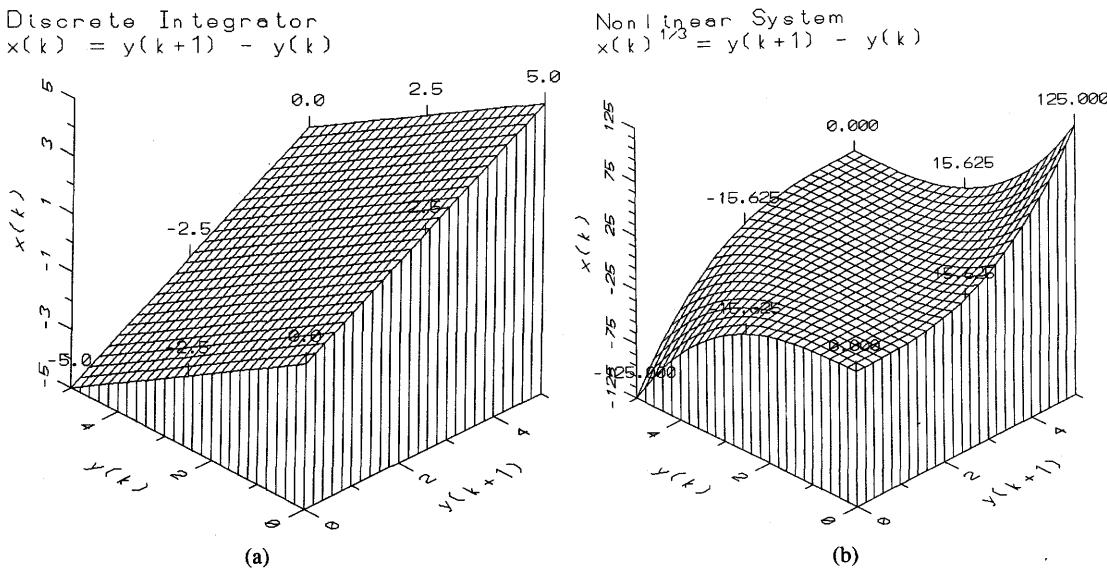
where  $y(k)$  is the system output,  $x(k)$  is the input,  $k$  is the time step number, and  $a$  and  $b$  are constant coefficients. Each point in plane  $y(k)$ ,  $y(k+1)$  corresponds to a required value of control input  $x(k)$ . Interpreting the present and the next following output  $y(k)$ , and  $y(k+1)$ , respectively, as two independent variables, input  $x(k)$  can be represented as a so-called *characteristic surface* of the plant.

Two examples of characteristic surfaces are illustrated in Figure 8.26. For the linear plant as in (8.22a) with  $a = b = 1$ , the planar characteristic surface is shown in Figure 8.26(a). Nonlinear systems exhibit more diversified characteristic surface shapes that cannot be expressed through a linear form as in (8.22a). A sample nonplanar surface for the equation

$$y(k+1) = y(k) + [x(k)]^{1/3} \quad (8.22b)$$

is shown in Figure 8.26(b). Obviously, both for the linear or nonlinear plant that is completely known, the characteristic surface can be calculated and used to produce the present control input value  $x(k)$ . A simple look-up table or formulas such as (8.22) for  $x(k)$  values at each coordinate pair  $y(k)$ ,  $y(k+1)$  would provide, without computation, the appropriate value of the desired control signal. In a more realistic situation, however, when the system parameters are not known in advance or they vary in time, the characteristic surface can be learned gradually from the plant input-output data. This can be accomplished in an on-line control mode.

The block diagram of the complete learning and control system is shown in Figure 8.27. Let us assume a first-order discrete-time plant. For such a plant, the control signal  $x(k)$  is determined by a pair of values  $y(k)$  and  $y(k+1)$ , which are the present and next plant outputs, respectively. The sequence of values  $y(k)$ , for



**Figure 8.26** Examples of plant characteristic surfaces: (a) planar surface (8.22a),  $a = b = 1$  (first-order linear plant) and (b) nonplanar surface (8.22b) (nonlinear dynamical plant).

$k = 1, 2, 3, \dots$ , is assumed to be known and is provided by the reference plant output generator.

The plant control signal  $x(k)$  is initially produced by a conventional controller. The controller is used in a negative feedback configuration similar to the one shown in Figure 8.16(b). The reference plant output generator inputs the present output value of the plant to the summing node  $S1$ . This input is approximately equal to the actual output of the plant due to the fact that the closed-loop system between the reference plant output to the summing node  $S1$  and the plant output has a transfer function of approximately unity value. For this condition to be valid, the closed-loop control system consisting of classical controller, plant, and node  $S1$  must be both stable and it must use a high-gain classical controller. Note also that the neurocontroller provides little or no input to the summing node  $S2$ , at least at the initial training stage of the entire control system.

In the initial stage of operation, the plant derives almost all of its input from the classical controller. The neurocontroller, called here the CMAC memory, is not yet trained, and it initially outputs a zero value to  $S2$ . Therefore, initially, the classical controller output is the dominant component of the total actuating signal  $x(k)$ . In each control step, however, the control signal produced by the classical controller,  $x(k)$ , is used to build the value of the CMAC characteristic surface  $x(k)$  having current coordinates  $y(k)$ ,  $y(k + 1)$ . The characteristic surface adapts in each control step according to the following learning law, which is similar to

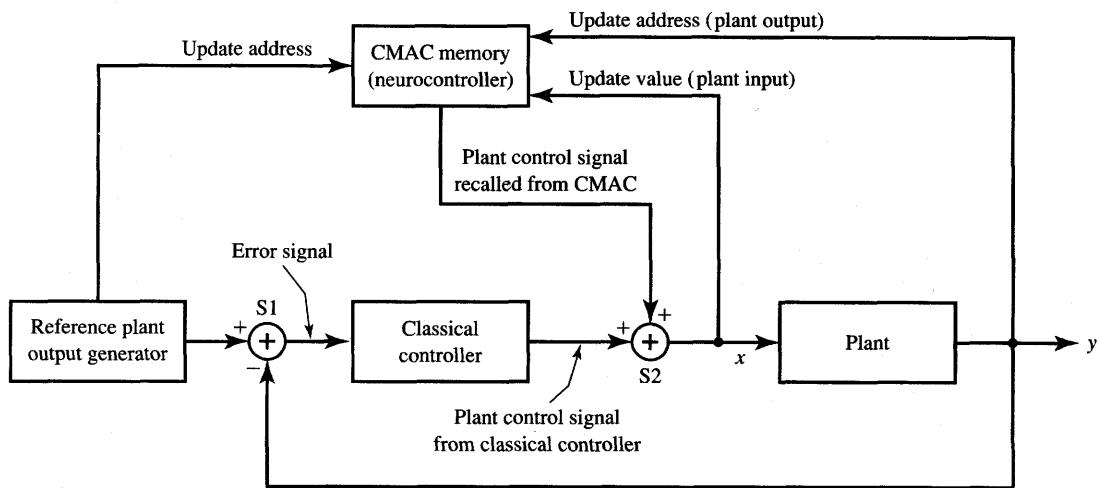


Figure 8.27 CMAC neurocontrol block diagram.

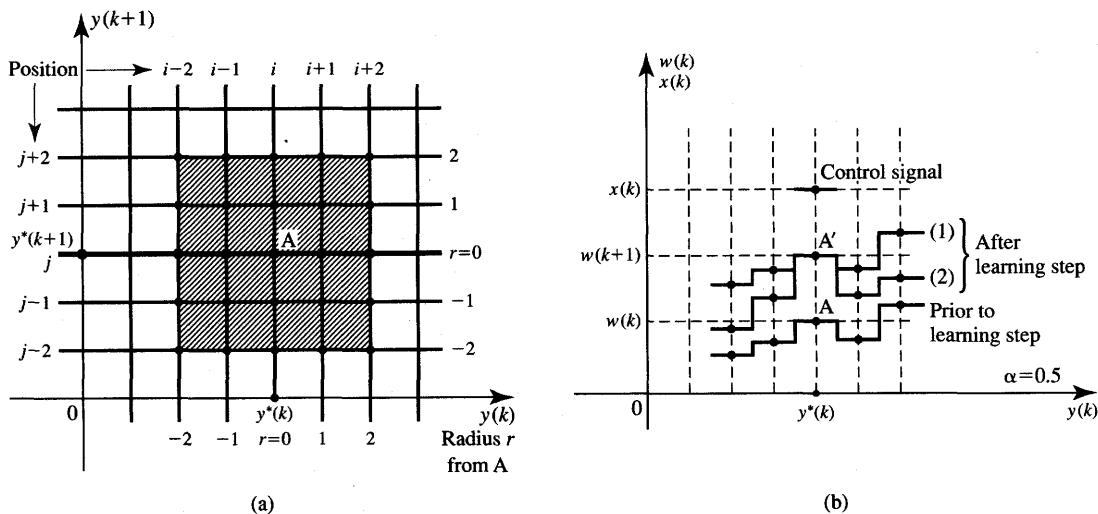
the learning rule (2.46):

$$w(k + 1) = w(k) + \alpha[x(k) - w(k)] \quad (8.23)$$

In this weight adaptation formula,  $\alpha > 0$  is a small positive learning constant,  $k$  denotes the learning (update) step number and also the number of the control step. As pointed out, the initial weight values are set to zero, which corresponds to setting the entire characteristic surface to zero at the beginning of training. The weights  $w(k)$ , which are built by the conventional controller, can be interpreted as iteratively updated control signal values  $x(k)$ , or memorized values of the characteristic surface that needs to be learned. The learning as in (8.23) should eventually end up at learned values  $w$ , which are averaged  $x(k)$  values over a number of CMAC controller learning steps.

Despite the lack of competition between the neurons and the absence of the winner, the learning as in (8.23) follows the unsupervised winner-take-all learning rule. This is to provide matching between vectors  $y(k)$ ,  $y(k + 1)$  and vectors  $x(k)$ . The winning neuron during learning is always the one designated by the coordinates  $y(k)$ ,  $y(k + 1)$ , of the learned characteristic surface. It is seen that CMAC characteristic surface learning represents an interesting case of supervised look-up memory training.

The described learning is of the local type, and more precisely can be called pointwise, since the only coordinates at which learning occurs are  $y(k)$ ,  $y(k + 1)$ . No information is conveyed to nearby memory locations. The generalization of learning can be used, however, in order to update a neighborhood of the point to be learned. To accomplish the learning with neighborhood sensitivity, which can be also termed as learning with embedded generalization, the memory values



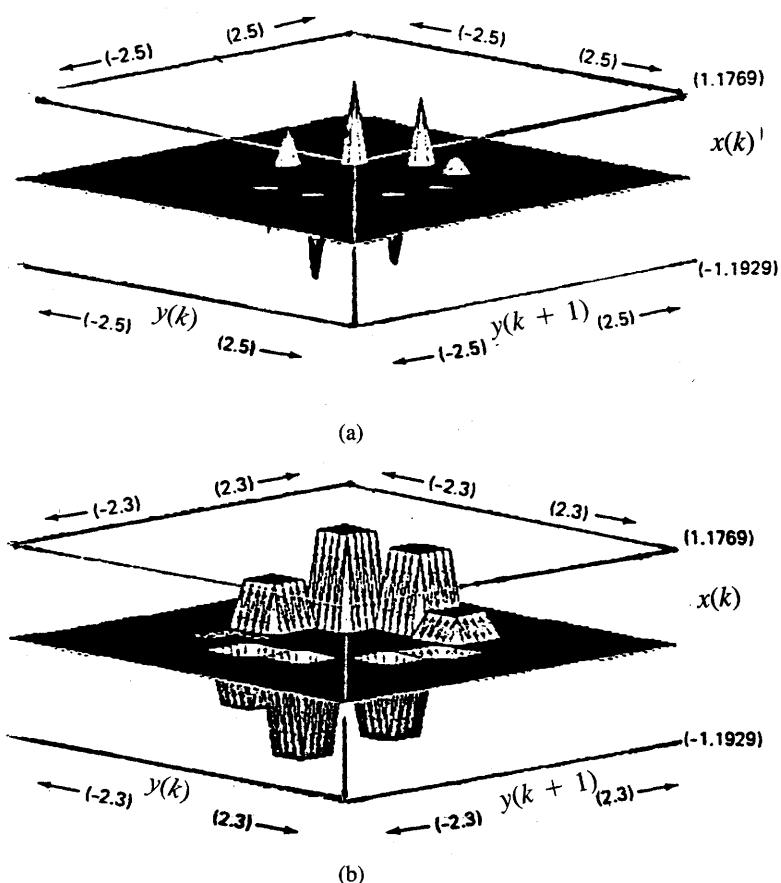
**Figure 8.28** Learning of CMAC neurocontroller with generalization: (a) example of neighborhood learning radius of 2 and (b) learning at cross-section  $y^*(k + 1)$  with generalization (two cases).

$w(k + 1)$  are assigned planar coordinates so that they become  $w_{ij}(k + 1)$ . It is now assumed that the memory has its values built over the discrete space with a certain neighborhood radius.

Figure 8.28(a) shows neighborhood sensitivity in the shaded region of generalized learning at point  $A$  with the neighborhood radius of 2. Learning at  $A$  in such cases and according to (8.23) is accompanied by the neighborhood learning at 24 adjacent memory locations falling within the shaded region. Learning at memory cross-section  $y^*(k + 1)$  is displayed in Figure 8.28(b). The figure shows example memory adjustments at five neighborhood points. Curve (1) shows adjusted memory values with equal adjustments for each of the updated locations independent of their distance from  $A$ . Curve (2) exemplifies tapered learning with adjustments of  $w_{ij}(k + 1)$  inversely proportional to the distance between the point  $A$  and the location  $i, j$  being adjusted within the neighborhood.

Figure 8.29 shows example learning of a characteristic surface without and with neighborhood generalization. Figure 8.29(a) depicts pointwise learning in several isolated locations of the controller's memory. Figure 8.29(b) depicts untapered neighborhood learning with uniform generalization built into the characteristic surface. It can be seen that the neighborhood weights are adjusted in this case by the same amount as for pointwise learning.

During the learning phase, the control signal coming from the classical controller of fixed high-gain is added in node S2 to the feedforward neurocontroller output. This output gradually increases from zero values as the weights are trained



**Figure 8.29** Learning of the characteristic surface: (a) pointwise learning mode (without generalization) and (b) neighborhood learning mode (with generalization). [©MIT Press; reprinted from Kraft and Campagna (1990b) with permission.]

to approximate the plant and as the characteristic surface is unfolding. As learning continues, the portion of the control signal derived from the neurocontroller increases. More precisely, the memorized weights of the CMAC controller are built up during control. As a result, the recalled control signal from the neural look-up memory gradually takes control of the plant. The conventional controller is excited by the error signal produced at the output of the summing node S1. Since this error signal gradually diminishes as the neurocontroller takes over the control function, the error signal input is reduced even further. As a result, the conventional controller is gradually eliminated from the control action.

For an accurately trained neurocontroller having the block diagram shown in Figure 8.27, the error signal of the classical controller reduces to zero, as does

its output signal. An immediate consequence of the feedforward CMAC neuro-controller taking over the control function is the increased speed of the system's response. In the presence of noise, or variable plant parameters, however, the normally idle classical controller would assume partial control of the plant to compensate for partial uncertainties that may be present in the system but have not been learned from earlier experience.

The neurocontroller using the described concept has been designed to control a linear discrete integrator plant described by (8.22a) with  $a = b = 1$  (Kraft and Campagna 1990b). The desired response  $y(k)$  of the sample plant has been chosen to be the response of the unity-gain discrete integrator-filter with unity low-frequency gain and a pole at  $z = 0.1$ . The difference equation of the integrator-filter, which produces the desired sequence  $y(k)$  at the output of the plant, is

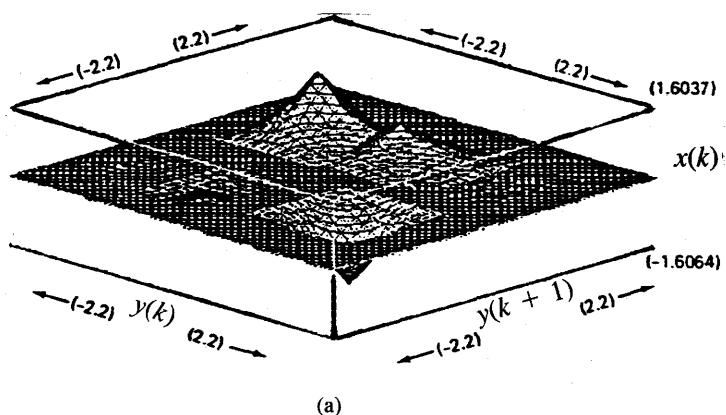
$$y(k + 1) - 0.1y(k) - 0.9r(k) = 0 \quad (8.24)$$

A unity magnitude, bipolar square wave of frequency 1 Hz has been chosen as the reference wave  $r(t)$ . Samples  $r(k)$  of the continuous wave  $r(t)$  have been assumed to be taken with frequency 9 Hz. Also, the plant was considered to be at rest initially. The reader can verify that this set of assumptions allows for computation of the entire sequence of plant output samples  $y(k)$ , for  $k = 0, 1, 2, \dots$ , using Equation (8.24).

Three experiments for learning the characteristic surface with simultaneous control of the plant have been performed (Kraft and Campagna 1990a, 1990b). The first experiment was performed with noise-free input. The plant input and output measurements in the second experiment were contaminated with a significant amount of noise. The noise was white and uniformly distributed between  $-0.4$  and  $0.4$ . The third experiment involved no noise, but the plant to be learned was nonlinear. The plant had the linear characteristic surface augmented by the nonlinear term  $-0.4[y(k)]^{1/3}$  to the form

$$y(k + 1) = y(k) - 0.4[y(k)]^{1/3} + x(k) \quad (8.25)$$

Each of the three networks has been trained for 625 weights, or memory locations. The characteristic surface learned in the first experiment for  $\alpha = 0.3$  and in the generalized learning mode is shown in Figure 8.30(a). The CMAC surfaces learned in the two remaining cases exhibit considerable similarities to the surface shown in the figure. The learning covers the domain  $-1 < y(k) < 1$  and  $-1 < y(k + 1) < 1$ . The reader may notice that the plant output must be constrained to this region because of the choice of both the reference wave and the form of the difference equation (8.24). Specifically, the parts of the characteristic surface that are close to corners  $(-1, 1), (1, -1)$  in coordinates  $y(k), y(k + 1)$  indicate the largest and the smallest control signal values, respectively. Ideally, the learned surface should have been exactly as shown in Figure 8.26(a). However, the learning of the initially unknown surface remains notably localized.



Time Instant	Reference Wave	Plant Output		Control Signal
		Present	Next	
$k$	$r(k)$	$y(k)$	$y(k+1)$	$x(k)$
0	0	0	0	0
1	1	0	0.9	0.9
2	1	0.9	0.99	0.09
3	1	0.99	0.999	0.009
4	1	0.999	0.9999	0.0009
5	-1	0.9999	-0.8	-1.7999
6	-1	-0.8	-0.98	-0.1799
7	1	-0.98	-0.998	-0.018
8	1	-0.998	-0.9998	-0.0018
9	0	-0.9998	-0.1	0.8998
10	1	-0.1	0.8	0.9
11	1	0.8	0.98	0.18
12	1	0.98	0.998	0.018

$$y(k+1) = 0.1y(k) + 0.9r(k) \text{ (filter)}$$

$$x(k) = y(k+1) - y(k) \text{ (plant)}$$

(b)

**Figure 8.30** CMAC learning: (a) characteristic surface learned and (b) control signals computed for the known plant. [Part (a), ©MIT Press; adapted from Kraft and Campagna (1990b) with permission.]

For the coordinates  $y(k)$ ,  $y(k+1)$  that occur frequently, the learning resulted in local peaks; some other areas of the characteristic surface have not deflected from the initial value of  $w_{ij} = 0$ .

Figure 8.30(b) tabulates 12 initial steps of CMAC learning for the experiment described, with no input noise present. Plant outputs  $y(k)$ ,  $y(k+1)$  have

been computed from Eq. (8.24). The data tabulated in the plant input column  $x(k)$  were obtained assuming a known plant rather than an unknown plant being identified using the signal provided by the conventional feedback controller. The controlled plant in this example is described by Eq. (8.22) with  $a = b = 1$ .

The measured signal tracking performance of the actual CMAC is plotted in Figure 8.31. The solid line represents the actual plant output, the dashed line indicates the desired output  $y(k)$ . It can be seen from the graphs that the trained controller's performance has proven to be at least satisfactory. For the no-noise case, 25 learning cycles involving a time frame of 25 full periods of the square wave  $r(k)$  have been performed prior to the 30 output samples displayed in the figures. Learning with noise took longer, but the controller's final performance shown in Figure 8.31(b) was approximately the same as if it had learned without noise. Especially high performance was achieved for nonlinear plant learning. As shown in Figure 8.31(c), the tracking achieved was very good and output was very close to the required wave.

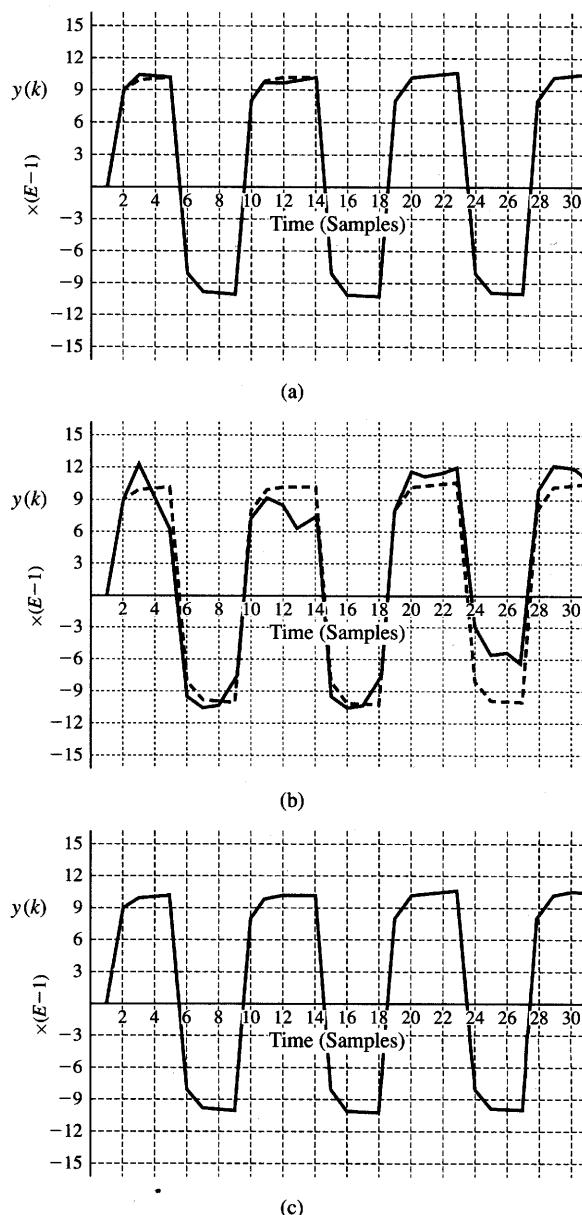
As can be seen, the controller under discussion is learning in a supervised mode. The controller's training involves characteristic surface buildup through weights gradually matching the plant inputs needed to produce the desired outputs. The neurons in this network are with the identity activation function and therefore they do not participate in the processing.

The CMAC, compared to conventional adaptive controllers performing the control task as described, has shown distinct desirable characteristics such as accuracy of learning and learning speed. In addition, it ranks favorably in terms of actual control speed. Upon learning, the control signal is instantaneously available, since it is derived directly from the look-up memory, which stores characteristic surface values. Moreover, the neurocontroller contains no restrictions regarding the plant and system linearity, and it seems to perform well in a noisy environment.

---

### Concluding Remarks

The section on neural network control applications explores several central issues such as identification, control of static and dynamical plants, and example designs of selected low-order neurocontrollers. Control of example plants operating in both continuous- and discrete-time modes was considered. However, the exposition does not unveil a complete range of problems and ready-to-use solutions. Rather, basic concepts, problems, and solutions are outlined along with some promising directions for development of neurocontrol systems. The main reason for such a preliminary exposition is that the potential of artificial neural systems as neurocontrollers still has to be researched before it can be fully realized (Barto 1990). The study of the technical literature available seems



**Figure 8.31** CMAC-generated plant response (dashed lines denote desired output): (a) plant as in (8.22a),  $a = b = 1$ , (b) same as part (a) but noise added, and (c) noise-free nonlinear plant as in (8.25). [©MIT Press; reprinted from Kraft and Campagna (1990b) with permission.]

to indicate that much has to be done before the systematic design of learning controllers can be undertaken.

At this stage of neurocontrol development, several areas of possible exploration are worth emphasizing. Neural networks can be used as models of dynamical systems with outputs dependent on input history. Suitable architectures and more systematic modeling approaches need to be devised for this purpose. Neural networks can also serve as controllers, and they seem to be especially promising for nonlinear dynamical systems control. Acquisition and storage of control information seems to be well suited to the applications of associative memory concepts. Such neurocontrollers would take full advantage of associative memory systems having high capacity, high speed, and the ability to interpolate and extrapolate control data in real time (Barto 1990). In addition, theoretical and experimental investigations of the stability of neurocontrollers needs to be undertaken. This would include both learning and control stability of such controllers.

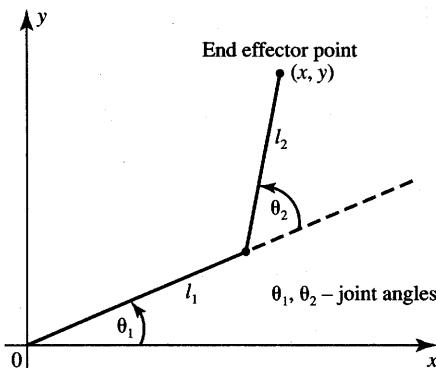
To achieve this aim, novel architectures and training approaches, and an appropriate mixture of conventional and neural control need to be developed. The choices of the traditional identifier and controller structures are usually based on the well-established results of linear systems theory. However, existing design procedures for stable nonlinear dynamical systems are valid only for system-by-system cases. Such procedures are currently not available for designing large classes of such systems. The first comprehensive theoretical and experimental results in the identification and control of nonlinear dynamical systems using neural networks have been reported only recently (Narendra and Partha Sarathy 1990, 1991).

One particularly interesting and promising concept is that of so-called dynamic back-propagation (Hecht-Nielsen 1990). Since most physical systems involve dynamics, their realistic modeling can involve a combination of linear dynamical systems and static multilayer neural networks. This combination can be considered as an extension of error back-propagation architectures for static systems. The study has shown that multilayer neural networks using dynamic error back-propagation can be very effective in performing identification and control tasks. The reader is encouraged to look to references for more advanced coverage of neurocontrol issues (Nguyen and Widrow 1990).

## 8.4

### NETWORKS FOR ROBOT KINEMATICS

Although neural networks applicable to the solution of robotics control problems are, in fact, neurocontrollers, their function is specialized mainly to provide solutions to robot arm motion problems. In this section, neural network models



**Figure 8.32** Geometry of planar manipulator with two degrees of freedom.

are considered for solving a number of robot kinematics problems. Robot kinematics involves the study of the geometry of manipulator arm motions. Since the performance of specific manipulator tasks is achieved through movement of the manipulator linkages, kinematics is of fundamental importance for robot design and control. Trajectory control and learning of the robot arms motions to achieve the desired final position are the main kinematics tasks covered below.

### Overview of Robot Kinematics Problems

Our focus in this section is to provide the design fundamentals for neuro-controllers used in robotics systems. Trajectory control of robotic manipulators traditionally consists of following a preprogrammed sequence of end effector movements. Robot control usually requires control signals applied at the joints of the robot while the desired trajectory, or sequence of arm end positions, is specified for the end effector. Figure 8.32 shows the geometry of an idealized planar robot manipulator with two degrees of freedom. The robot arms operate in a plane. To make the arm move, the desired coordinates of the end effector point ( $x, y$ ) are fed to the robot controller so that it generates the joint angles ( $\theta_1, \theta_2$ ) for the motors that move the arms. To perform end effector position control of a robotic manipulator, two problems need to be solved (Craig 1986):

1. *Inverse kinematics problem:* Given the Cartesian coordinates of the end effector, specified either as a single point or as a set of points on a trajectory, joint angles or a set of joint angles need to be found.
2. *Target position control:* Given the final end effector position, a joint angles sequence suitable for achieving the final position needs to be found.

The kinematics considerations for the manipulator shown in Figure 8.32 are based on the *forward kinematic equation*. The forward kinematic equation involves mapping of joint angle coordinates  $(\theta_1, \theta_2)$  to the end effector position  $(x, y)$ . The mapping expressions can be obtained by inspection of the figure as follows

$$\begin{aligned} x &= l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ y &= l_2 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2) \end{aligned} \quad (8.26)$$

where  $\theta_1$  and  $\theta_2$  are the joint angles of the first and second arm segments, respectively, and  $l_1$ , and  $l_2$  are respective arm segment lengths. Relation (8.26) expresses the forward kinematic problem and implements unique mapping from the joint angle space to the Cartesian space.

The inverse kinematic problem is described by (Craig 1986):

$$\theta_2 = \cos^{-1} \left[ \frac{(x^2 + y^2 - l_1^2 - l_2^2)}{(2l_1 l_2)} \right] \quad (8.27a)$$

$$\theta_1 = \tan^{-1}(y/x) - \tan^{-1} \left[ l_2 \sin \theta_2 / (l_1 + l_2 \cos \theta_2) \right] \quad (8.27b)$$

Since  $\cos^{-1}$  is not a single-valued function in the range of angles of interest, two possible orientations typically result from (8.27) for the robot arm joint angles. The arm can be positioned with the elbow up or down, with the end effector still at the required  $(x, y)$  point. The inverse kinematic transformation (8.27) implementing mapping from Cartesian space to joint space is thus not unique.

As can be seen from Equation (8.27), control of robot arms requires the solution of the inverse kinematic equation system in real time. In the general case of an end effector with  $n$  degrees of freedom, the forward and inverse kinematic problems can be formulated, respectively, as

$$\mathbf{x} = \mathbf{h}(\boldsymbol{\theta}) \quad (8.28)$$

$$\boldsymbol{\theta} = \mathbf{h}^{-1}(\mathbf{x}) \quad (8.29)$$

where  $\boldsymbol{\theta}$  and  $\mathbf{x}$  are joint angles and end effector Cartesian coordinate vectors, respectively, which are defined as follows:

$$\boldsymbol{\theta} \triangleq [\theta_1 \ \theta_2 \ \dots \ \theta_k]^t$$

$$\mathbf{x} \triangleq [x_1 \ x_2 \ \dots \ x_n]^t$$

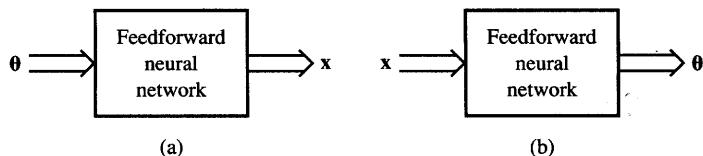
Even though the numerical solution of (8.29) can be found, this requires large, real-time computational resources. The neural network approach may be used here to solve both the problem of kinematic task formulation and the setting up of the equations. It also allows for circumvention of the computational complexity involved in their numerical solution.

### Solution of the Forward and Inverse Kinematics Problems

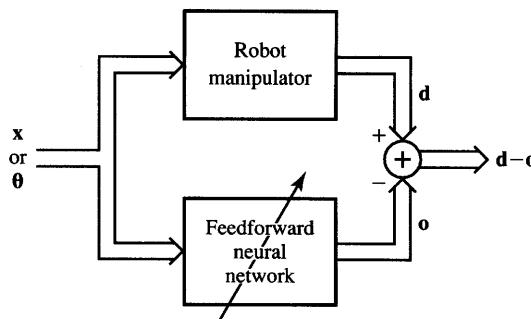
The forward and inverse kinematics problems can be solved by multilayer feedforward networks as shown in the form of a block diagram in Figure 8.33. The network in Figure 8.33(a) needs to be trained to learn the set of end effector positions  $x$  from the given set of joint angle examples. The network in Figure 8.33(b) is trained to learn the set of joint angles  $\theta$  from the given set of end effector position examples. Each of the neural networks involved performs static mapping of two two-variable functions as discussed in earlier chapters.

The supervised learning diagram of both configurations from Figure 8.33 is shown in Figure 8.34. The reader may notice that this configuration falls into the general category of forward plant identification first introduced in Figure 8.17(a). The robot manipulator output  $d$  provides the teacher's signal, or the desired output value, which is compared with the network actual output  $o$ . Based on the error value  $\|d - o\|$ , the neural network weights are tuned to achieve the required accuracy of mapping. Note that the teacher's signal  $d$  for the training set can be provided either by solving (8.28) or (8.29), or by taking the respective measurements of the Cartesian or joint coordinates of the actual manipulator. The learning of the manipulator proceeds off-line. Once the network has been trained and installed in the system, no more training is allowed since the neurocontroller is placed in front of the robot manipulator as discussed in the following paragraph. Therefore, the set of training data should be uniformly distributed over the robot's working area so that the network can make good generalizations for the intermediate points.

Obviously, once the network has been trained to perform the required kinematic transformation, it then needs to be employed in the inverse feedforward control mode. If the neural network has been trained to solve a forward kinematics problem, its natural application is to control the manipulator, which needs Cartesian coordinates  $x$  as input and produces joint angles  $\theta$  as outputs. This is shown in Figure 8.35(a). The input to the neural controller is the vector of desired joint angles,  $\theta_d$ . After error-free training of the controller, we should have  $\theta = \theta_d$  at the output of the robot manipulator. Using the trained network shown



**Figure 8.33** Neural networks for robot kinematics transformation: (a) forward kinematics problem and (b) inverse kinematics problem.

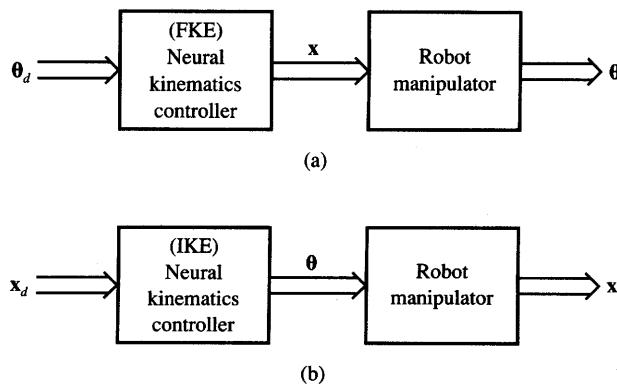


**Figure 8.34** Robot manipulator supervised training.

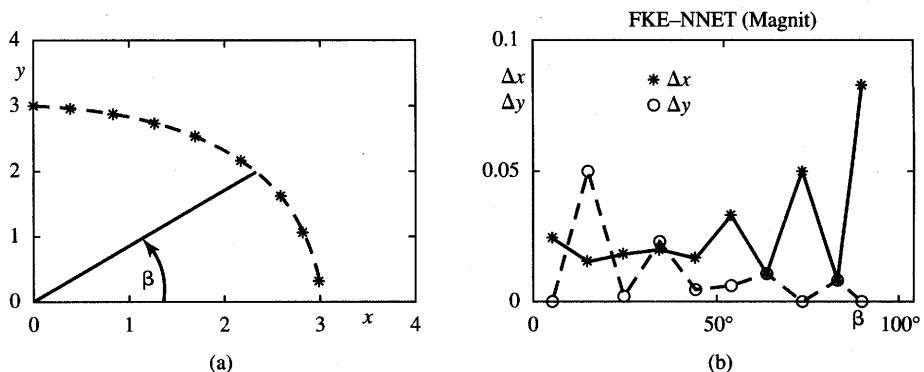
in Figure 8.35(b) requires specification of the desired end effector coordinates  $x_d$ . The network translates these coordinates to angles  $\theta$  and the robot manipulator should respond with  $x = x_d$ .

Let us review the forward kinematics trajectory learning for the two degrees of freedom (planar) manipulator. The concept of “trajectory” is understood here as the end effector workspace rather than as a sequence of points to be followed. In designing trajectory we are therefore dealing with specializing the robot end effector movement in a specific work area rather than training it to reach sequence of points ending at a target position. Target position learning is studied separately later in this chapter.

In this example the robot is required to learn Cartesian coordinates  $x, y$  for the circular trajectory of radius  $r$ . A network with two inputs ( $\theta_1$  and  $\theta_2$ ), four hidden nodes, and two output neurons yielding outputs  $x, y$  has been selected



**Figure 8.35** Controlling robot manipulator using neural controller: (a) joint angles input and (b) target coordinates (end effector) input.



**Figure 8.36** Forward kinematics solution for circular trajectory,  $r = 3$ : (a) response of the trained network and (b) mapping error for training data reused in the test mode. [©IEEE; adapted from Arteaga-Bravo (1990) with permission.]

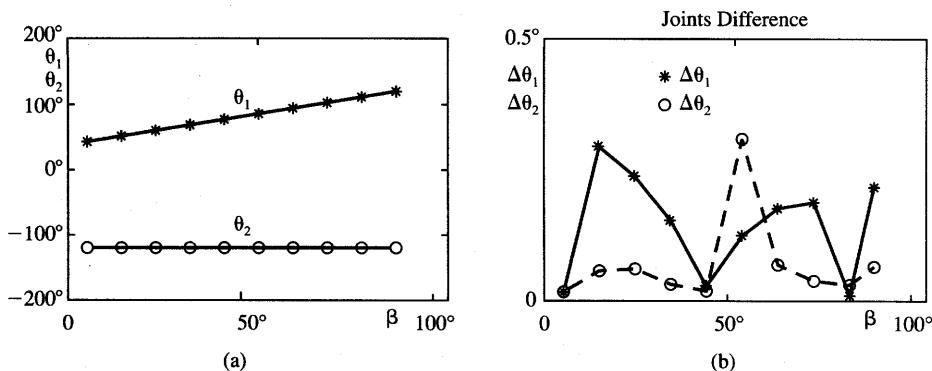
to simulate the expression (8.26) (Arteaga-Bravo 1990). The constraint on the circular trajectory, which needs to be learned, is

$$\begin{aligned}x &= r \cos \beta \\y &= r \sin \beta\end{aligned}\quad (8.30)$$

where angle  $\beta$  is covering the first quadrant in this example. The training has been performed for  $l_1 = 3$ ,  $l_2 = 2$ , and  $r = 3$  at 10 points of the selected trajectory. The 2-4-2 network has been trained using the error back-propagation technique with learning constant  $\eta = 0.9$ , and  $\alpha = 0.9$  for the momentum term. The training data of the trajectory with  $r = 3$  contains angle  $\beta$  between 5 and  $85^\circ$ , spaced by  $10^\circ$ , and  $\beta = 90^\circ$ .

The trained 2-4-2 network has subsequently been evaluated using the training data in the test mode. The results of training are shown in Figure 8.36(a). The figure illustrates an accurate mapping of angles into the desired end effector positions forming the trajectory  $\sqrt{x^2 + y^2} = 3$ . The differences between the accurate forward kinematics solutions and those computed by the neural network are illustrated in Figure 8.36(b). The maximum magnitude difference, or maximum absolute error, has been observed to be about 0.08 for the  $x$  coordinate at  $\beta = 90^\circ$ .

A multilayer feedforward network with two hidden layers containing four neurons each has then been employed to perform the manipulator's inverse kinematic transformation for the same circular trajectory. The 2-4-4-2 network has been selected for training with parameters  $\eta = 0.9$  and  $\alpha = 0.3$ . The results of the evaluation of the trained network for the training data  $\theta_1$  and  $\theta_2$  are illustrated in Figure 8.37(a). The figure illustrates a mapping of points from the trajectory (8.30) with  $r = 3$  into joint angles  $\theta_1$  and  $\theta_2$ . The differences between the accurate inverse kinematics solutions and those computed by the network at the training



**Figure 8.37** Inverse kinematics solution for circular trajectory,  $r = 3$ : (a) response of the trained network and (b) mapping error for training data in the test mode. [©IEEE; adapted from Arteaga-Bravo (1990) with permission.]

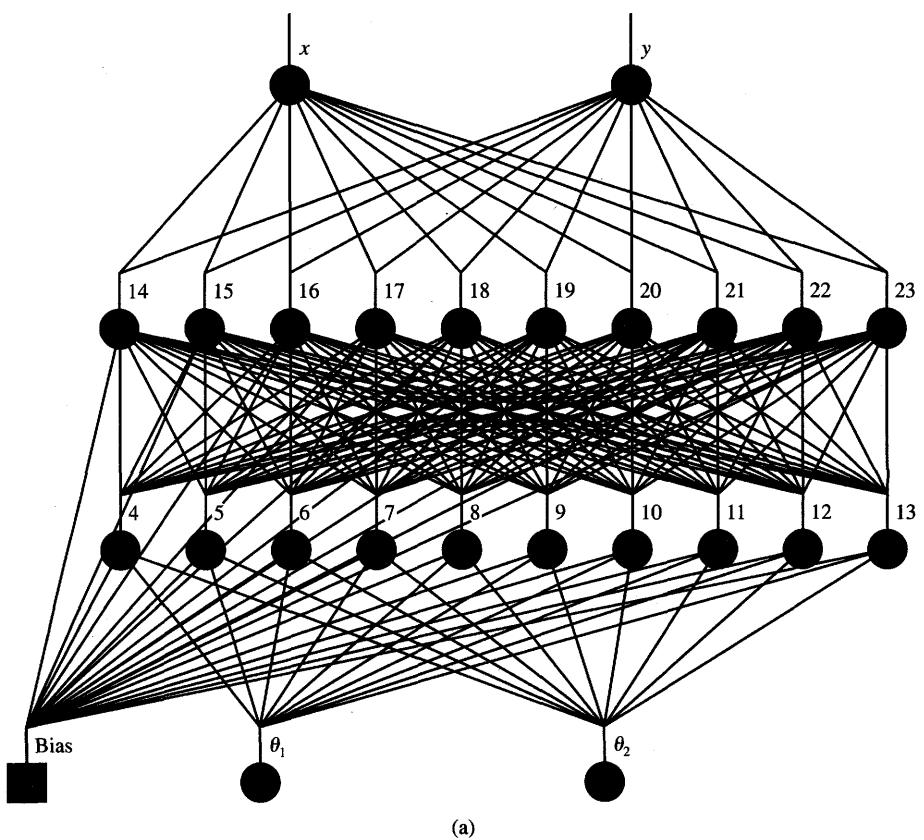
points are depicted in Figure 8.37(b). The largest absolute error of the joint angle has been approximately  $0.33^\circ$  for  $\theta_2$  at  $\beta = 55^\circ$ .

Neurons with unipolar activation functions have been used in this experiment. Training input data for both cases have been mapped to the range  $(0,1)$  in order to make more efficient use of the unipolar activation function and to obtain better results in training the networks.

### Comparison of Architectures for the Forward Kinematics Problem

In this section, various feedforward neural networks are considered for solving the forward kinematics problem for robots with two degrees of freedom. Several network configurations have been designed and compared for modeling of the manipulator kinematics (Nguyen, Patel, and Khorasani 1990). A manipulator is treated in the description below as a black box for which the training samples are collected from geometrical measurements. The specific trajectory of the end effector is used to test the outcome of the training for selected neural architectures.

Each of the networks used for evaluation has two input nodes,  $\theta_1$  and  $\theta_2$ , in addition to the bias node, and two output nodes,  $x$  and  $y$ . Since the unnormalized output values are Cartesian coordinates and exceed the range  $(-1, 1)$  covered by the bipolar sigmoidal-type activation functions, output neurons are selected such that they have an identity activation function and perform scalar product computation only. Architectures studied have been: 2–10–10–2 for the network trained with the error back-propagation algorithm (BP), 2–10–10–2 for the network trained with the back-propagation algorithm but with output splitting (BPOS), 2–8–2 for the functional link network (FL). In addition, a counterpropagation

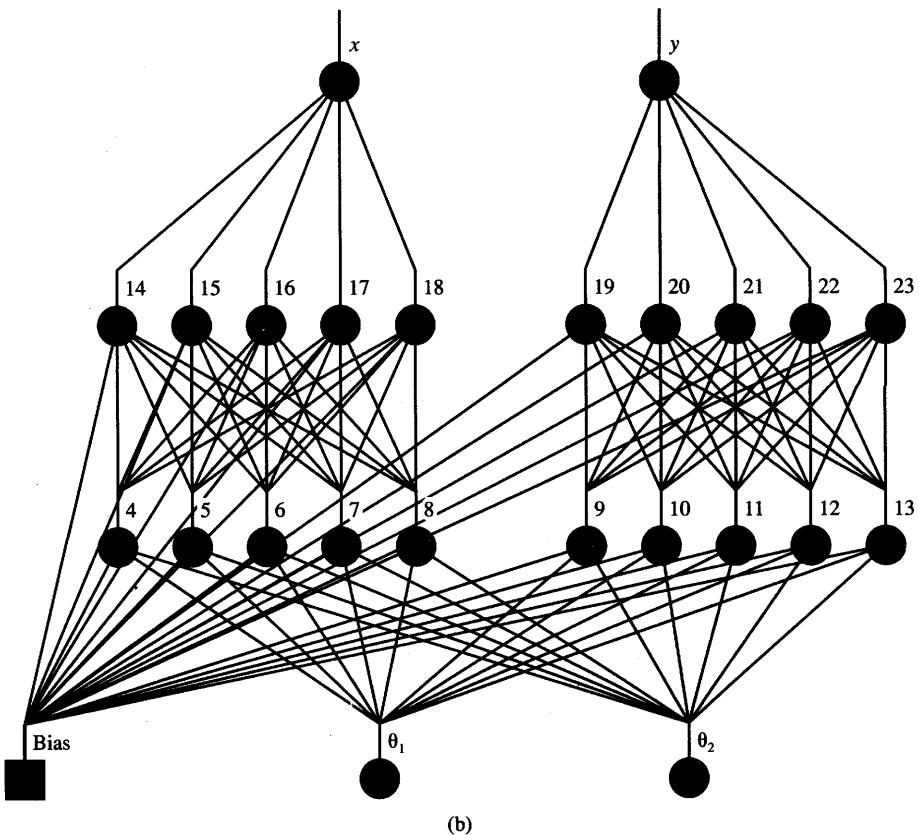


**Figure 8.38a** Architectures for a learning manipulator's forward kinematics problem: (a) BP [©IEEE; adapted from Nguyen, Patel, and Khorasani (1990) with permission.]

network (CP) has been used to provide trajectory mapping (Nguyen, Patel, and Khorasani 1990).

The networks have been trained using a set consisting of 64 input/output pairs obtained from measurements taken in the robot working area. Weights have been initialized at random values between  $-0.5$  and  $0.5$ , except for output layer weights, which have been initialized at zero. The joint angle values for training have been selected in the first quadrant in such a way that they uniformly cover the entire work area of interest defined by  $0 < \theta_1 < 90^\circ$  and  $0 < \theta_2 < 90^\circ$ .

BP and BPOS error back-propagation architectures have been trained with  $\eta = 0.1$  and  $\eta = 0.4$  for the first and second hidden layers, respectively. The architecture of the BPOS network is the modified version of the common BP network. The nodes in both hidden layers of the BPOS network have been divided symmetrically into two groups. Both networks are shown in Figure 8.38. The FL network is composed of eight input nodes providing the following signals:



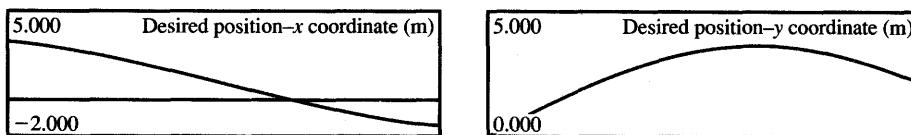
**Figure 8.38b** Architectures for a learning manipulator's forward kinematics problem (continued): (b) BPOS. [©IEEE; adapted from Nguyen, Patel, and Khorasani (1990) with permission.]

$\theta_1$ ,  $\theta_2$ ,  $\sin \theta_1$ ,  $\sin \theta_2$ ,  $\cos \theta_1$ ,  $\cos \theta_2$ ,  $\sin(\theta_1 + \theta_2)$ , and  $\cos(\theta_1 + \theta_2)$ . A simple delta rule was used for learning of the single-layer feedforward FL network with a learning rate of  $\eta = 0.4$ .

The CP network design has followed the guidelines of Chapter 7 with the Kohonen layer used as a first CP network layer. This part of the network layer has been trained using the winner-take-all learning rule. The layer is followed by the Grossberg layer producing the desired network output. The input to the first layer of the CP network is provided by normalized input vectors. The CP network has performed here as a look-up table responding with the output, which is paired with the stored input that matches the output by being in the same training pair of vectors.

The following trajectory was used to train and test the discussed four architectures

$$\begin{aligned}\theta_1(t) &= 3 \cdot 10^{-3}t \\ \theta_2(t) &= 3 \cdot 10^{-3}t\end{aligned}\quad (8.31)$$



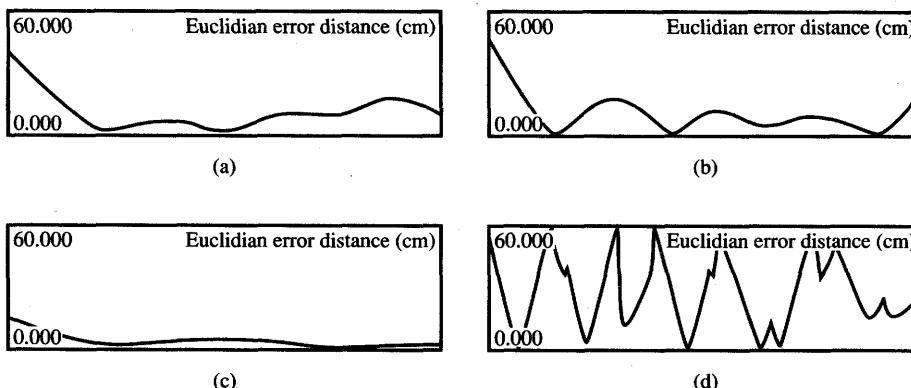
**Figure 8.39** Trajectory for network testing. [©IEEE; adapted from Nguyen, Patel, and Khorasani (1990) with permission.]

where  $0 < t < 500$  s. Figure 8.39 illustrates the trajectory used for network learning in Cartesian coordinates. The FKE equation (8.26) provides here the results for the Cartesian coordinates of the end effector.

The training of all four network architectures resulted in quick learning convergence to the desired trajectory. The basic BP network with two hidden layers gave a reasonably accurate solution. The error defined as distance (in centimeters) between the trajectory (8.31) and the one produced by the trained network is shown in Figure 8.40(a). The training of the BPOS network, which involved considerably fewer connections than the BP network, yielded comparable accuracy. This is shown in Figure 8.40(b). Although BPOS network training required 300 training cycles compared to 200 cycles for the same error of 10 percent for the BP network, its total training time has been shorter due to the smaller number of weights in the BPOS network.

The best solution of the trajectory learning task has been achieved with the FL network. The network required only 20 training cycles and the training resulted in the best approximation of the target trajectory (8.31) for all four cases studied. This can be seen from Figure 8.40(c).

In contrast with the multilayer feedforward networks and functional link network, the CP network implements a steplike approximation of the trajectory.



**Figure 8.40** Trajectory error for the test mode: (a) BP, (b) BPOS, (c) FL, and (d) CP. [©IEEE; adapted from Nguyen, Patel, and Khorasani (1990) with permission.]

The training results in relatively fast approximation of the trajectory, but the end effector position error is rather large and uneven, as can be seen from Figure 8.40(d). Therefore, the CP network in this application can be recommended only if a short development time is required for low-precision trajectory tracking.

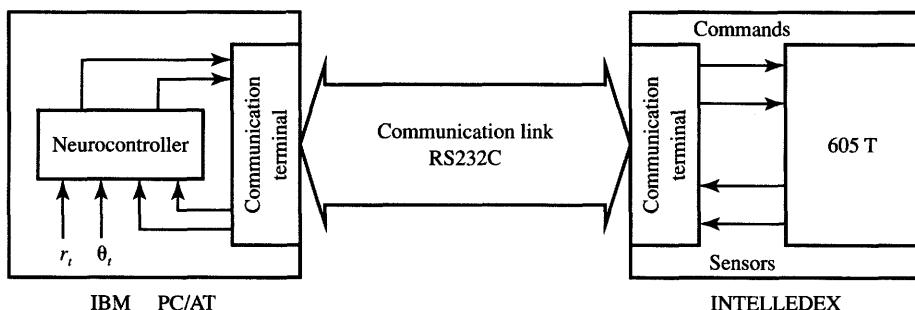
The discussion of neural network applications for solution of the forward and inverse kinematics problems and generation of the desired trajectories indicates that neural networks can efficiently learn manipulator kinematics. Modeling of the desired trajectories or kinematic transformations can be achieved with a considerable degree of accuracy using a number of network architectures. Training of a suitable neural network architecture can be performed based on geometrical measurements. Alternatively, training can be based on computed solutions of either forward or inverse kinematics problem for the robot working area of interest to the designer.

### Target Position Learning

So far our discussion of applications of neural networks for robotics has only focused on memoryless mapping of manipulator kinematics. Neural networks, however, can also be employed to learn a *sequence of mappings*. This is understood as learning a sequence of space transformations. In this sense, the learning of a target position can be understood as dynamical control learning. Robotic arms can be treated as dynamical plant. In addition to learning the trajectory or working area, which is essentially static, or point-to-point transformation, robot arms can be controlled to follow a sequence of steps and to reach a desired target position. In this section the study of robot arm movement with two degrees of freedom and with a desired final end effector position is discussed based on the work of Sobajic, Lu, and Pao (1988). In particular, the presence of trajectory disturbances is evaluated in this study.

Our objective is to move the end effector of a two-link planar robot manipulator toward a target point until the positions of the two coincide. The specific task for the neural network designer is to generate the sequence of joint vector increments, thus providing the appropriate sequence of incremental control signals. In this discussion we assume that the target end effector position that needs to be learned is specified in polar coordinates  $r_t \angle \theta_t$ . An error back-propagation network with architecture 4–6–4–2 has been applied to perform the task (Sobajic, Lu, and Pao 1988). Network input signals are equal to the currently measured joint angles  $\theta_1(t)$  and  $\theta_2(t)$ , and also to the fixed target coordinates  $r_t$  and  $\theta_t$ . The network needs to map angles  $\theta_1$  and  $\theta_2$  into the two control actions  $k_1\Delta\theta_1$  and  $k_2\Delta\theta_2$ , respectively. The control state equations have the following form:

$$\begin{aligned}\theta_1(t+1) &= \theta_1(t) + k_1(t)\Delta\theta_1 \\ \theta_2(t+1) &= \theta_2(t) + k_2(t)\Delta\theta_2\end{aligned}\tag{8.32}$$



**Figure 8.41** The communication scheme between the Intelledex 605T and the neurocontroller.  
[©IEEE; reprinted from Sobajic, Lu, and Pao (1988) with permission.]

The actual outputs of the neural network are  $k_1(t)$  and  $k_2(t)$ . The values are limited to the range  $(-1, 1)$  since bipolar activation functions are used. An Intelledex 605T-type robot and the neural network controller for that experiment have been configured in a closed-loop feedback system (Sobajic, Lu, and Pao 1988). As shown in Figure 8.41, the information exchange between the neurocontroller and the robot is carried by the RS232 serial port.

Two experiments have been performed. In the first experiment, the neural network controller has been trained to guide the arm end effector from any initial location to the specified single fixed target position. The training has been implemented based on the set of 26 input vectors covering the anticipated work area and involving a number of different initial points. Network training results are tabulated in Figure 8.42(a). Eight different initial point positions have been used for training as shown in the learning control experiment. The maximum relative target distance error reported in the experiment is 1.46 percent. All remaining relative target distance errors were below 1 percent.

The second experiment has been designed to move the robot arm from any initial location to one of a number of specified target positions. The training has been performed using the set of 64 training input vectors for each of the 14 specified target points being entrained. Results of the training have been tabulated in Figure 8.42(b) in 14 rows, with each row corresponding to one target point. All trials originated at  $\theta_1 = \theta_2 = 0$ . The maximum relative target distance error reported was 2.36 percent. The final weight values of the trained network of size 4–6–4–2 are listed in Figure 8.43.

In addition to testing the trained network under normal working circumstances as reported above, the controller was also investigated when affected by the presence of unforeseen disturbances. First, the network trajectory control was overridden first at time  $t = t_1$ , and then at time  $t = t_2$ . Thus, the robot arm was deflected twice to undesirable positions off the trajectory. In each case, the neural

Learning Experiment	Initial Position		Required Final Position		Implemented Final Position		Error
	$\theta_1$	$\theta_2$	$r_t$	$\theta_t$	$r_f$	$\theta_f$	
1	30.	30.	40.	-30.	39.96	-29.48	0.69%
2	30.	-90.	40.	-30.	39.66	-29.67	0.77%
3	-10.	30.	40.	-30.	39.91	-30.18	0.28%
4	-10.	-90.	40.	-30.	39.49	-30.86	1.46%
5	-50.	-30.	40.	-30.	40.24	-29.47	0.82%
6	-50.	90.	40.	-30.	39.92	-29.32	0.90%
7	-100.	-30.	40.	-30.	40.31	-30.36	0.75%
8	-100.	90.	40.	-30.	40.10	-30.39	0.54%

(a)

Learning Experiment	Initial Position		Required Final Position		Implemented Final Position		Error
	$\theta_1$	$\theta_2$	$r_t$	$\theta_t$	$r_f$	$\theta_f$	
1	0.	0.	40.	-45.	39.71	-44.99	0.54%
2	0.	0.	30.	-45.	30.00	-42.88	2.08%
3	0.	0.	20.	-45.	18.65	-46.63	2.73%
4	0.	0.	40.	-30.	39.71	-30.10	0.56%
5	0.	0.	30.	-30.	30.17	-27.70	2.28%
6	0.	0.	20.	-30.	18.81	-31.24	2.36%
7	0.	0.	40.	-15.	39.73	-15.28	0.62%
8	0.	0.	30.	-15.	30.26	-12.87	2.15%
9	0.	0.	20.	-15.	19.12	-15.82	1.72%
10	0.	0.	40.	0.	39.79	-0.62	0.90%
11	0.	0.	20.	0.	19.40	-0.43	1.16%
12	0.	0.	40.	15.	39.79	14.05	1.25%
13	0.	0.	30.	15.	30.57	16.63	1.93%
14	0.	0.	20.	15.	19.72	14.50	0.62%

$(\theta_1, \theta_2)$ —initial position of the robot arm.  
 $(r_t, \theta_t)$ —target position.  
 $(r_f, \theta_f)$ —final position of the robot arm.

error—the relative error.

(b)

**Figure 8.42** Result of the 4–6–4–2 network training: (a) fixed target position, different starting points, and (b) different target positions, different starting points. [©IEE; reprinted from Sobajic, Lu, and Pao (1988) with permission.]

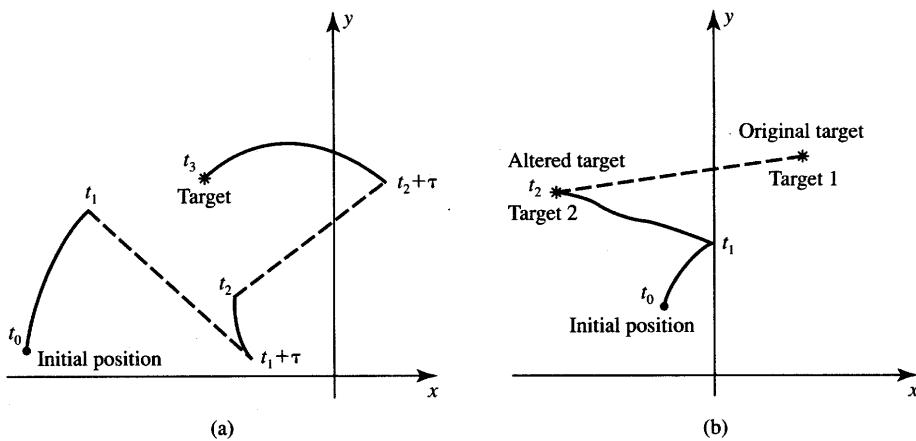
controller has shown remarkable response and it produced a new and correct trajectory, as illustrated in Figure 8.44(a). In another experiment, the target position was altered at  $t = t_1$  while the robot arm was already following the desired trajectory. As a result of the sudden alteration of the target, the neural controller

Level 1		Weights			
		-0.028	8.578	-0.198	14.050
		1.504	10.399	0.889	10.752
		-1.535	16.108	0.417	14.590
		-5.790	-5.707	3.421	-1.213
		-1.192	-0.606	3.323	-3.638
		18.238	2.350	-18.866	-0.189
Level 2		Thresholds			
	-12.141	-12.409	-18.521	2.801	1.217
					-5.061
Level 2		Weights			
		-0.472	-2.300	-3.269	-1.451
		5.670	5.270	8.206	-1.576
		1.330	0.618	-0.563	4.573
		-0.595	-0.685	-0.402	-1.648
Level 3		Thresholds			
		0.921	-6.032	3.971	-0.422
Level 3		Weights			
		-2.656	-0.059	-5.215	-2.623
		4.642	-4.596	0.346	0.434
Level 3		Thresholds			
		3.180	1.884		

**Figure 8.43** Tabulated weights of the trained network for the different target points experiment. [©IEEE; reprinted from Sobajic, Lu, and Pao (1988) with permission.]

generated the revised set of command actions, and the new target position was reached. This is illustrated in Figure 8.44(b).

The described project with target position guidance control has made use of multilayer feedforward controllers trained using the error back-propagation techniques. The results achieved demonstrate their unusual efficiency and potential for use in robot kinematics control systems. The performance of neural controllers described in this section can be based entirely on actual geometrical conditions measured in the robot working space. Therefore, robot neurocontrollers can be developed without the need for complex mathematical transformations or even mathematical modeling of the problem. The discussed networks have been able to learn and generalize not only different static space-to-space mappings, but also the robot movement dynamics. The networks were also able to memorize the sequence of control signals needed to achieve one of the specified final positions.



**Figure 8.44** The trajectory of the robot arm in the presence of disturbances: (a) the arm position disturbance and (b) the target coordinates alteration. [©IEEE; reprinted from Sobajic, Lu, and Pao (1988) with permission.]

## 8.5

### CONNECTIONIST EXPERT SYSTEMS FOR MEDICAL DIAGNOSIS

This section presents examples of connectionist expert systems for medical diagnosis. The introductory concepts of multilayer feedforward networks with error back-propagation training used for medical diagnosis have already been formulated in Section 4.6 and illustrated in Figure 4.24. This section discusses in more detail the specific capabilities and limitations of medical diagnosis connectionist expert systems. Example expert systems for diagnosis of skin, low back pain, and heart infarction diseases are presented.

Medical diagnosis is performed with multiple, often related, disease states that can occur with identical symptoms and clinical data. In addition, specific diseases are not always accompanied by the same symptoms and clinical and historical data. As a result of these difficulties, both physician accuracy and computer-aided diagnosis systems leave room for improvement. This is one of the reasons for our study. Let us review the basic issues of computer-aided diagnosis.

Computer-aided diagnosis systems have been successfully used in a number of areas of medicine. Among several approaches to automated diagnosis, rule-based systems and knowledge-based systems using the Bayesian approach seem to be the most widespread. Both approaches, however, seem to suffer from some problems in medical diagnosis applications. Rule-based systems typically require formulation of explicit diagnostic rules. Much of the human knowledge, and

especially expert or medical expert knowledge, however, remain implicit. Implicit knowledge can be expressed neither as a mathematical function for numerical computation nor as an if-then rule for symbolic processing. Even expressing the implicit knowledge as a conditional probability density function encounters difficulties because of statistical interdependencies between the events.

Generally, implicit knowledge is difficult to quantize, formalize, or sometimes even express verbally. In addition to these difficulties, translation of implicit knowledge into explicit rules would thus lead to loss and distortion of information content. In other words, domains with implicit knowledge are difficult to express in terms of accurate specifications of decision rules. In addition, the knowledge engineer would have to understand thoroughly the domain and the expert reasoning in order to extract the rules for the system. Building an expert system for diagnosis using decision rules remains an expensive and time-consuming task.

The approach based on Bayes theorem involves comparisons of relative likelihoods of different diagnoses. The individual likelihoods of diseases are computed from individual patient information. Symptoms of diseases have to be assumed to be statistically independent in the Bayesian approach. This assumption underlying the Bayes theorem, however, does not apply widely in diagnostic practice. This is because several symptoms often arise due to the same organic cause. On the other hand, such symptoms have to be kept separated because they may provide vital decision information. Thus, a useful performance by practical diagnosis systems that are based on Bayes theorem is not always guaranteed.

The connectionist expert system can be trained to associate diagnoses with the symptom data. This presents an alternative approach that is essentially data driven. The connectionist approach lies in the middle ground between the rule-based and Bayesian approach. The experimental and heuristic approach prevailing in building connectionist expert systems focuses on repeated presentation of training examples of the various symptoms along with the diagnosed diseases. After training of a suitable network architecture is complete, test data and generalization by the network should lead to correct disease diagnosis.

---

### Expert System for Skin Diseases Diagnosis

The expert system called DESKNET has been designed for instruction of medical students in the diagnosis of papulosquamous skin diseases. These are diseases that exhibit bumpiness or scaliness of the skin (Yoon et al. 1989). The system has been developed and used mostly for symptom gathering and for developing diagnosing skills rather than for replacing the doctor's diagnosis. The expert system uses multilayer feedforward network and employs the 96-20-10 architecture with a single hidden layer and continuous unipolar neurons.

Input for the system consists of the following skin disease symptoms and their parameters: location, distribution, shape, arrangement, pattern, number of

lesions, presence of an active border, amount of scale, elevation of papules, color, altered pigmentation, itching, pustules, lymphadenopathy, palmar thickening, results of microscopic examination, the presence of herald patch, and the result of the dermatology test called KOH. In addition to the current symptom-generated data from the above list, the duration of skin lesions in days and weeks are also represented at the input.

The system has a total of 96 inputs plus the fixed bias input, and 10 output neurons. Inputs of values 0 and 1 are fed to the network signifying the absence or presence of the symptoms and their respective durations. The output neurons used in the local representation mode are indicative of the following 10 diseases diagnosed: psoriasis, pityriasis rubra pilaris, lichen planus, pityriasis rosea, tinea versicolor, dermatophytosis, cutaneous T-cell lymphoma, secondary syphilis, chronic contact dermatitis, and seborrheic dermatitis.

The training data for the DESKNET system consisted of input specifications of 10 model diseases collected from 250 patients. If specific symptoms or their parameters were not known, the input was coded as 0.5. In this way the outputs of unipolar neurons used in this network were made independent of that input. The network was trained using the standard error back-propagation algorithm.

The performance of the developed network was tested for previously unused symptom and disease data collected from 99 patients. The correct diagnosis was achieved for 70% of the papulosquamous group skin diseases. The success rate was above 80% for the remaining diseases except for psoriasis. Psoriasis patients were diagnosed correctly only in 30% of the cases. According to the domain expert, however, psoriasis often resembles other diseases within the papulosquamous group, which makes it somewhat difficult to recognize even for specialists.

In case of a rule-based system whose knowledge base consists of a set of explicit rules, the decision path can be traced and explanation of the decision can be provided to the user. Connectionist expert systems reach conclusions through rather complex, nonlinear and simultaneous synergistic interaction of many units at a time. The accurate justification of an hypothesis, therefore, involves all nodes of the network. If the justification of a conclusion is expected from a discussed system, analyzing the effect of a single input, or selected group of inputs, would be very difficult and would yield inaccurate results.

However, some explanation capability exists within a trained connectionist expert system. Despite its limitations, this capability can offer certain interpretations to the user, and also reveals those input variables that are more important and are likely to contribute to decisions. For the local representation network, one of the output nodes is activated stronger than any of the remaining ones. Weights leading to this node with relatively large magnitudes are therefore contributing more to the decisions than the remaining weights, each considered separately. The positive and negative signs of relatively large weights can be looked at as indicative of positive and negative contributions, respectively. An additional consideration is that the hidden layer re-represents the input data and the outputs

of hidden nodes are neither symptoms nor diagnostic decisions. In a network with a single hidden layer, internal data representation provided by the hidden layer rather than the original input data is used for generating the final decision. Internal representation can, however, offer further heuristic interpretations of the knowledge processing within the system.

In the case of the DESKNET expert system, which uses 20 hidden-layer nodes, the total of 20 hidden-layer factors attributed to the level of hidden node responses possess the discrimination capability for the set of 10 diseases. The factors are transmitted by large weights connected to the strongest activated output neuron of the output layer. With reference to Figure 4.7, such weights can be labeled  $w_{k_1 j_1}, w_{k_1 j_2}, \dots, w_{k_1 j_l}$ , where  $k_1$  is the output node number indicating the disease diagnosed, and there are  $l$  internal factors supporting it.

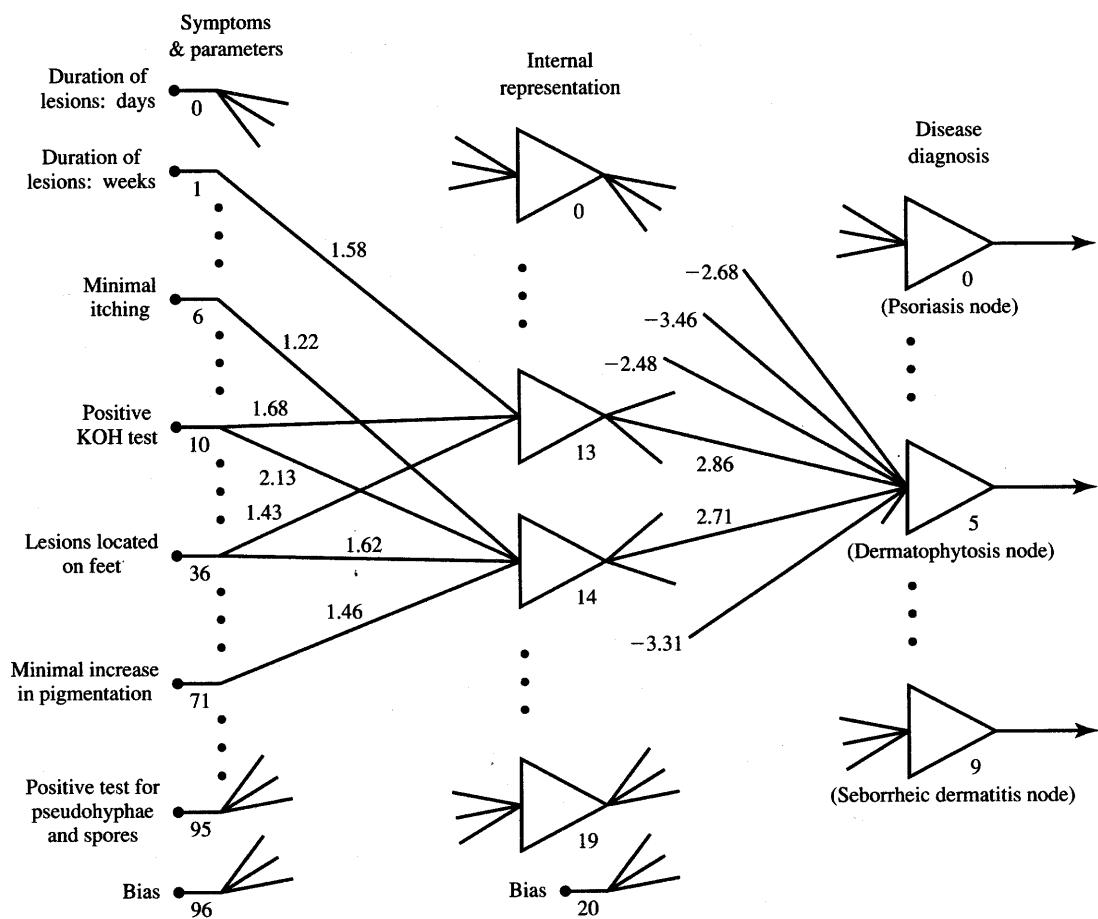
An example of the explanation potential of DESKNET is shown in Figure 8.45. The figure shows only the weights that are of relative importance for diagnosing a dermatophytosis disease. It can be seen that, although unnamed, internal factors numbered 13 and 14 are the two strongest in causing disease number 5. Thus, for this example case, we have  $k_1 = 5$ ,  $j_1 = 13$ ,  $j_2 = 14$ , and  $l = 2$ . Relatively large input weights now need to be found for each of the hidden nodes  $j_1, j_2, \dots, j_l$ . Input nodes 1, 10, and 36 and 6, 10, 36, and 71 were identified as those that are affecting the internal factors of 13 and 14, respectively. Consequently, symptoms and their parameters numbered 1, 6, 10, 36, and 71 were found to be of dominant importance for the diagnosis of dermatophytosis.

Below we explain the methodology that has been used for identification of relatively large and contributive weights and symptoms in DESKNET. To determine internal decision factors, the largest and smallest weights of the output layer have been selected among all weights  $w_{k_1 j_i}$ ,  $j = 1, 2, \dots, J$ , for a given diagnosis  $k_1$ . The maximum difference between the largest and smallest weight constitutes the weight range. The range is then divided by 6 to estimate the standard deviation of weights. The doubled standard deviation is used as the cutoff point for discrimination of relatively large weights. For the data of Figure 8.45 with diagnosed disease 5, or  $k_1 = 5$ , weights with the largest magnitudes are  $w_{5,13} = 2.86$ ,  $w_{5,14} = 2.71$ ,  $w_{5,2} = -2.68$ ,  $w_{5,6} = -3.46$ ,  $w_{5,10} = -2.38$ , and  $w_{5,17} = -3.31$ . The range for this set of weights is

$$w_{5,13} - w_{5,6} = 6.32$$

This yields the standard deviation of approximately 1.05, and the cutoff points equal to  $\pm 2(1.05) = \pm 2.1$  for selection of relatively large weights. Therefore, only two weights,  $w_{5,13}$  and  $w_{5,14}$ , are found to support the decision of node  $k_1 = 5$ . The weights are shown in the figure and other weights are omitted. This analysis performed for the output layer needs to be repeated by using hidden nodes 13 and 14 as decision nodes. The full probability profile obtained for the diagnosis path of dermatophytosis is illustrated in the figure.

The discussion indicates the presence of the following symptoms and their parameters for this particular disease: lesions of weeks of duration, minimal



**Figure 8.45** Explanation of dermatophytosis diagnosis using the DESKNET expert system.  
[Adapted from *Journal of Neural Network Computing*, New York: Auerbach Publishers, © Warren, Gorham & Lamont, Inc., used with permission, from Yoon et al. (1989).]

itching, positive KOH test, lesions on feet, minimal increase in pigmentation, and microscopic evaluation for pseudohyphae. The profile produced by this methodology rather closely matches the probability profile provided by the domain expert. Similar probability profiles can be obtained for the negative responses of the hidden nodes numbered 2, 6, 10, and 17 (not shown on Figure 8.45). These profiles indicate that absence of certain symptoms contribute to the positive diagnosis of a particular disease.

It should be noted that the discussed method of explanation extraction from the expert system is rather heuristic and somewhat inconclusive. Indeed, factors influencing the decision are usually distributed within the network in a complex,

superimposed way. Thus, only fairly general and somewhat tentative conclusions can be drawn from the search for the largest and smallest weights connected to the neurons with the strongest responses within the layer.

---

### Expert System for Low Back Pain Diagnosis

The neural network-based expert system described in this section is used for low back pain disease diagnosis (Bounds et al. 1988). This expert system is somewhat better validated in practice and less experimental than DESKNET. After training, the system is able to provide correct diagnosis more often than any of the three groups of doctors or the fuzzy logic system when they are used for comparisons on the same data. This project offers an interesting and suggestive example of the useful performance of a neural network in an actual diagnosis application.

Back pain is one of the most common complaints encountered by doctors, and yet the diagnosis of the cause underlying the back pain is difficult. Common symptoms are often found in patients with serious spinal problems and in those with much less serious problems. In addition, nonorganic symptoms make the diagnosis of this disease more difficult.

For the purpose of patient treatment the back pain needs to be classified into four categories as follows (Bounds et al. 1988):

- Simple low back pain (SLBP)—simple backache
- Root pain (ROOTP)—nerve root compression due, for example, to disk prolapse
- Spinal pathology (SPATH)—due to tumor, inflammation, or infection
- Abnormal illness behavior (AIB)—back pain with significant psychological overlay.

Patients with AIB need to be separated from the population due to the difficulties of interpretation of clinical findings in patients of this group. Further comprehensive assessment is needed for such patients before a more detailed diagnosis is provided.

The set of patient data used referred to 200 patients with back pain. Lists of symptoms and medical history data were compiled for each patient for the purpose of producing the training and test sets. To ensure that the patients' problems were truly representative of one of the four classes, patients were followed up over a long period. Fifty examples of each class were used; 25 formed the training set and the remaining formed the test set for each class.

The input data were coded into 50 inputs in the unipolar values range [0, 1]. Many of the symptoms and medical history questions were responded to with

"yes/no" answers and they were assigned binary 1/0 input values, respectively. The input was coded 0.5 where none of the answers was applicable. Answers to a few questions that were not binary were coded with the input values between 0 and 1. Unipolar continuous neurons arranged first in a single- and then in a double-hidden layer architecture were used for experiments. Both local and distributed representations were tested. Although local representation with a separate output neuron for each class worked reasonably well, better results were obtained for distributed coding as follows:

$$\mathbf{d} = \begin{cases} [0 \ 0]^t & \text{for SLBP} \\ [0 \ 1]^t & \text{for AIB} \\ [1 \ 0]^t & \text{for ROOTP} \\ [1 \ 1]^t & \text{for SPATH} \end{cases}$$

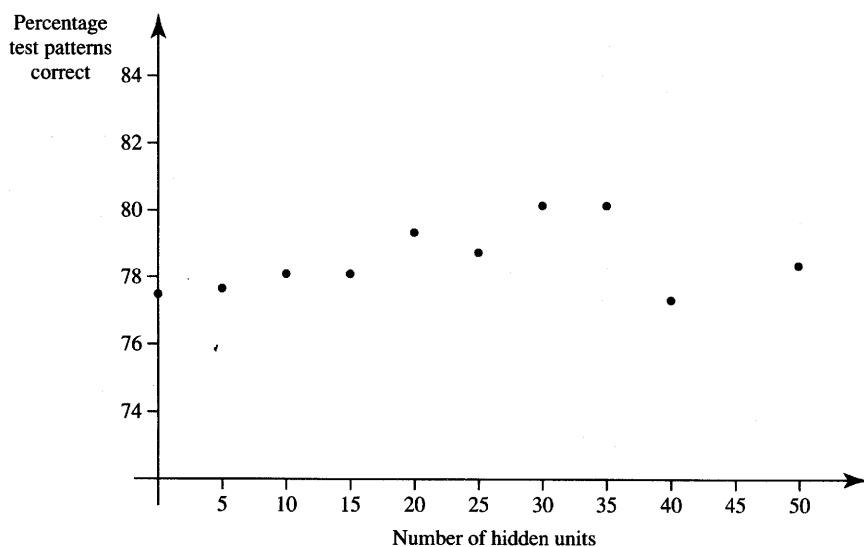
This coding improved the chances of a less serious cause (SLBP) being incorrectly attributed as the most serious cause (SPATH), and vice versa.

The training was implemented for the learning rate of 0.1 and with the momentum term of 0.9. Somewhere between 10 and 40 training runs were carried out, each starting from different initial random weights for each architectural configuration tested. A different number of hidden neurons were tried, ranging from 0 to 50 in a single hidden layer. Also, various sizes of double hidden layers were evaluated. Subsequent to training, feasible network architectures were selected. Final tests were made on the test set with a thresholding value of 0.5 for the output neurons so that they provided binary decisions. All units of the trained networks were fully connected.

The results were found to be somewhat insensitive for the number of single hidden layer units if it was selected between 0 and 50. The average success rate for test runs is depicted in Figure 8.46. Ten runs for each tested architecture are included in the average shown. Markedly, the large dimensionality of the input vector caused only a marginally worse decision rate for a single-layer network ( $J = 0$ ) compared to a single hidden-layer network containing dozens of hidden neurons. Networks with double hidden layer have shown no practical advantage over the results illustrated in the figure for a single hidden layer. This conclusion, however, remains valid only for this particular problem, network input/output size, and the data used to train it.

To evaluate the performance of the developed expert system, the set of test data on symptoms and medical history of patients was given to both general practitioners and specialists. In addition, the set was used for testing a CAD system based on fuzzy logic (Bounds et al. 1988; Norris 1986). Comparison of performance has been made for the following diagnosis methods:

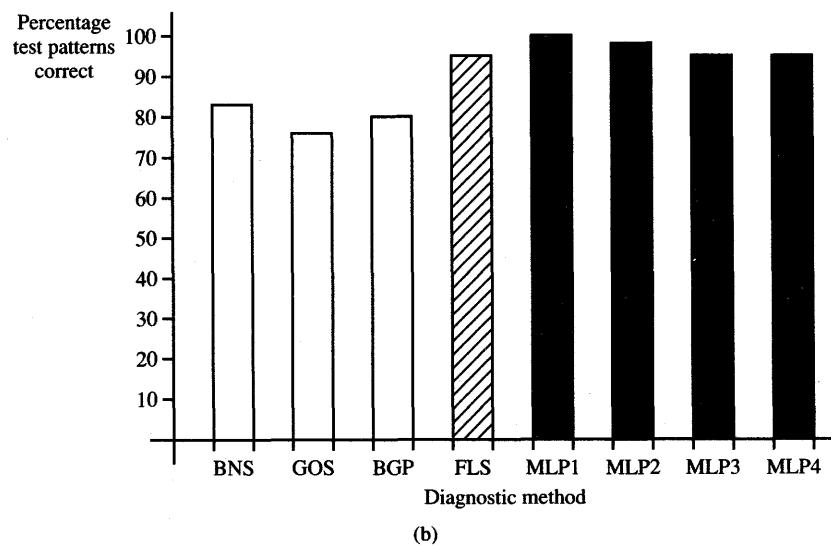
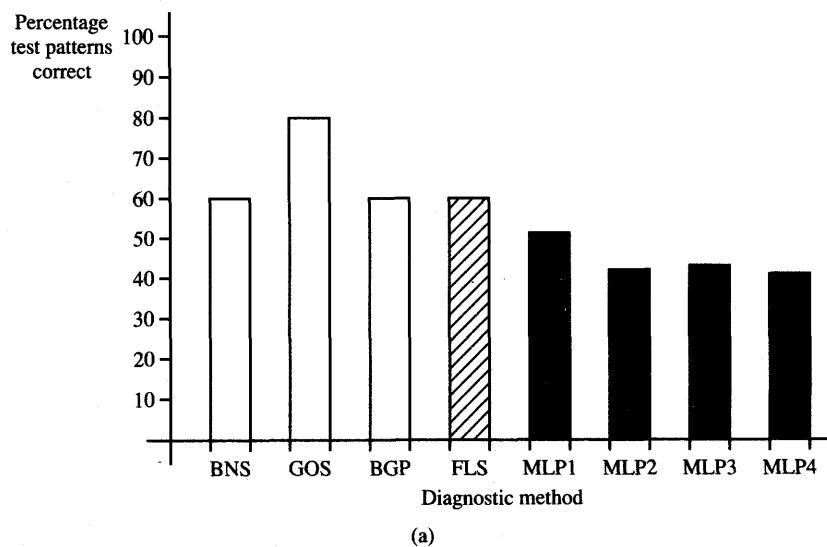
- BNS—Bristol neurosurgeons
- GOS—Glasgow orthopaedic surgeons



**Figure 8.46** Test set performance of the back pain diagnosis expert system using a single hidden layer network with  $J$  hidden nodes. [©IEE; adapted from Bounds et al. (1988) with permission.]

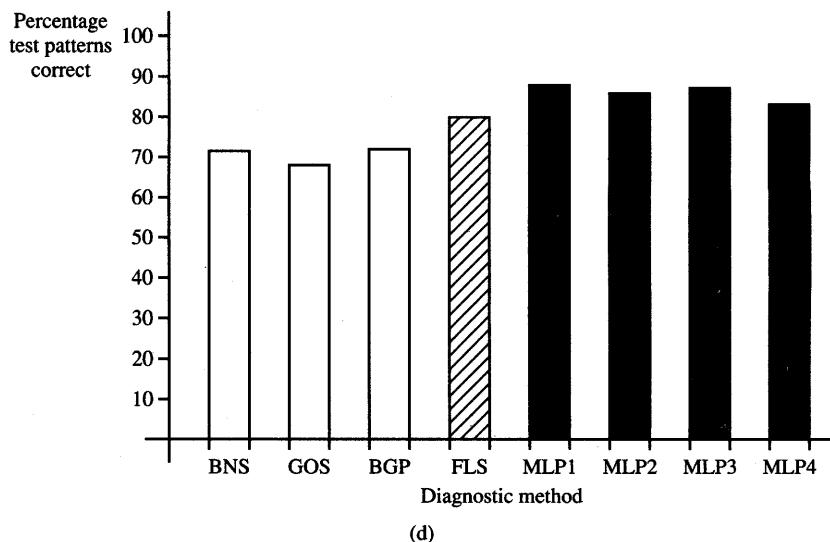
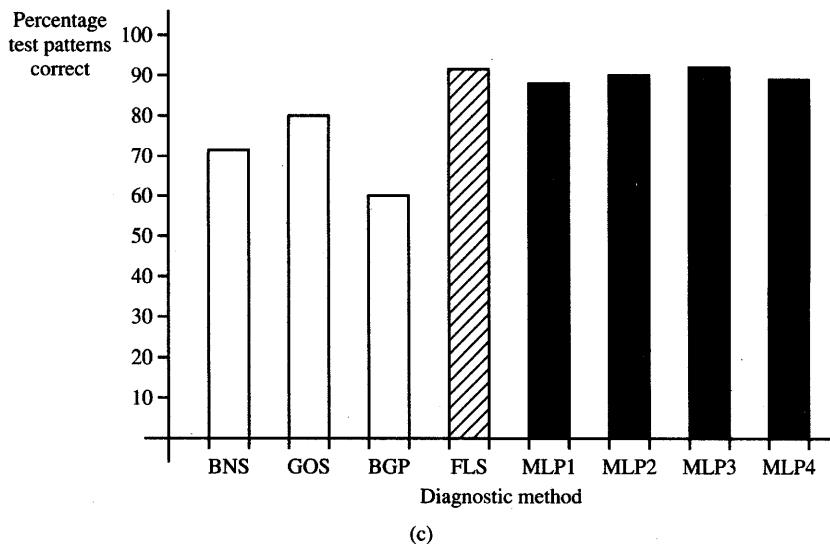
- BGP—Bristol general practitioners
- FLS—Fuzzy logic system (CAD)
- MLP1—Best multilayer perceptron with architecture 50–30–2
- MLP2—Mean of 10 MLP runs for 50–30–2 network
- MLP3—Best multilayer perceptron with architecture 50–0–2
- MLP4—Mean of 10 MLP runs for 50–0–2 network.

The results for each of the four diagnoses are summarized as bar charts in Figure 8.47. The results indicate that the overall performance of the multilayer perceptron network (black bars) exceeds that of three groups of doctors (blank bars). Also, the multilayer perceptron network reaches the level of performance of the fuzzy logic system (shaded bar). For the most serious disease, spinal pathology, the neural expert system outperforms the other methods as shown in Figure 8.47(d). It can be seen that any of the four selected architectures yields an equal or better diagnosis than each of the three teams of doctors and the fuzzy logic system. This outcome of the diagnosis is of special importance since undetected spinal pathology may have serious implications for a patient's health. On the other hand, the neural expert system performed much worse than any other method for diagnosis of simple low-back pain as illustrated in Figure 8.47(a).



**Figure 8.47a,b** Comparison of diagnostic methods for back-pain diagnosis: (a) simple low back pain and (b) abnormal illness behavior. [©IEEE; reprinted from Bounds et al. (1988) with permission.]

In the opinion of the authors of the project, who are pattern recognition specialists and medical school faculty, the neural network could perform even better than reported provided a larger number of representative training examples were



**Figure 8.47c,d** Comparison of diagnostic methods for back-pain diagnosis (*continued*): (c) root pain, and (d) spinal pathology. [©IEE; reprinted from Bounds et al. (1988) with permission.]

used. Insufficient examples which do not span the entire spectrum of possible input patterns may have caused certain features of patterns not to be encoded in the network during the reported training experiment.

---

### Expert System for Coronary Occlusion Diagnosis

Acute myocardial infarction, also called coronary occlusion, is an example of another serious disease that has been difficult to diagnose accurately. A considerable number of methodologies have been developed in attempts to improve on the diagnosis accuracy of predicting acute myocardial infarction. The best of the diagnostic approaches using conventional computer-aided diagnostic systems has performed with a detection rate of 88 percent and a false alarm rate of 26 percent. These prediction rates are comparable to those by physicians, reported as equal to 88 and 29 percent for detection and false alarm rates, respectively (Baxt 1990).

A multilayer unipolar perceptron network trained using the error back-propagation technique was applied to the prediction of acute myocardial infarction in patients admitted to the emergency room with acute anterior chest pain. Because most patients in this population are not suffering from acute myocardial infarction, a subset of patients with a much greater probability of the disease was chosen for the study. Moreover, only patients later admitted to the coronary care unit were included in the evaluation. Understandably, only for these patients, the full present and retrospective data were available for analysis (Baxt 1990) and subsequent neural network training.

Consider that the present coronary care unit data confirm or rule out the presence of infarction. These data were presented as desired output and coded as 0 or 1 for an absent or present infarction, respectively. The retrospective data were clinical variables collected in the emergency department and submitted as input patterns to the network. The full list of predictive input variables for the diagnosis of coronary occlusion may include as many as 41 items. The tests performed on the neural network indicated, however, that only 20 of these variables were relevant for representing essential retrospective diagnostic data. The variables are tabulated in Figure 8.48(a). The inputs to the network were generated by a specially written program that coded most of these clinical variables into binary data 0 and 1 for the absence or presence of a finding.

The multilayer feedforward perceptron network that yielded the best results in the project has 20 input nodes, 1 output node, and two hidden layers consisting of 10 neurons each. The data involved a population of 356 patients randomly divided into two equal and permanent groups. In each of the groups there were 118 patients without acute myocardial infarction and 60 patients who sustained the disease. The network training was simulated starting at random initial weights with a learning rate of 0.05 and momentum term of 0.9.

The data on the first group of patients were first used to train the network, while the data on the second group were used to perform a test of the diagnostic expert system. It was found during tests that the output neuron indicating the

History	Past History	Examination	Electro-Cardiogram Findings
Age	Past AMI	Jugular venous distention	2 mm ST elevation
Sex	Angina		1 mm ST elevation
Location of pain	Diabetes	Rales	ST depression
Response to nitroglycerin	Hypertension		T wave inversion
Nausea and vomiting			Significant ischomic change
Diaphoresis			
Syncope			
Shortness of breath			
Palpitations			

(a)

Network Prediction	Absence of infarction		Infarction	
	cases of 236	%	cases of 120	%
Correct	226	95.7	111	92.5
Incorrect	10	4.3	9	7.5

(b)

Figure 8.48 Connectionist expert system for coronary occlusion diagnosis: (a) input variables and (b) test data performance summary. [Adapted from Baxt (1990)]

disease always exceeded 0.8, and the output indicating its absence was below 0.2. The network correctly diagnosed 55 of 60 test patients with infarction and 113 of 118 test patients without infarction. Secondly, the training and test sets were swapped and the training and test sessions repeated. The newly obtained network correctly diagnosed 56 of 60 test patients with infarction and 113 of 118 test patients without infarction on the new test data.

Figure 8.48(b) summarizes the network performance achieved in this project. The analysis of results reveals that the connectionist expert system has a disease detection ratio above 92 percent among sick patients. However, among healthy patients the system provides false alarms of disease in 4.3 percent of the cases. In spite of the errors, these results outperform the best previously reported diagnostic schemes, including trained physicians, which yield results of 88 and 26 percent, respectively.

Although the results presented above are very encouraging, further study needs to be made. Specifically, to draw conclusive comparisons, the experiments with different diagnosing techniques would have to use the same set of clinical and diagnostic data. Further, the study involved only patients admitted from the

emergency department to the coronary care unit, which could have had consequences in terms of disease development. Despite the preliminary scope of the study, if the reported results hold up to further scrutiny, the connectionist expert system for infarction diagnosis could have a substantial impact on the reduction of health care costs.

---

### Concluding Remarks

The three medical diagnosis expert systems presented in this section are able to store the model patient knowledge base. Moreover, the connectionist methodology to store knowledge can substantially ease the task of building an expert system. It is no longer necessary for a knowledge engineer to have a thorough understanding of implicit rules, links, and relations within the domain. Moreover, the amount of time for system development is substantially reduced compared to the systems that use either a tree structure of rule-based relationships or statistical calculations.

Although the performance achieved by the example expert systems discussed is impressive, the systems built and tested should be considered to be preliminary and tentative (Jones 1991). Although they compare rather advantageously and sometimes are even clearly superior to human experts, more detailed and exhaustive study is needed. Such a study would allow better assessment of all capabilities and limitations of connectionist methodology for expert systems applications. Also, a more comprehensive and versatile approach to the retrieval of the decision explanations of such expert systems is needed. It is almost certain that these and other relevant questions shall be explored by knowledge scientists in the near future.

## 8.6

---

### SELF-ORGANIZING SEMANTIC MAPS

The semantic maps discussed in this section are implemented based on the self-organizing feature map concept described in Sections 7.4 and 7.5. This application example demonstrates that, in addition to processing quantitative data for feature extraction, the maps make it possible to display abstract data, such as words, arranged in topological neighborhoods. The maps extract semantic relationships that exist within the set of language data considered here as a collection of words. The relationships can be reflected by their relative distances on the map containing words positioned according to their meaning or context. This indicates that the trained network can possibly detect the logical similarity between words from the statistics of the contexts in which they are used. The

context is understood here as an element of a set of attribute values that occur in conjunction with the words. In another approach, the context can be determined by the frequency of neighboring words without regard to the attribute values that are occurring. The discussion below is based on experiments and results described by Ritter and Kohonen (1989) and Kohonen (1990).

The self-organizing feature network consists of a number of laterally interacting adaptive neurons, usually arranged as a two-dimensional array. Although each array neuron is connected to each input node, only neighborhood neurons respond, after training, to specific inputs. The planar position of the excitation specifies a mapping of the input pattern onto the two-dimensional topographic map. The map represents the distance relations within the original high-dimensional input space of input patterns compressed to the two-dimensional image space represented by a neural array. This remarkable property follows from the assumption of lateral interactions within the array and from the assumed weight adaptation law (7.28).

Let us restate that the resulting maps are nonlinear projections of the input space onto the feature array. The projection preserves the distance relationships between the patterns in the original multidimensional pattern space. Such implicit distances become explicit planar distances in low-dimensional image space called here the feature space. However, a mapping from a high-dimensional space to a low-dimensional one preserves only the most important neighborhood relationships within the input pattern. The less relevant properties are ignored. Moreover, the inputs that appear more frequently are mapped to larger and stronger domains at the expense of less frequent inputs. Data that are close, or similar, in the input space and appear frequently at the input are mapped into a cluster of images forming localized images on the map. Furthermore, the clusters of images are also arranged into topological neighborhoods that display the overall relationships within the entire input data set.

Extraction of features from geometrically or physically related input pattern data is usually a concrete task, as demonstrated in example of Figure 7.19. This and other self-organizing feature map examples of Chapter 7 have made use of the numerical values of data in the input space. Much more abstract data need to be processed, for example, in the case of cognitive and linguistic relations that may exist in the input space. Such processing reflects relationships that neural networks could extract for linguistic representations and relations. Self-organizing maps that display semantic relations between abstract language data are called *semantic maps* (Ritter and Kohonen 1989).

The most general concepts or abstractions needed to interpret the empirical world are called *categories*. They form an elementary lexicon embracing the whole domain of knowledge. They are commonly used for thinking and communication. The most common categories are (1) items (objects), (2) qualities (properties), (3) states (or state changes), and (4) relations (spatial, temporal, or other). The categories are formed on the basis of consciousness, and their typical

counterparts in most languages are (1) nouns, (2) adjectives, (3) verbs, and (4) adverbs, prepositions, etc., but also order of words (syntax). Each category usually includes many subcategories such as persons, animals, and inanimate objects that are usually found within each category of items.

Difficulty results when trying to express the human language as neural network input data as opposed to nonsymbolic input data encoding. These data can be represented as continuous or discrete values arranged into a vectorial variable. The difficulty is due to the fact that for symbolic objects, such as words, no adequate metric has been developed. In addition, except for a few special words denoting sounds, the meaning of a word is usually disassociated from its encoding into letters or sounds. Moreover, no metric relations exist whatsoever between the words representing similar objects. To overcome the difficulties mentioned, we will therefore try to present the coded value for each symbol word expressed in appropriate context containing values of attributes.

One simple model for contextual representation of a symbol and its attribute enabling the topographical mapping on the semantic map is to concatenate the data vector as follows:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{x}_a \end{bmatrix} \quad (8.33)$$

where the symbol and the attribute part of the data vector are expressed as column vectors  $[\mathbf{x}_s \ 0]^t$  and  $[0 \ \mathbf{x}_a]^t$ , respectively.

We assume here that the symbol and attribute vectors are orthogonal. To reflect the metric relationship of the attribute sets, the norm, or the vector length, of the attribute vector is chosen as dominant over the norm of the symbol vector during the self-organizing process. Since both symbol and attribute part are included in each training input, however, topological neighborhoods are created by symbols and their attributes. The symbols can become encoded into a planar array so as to reflect their logic similarities. We would expect that the recall of only the symbol part with the attribute part of the data vector missing would provide us with the calibrated topological map of symbols. Such a map would show implicit logical similarities that potentially exist within the set of symbols in the form of geometrically explicit distances between symbols.

Assuming the local representation of the symbols, such that the  $i$ 'th symbol in the set of  $P$  symbols is assigned a  $P$ -dimensional symbol vector  $\mathbf{x}_{si}$  whose  $i$ 'th component is equal to a fixed value of  $c$ , and all remaining components are zero. We thus have the symbol vectors of value

$$\begin{aligned} \mathbf{x}_{s1} &= [c \ 0 \ \dots \ 0]^t \\ \mathbf{x}_{s2} &= [0 \ c \ \dots \ 0]^t \\ \mathbf{x}_{sP} &= [0 \ 0 \ \dots \ c]^t \end{aligned} \quad (8.34)$$

The attributes are present or absent for each of the  $P$  symbols. The attribute

		dove	hen	duck	goose	owl	hawk	eagle	fox	dog	wolf	cat	tiger	lion	horse	zebra	cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	mane	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8.49 Table of attributes for the set of  $P = 16$  animal objects. [© Springer-Verlag; reprinted from Ritter and Kohonen (1989) with permission.]

column vector  $\mathbf{x}_a$  contains the number of entries equal to the number of relevant attributes allocated to the set of symbols under consideration. The presence, or absence, of a particular attribute is indicated by a binary entry 1 or 0, respectively. Note also that the unnormalized similarity measure of two symbols is the scalar product of their respective attribute vectors.

Figure 8.49 illustrates a specific example input data case used for simulation by Ritter and Kohonen (1989). For  $P = 16$  elements in the set of symbols being animals, the attribute vectors are specified by the attribute column vectors of the table shown. An example data vector for a cow is given as

$$\mathbf{x} = [\mathbf{x}_{s16} \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ \dots \ 0]^t \quad (8.35)$$

where  $\mathbf{x}_{s16} = [0 \ \dots \ c]^t$  and  $\mathbf{x}$  consists of  $n = 29$  entries.

Let us note that the norm of the difference vector between each pair of symbol vectors is identical and equal to  $\sqrt{2}c$ . The numerical value of  $c$  is the norm of each symbol vector and it determines the relative weight of the symbol vector as compared to its attribute counterpart vector within the 29-entry data field. Input data vectors shown in the figure and used for training have been normalized to the unity length. To draw a comparison, this may correspond to the biological intensity normalization of the incoming stimuli.

Sixteen data vectors containing a symbol (object) part concatenated with its attribute part as displayed in Figure 8.49 have been used to produce a semantic map of the set of animals. The data vectors have been used to train a  $10 \times 10$  planar array of neurons. The initial weights have been selected of random values so that no initial ordering was imposed. After about 2000 training input

presentations, the neighborhood sensitivity has been ascertained. The neurons' responses became consistently stronger or weaker in certain regions. This has been learned due to either of the excitations  $\mathbf{x} = [\mathbf{x}_{si} \ 0]^t$ , or  $\mathbf{x} = [\mathbf{0} \ \mathbf{x}_{ai}]^t$ . The neurons with the strongest response due to the symbol vector excitation only (attribute part missing) are shown in Figure 8.50(a). They are labeled using the symbol that elicits the particular response. The neurons represented by dots indicate their nondominant responses on the map of symbols.

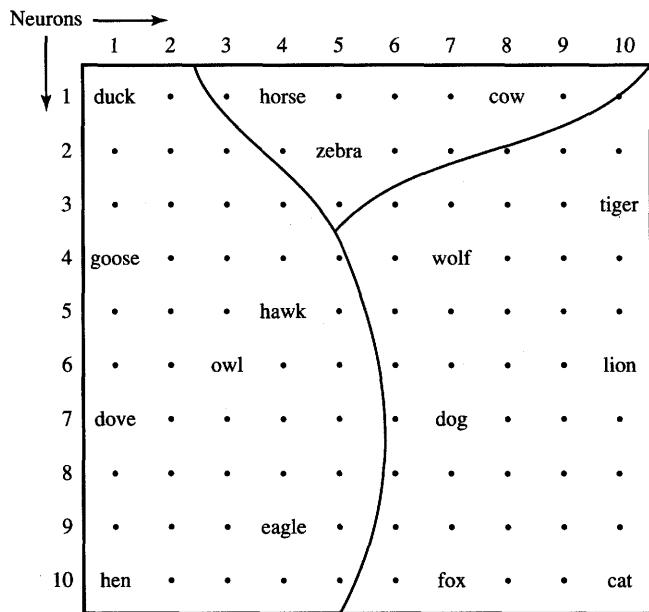
The spatial order of the responses has emerged on the semantic map as a result of the training. The order divides the symbols into three distinct groups of birds (left), hunters (right), and grass-eating animals (top). Within each group, certain similarities are also recognizable. "Dog" is located predominantly between "fox" and "wolf." It is somewhat less related to "cat" and "lion" and even less similar to hunting birds, however, it does not seem to be related at all to nonhunting birds such as a "duck" or "goose." This is also visible from Figure 8.50(b) showing the strongest response domains. Each neuron of this map is marked with the stimulus eliciting its strongest response.

The semantic maps of objects displaying logical distances between the animal symbols have been produced in this experiment using the static context of the relevant attribute vector. In language use, however, objects and their attributes often occur in temporal sequences. For a more comprehensive evaluation of the potential of self-organizing semantic maps, the concept of context needs to be broadened and to include the time domain. In the discussion to follow, the time variable present in semantic data will be represented as an order of word occurrence in their serial sequence.

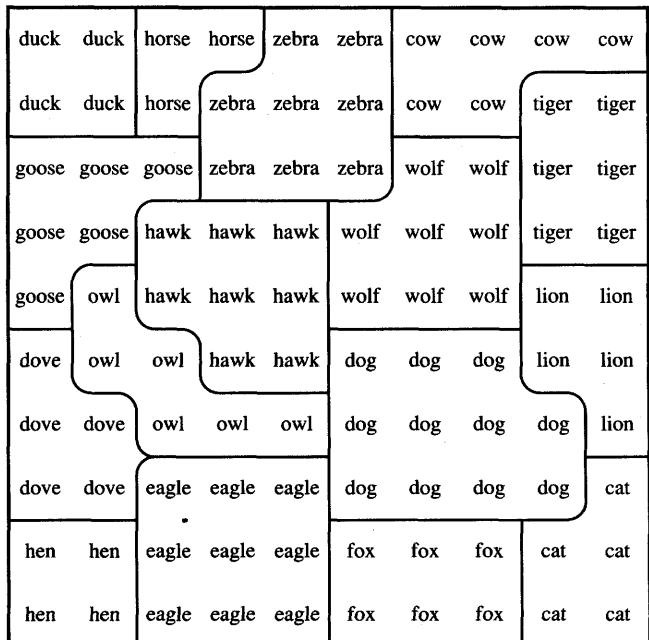
Let us look at a mapping of a set of 30 test words listed as shown in Figure 8.51(a). The test set consists of nouns, verbs, and adverbs. The set is subdivided into 14 pairs or triples of related words. Let us attempt to build sensible sentence patterns on this set. Three-word sentences created from this semantic map-building lexicon and allowed in this experiment have to obey some rudimentary rules of grammar and semantic correctness.

Thirty-nine basic sentence patterns as shown in Figure 8.51(b) have been produced based on such rules. Test sentences are then created by plugging each member of the pair, or the triple, into the sentence patterns. In this mode, the basic sentence pattern 1-5-12, or "Bob runs fast" generates test sentences "Jim runs fast," "Mary runs fast," but also "Bob walks fast," etc. This way the total of 498 grammatically and semantically correct test sentences as in Figure 8.51(c) have been produced. The temporal context of a word in the experiment is made by the immediate predecessor and successor word. To build pairs for the first and the last word in the sentence, the randomly generated sentences were concatenated in the order in which they were produced.

Each word from the 30-word dictionary involved in the experiment can be defined by a vector with 30 entries with only one of them having a nonzero value. This would lead, however, to a rather sparse description of the input data.



(a)



(b)

**Figure 8.50** Semantic self-organizing symbol map obtained through training using attributes: (a) strongest responses due to the symbol part only excitation and (b) strongest response domains. [©Springer-Verlag; reprinted from Ritter and Kohonen (1989) with permission.]

		Sentence patterns:			
Bob/Jim/Mary	1				Mary likes meat
horse/dog/cat	2	1-5-12	1-9-2	2-5-14	Jim speaks well
beer/water	3	1-5-13	1-9-3	2-9-1	Mary likes Jim
meat/bread	4	1-5-14	1-9-4	2-9-2	Jim eats often
runs/walks	5	1-6-12	1-10-3	2-9-3	Mary buys meat
works/speaks	6	1-6-13	1-11-4	2-9-4	dog drinks fast
visits/phones	7	1-6-14	1-10-12	2-10-3	horse hates meat
buys/sells	8	1-6-15	1-10-13	2-10-12	Jim eats seldom
likes/hates	9	1-7-14	1-10-14	2-10-13	Bob buys meat
drinks/eats	10	1-8-12	1-11-12	2-10-14	cat walks slowly
much/little	11	1-8-2	1-11-13	1-11-4	Jim eats bread
fast/slowly	12	1-8-3	1-11-14	1-11-12	cat hates Jim
often/seldom	13	1-8-4	2-5-12	2-11-13	Bob sells beer
well/poorly	14	1-9-1	2-5-13	2-11-14	(etc.)

(a)

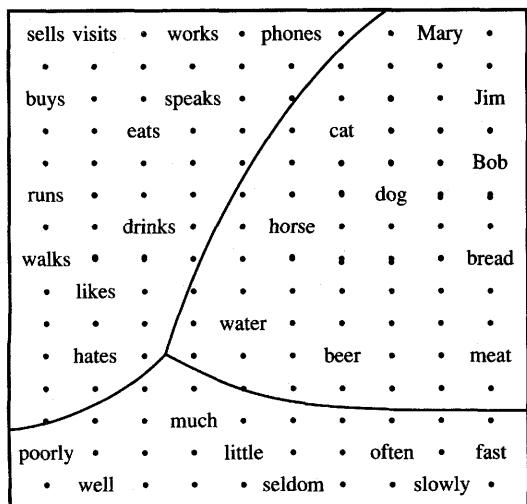
(b)

(c)

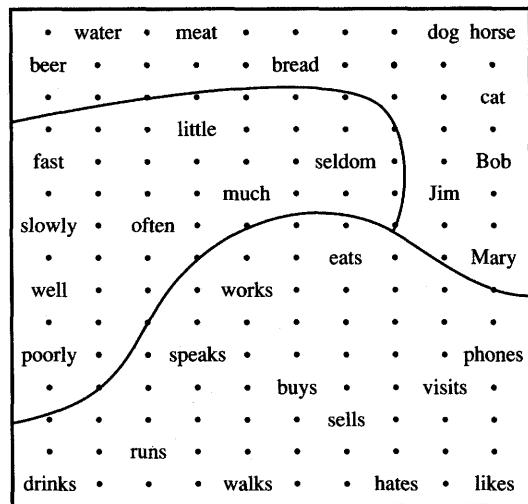
**Figure 8.51** Words and sentences used for context production: (a) nouns, verbs, and adverbs, (b) sentence patterns, and (c) selected examples of sentences. [©Springer-Verlag; reprinted from Ritter and Kohonen (1989) with permission.]

For more economical encoding, a 7-tuple vector of unit length has been used (Ritter and Kohonen 1989). Such 7-tuple vectors have been generated at random for each word in the experiment, thus ensuring quasi-orthogonality of different word vectors. It follows then that the word in the context can be expressed using a single vector of 21 entries, where 14 entries represent the predecessor and successor context words. To perform training more efficiently, the average context of each word under consideration has been used instead of step-by-step learning with each sentence applied separately. Ten thousand sentences having all vectors have been used for averaging. The average word contexts were represented as 14 entries with word vectors scaled to unit length. They have been given a role similar to the attribute vector  $x_a$  described in the preceding simulation.

Each of the 7-tuple symbol-word vectors was scaled to the length  $c = 0.2$  and combined with the 14-tuple average context vector. As a result, 21 scalar input data have been supplied as inputs to the array. A planar array of  $10 \times 15$  neurons with 21 input nodes and weights randomly initialized prior to the training has been used for this semantic map design. Each of the 21 inputs has been connected to each of 150 neurons. After 2000 training presentations, the responses to symbol words alone have been tested. As can be seen from Figure 8.52(a), the training contexts have positioned the words in such an order that their mutual arrangements reflect existing semantic and grammatical relationships. A separate grouping for nouns, verbs, and adverbs has resulted, as marked by the partitioning curves. Each of the three domains clustering parts of speech has discernible subregions. For instance, names of persons or adverbs of opposite meaning tend to cluster together.



(a)



(b)

**Figure 8.52** Semantic maps of words obtained by using their mutual context for training:  
 (a) preceding-succeeding word context and (b) preceding word context. [©Springer-Verlag;  
 reprinted from Ritter and Kohonen (1989) with permission.]

In a related semantic map generation experiment, the word context was limited to the immediately preceding word only. Even with that limited context, a map showing similar distance relationships has resulted as shown in Figure 8.52(b). It can be seen that even with the restricted context, a meaningful semantic map can be produced.

These feature extraction experiments have shown the extension of self-organizing neural arrays to processing of expressions that are of symbolic nature and display rather subtle and even evasive relationships. The simulations of semantic structures have resulted in semantic neighborhood maps that encode words into relative planar regions of localized responses. As seen, generated semantic maps allow the segregation and grouping of symbolic data based on an unsupervised learning approach. The topographic organization of concepts as discussed above may be offering us some clues as to why the information processing in the brain is spatially organized and distributed.

## 8.7

### CONCLUDING REMARKS

The presentation focus of this chapter has been on neurocomputing engineering projects. Our concern has been analyzing and designing sample information processing systems that are nonprogrammable. We followed a fundamentally new and different information processing paradigm that is based on developing

associations, mapping, or transformations between data in response to the environment. To achieve these objectives, we essentially pursued nonalgorithmic paths for solving specific engineering problems.

Well above a dozen representative engineering application problems have been explored in this chapter. Application examples have been selected so that they reinforce and illustrate basic theoretical concepts discussed in preceding chapters. Projects that seem exciting but are not easily traceable and reproducible have been excluded from coverage. Networks for constrained optimization, for printed and handwritten character recognition, for controlling static and dynamical plants and actuating robot arms, connectionist expert systems for medical diagnosis, and finally networks for processing semantic information have been studied. It is believed that the depth and diversity of the coverage in this chapter has prepared the reader at least for pursuing new application-specific projects, if not for independent research.

Due to the abundance and diversity of technical publications on artificial neural systems applications, many other interesting problem solutions could not be addressed in this book. A number of edited volumes on artificial neural systems (Denker 1986; Anderson and Rosenfeld 1988; Anderson 1988; Touretzky 1989, 1990; Omidvar 1992) and special issues of technical journals (*IEEE Computer*, March 1988; *IEEE Transactions on Circuits and Systems*, June 1990; *Proceedings of the IEEE*, September 1990 and October 1990) include discussion of different applications. In addition, the reader may refer to specific applications from the following partial list of projects:

- Trainable networks for solving matrix algebra problems (Wang and Mendel 1990)
- Reinforcement and temporal-difference learning controller for an inverted pendulum problem (Anderson 1989)
- Truck "backer-upper" neurocontroller and control of complex dynamical plant (Nguyen and Widrow 1990)
- Trajectory generation based on visual perception of mobile robots (Kawato et al. 1988)
- Sensor-based control of robots with vision (Miller 1989; Hashimoto et al., 1990)
- Networks for robot control (Liu, Iberall, and Bekey 1989; Wilhelmsen and Cotter 1990)
- Autonomous land vehicle neurocontroller/driver (Pomerleau 1989)
- Neural system NETTALK for continuous speech recognition (Sejnowski and Rosenberg 1987)
- General review of neural network performance for speech analysis (Lippman 1989)
- Temporal signals recognition networks (Gorman and Sejnowski 1988; Eberhart and Dobbins 1990)

- Inspection of machined surfaces (Villalobos and Gruber 1990)
- Connectionist expert system for bond rating (Dutta and Sekhar 1988).

Typical applications of neurocomputing indicate the ability of neural networks to handle multiple input and output variables, nonlinear relationships, past states, and implement fast network learning when suitable architectures are used. It is important to realize that these are not special, magical, or stand-alone techniques and applications. Rather, they can be looked at as outgrowth of long periods of research in function approximation, optimization, signal processing, pattern classification and decision theory, and stochastic approximation (Anderson 1989; Poggio and Girosi 1990; Tsyplkin 1973). What makes neural network approaches different from conventional engineering is the methodology, specifically, the experimental methodology of learning and problem formulation that underlies neural network methods.

It seems somewhat easy to perceive the free-wheeling nature of many approaches we have taken during our discussion and the absence of theoretical guarantees, derivations, and proofs. Such approaches have often been present in earlier theoretical presentations and perhaps even more in the practical applications of artificial neural systems addressed in this chapter. However, we have made no restrictive assumptions or simplifications such as linearization, convergence, and stability. Interestingly, they have not been needed to accomplish a number of successful artificial neural system designs (Barto 1990).

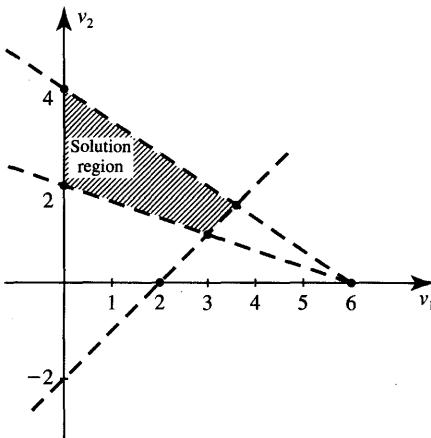
As time passes, new neurocomputing engineering methods and applications will undoubtedly emerge. Despite spectacular successes, one should not be misled, however, about the still existing limitations of neural network technology. Currently neural networks can already be of great value in helping to solve real problems without exactly mimicking many functions performed by biological neural networks. Animals and humans can attain complex combinations of information processing, pattern recognition and feedback, and feedforward transmissions and of cognitive acts. While these very complex facets of neurocomputing may be considered as eventual aspirations for artificial neural systems, they should not be used as criterion for their present value.

## PROBLEMS

Please note that problems highlighted with an asterisk (\*) are typically computationally intensive and the use of programs is advisable to solve them.

*P8.1* The cost function for a linear programming problem is  $c = v_1 + 2v_2$ . The constraints are known as

$$\begin{aligned} v_1 - v_2 &\leq 2 \\ 2v_1 + 3v_2 &\leq 12 \\ v_1 &\geq 0 \\ v_2 &\geq 0 \end{aligned}$$



**Figure P8.2** Linear programming problem of Problem P8.2.

(a) Find the solution of the problem by analyzing the conditions on the solution plane.

(b) Find the weights and currents of the gradient-type neural network similar to that shown in Figure 8.1 that would solve this task.

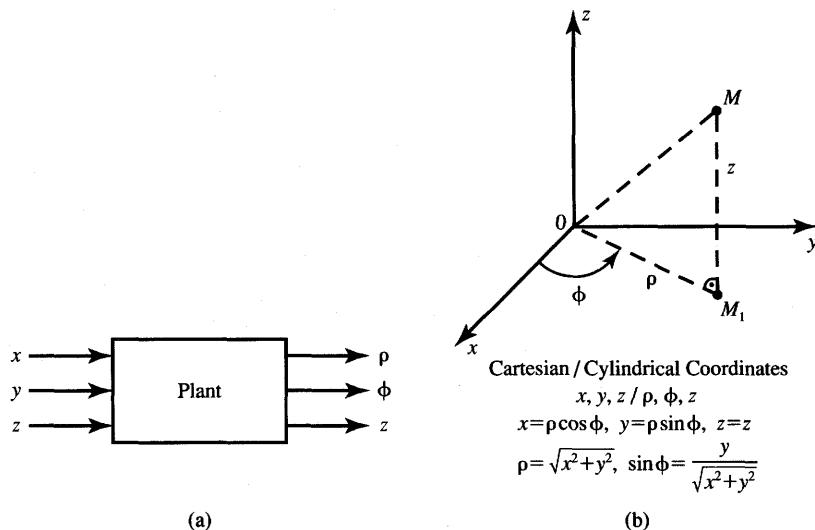
**P8.2** For the linear programming problem with the cost function  $c = 2v_1 + v_2$  and the constraints as shown in Figure P8.2, find

(a) the solution of the linear programming problem.

(b) the weights and currents of the gradient-type neural network similar to that shown in Figure 8.1 that would solve this task.

**P8.3\*** The static plant shown in Figure P8.3 needs to convert Cartesian coordinates  $x$ ,  $y$ ,  $z$ , into cylindrical coordinates  $\rho$ ,  $\phi$ ,  $z$ . The relations between input and output variables are also shown in the figure. Design a multilayer feedforward network that performs the forward identification and models the plant in the forward mode as shown in Figure 8.17(a). The training test points should be selected for the plant outputs in the range  $0 < \rho < 1$ ,  $0 < \phi < \pi/2$ , and  $0 < z < 1$ . Select suitable architecture for the multilayer feedforward network and train the network.

**P8.4\*** Repeat Problem P8.3 for the plant inverse identification by modeling the plant, which converts the Cartesian  $x$ ,  $y$ ,  $z$  coordinates into cylindrical coordinates  $\rho$ ,  $\phi$ ,  $z$  in the same working region. [Hint: Use Figure 8.17(b) and model the inverse of the plant by training a multilayer feedforward network of suitable architecture. Study the mapping accuracy produced by the network by computing the differences between the network response and the accurate response value.]



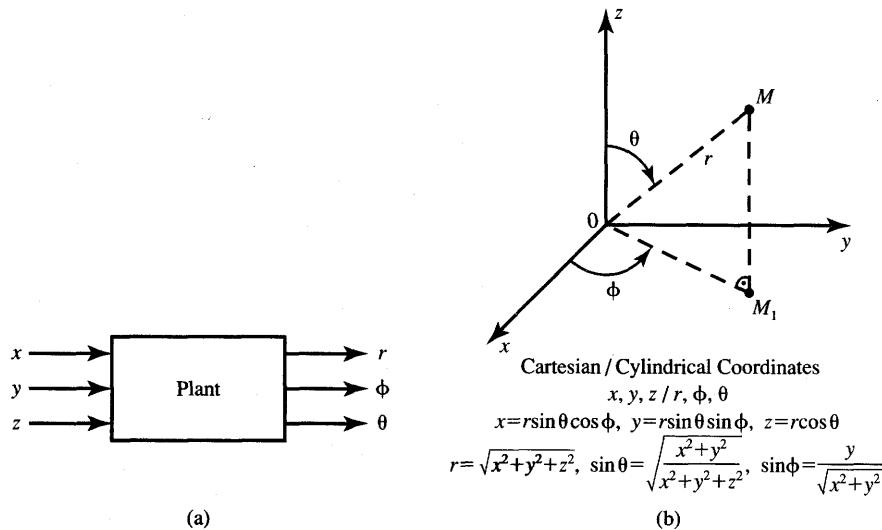
**Figure P8.3** Conversion of Cartesian to cylindrical coordinates: (a) plant and (b) notation and relationships.

P8.5\* The static plant shown in Figure P8.5 converts Cartesian coordinates  $x$ ,  $y$ ,  $z$ , into spherical coordinates  $r$ ,  $\phi$ ,  $\theta$ . The relations between the input and output variables are also shown in the figure. Design a multilayer feedforward network that performs the forward plant identification and models it in the forward mode as shown in Figure 8.17(a). The plant outputs, or training test points, should be selected in the range:  $0 < r < 1$ ,  $0 < \phi < \pi/2$ , and  $0 < \theta < \pi/2$ .

P8.6\* Repeat Problem P8.5 for plant inverse identification for the plant converting the Cartesian  $x$ ,  $y$ ,  $z$  coordinates into spherical ones,  $r$ ,  $\phi$ ,  $\theta$ , in the same working region. [Hint: Use Figure 8.17(b) and model the inverse of the plant by training a multilayer feedforward network of suitable architecture.]

P8.7 Network  $A$  needs to be trained to perform as plant inverse with its copy serving as neurocontroller  $B$ . The learning control system is shown in Figure P8.7(a). The expressions describing the plant are, in general, not known for this control arrangement. In this problem, however, it is assumed that the plant is known to be a Cartesian-to-cylindrical coordinates converter as illustrated in Figure P8.3. For the input  $\mathbf{d}$  specified in Figure P8.7(b), the measurements of  $\mathbf{x}$  and  $\mathbf{o}$  have been obtained as indicated on the figure. Compute

(a) plant output vector  $\mathbf{y}$



**Figure P8.5** Conversion of Cartesian to spherical coordinates: (a) plant and (b) notation and relationships.

- (b) control error  $\|\mathbf{d} - \mathbf{y}\|$  at the plant output
- (c) error  $\|\mathbf{x} - \mathbf{o}\|$  which is used for training of network  $A$ .

Now assume that following the training, both networks  $A$  and  $B$  are ideally in tune with the plant inverse. Compute  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\|\mathbf{x} - \mathbf{o}\|$  for this condition if the neurocontroller input has changed to

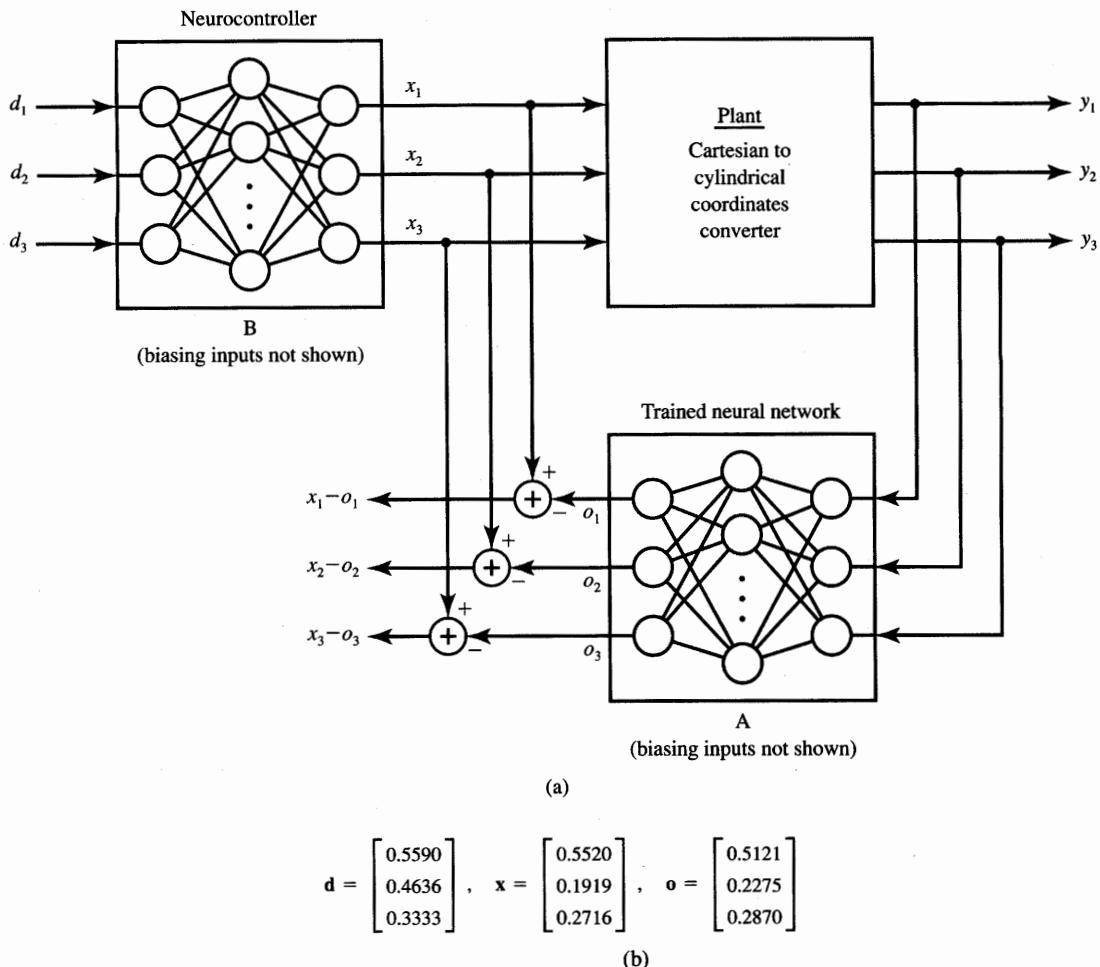
$$\mathbf{d} = \begin{bmatrix} 0.275 \\ 0.725 \\ 0.300 \end{bmatrix}$$

- P8.8** The plant shown in Figure P8.8 is controlled in a specialized on-line learning control architecture. In general, the plant equations  $y_i = y_i(o_1, o_2)$ ,  $i = 1, 2$ , are not known; however, for the purpose of this exercise it is assumed that

$$y_1 = \frac{1}{2}o_1^2 + \sin o_2$$

$$y_2 = o_1 + o_1\sqrt{o_2}$$

- (a) Compute the Jacobian matrix from the analytical description of the plant.
- (b) Knowing that the operating point of the plant is  $o_1^* = 0.472$  and  $o_2^* = 0.125$ , compute the four entries of the Jacobian matrix  $\mathbf{J}$  at

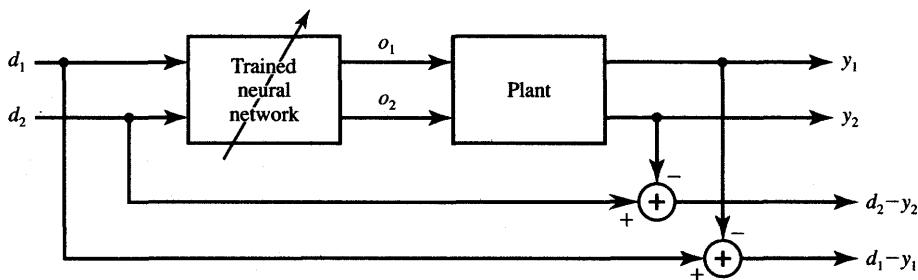


**Figure P8.7** Feedforward control with plant inverse learning: (a) block diagram and (b) results of measurements before completion of training.

this point  $\mathbf{o}^*$ :

$$\mathbf{J}(\mathbf{o}^*) = \begin{bmatrix} \frac{\partial y_1}{\partial o_1} & \frac{\partial y_1}{\partial o_2} \\ \frac{\partial y_2}{\partial o_1} & \frac{\partial y_2}{\partial o_2} \end{bmatrix}$$

- P8.9 The plant of Problem P8.8 is working in the same learning control architecture as shown in Figure P8.8. Compute numerically the approximations



**Figure P8.8** Specialized on-line learning control architecture.

for the Jacobian matrix entries at the quiescent operating point  $\mathbf{o}^*$  given as

$$\mathbf{o}^* = \begin{bmatrix} o_1^* \\ o_2^* \end{bmatrix} = \begin{bmatrix} 0.472 \\ 0.125 \end{bmatrix}$$

using the perturbation method. The disturbed values are to be assumed as deviated by +5 percent from the operating point, thus making the plant signal equal to 1.05  $\mathbf{o}^*$ .

**P8.10** The plant described with the following nonlinear equations

$$\begin{aligned} y_1 &= o_1 o_2 + \frac{1}{\sqrt{o_2}} \\ y_2 &= \sqrt{o_1} + o_1 \sin o_2 \end{aligned}$$

is used in the control arrangement shown in Figure P8.8. Its operating point is

$$\mathbf{o}^* = \begin{bmatrix} 0.721 \\ 0.293 \end{bmatrix}$$

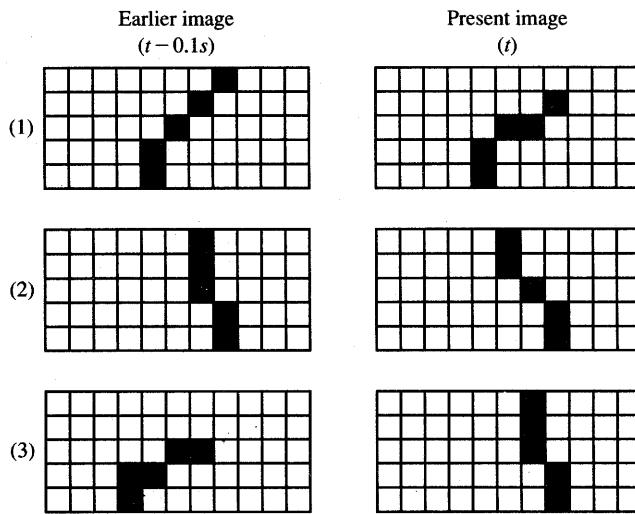
and the desired response for this input is known to be

$$\mathbf{d} = \begin{bmatrix} 1.725 \\ 1.293 \end{bmatrix}$$

Compute numerically

- (a) the plant output  $\mathbf{y}^*$
- (b) the output error  $E = (1/2)\|\mathbf{d} - \mathbf{y}^*\|$
- (c) the relative error values transferred to the plant input, i.e.,  $-\partial E / \partial o_1$  and  $-\partial E / \partial o_2$ , at this operating point.

**P8.11** You need to use common-sense judgment and provide advice to balance the cart-pole system shown in Figures 8.24 and 8.25. Three sample pairs of quantized images are displayed in Figure P8.11. By analyzing each sequence of two images, determine what sign of the horizontal force needs



**Figure P8.11** Three pairs of quantized images of the cart-pole system.

to be applied at present to improve the system's balance. Each of the three cases shown in rows (1), (2), and (3) of the figure should be treated separately. (Note: The force toward the right is positive; the force toward the left is negative.)

- (a) Use your own judgment; assume that you are the teacher and you are trying to train the controller.
- (b) Use the weights of the trained controller to obtain a crude approximation of the discrete perceptron activation value. Inspect Figure 8.25(b) for this purpose and draw conclusions as to the sign of the force needed for balancing.

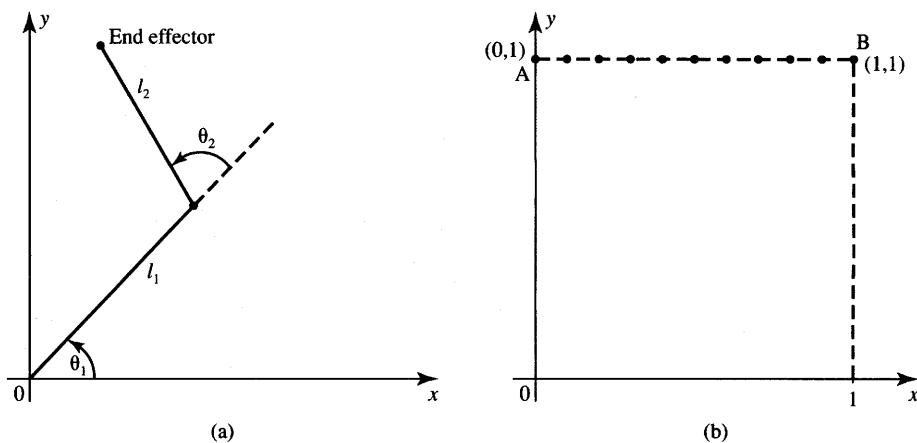
**P8.12** The discrete-time plant is described with the following difference equation:

$$x(k) = 2y(k + 1) - y(k) - 0.5$$

where  $x(k)$  and  $y(k)$  denote input and output, respectively, at time instant numbered  $k$ . The desired output wave of the plant is specified as

$$y(k) = \sin\left(\frac{k\pi}{6}\right) + 1$$

Assuming that a characteristic surface  $x(k) = x[y(k), y(k + 1)]$  needs to be learned by the CMAC neurocontroller, identify the initial sequence of plant control signal values  $x(0), x(1), \dots, x(12)$ , that would be located on the characteristic surface.



**Figure P8.13** Two-link planar manipulators: (a) diagram and (b) trajectory to be learned.

**P8.13** A multilayer feedforward network needs to be designed that solves the inverse kinematics problem as shown in Figure 8.33(b). The network's eventual use will be in the control arrangement from Figure 8.35(b).

The network should model the two-link planar manipulator as in Figure P8.13(a), with its end effector on a desired trajectory being a straight segment  $AB$  as shown in Figure P8.13(b).

- (a) Draw the network architecture that would be expected to solve this problem.
- (b) Prepare a training set of 11 pairs of input/output data. Note that the desired output data are  $\theta_1$ ,  $\theta_2$ , in response to the end effector Cartesian coordinates  $x, y$  provided as inputs at each point. Assume  $l_1 = l_2 = 1$  for this manipulator.

**P8.14** Repeat Problem P8.13 for the elliptic end effector trajectory having the equation

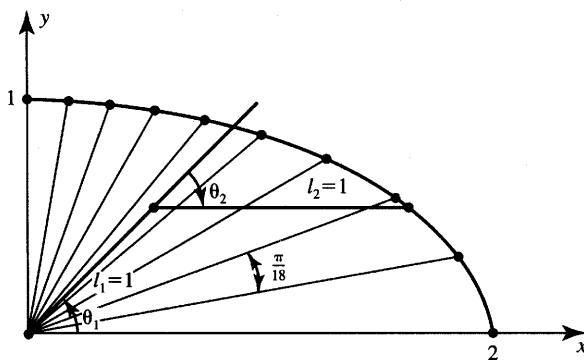
$$\left(\frac{x}{2}\right)^2 + y^2 = 1$$

Select ten training points on the trajectory in the first quadrant. The points should be  $\pi/18$  apart as indicated in Figure P8.14. Prepare complete training data for  $l_1 = l_2 = 1$ .

**P8.15** A multilayer feedforward network needs to be trained to solve the forward kinematics problem. The desired trajectory of a two-link planar manipulator, in joint space, is given as

$$\theta_1(t) = \frac{\pi}{40}t$$

$$\theta_2(t) = \frac{\pi}{20} \left(1 - \frac{t}{20}\right)$$



**Figure P8.14** Trajectory to be learned by the two-link planar manipulator.

- (a) Produce 21 training pairs for trajectory points taken at  $t = 0, 1, 2, \dots, 20$ ; assume  $l_1 = 2$  and  $l_2 = 3$ .
- (b) Devise a network architecture that would be expected to train properly for this problem.

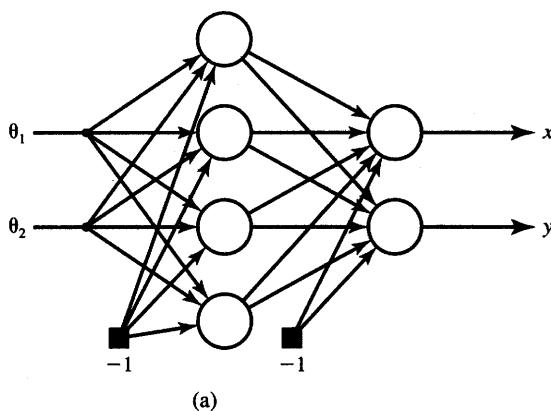
**P8.16\*** The multilayer feedforward network shown in Figure P8.16(a) needs to be designed to solve the forward kinematics problem. The network working area is the circular trajectory with  $r = 3$  shown in Figure 8.36(a). Implement the training of the network using the training data from Figure P8.16(b) [normalization of data may be needed to fit within the  $(-1, 1)$  range].

**P8.17\*** Repeat Problem P8.16 using a multilayer feedforward network of the architecture 2–4–4–2 as shown in Figure P8.17 to model the solution of the inverse kinematics problem. Use the training data set of Figure P8.16(b).

**P8.18** In this problem you are asked to design a simple “mini expert system” called *free time advisor* (FTA) that would suit your personal needs. The FTA is supposed to advise you which of the free time activities to choose from given the circumstances preceding your decision making. The five choices for the activities are

- (a) Socializing (date or visiting with friends, etc.)
- (b) Physical exercise (tennis, jogging, swimming, etc.)
- (c) Shopping or repairing things (taking care of necessities)
- (d) Cultural activities (seeing a good movie or reading a book, etc.)
- (e) Doing nothing (no suggestions needed here).

Factors determining your final choice of activity are virtually countless. Limiting their number to a dozen or so (but no more than two dozen) allows us to keep the expert system development work traceable. Prepare



Point	Angle (deg)	End Eff. (Cartesian)		Joint Angles (rad)	
		x	y	$\theta_1$	$\theta_2$
1	5	2.9886	0.2615	0.7669	-1.9106
2	15	2.8978	0.7765	0.9414	-1.9106
3	25	2.7189	1.2679	1.1160	-1.9106
4	35	2.4575	1.7207	1.2905	-1.9106
5	65	2.1213	2.1213	1.4608	-1.9106
6	55	1.7207	2.4575	1.6396	-1.9106
7	65	1.2679	2.7189	1.8141	-1.9106
8	75	0.7765	2.8978	1.9886	-1.9106
9	85	0.2615	2.9886	2.1632	-1.9106
10	90	0	3.0000	2.2504	-1.9106

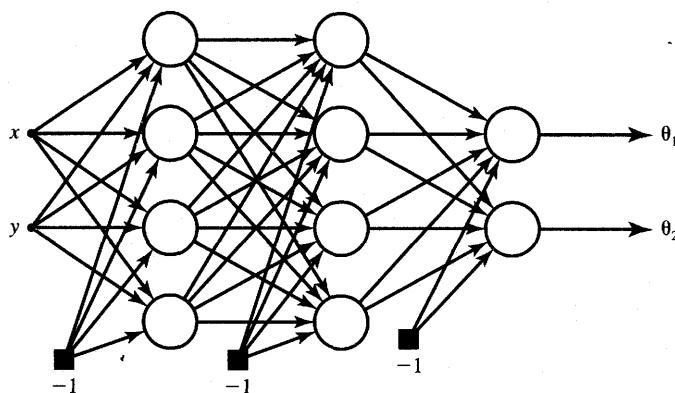
(b)

**Figure P8.16** Figure for forward kinematics Problem P8.16: (a) network architecture and (b) training data.

your list of factors. These factors typically can be

- (1) Money available
- (2) Weather predicted
- (3) Amount of time available
- (4) A good book at hand
- (5) Recent history of free time activities
- (6) Empty refrigerator and pantry
- (7) Car makes unusual noises and needs repair
- (8) More sleep or rest needed.

For final lists of activities and factors that influence your decision, propose a set of test data consisting of the total of 10 to 20 pairs of input/output

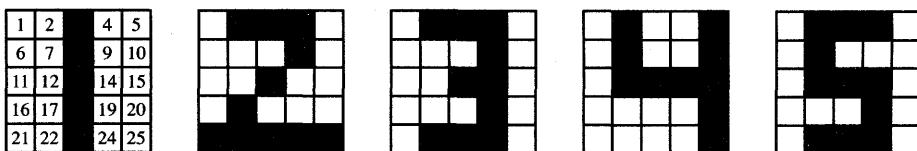


**Figure P8.17** Network for inverse kinematics Problem P8.17.

vectors, with at least 2 pairs for each specific activity from the list. Code inputs as mixed variables (continuous and binary) as you see appropriate in the range  $[-1, 1]$  and assign 0 to those inputs that do not matter.

**P8.19\*** Develop a multilayer feedforward character classifier for five printed digits shown as  $5 \times 5$  black-white pixel maps on Figure P8.19.

- Devise a suitable network architecture for a local representation classifier.
- Prepare the set of five input/output binary training vector pairs.
- Train the network for zero decision errors.
- Perform the recall of nondistorted digits by reusing the training input data.
- Perform the evaluation of the classifier by recalling digits distorted by the center pixel (pixel 13) of the  $5 \times 5$  field being white rather than black.
- Evaluate the classifier by recalling digits distorted by reversal of input pixels 12, 13, and 14.



**Figure P8.19** Pixel maps for digit recognition network in Problem P8.19.

---

## REFERENCES

- Anderson, C. W. 1989. "Learning to Control an Inverted Pendulum Using Neural Networks," *IEEE Control Systems Magazine*, (April): 31-37.
- Anderson, D. Ed. 1988. *Neural Information Processing Systems*. New York: American Institute of Physics.
- Anderson, J. A., and E. Rosenfield. Eds. 1988. *Neurocomputing: Foundations of Research*. Cambridge, Mass.: The MIT Press.
- Apple Computer. 1986. *Imagewriter User's Manual, Part 1: Reference*. Cupertino, Calif.: Apple Computer.
- Arteaga-Bravo, F. J. 1990. "Multi-Layer Back-Propagation Network for Learning the Forward and Inverse Kinematics Equations," in *Proc. Joint 1990 IEEE Int. Neural Network Conf.*, pp. II-319-321.
- Barto, A. G., R. S. Sutton, C. W. Anderson. 1983 "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE Trans. Systems, Man, and Cybernetics SMC-13*(5): pp. 835-846.
- Barto, A. G. 1990 "Connectionist Learning for Control: An Overview," in *Neural Networks for Control*, Eds. W. T. Miller, III, R. S. Sutton, P. J. Werbos. Cambridge, Mass.: The MIT Press.
- Baxt, W. G. 1990. "Use of an Artificial Neural Network for Data Analysis in Clinical Decision-Making: The Diagnosis of Coronary Occlusion," *Neural Computation*, 2: 480-489.
- Bounds, D. G., P. J. Lloyd, B. Mathew, and G. Wadell. 1988. "A Multilayer Perceptron Network for the Diagnosis of Low Back Pain," in *Proc. IEEE Int. Conf. on Neural Networks*, San Diego, California. pp. II-481-489.
- Chandra, V., and R. Sudhakar. 1988. "Recent Developments in Artificial Neural Network Based Character Recognition: A Performance Study," *Proc. IEEE Southeastcon*, Knoxville, Tennessee. pp. 633-637.
- Craig, J. J. 1986. *Introduction to Robotics Mechanics and Control*. Reading, Mass.: Addison Wesley Publishing Co.
- Dayhoff, R. E. and J. E. Dayhoff. 1988. "Neural Networks for Medical Image Processing" in *Proc. IEEE Symp. on Computer Applications in Medical Care*, Washington, D.C. pp. 271-275.
- Denker, J. S. Ed. 1986. *Neural Networks for Computing*. AIP Conference Proceedings. New York: American Institute of Physics.
- Dutta, S., and S. Shekhar. 1988 "Bond Rating: A Non-conservative Application of Neural Networks," in *Proc. IEEE Int. Conf. on Neural Networks*, San Diego, California.

- Eberhart, R. C. and R. W. Dobbins. 1990. *Neural Network PC Tools, A Practical Guide*. San Diego, Calif.: Academic Press.
- Feldman, R. S. 1987. *Understanding Psychology*. New York: McGraw-Hill Book Co.
- Fu, K. S. 1986. "Learning Control Systems—Review and Outlook," *IEEE Trans. Pattern Analysis and Machine Intelligence* 8(3): 327–342.
- Gallant, S. I. 1988. "Connectionist Expert Systems," *Comm. ACM* 31(2): 152–169.
- Gorman, R., and T. Sejnowski. 1988. "Learned Classification of Sonar Targets Using a Massively Parallel Network," *IEEE Trans. Acoustics, Speech, and Signal Proc.* 36: 1135–1140.
- Graf, H. P., L. D. Jackel, and W. E. Hubbard. 1988. "VLSI Implementation of a Neural Network Model," *IEEE Computer* (March): 41–49.
- Guez, A., J. L. Eilbert, and M. Kam. 1988. "Neural Network Architecture for Control," *IEEE Control Systems Magazine* (April): 22–25.
- Hashimoto, H., T. Kubota, M. Kudon, and F. Harashima. 1990. "Visual Control of a Robotic Manipulator Using Neural Networks," in *Proc. 29th Conf. on Decision and Control*, Honolulu, Hawaii. pp. 3295–3302.
- Hecht-Nielsen, R. 1990. *Neurocomputing*. Reading, Mass.: Addison-Wesley Publishing Co.
- Hertz, J., A. Krogh, and R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation*. Redwood City, Calif.: Addison-Wesley Publishing Co.
- Howard, R. E., B. Boser, J. S. Denker, H. P. Graf, D. Henderson, W. Hubbard, L. D. Jackel, Y. LeCun, and H. S. Baird. 1990. "Optical Character Recognition: A Technology Driver for Neural Networks," *Proc. IEEE Int. Joint Neural Network Conf.* pp. 2433–2436.
- Jackel, L. D., H. P. Graf, W. Hubbard, J. S. Denker, and D. Henderson. 1988. "An Application of Neural Net Chips: Handwritten Digit Recognition," in *Proc. IEEE Int. Conf. on Neural Networks*, San Diego, California. pp. II-107–115.
- Jones, D. 1991. "Neural Networks for Medical Diagnosis," in *Handbook of Neural Computing Applications*. New York: Academic Press.
- Josin, G., D. Charney, and D. White. 1988. "Robot Control Using Neural Networks," *Proc. IEEE Int. Conf. on Neural Networks*, San Diego, California. pp. II-615–631.
- Kawato, M., Y. Uno, M. Isobe, and R. Suzuki. 1988. "Hierarchical Neural Network Model for Voluntary Movement with Applications to Robotics," *IEEE Control Systems Magazine* (April): 8–16.

- Kohonen, T. 1990. "The Self-Organizing Map," *Proc. of the IEEE* 78(9): 1464–1480.
- Kraft, L. G., and D. S. Campagna. 1990a. "A Comparison Between CMAC Neural Network Control and Two Traditional Adaptive Control Systems," *IEEE Control Systems Magazine* (April): 36–43.
- Kraft, L. G., and D. S. Campagna. 1990b. "A Summary Comparison of CMAC Neural Network and Traditional Adaptive Control Systems," in *Neural Networks for Control*, ed. T. W. Miller III, R. S. Sutton, and P. J. Werbos. Cambridge, Mass.: The MIT Press.
- LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation* 1: 541–551.
- Liu, H., T. Iberall, and G. A. Bekey. 1989. "Neural Network Architecture for Robot Hand Control," *IEEE Control Systems Magazine* (April): 38–43.
- Lippman, R. P. 1989. "Review of Neural Networks for Speech Recognition," *Neural Computation* 1(1): 1–38.
- Miller, W. T., III. 1989. "Real-Time Application of Neural Networks for Sensor-Based Control of Robots with Vision," *IEEE Trans. Systems, Man, and Cybernetics* 19(4): 825–831.
- Mishkin, E., and L. Braun. Eds. 1961. *Adaptive Control Systems*. New York: McGraw-Hill Book Co.
- Narendra, K. S., and K. Parthasarathy. 1990. "Identification and Control of Dynamical Systems using Neural Networks," *IEEE Trans. Neural Networks* 1(1): 4–21.
- Narendra, K. S., and K. Parthasarathy. 1991. "Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks," *IEEE Trans. Neural Networks* 2(2): 252–262.
- Nguyen, D. H., and B. Widrow. 1990. "Neural Networks for Self-learning Control Systems," *IEEE Control Systems Magazine* (April): 18–23.
- Nguyen, L., R. V. Patel, and K. Khorasani. 1990. "Neural Network Architectures for the Forward Kinematics Problem in Robotics," in *Proc. Joint IEEE Int. Neural Networks Conf.*, San Diego, California. III-393–399.
- Norris, D. E. 1986. "Machine Learning Using Fuzzy Logic with Applications in Medicine," Ph.D. diss., University of Bristol, United Kingdom.
- Omidvar, O. M. Ed. 1992. *Progress in Neural Networks*. Norwood, N.J.: Ablex Publishing Co.
- Poggio, T., and F. Girosi. 1990. "Networks for Approximation and Learning," *Proc. IEEE* 78(9): 1481–1497.

- Pomerleau, D. A. 1989. "ALVINN: An Autonomous Land Vehicle in a Neural Network," in *Advances in Neural Information Processing Systems*, vol. 1, D. Touretzky. San Mateo, Calif.: Morgan Kaufmann Publishers.
- Psaltis, D., A. Sideris, and A. Yamamura. 1987. "Neural Controllers," in *Proc. IEEE Int. Neural Networks Conf.*, San Diego, California. pp. IV-551-558.
- Psaltis, D., A. Sideris, and A. Yamamura. 1988. "A Multi-layer Neural Network Controller," *IEEE Control Systems Magazine* (April): 17-21.
- Ritter, H., and T. Kohonen. 1989. "Self-Organizing Semantic Maps," *Biolog. Cybern.* 61: 241-254.
- Saerens, M., and A. Soquet. 1991. "Neural Controller Based on Back-Propagation Algorithm," *IEE Proc., Part F* 138(1): 55-62.
- Sejnowski, T., and C. Rosenberg. 1987. "Parallel Networks that Learn to Pronounce English Text," *Complex Systems* 1: 145-168.
- Simmons, D. M. 1972. *Linear Programming for Operations Research*. San Francisco: Holden-Day.
- Sobajic, D. J., J. J. Lu, and Y. H. Pao. 1988. "Intelligent Control for the Intelledex 605 T Robot Manipulator," in *Proc. 1988 IEEE Int. Neural Networks Conf.*, San Diego, California. pp. II-633-640.
- Tank, D. W., and J. J. Hopfield. 1986. "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit and a Linear Programming Circuit," *IEEE Trans. Circ. Syst. CAS-33(5)*, 533-541.
- Tolat, V. V., and B. Widrow. 1988. "An Adaptive 'Broom Balancer' with Visual Inputs," in *Proc. 1988 IEEE Int. Neural Networks Conf.*, San Diego, California. pp. II-641-647.
- Touretzky, D. Ed. 1989. *Advances in Neural Information Processing Systems*, vol. 1. San Mateo, Calif.: Morgan-Kaufmann Publishers.
- Touretzky, D. Ed. 1990 *Advances in Neural Information Processing Systems*, vol. 2. San Mateo, Calif.: Morgan-Kaufmann Publishers.
- Tsyplkin, Ya. Z. 1973. *Foundations of the Theory of Learning Systems*. New York: Academic Press.
- Villalobos, L., and S. Gruber. 1990. "Interpolation Characteristics and Noise Sensitivity of Neural Network Based Inspection of Machined Surfaces," in *Proc. 1990 IEEE Int. Conf. on Robotics and Automation*.
- Wang, L., and J. M. Mendel. 1990. "Structured Trainable Networks for Matrix Algebra," in *Proc. IEEE 1990 Joint Conf. on Neural Networks*, San Diego, California. pp. 125-132.
- Widrow, B., and R. Winter. "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition," *IEEE Computer* (March): 25-39.

- Widrow, B., and M. A. Lehr. 1990. "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *Proc. IEEE* 78(9): 1415–1441.
- Wilhelmsen, K., and N. Cotter. 1990. "Neural Network Based Controllers for a Single-Degree-of-Freedom Robotic Arm," in *Proc. 1990 Int. Joint Conf. on Neural Networks*, San Diego, California. pp. III-407–413.
- Yoon, Y. O., R. W. Brobst, P. R. Bergstresser, and L. L. Peterson. 1989. "A Desktop Neural Network for Dermatology Diagnosis," *Journal of Neural Network Computing*, (Summer): 43–52.
- Zigoris, D. M. 1989. "Performance Evaluation of the Back-Propagation Algorithm for Character Recognition," M. Eng. thesis, University of Louisville, Kentucky.
- Zurada, J., D. M. Zigoris, P. B. Aronhime, and M. Desai. 1991. "Multi-Layer Feedforward Networks for Printed Character Classification," *Proc. 34th Midwest Symp. on Circuits and Systems*, Monterey, California, May 14-16.



# 9

---

## NEURAL NETWORKS IMPLEMENTATION

*Physical models are as different from the world as a geographical map is from the surface of the earth.*

L. BRILLOUIN

- 9.1 Artificial Neural Systems: Overview of Actual Models
- 9.2 Integrated Circuit Synaptic Connections
- 9.3 Active Building Blocks of Neural Networks
- 9.4 Analog Multipliers and Scalar Product Circuits
- 9.5 Associative Memory Implementations
- 9.6 Electronic Neural Processors
- 9.7 Concluding Remarks

**A**rtificial neural systems are currently realized in a number of ways. As we have seen, they can be implemented as computer simulations. This approach has been used for a number of examples in this text. Numerical simulations also support most of the computational end-of-chapter exercises involving training and recall. For the best results and real-life applications, however, artificial neural networks need to be implemented as analog, digital, or hybrid (analog/digital) hardware. Moreover, neural network processors rather than digital computer simulations seem to be the key ingredient to further expansion and commercialization of neural network technology. Fortunately, study of the technical reports and of products available indicates that electronic implementations of neural networks are feasible and promising. This provides us with the focus and justification for neural network hardware implementation study in this chapter.

While neural network hardware development is being pursued in a number of research centers, neurocomputing concepts and applications are usually tested through simulations of neural algorithms on digital computers. To this aim, a very wide spectrum of computers, called *programmable neurocomputers*, can be used.

A simple microcomputer running neural network training or recall software can therefore be termed a programmable neurocomputer. We should realize, however, that an especially efficient programmable neurocomputer can be built by tailoring its architecture to the flow of the neural algorithm. This would involve organizing the computations in parallel, scheduling appropriate data communication, and specializing in arithmetic or transfer operations that are performed frequently.

A number of companies have brought coprocessor accelerator boards on the market. They typically interface to personal computers or conventional workstations and are able to enhance the speed of numerical computations considerably. Coprocessor accelerator boards also make the simulations of neural algorithms faster and more accurate. A natural progression of these coprocessors has produced neural array processors with dedicated architecture and computation flow. A number of specialized parallel neurocomputers which are programmable have been developed.

However, any programmable sequential general-purpose neurocomputer is an order of magnitude slower than neural hardware, which could be directly produced using the same fabrication technology as the programmable neurocomputer. Our objective in this chapter is to review briefly programmable neurocomputers and provide extensive fundamentals on neural hardware. Such neural hardware is called a *neurocomputer*. Neurocomputers are dedicated computing devices working with embedded neural processing algorithms and employ parallel processing to increase the throughput. Programming is not required for their operation, but they often interface with a programmable training controller. The high-performance parallel computing unit must be matched with an efficient interface to avoid bottlenecks in training and recall network functions. We will focus in this chapter on silicon-based neural hardware which represents the largest category among special-purpose analog and digital neurocomputers.

## 9.1

---

### ARTIFICIAL NEURAL SYSTEMS: OVERVIEW OF ACTUAL MODELS

Throughout the preceding chapters we have covered neural network concepts and applications, but only marginal attention has been paid to building actual working models of artificial neural systems. Our study thus far indicates, however, that the computation by neural networks whether implemented on a programmable or nonprogrammable neurocomputer is performed differently than computation by ordinary digital computers. In this section we attempt to address these differences. In addition, an overview of computational requirements for implementing connectionist models is the focus of this section.

The diversity of neural network concepts, algorithms, and abstractions, which dominated the coverage in previous chapters, also extends into the area of building their physical models. We will see that artificial neural networks can be built using analog, digital, or hybrid dedicated electronic or optical hardware. As mentioned before, neurocomputing algorithms can be tested and implemented on either dedicated or general-purpose conventional computers, called programmable neurocomputers. In addition, mixtures of computer simulations with digital or analog integrated circuit computations are often applied. These usually dedicated circuits can be built in a variety of technologies to assist with efficient implementations of specialized neurocomputing algorithms.

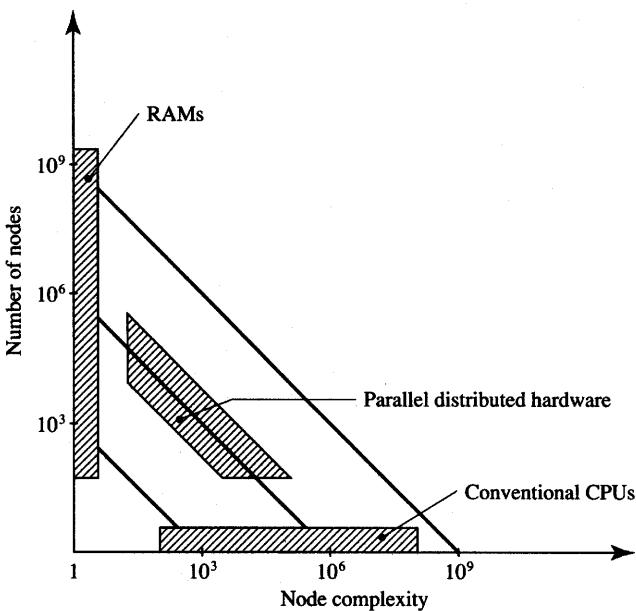
Let us look at an example of a neurocomputer that involves such mixed computations. One of the commercial neurocomputers involves simulation of error back-propagation training on a conventional programmable host computer. Subsequent transfer (downloading) and storage of weights, which result from the training, are performed by the digital circuitry. However, information stored in the network is recalled through analog computation within the designated electronic circuitry. In fact, the analog computation mode and parallel form of information processing and retrieval are responsible for the fast recall performed by the discussed network. In this neurocomputer we thus have numerical simulation of training, which is performed off-chip, digital transfer and data storage circuitry, and a dedicated analog neural network that performs recall only.

Let us focus on efficient hardware implementation of neural algorithms. Review of processing algorithms indicates that batch learning and incremental learning either from current errors or from past experience, as well as recall of stored information, are all executed as sequences of relatively simple arithmetic operations such as weighted product summation followed by a nonlinear mapping. The most commonly encountered operations are scalar and outer product vector multiplication and matrix-vector multiplication. These operations are performed as a series of multiply and/or multiply and add operations. Additions and subtractions of matrices and vectors, and ordinary multiplications often involving the generation of a nonlinear activation function, are less frequent, but are also indispensable for most learning and recall tasks.

---

### **Node Numbers and Complexity of Computing Systems**

Let us attempt to position neural computation in the more general context of computation. One of the most important features of artificial neural systems is that they perform a large number of numerical operations in parallel. These operations involve simple arithmetic operations as well as nonlinear mappings and computation of derivatives. Almost all data stored in the network are involved in recall computation at any given time. The distributed neural processing is



**Figure 9.1** Comparison of the number of nodes and single node complexity of computing hardware. [Adapted from Treleaven, Pacheco, and Vellasco (1989).]

typically performed within the entire array composed of neurons and weights. This indicates that parallel distributed computing makes efficient use of the stored data (weights) and of the input data (inputs).

In contrast to the neural processor arrays performing parallel computations, conventional computers operate on a very limited number of data at a time. The arithmetic-logic unit within the processor of a conventional computer manipulates only two words just recently fetched from memory. Although the same processors can perform a variety of complex instructions very quickly, many megabytes of data are idle during any instruction cycle. The duty cycle of a processor is close to 100%, but that of the stored data is close to zero in conventional computation. This explains the computational bottleneck when conventional computers must deal with very large input/output data rates and perform tasks such as computer vision or speech recognition. This also explains why neural networks can handle massive amounts of input/output data more efficiently than conventional computers.

Figure 9.1 illustrates the comparison of conventional computers with the parallel distributed processing hardware. The parameters compared are the number of addressable computing nodes versus their complexity. We adopt for this comparison a rather broad concept of the computing node. Any addressable unit ranging from RAM cell to processor is understood in this discussion to be a

computing node. The simplest processing node is the RAM cell, which typically consists of one and rarely more than three transistors. However, it has no processing power beyond its ability to retain the present binary content or to acquire a new one. In spite of their simplicity, millions of memory cells are needed for operation of a modern digital computer. We may place the central processing unit (CPU) of a computer at the other end of the spectrum of nodes. The CPU is the most complex processing node; however, only a limited number of CPUs are needed for a computer to operate.

Neural networks fall by nature of their parallel distributed processing into the middle ground between the two extreme examples of nodes of RAM cell and CPU. A medium number of processing nodes, each of low to medium complexity, perform local neurocomputing on local data. Ideally, each processing node, or neuron, should be rather simple and small to accommodate the placement of many of them on an integrated circuit chip. Since the neural processing node does not need to perform dozens of diversified logic and arithmetic operations, it can be hard-wired to perform just its basic computational function (Mackie et al. 1988). Its operation is rather straightforward and can often be tailored to a particular application or network paradigm.

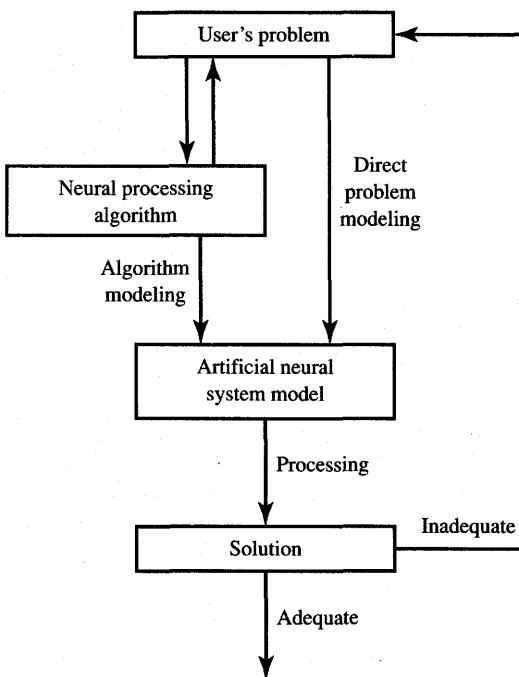
Let us review the choice of an artificial neural system model, a choice that has to precede solution of a neurocomputing task. A specific neurocomputing task needs to be translated onto an artificial system model so that it can be implemented. Since the choice of an algorithm depends on the specific user's task to be accomplished, the user must first consider the selection of an optimal processing algorithm. The problem for which a suitable algorithm has been found is then transferred to the model to compute a solution.

This sequence of steps is illustrated in Figure 9.2 showing the problem-algorithm-model flowchart. Alternatively, the user's problem can be transferred directly to the artificial neural system model for execution. This may be the case for well-defined and known tasks for which the choice of the successful model has been made, and parameters of the algorithm have already been developed based on the previous experience of the user.

---

### Neurocomputing Hardware Requirements

Artificial neural system models contain both invariant and variable parameters, which are called *characteristics* here and are essential for the system's operation (Reece and Treleaven 1988). Invariant characteristics are fixed in the neurocomputing model, while variable characteristics can be modified. Invariant characteristics usually include, among others, network topology, number of bits, or simply accuracy in expressing the weight value, number of bits describing the activation and output values of the neuron, and interconnection density. Variable characteristics of the model typically include its learning parameters, sequence



**Figure 9.2** Problem-algorithm-model flowchart.

of learning and recall operations, as well as the presence or lack of synchronous neuron updates, etc.

Simulation of most neural network models in real-life applications can become computationally intensive to the degree that the processing bottleneck on conventional computers imposes constraints on practical explorations of large neural networks. In fact, limitations of computational power available has dwarfed the learning machines' research wave of a quarter century ago.

As mentioned before, there are two distinct approaches currently being taken for supporting artificial neural system modeling:

1. General-purpose computers, which are programmable and therefore able to simulate a variety of models
2. Special-purpose hardware, often dedicated to a specific neural processing model.

Let us first outline the requirements for neurocomputing hardware. This outline will allow us to specify the set of features that neuroemulators should have. Understandably, the features of any given artificial neural network depend primarily on its specific application. Let us look first at such properties as recall time, network size, and input layer size. Figure 9.3(a) tabulates five applications of layered feedforward networks. The applications listed differ widely from each

Application	Recall Time s	Network Size		Input Layer Size
		Neurons	Weights	
Radar Pulse Identification	$50 \cdot 10^{-6}$	384	32 k	128
Robot Arm Controller	0.01–1	312	3.6 k	5
Isolated Words Recognition (1024-word vocabulary)	.01	10 k	2.5 k	25
Low Level Vision 100 weights/neuron	.04	64 k	6400 k	64 k
Financial Risk Evaluation	1	5–10 k	4000 k	?

(a)

**Figure 9.3a** Neural network comparison for typical applications: (a) typical sizes and recall times. [Adapted from DARPA (1988).]

other in required processing speed and memory size. For example, for identification of radar pulse signals, processing time is the most severe constraint but memory size is not. On the other end, financial risk evaluation modeling requires a very large, but not necessarily very fast, network. The response time of financial risk evaluation can be in the range of a few seconds, while the radar pulse identification must be performed within  $50 \mu\text{s}$  (Jutten, Guerin, and Herault 1990). Some other applications, such as vision, may require unusually large input/output bandwidth. The input/output bandwidth is understood here to be the number of bits per second that can be processed by the input/output unit. This brief comparison of computational requirements for various tasks indicates that a trade-off may be necessary between a network's processing time, memory size, and the input/output bandwidth. When considering particular applications, these three features should be balanced and geared toward the specific problem requirements.

Let us now consider the memory and network size of an artificial neural system. The size of a neural network is determined by the overall number of neurons and the total number of inputs, but mainly by the number of its weights. Thus, neural network size is about directly proportional to the number of weights. For example, there are  $(n - 1)^2$  weights in fully coupled single-layer networks with  $n$  neurons. Since the number of weights drastically exceeds the number of neurons, the size of the network is determined primarily by the number of weights.

Assume that a neural network model is simulated on a digital computer. Each weight is typically characterized by a single value, and only the current

Application	Memory Size	Word Length	I/O Bandwidth	CPS Measure (Recall)	MFLOPS Measure (Recall)
	Words	Bits	W/s or b/s		
Radar Pulse Identification	< 32 k	1/neuron 32/synapse	2.5 Mb/s	640 M	1.3 G
Robot Arm Controller	4.5 k	32	< 600 W/s	< 350 k	< 700 k
Isolated Words Recognition (1024-word vocabulary)	300 k	32	2.5 kW/s	24 M	48 M
Low Level Vision 100 weights/neuron	6.4 M	1/neuron 8/synapse	1.6 Mb/s	156 M	312 M
Financial Risk Evaluation	4.1 M	?	< 7 kW/s	4 M	8 M

(b)

**Figure 9.3b** Neural network comparison for typical applications (*continued*): (b) hardware requirements. [Adapted from DARPA (1988).]

weight value needs to be stored during learning. This also applies to error back-propagation training of multilayer feedforward networks. However, if either cumulative weight adjustment training, or training with a momentum term, is implemented, the most recent weight value must be memorized in addition to its present value. Furthermore, activation values, output values, and the error term value must be stored for each neuron of the layer that undergoes training.

At the data format level, the machine representation must be more accurate for weights than for any of the remaining data. In particular, weight updates resulting during incremental supervised or unsupervised learning are relatively small to provide training stability and convergence. A *floating-point* format for representing weights by number is therefore necessary. Floating-point representation for weights typically requires a mantissa with 24 or 32 bits (Bessiere et al. 1991).

The external environment in which the neural network is used usually imposes the speed of data transfer. A high sampling rate, for example, would be necessary to process large video images. In addition to communication speed requirements, a neural network can be characterized by its computational bandwidth. The computational bandwidth for conventional computers is typically expressed in units such as MFLOPS (million of floating-point operations per second), or MIPS (million of instructions per second). For simulation of artificial neural systems the "CPS" (connections per second) unit has recently emerged as an efficiency measure (Bessiere et al. 1991).

The CPS unit has become the popular catch phrase for neurocomputing products. Its generality is severely limited, however. It describes only the processing speed for the recall mode of a multilayer feedforward network. CPS units neither reflect the speed of learning of any architecture nor the processing speed within recurrent networks. Also, the number of CPS units does not include auxiliary system operations, which support inputs and outputs.

Let us compare the units of CPS and FLOPS for feedforward networks simulated on a conventional computer. The recall pass through a single neuron with  $n$  connections requires approximately  $n$  multiplication-accumulation operations, one for each weight. Assuming that multiplication-accumulation is equivalent to approximately two floating-point operations of the CPU, we can establish that there is 1 : 2 ratio between CPS measure and FLOPS measure.

Figure 9.3(b) outlines the hardware requirements for the five artificial neural systems discussed earlier in Figure 9.3(a). The table lists the required CPS and FLOPS measures expected for each of the five applications. It can be seen from the table that each of the networks not only requires a different size of the memory measured in terms of memory words, but memory word lengths also differ. In addition, I/O bandwidth requirements differ widely among networks for different applications.

As an example, the application of radar pulse identification requires 2.5 Mb/s I/O bandwidth, which corresponds to the transfer of a 128-tuple binary vector every 50  $\mu$ s. The network performing this task is of rather moderate size starting at 32 000 weights and having 256 binary neurons arranged in two layers plus one layer of input nodes. If a binary input vector must be processed every 50  $\mu$ s, then 32 000 connections must be implemented 20,000 times per second. We would thus have the required CPS measure of approximately 640 MCPS. This corresponds to a computational power of about 1.3 MFLOPS of the computer simulating efficiently the recall stream of radar pulses that needs to be recognized.

Let us attempt to outline the general computational requirement estimates for two important cases of neural processing. Our concern will be digital computer simulation of learning in feedforward networks of multilayer perceptrons, and simulation of recall in discrete-time, fully coupled recurrent networks. Figure 9.4 lists the average number of multiplication-accumulation operations per weight. Single-layer and multilayer feedforward network learning, and single-layer and recurrent network recall are included in this comparison. The number of operations listed in the figure are understood to be estimates and are specified as range of values. This allows accounting for supporting operations other than multiplication and addition. Also, the precise average number of operations would depend on the complexity of the activation function computation. The function can be defined for computation as a permanent look-up table, generated by polynomial expansion or calculated directly. Each of the methods requires a different computational expense.

Processing Mode	Single-layer Network	Multilayer Network	Recurrent Network (Discrete-time)
Learning	4 – 10	7 – 12	Batch
Recall	2 – 3	2 – 3	$(2 - 3) \times R$
Total	6 – 13	9 – 15	

R—number of recurrent updates of all neurons until stability.

**Figure 9.4** Average estimated number of multiplication-accumulation operations per weight for three selected architectures. [Adapted from Jutten, Guerin, and Herault (1990).]

It can be seen that the learning of a multilayer feedforward architecture is the most computationally intensive neural processing form. Let us use the data of Figures 9.3 and 9.4 to estimate the required computational power of a conventional computer performing efficient learning of isolated word recognition using a 1024-word vocabulary. Assume that the training needs to take 4000 full presentation cycles. Thus, about 4 M input vectors are submitted to the network during learning, or 4 M incremental learning steps are needed. Since the network contains 2500 weights, assuming 12 operations per single weight adjustment would yield the total of 120 G required operations. It then takes about 42 min of computing time for a 48-MFLOPS computer to complete the training, or only 1 min for a 2000-MFLOPS computer to accomplish the same task.

The calculation shows that the powerful computer can produce the multilayer feedforward network in question within a reasonable training time, provided the training converges to an acceptable solution. Improvements of training efficiency can be achieved in a number of forms. Some of them are:

- Dedicated hardware architectures of conventional programmable computers
- Parallel distributed architectures of densely interconnected computing neural nodes containing conventional multiply or multiply-add processors
- Reduction of the volume of the training data through the use of the generalization properties of networks
- Designing neurocomputing integrated circuits using digital, analog, or digital/analog arrays performing simultaneous local computations within the entire array.

As stated before, the availability of neurocomputing integrated circuits offers the most advantageous alternative for multifaceted utilization of neural networks.

Not only would the processing time drastically decrease for integrated neurocomputing circuits, but also the size, power supply requirements, and the retraining possibility of networks would render miniature neurocomputers very attractive.

---

### Digital and Analog Electronic Neurocomputing Circuits

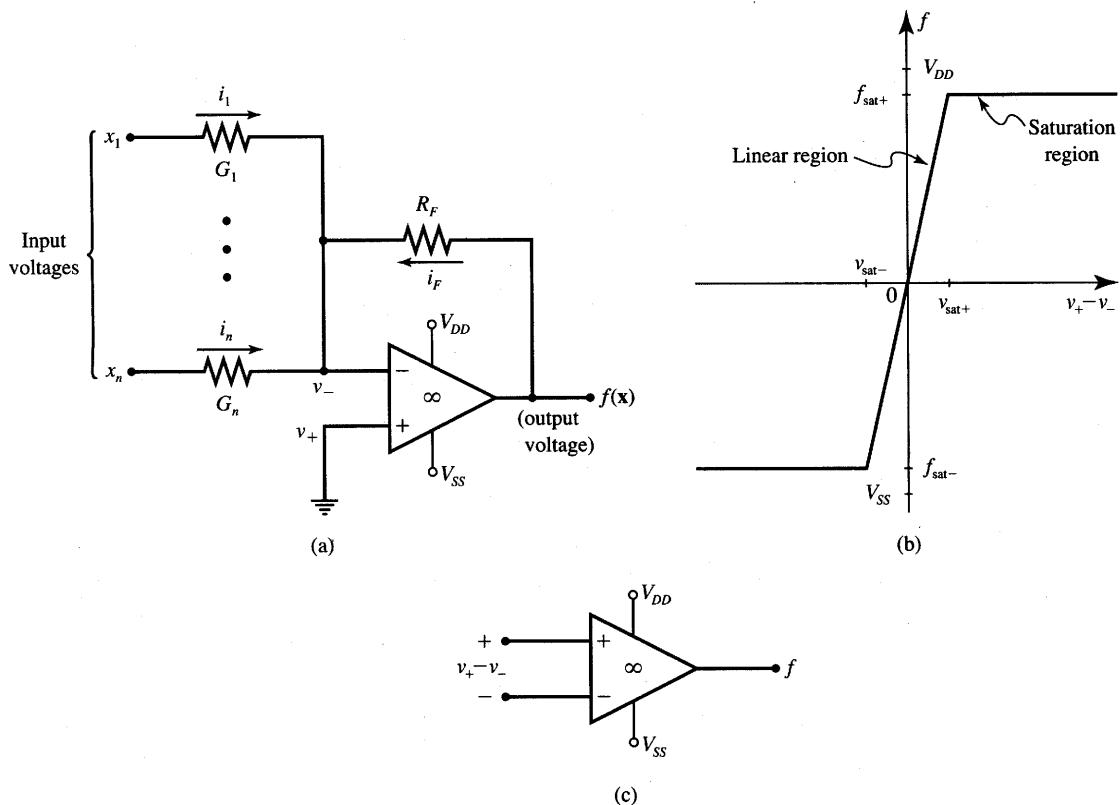
To review special-purpose neurocomputing hardware, the exposition of artificial neural systems must descend to the level of the neuron. The basic processing unit defined as the processing node (neuron) and weights (synaptic connections) was shown earlier in Figure 2.4. The processing unit needs to memorize data stored as connection weights, to generate a neuron's activation value, and to compute the unit's output signal. These features would make it possible for the unit to perform recall functions. In addition, the ability to store the learning rule and to adapt weights according to the rule would be the most desirable property of the processing node. The processed signals and functions of the node can be either analog, digital, or of a combined type (Goser et al. 1989).

Let us review some basic issues of building a neural processing node. To calculate the activation value using the conventional digital computer simulation method, the weight values stored in a local memory cell are first multiplied by the input values and then the products are added. More efficiently, the activation value can be compounded through serial calculation as follows

$$\begin{aligned} \text{net}_i &\leftarrow 0 \\ \text{net}_i &\leftarrow \text{net}_i + w_{ij}x_j, \quad \text{for } j = 1 \text{ TO } n \text{ STEP } 1 \end{aligned} \tag{9.1}$$

This commonly used multiply-accumulate sequence (9.1) requires synchronous operation of related digital hardware. High-precision calculations and considerable noise immunity are inherent in this computing method. While high-precision computation of a neuron's response as in (9.1) is often indispensable for the learning mode, the recall function usually does not require the same degree of accuracy. Furthermore, it is difficult to avoid the view that the computation algorithm (9.1) cannot be substantiated by any biological analogies. Let us now therefore look at analog computation performed by a neural processing node. Analog computation is not only more plausible in biological neural networks, but constitutes the fundamental form of processing for nonprogrammable neural hardware.

In the case of an analog processing node, knowledge is encapsulated in analog weights  $w_{ij}$ . Learning of the network proceeds via relatively slow changes of weights  $w_{ij}$  and only in the training mode. Recall proceeds as simultaneous analog computing within the entire array encompassing all nodes. In a simple version, weights can be determined by resistance values, and the analog computation of



**Figure 9.5** Neuron and weight implementation using resistors and an operational amplifier:  
 (a) circuit diagram, (b) transfer characteristics of an operational amplifier (open-loop), and  
 (c) open-loop operational amplifier symbol.

the scalar product and subsequent nonlinear mapping can be performed by a summing amplifier with saturation.

A conventional operational amplifier can implement these node functions as follows

$$f(\mathbf{x}) = f \left( \sum_{j=1}^n w_j x_j \right) \quad (9.2)$$

A sample analog implementation of the processing node performing Equation (9.2) is shown in Figure 9.5(a). The neuron, or cell body, is formed from an infinity-gain operational amplifier with strong negative feedback. Weight value \$w\_j\$ is proportional to the conductance \$G\_j\$. As derived below, the feedback resistance \$R\_F\$ provides here a proportionality factor common for all weights. Let us analyze the computing node circuit and evaluate its weights in terms of circuit components.

Figure 9.5(b) illustrates the voltage transfer characteristics  $f(v_+ - v_-)$  of the open-loop high-gain operational amplifier. Figure 9.5(c) shows the circuit symbol of the open-loop operational amplifier for which the characteristics shown are valid. In the linear region, the open-loop operational amplifier performs as a constant-gain amplifier of high gain. The open-loop gain value of the operational amplifier in the linear region corresponds to the slope of the steep part of the characteristics displayed in Figure 9.5(b). The output voltage  $f$  of the operational amplifier, however, can neither exceed the  $f_{\text{sat}+}$  value, nor can it drop below the  $f_{\text{sat}-}$  value. This is due to the saturation of transistors within the device. Typically, saturation voltages  $f_{\text{sat}-}$  and  $f_{\text{sat}+}$  are of equal magnitude and of opposite signs. The values of  $f_{\text{sat}+}$  and  $f_{\text{sat}-}$  are about 1 V below the operational amplifier supply voltage  $V_{DD}$ , and 1 V above  $V_{SS}$ , respectively.

As a consequence, an operational amplifier operates within a very narrow range of input voltages. This range is  $(v_{\text{sat}-}, v_{\text{sat}+})$ . However, the differential input voltage  $v_+ - v_-$  remains in this range only if the operational amplifier is in its linear region of operation. Note that only the operational amplifier with a closed negative feedback loop remains in the linear region. The negative feedback loop is closed when the resistance  $R_F$  connects the output with an inverting input of the operational amplifier. The neuron's circuit from Figure 9.5(a) can be considered as having the discussed properties.

It can be seen that any rise in output voltage  $f$  causes an increase of current  $i_F$ , which in turn elevates the potential  $v_-$ . Such an increase is caused as the result of an additional voltage drop across the device input resistance  $R_i$ . This resistance can be measured between the noninverting and inverting input terminals and is typically of large value. The increase of voltage across  $R_i$ , however, results in a decrease in the output voltage since an increase in  $v_-$  brings about a decrease in the voltage  $f$ . The final outcome of this process is that the negative feedback around the operational amplifier enforces its stabilization in the linear region. More precisely, the existing negative feedback attempts to stabilize its differential input voltage  $v_+ - v_-$  near 0 V.

Inspection of Figure 9.5(a) indicates that the negative feedback enforces the condition  $v_+ - v_- \approx 0$  V. Since the noninverting input of the operational amplifier is grounded and thus  $v_+ = 0$  V, potential  $v_-$  remains very close to 0 V. This phenomenon is called *virtual ground* and it makes the analysis of operational amplifier circuits rather simple. The advantage of the virtual ground node is that while being separated from the ground by a very large resistance  $R_i$ , the node behaves as grounded. The property holds for the operational amplifier inverting input node of Figure 9.5(a). This remains true as long as the negative feedback prevails.

The KCL equation for the inverting input node of the circuit from Figure 9.5(a) can be obtained as

$$i_1 + i_2 + \dots + i_n = -i_F \quad (9.3a)$$

which is equivalent to

$$(x_1 - v_-)G_1 + (x_2 - v_-)G_2 + \dots + (x_n - v_-)G_n = (v_- - f) \frac{1}{R_F}$$

Using the virtual ground property we obtain

$$x_1G_1 + x_2G_2 + \dots + x_nG_n = -\frac{f}{R_F} \quad (9.3b)$$

The output voltage of the neuron can now be obtained from (9.3b) for the linear range of operation as

$$f(\mathbf{x}) = \sum_{j=1}^n x_j(-R_F G_j) \quad (9.4a)$$

where

$$\sum_{j=1}^n x_j(-R_F G_j) \stackrel{\Delta}{=} \text{net} \quad (9.4b)$$

Comparison of the coefficients in expressions (9.2) and (9.4) leads to the conclusion that the weights implemented by the circuit of Figure 9.5(a) are

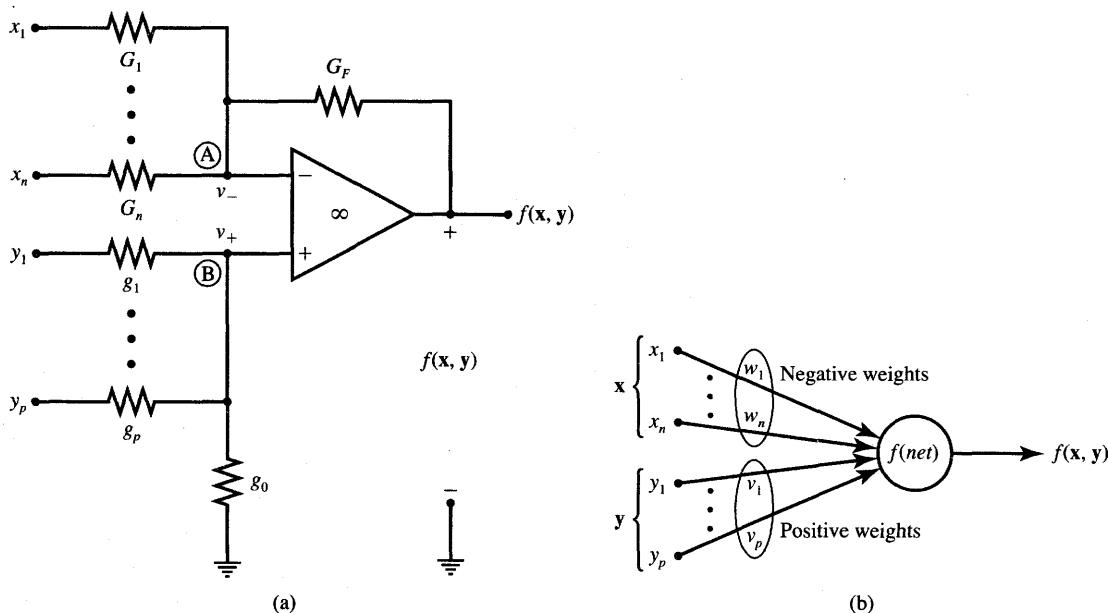
$$w_j = -R_F G_j \quad (9.5a)$$

Furthermore, considering the definition of activation *net* as in (9.4b), the neuron implements here a unity-gain activation function in the linear region. In fact, the amplifier produces a linear combination of input voltages with weights (9.5a). The amplifier saturates whenever  $f(\mathbf{x})$  reaches and attempts to exceed  $f_{\text{sat}+}$ . It also saturates when  $f(\mathbf{x})$  attempts to drop below  $f_{\text{sat}-}$ . This allows for expression of the activation function in the following form:

$$f(\text{net}) = \begin{cases} f_{\text{sat}-}, & \text{for } \text{net} < f_{\text{sat}-} \\ \text{net}, & \text{for } f_{\text{sat}-} < \text{net} < f_{\text{sat}+} \\ f_{\text{sat}+}, & \text{for } \text{net} > f_{\text{sat}+} \end{cases} \quad (9.5b)$$

The activation value *net* is not measurable or physically present anywhere in the discussed neural computing node circuitry. It is helpful, however, to define and use this variable as a hypothetical input voltage of a unity-gain saturating amplifier.

Note that the discussed processing node configuration is somewhat preliminary. The unit basically implements the identity activation function. Also, positive weights cannot be achieved in this circuit without some additional modifications. However, it is possible to produce a nonzero bias level in this configuration. This can be easily accomplished by connecting the noninverting input terminal of the operational amplifier to the bias voltage  $T$ . As a consequence, the virtual ground potential of the inverting operational amplifier input is shifted to the value of  $T$ . Alternatively, the fixed bias can be produced by fixing at a constant value one of the input voltages  $x_1, x_2, \dots, x_p$  in the circuit of Figure 9.5(a).



**Figure 9.6** Resistor-operational amplifier implementation of neuron with weights: (a) circuit diagram and (b) computing node symbol.

Figure 9.6(a) shows the resistor-operational amplifier implementation of an electronic neuron with the output voltage being weighted input voltages. Both negative and positive signs of weighting coefficients can be produced in this circuit. Negative weights are formed in the same way as in the configuration of Figure 9.5(a). In addition, positive weights are produced by a voltage divider circuitry connected to the noninverting input of the operational amplifier. We assume here that negative weights  $w_i$  are associated with inputs  $x_i$ , for  $i = 1, 2, \dots, n$ , and positive weights  $v_i$  are associated with inputs  $y_i$ , for  $i = 1, 2, \dots, p$ .

Let us perform the analysis of this network by using the superposition principle. Our objective is to express the neuron's weight values and its range of linear operation in terms of circuit component values. We assume momentarily that the neuron made of an operational amplifier operates in the linear region. According to the superposition technique,  $f(\mathbf{x}, \mathbf{y})$  can be computed as the sum of individual  $n + p$  responses, which are the results of each separate excitation. The resulting neuron's weights that need to be computed are shown in symbolic form in Figure 9.6(b). Markedly, results obtained in (9.4a) and (9.5a) are already available for negative weights. These weights are responsible for computing the contribution of inputs  $x_1, x_2, \dots, x_n$  to the total neuron's response.

Let us note that the response due to the individual input  $y_k$  when all remaining inputs are short-circuited to ground can be computed using the following

expansion of the output voltage  $f(y_k)$

$$f(y_k) = \left( \frac{f(y_k)}{v_+} \right) \cdot \left( \frac{v_+}{y_k} \right) y_k \quad (9.6)$$

The first ratio of (9.6) can be computed by setting  $v_+ \equiv v_-$  due to the virtual ground property, and by writing the KCL equation for the node labeled  $A$ :

$$(f(y_k) - v_+) G_F - v_+ \sum_{j=1}^n G_j = 0 \quad (9.7a)$$

We thus obtain

$$\frac{f(y_k)}{v_+} = 1 + R_F \sum_{j=1}^n G_j \quad (9.7b)$$

The second ratio of (9.6) can be easily computed from the voltage division at node  $B$  as follows:

$$\frac{v_+}{y_k} = \frac{g_k}{\sum_{j=0}^p g_j}, \quad \text{for } k = 1, 2, \dots, p \quad (9.8)$$

Combining expression (9.7b) and (9.8) yields the neuron's response  $f(y_k)$  due to  $y_k$  from (9.6) as

$$f(y_k) = y_k \left( 1 + R_F \sum_{j=1}^n G_j \right) \frac{g_k}{\sum_{j=0}^p g_j} \quad (9.9)$$

Summarizing the results of the analysis for this elementary neural processing node, the final negative and positive weight values expressed in terms of circuit components result, respectively, as

$$\begin{aligned} w_i &= -R_F G_i, \quad \text{for } i = 1, 2, \dots, n \\ v_k &= \left( 1 + R_F \sum_{j=1}^n G_j \right) \frac{g_k}{\sum_{j=0}^p g_j}, \quad \text{for } k = 1, 2, \dots, p \end{aligned} \quad (9.10)$$

The total output voltage of the analyzed neuron is expressed as

$$f(\mathbf{x}, \mathbf{y}) = \begin{cases} f_{\text{sat}-}, & \text{for } \text{net} < f_{\text{sat}-} \\ \text{net}, & \text{for } f_{\text{sat}-} < \text{net} < f_{\text{sat}+} \\ f_{\text{sat}+}, & \text{for } \text{net} > f_{\text{sat}+} \end{cases} \quad (9.11)$$

where

$$\text{net} = \mathbf{w}' \mathbf{x} + \mathbf{v}' \mathbf{y}$$

Noticeably, the assumption of linear operation remains valid only when the condition  $f_{\text{sat}-} < \text{net} < f_{\text{sat}+}$  holds. For a  $\text{net}$  value outside this range, the performed analysis of the neuron of Figure 9.6 remains invalid and the neuron's output latches at one of its saturation voltage levels  $f_{\text{sat}-}$  or  $f_{\text{sat}+}$ .

**EXAMPLE 9.1**

This example illustrates the typical design steps of an electronic neural network for cluster detection. Assume that a three-neuron network has been trained in the winner-take-all mode as described in Section 7.2. The network needs to be able to identify one of the three input vector clusters. Network weight vectors that need to be implemented are

$$\mathbf{w}_A = \begin{bmatrix} -2.897 \\ 0.776 \end{bmatrix}, \quad \mathbf{w}_B = \begin{bmatrix} 2.121 \\ 2.121 \end{bmatrix}, \quad \mathbf{w}_C = \begin{bmatrix} -1.026 \\ -2.819 \end{bmatrix} \quad (9.12)$$

A network with three operational amplifiers, each using four resistors, as shown in Figure 9.7, can be employed to fulfill the design specifications (9.12). Detectors of clusters 2 and 3 require two positive and two negative weights, respectively, while cluster 1 needs weights of both signs. Let us observe that each of the three computing neurons has only two associated weights and four or five resistors. Since we have some freedom of choice here, all but two weight-determining resistor values will thus need to be assumed. Design assumptions for further considerations are  $R_F = r_0 = 1 \text{ k}\Omega$ .

**Cluster 1 Neuron:** Using Equations (9.10) for  $n = p = 1$ , we obtain from (9.12)

$$\begin{cases} -R_F G_1 = -2.897 \\ (1 + R_F G_1) \frac{g_1}{g_0 + g_1} = 0.776 \end{cases} \quad (9.13a)$$

Letting  $R_F = r_0 = 1/g_0 = 1 \text{ k}\Omega$  simplifies the expressions (9.13a) to the form

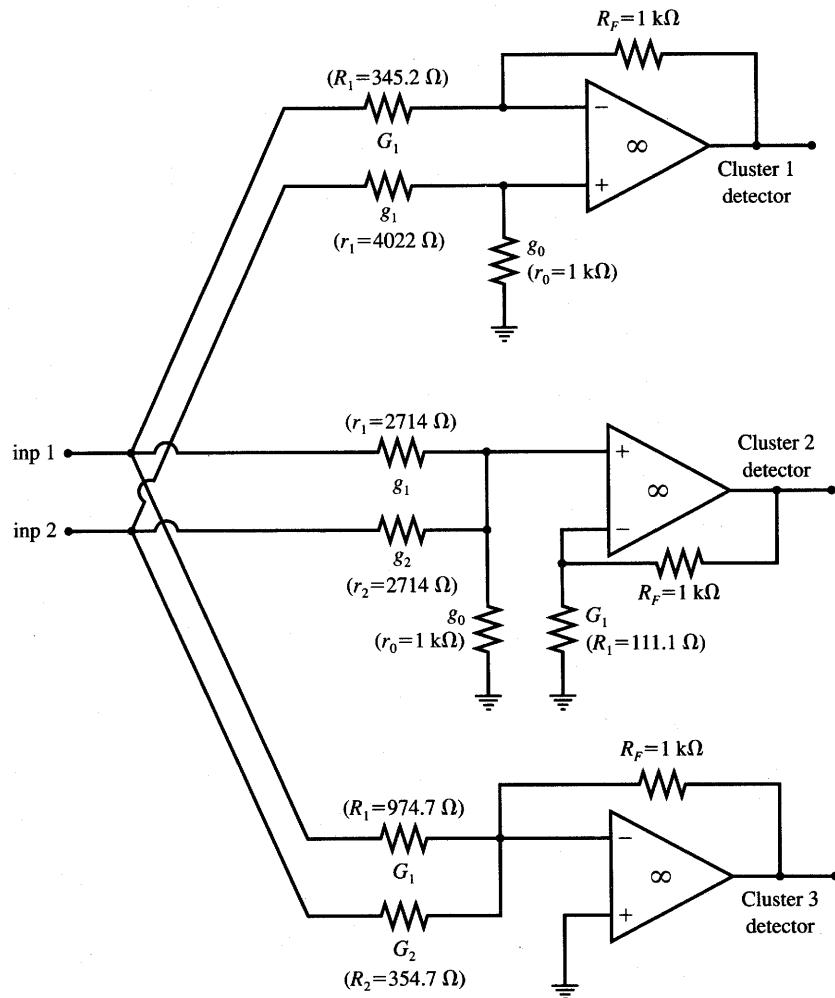
$$\begin{cases} G_1 \cdot 10^3 = 2.897 \\ 3.897 \frac{g_1}{10^{-3} + g_1} = 0.776 \end{cases} \quad (9.13b)$$

The reader can verify that  $R_1 = 1/G_1 = 345.2 \Omega$ , and  $r_1 = 1/g_1 = 4022 \Omega$  are the solutions of (9.13b). This completes the design of the neuron that detects cluster 1.

**Cluster 2 Neuron:** Using Equations (9.10) for  $p = 2$ , we obtain from (9.12)

$$\begin{cases} (1 + R_F G_1) \frac{g_1}{g_0 + g_1 + g_2} = 2.121 \\ (1 + R_F G_1) \frac{g_2}{g_0 + g_1 + g_2} = 2.121 \end{cases} \quad (9.14)$$

Note that a dummy  $G_1$  conductance between the inverting node of the operational amplifier and ground is used here to provide the fixed gain  $f/v_+$  of the circuit. This gain is now  $1 + R_F G_1$ . Since the ratios on the left side



**Figure 9.7** Cluster detecting network with the three resistor-operational amplifiers neurons of Example 9.1.

of (9.14) are smaller than unity,  $1 + R_F G_1$  should be chosen in the order of tens so that the positive weights can be in the required range. Since  $R_F = 1 \text{ k}\Omega$ , then for  $1 + R_F G_1 = 10$  we obtain  $R_1 = 1/G_1 = 111.1 \Omega$ , and (9.14) simplifies as follows

$$\begin{cases} \frac{g_1}{g_0 + g_1 + g_2} = 0.2121 \\ \frac{g_2}{g_0 + g_1 + g_2} = 0.2121 \end{cases} \quad (9.15)$$

Since we assume  $r_0 = 1/g_0 = 1 \text{ k}\Omega$ , the solutions of (9.15) are obtained as follows:

$$r_1 = r_2 = \frac{1}{g_1} = \frac{1}{g_2} = 2.714 \text{ k}\Omega$$

This completes the design of a neuron that detects cluster 2.

**Cluster 3 Neuron:** Using the first of two equations (9.10) for case  $n = 2$  we have

$$\begin{cases} R_F G_1 = 1.026 \\ R_F G_2 = 2.819 \end{cases} \quad (9.16)$$

Again using  $R_F = 1 \text{ k}\Omega$ , the remaining resistances of the network can be computed from (9.16) as  $R_1 = 1/G_1 = 974.6 \Omega$ , and  $R_2 = 1/G_2 = 354.7 \Omega$ . This completes the design of the entire cluster detecting network. ■

The techniques described in this subsection apply equally both to neuron circuits with zero and nonzero bias levels. Assume that a fixed bias of nonzero value is needed for a neuron. This can be achieved by augmenting the input vector  $\mathbf{x}$  by a single fixed component  $x_{n+1}$ . This component is used to produce the bias signal and can be called the reference voltage. Stage 1 of the circuit from Figure 9.8(a) depicts an example neuron with the weights of values  $-w_i$ , for  $i = 1, 2, \dots, n$ , and the nonzero bias produced by the input  $x_{n+1}$ . Applying relationships (9.10) and (9.11) allows for computation of the output voltage of the first stage as

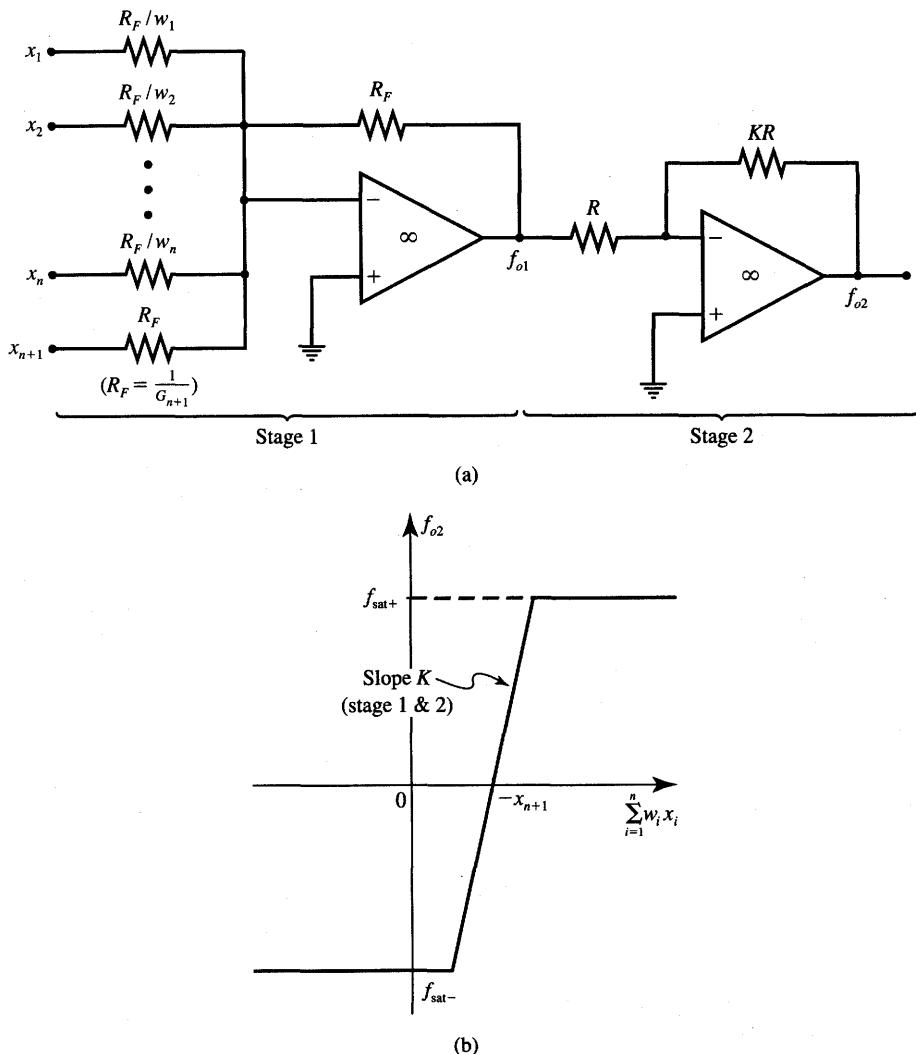
$$f_{o1}(x_1, \dots, x_{n+1}) = \sum_{i=1}^n (-w_i)x_i - x_{n+1} \quad (9.17)$$

As for previous configurations, the stage from Figure 9.8(a) also works with the linear transfer characteristics of unity slope, provided  $f_{\text{sat}-} < f_{o1} < f_{\text{sat}+}$ . Otherwise, the stage saturates at  $f_{\text{sat}-}$  or  $f_{\text{sat}+}$  for the output voltages that attempt to exceed the linear range of an operational amplifier.

In cases for which a high-gain neuron is needed, the unity-gain neurons, examples of which are shown in Figures 9.5, 9.6, and 9.7, should be connected in cascade with a high-gain amplifier. Stage 2 of the circuit from Figure 9.8(a) provides the gain of value  $-K$ , but it also reverses the polarity of all weights. The overall transfer characteristics, or activation function of this two-stage neuron, is given by the following formula:

$$f_{o2}(\mathbf{x}) = K \left( \sum_{i=1}^n w_i x_i + x_{n+1} \right) \quad (9.18)$$

Again, whenever the value  $f_{o2}(\mathbf{x})$  computed from (9.18) falls outside the range  $(f_{\text{sat}-}, f_{\text{sat}+})$ , the neuron's output saturates at either  $f_{\text{sat}-}$ , or  $f_{\text{sat}+}$ . The resulting transfer characteristics of the two-stage neuron is shown in Figure 9.8b. It can



**Figure 9.8** Neuron of gain  $K$  with positive weights and nonzero bias value: (a) circuit diagram and (b) activation function versus  $\sum_{i=1}^n w_i x_i$ .

be seen that the neuron's bias, or threshold, value is  $T$  and has now an opposite sign with respect to the reference voltage  $x_{n+1}$ , since we have  $T = -x_{n+1}$ .

In this subsection we introduced the simple electronic model of a processing neural node. It is a node that sums a number of weighted input voltages and produces an output voltage that is a nonlinear function of that sum. The nonlinear function of a processing node plays the same role as an activation function of a neuron. The activation functions implemented thus far with resistors and

operational amplifiers have been piecewise linear with controlled slope near the origin, and having symmetrical saturation for large activation levels.

The resistor-operational amplifier circuits introduced in this section rather lucidly illustrate the basic design concepts of the neural network computing nodes. Nevertheless, they are of limited practical significance due to several important reasons. Some of them are limitations of the shape of the produced transfer characteristics, excessive power consumption, and large area occupied by discrete resistors and operational amplifiers. In addition, the main problem with the electronic neural networks just discussed is that weight values are encapsulated in the values of resistances of fixed, discrete resistors. Such resistors are not easily adjustable or controllable. As a consequence, they can be used neither for learning, nor can they be reused for recall when another task needs to be solved.

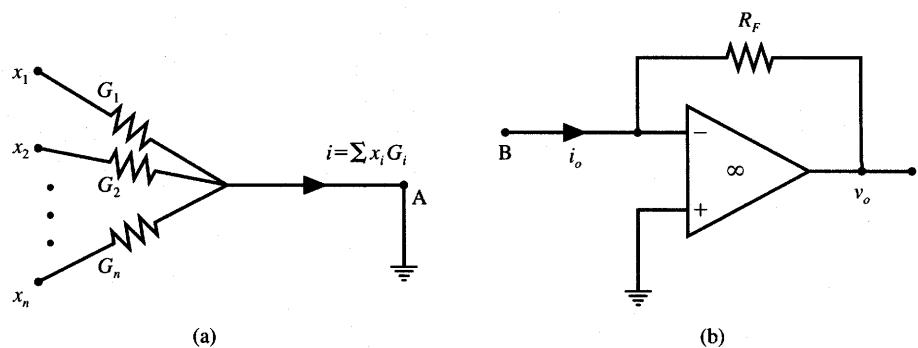
As will be shown in the subsequent sections, the discussed circuits from Figures 9.5 through 9.8 play an important role in more advanced neural network implementations. More sophisticated, integrated circuit neural networks are often based on the simple principles just illustrated with resistive networks and operational amplifiers. In fact, conversion of input voltages into weighted currents highlighted in (9.2) is one of the most frequent intermediate operations encountered in electronic neural network implementations. Another commonly used technique is the summation of currents into a proportional voltage response with nonlinear saturation. More specialized design techniques for electronic neural processing nodes are discussed below in more detail.

## 9.2

### INTEGRATED CIRCUIT SYNAPTIC CONNECTIONS

Artificial neural networks usually contain a very large number of synaptic connections and much fewer processing neurons. As mentioned before, conventional resistors can be used to produce synaptic connections in the form of weights. Nonlinear voltage amplifiers can be used to accomplish weighted summation of input voltages into the activation value, and also produce nonlinear mapping of an activation into the output voltage.

The central problem of implementation of artificial neural networks is to make weights that are continuously adjustable, preferably in response to an analog control signal. In addition, weights should not require many transistors for their implementation and for control of their values. Ideally, weights should also be capable of learning under a specified learning rule. We should realize from previous study that the requirement of adjustable weights is easier to implement than this one of learning weights. Indeed, a learning weight must not only be adjustable but it also must respond to a learning signal according to the training rule it implements.



**Figure 9.9** Functional blocks of the basic circuit of a neuron with fixed weights: (a) resistive synaptic connections and (b) current-to-voltage converter.

*Adjustable weights* are weights that can be modified off-line as a result of the learning process. *Learning weights* are understood as weights that modify by themselves on-line during learning. The learning signal in such cases is provided to the weight circuitry that allows the weight modification. The learning weight modification can be an outcome of either the current continuous or step-like learning. Usually, weight modification depends on the neuron's output signal, but it may also involve other quantities.

Neurobiology solves an on-line learning problem by using chemical modifications of ionic conductances, and by constructing a synapse that is a complex signal transmitting device with history-dependent properties. Making artificial neural network connections at least partially as clever as the natural connections could have great impact on the effectiveness of analog implementations of neural networks (Hopfield 1990). Let us focus, however, on studying microelectronic techniques that are presently available rather than on imitating neurobiological processes that are too involved to be duplicated in present-day integrated circuits.

As discussed in the previous section, inputs to the electronic model of a neural network are assumed to be voltages  $x_i$ , for  $i = 1, 2, \dots, n$ . To produce a weighted sum of inputs, the input voltages are first converted into branch currents  $x_i G_i$  as shown in Figure 9.9(a). The values  $G_i$  are branch conductances of resistors connected between the  $i$ 'th input and ground. The branch currents must then be summed and the sum converted back to voltage using, for example, a current-to-voltage converter depicted in Figure 9.9(b). The converter output voltage  $v_o$  is equal

$$v_o = -i_o R_F \quad (9.19)$$

To implement a neural processing node following the description above, two functional blocks from Figure 9.9(a) and (b) must be connected in cascade. Node

*A* needs to be connected to virtual ground node *B* instead of to the actual ground node as in Figure 9.9(a). This connection enforces  $i = i_o$ , and the output voltage value becomes as expressed by (9.3) and (9.4). The two connected networks yield a basic neural processing node identical to the one introduced earlier in Figure 9.5(a).

In this section we will study analysis and design of adjustable microelectronic synaptic connections. Our focus is on voltage-controlled resistors that implement such connections. We will see in the following discussion that both analog and digital control can be used to produce variations of weight values. The performance of learning synapses that are trained according to the Hebbian rule will also be studied.

### Voltage-controlled Weights

The resistive synaptic connection circuit with electrically tunable weights can be developed using the basic electronic neuron concept of Figure 9.9. Assume that there is a voltage-controlled switch available with performance as depicted in Figure 9.10. Figure 9.10(a) shows a symbol of the switch, and the typical characteristics of a switch are displayed in Figure 9.10(b). The switch can operate between an ON state (short-circuit),  $V_{AB} = 0$ , and an OFF state (open-circuit),  $I_{AB} = 0$ . If neither of the two states applies, the switch assumes the resistance between terminals *A* and *B*, equal to  $R_{AB}$ . The value of  $R_{AB}$  can be made dependent on the controlling voltage  $V_C$ . It can be seen from Figure 9.10(b) that  $R_{AB}$  decreases with the increase of the control voltage  $V_C$ .

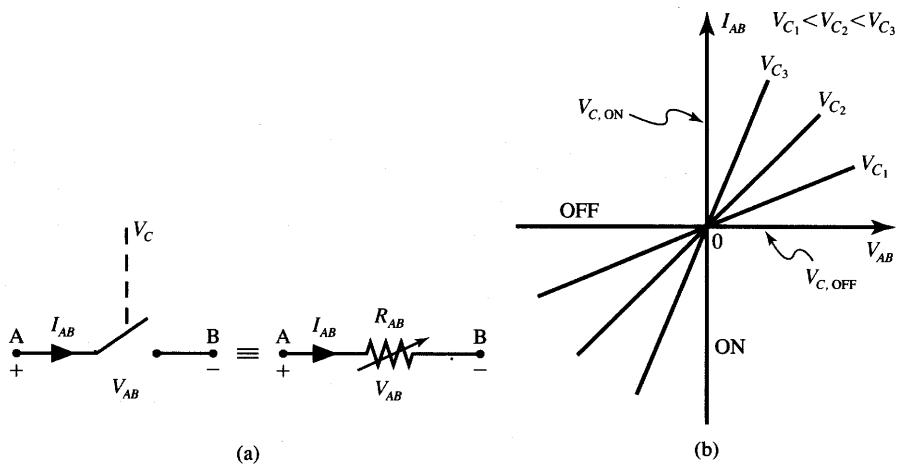
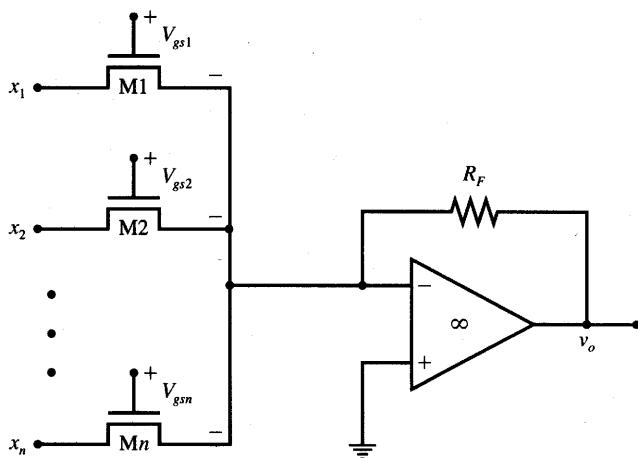


Figure 9.10 Voltage-controlled switch: (a) symbol and (b) graphical characteristics.



**Figure 9.11** Scalar product circuit with electrically tunable weights.

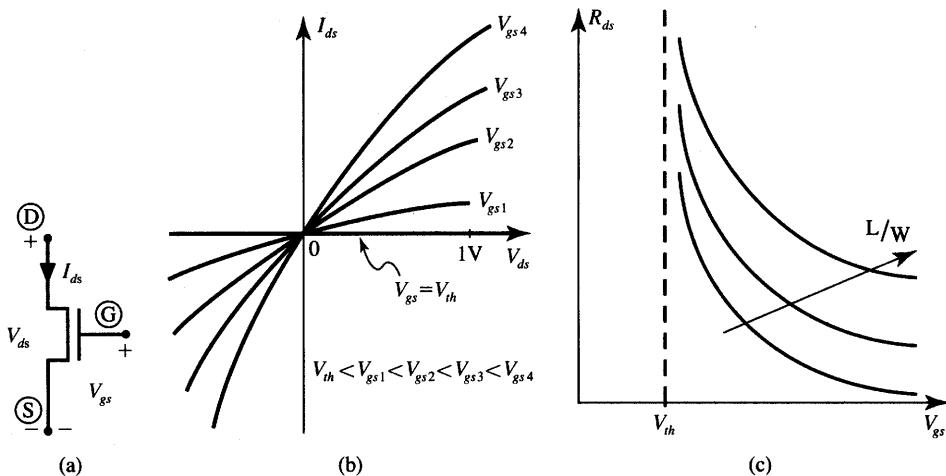
The voltage-controlled switch can be implemented using a single field-effect or MOS transistor operating in the resistive (ohmic, also called linear) region. Figure 9.11 shows a neuron with  $n$  synaptic connections, each of them consisting of a single enhancement mode NMOS transistor performing as a voltage-controlled switch. With reference to Figures 9.10 and 9.11, terminals *A* or *B* of the switch can be either the drain or source of the transistor since the device is electrically symmetric. Node *B* is the source for positive input voltage, otherwise node *A* becomes the source, so that the channel current always flows toward the source terminal. For this discussion, we assume that the source is at the virtual ground potential and input voltages are positive. The input voltage is thus applied at the drain terminal of an enhancement mode NMOS transistor. The channel resistance  $R_{ds}$ , or the resistance between the drain and source of the  $i$ 'th transistor, is controlled by the gate to source voltage  $V_{gs}$ .

Let us express the transistor channel resistance  $R_{ds}$  as a function of controlling voltage  $V_{gs}$ . The drain to source current,  $I_{ds}$ , for the transistor depicted in Figure 9.12(a) is given by the following expression (see Appendix)

$$I_{ds} = k' \frac{W}{L} \left[ (V_{gs} - V_{th})V_{ds} - \frac{V_{ds}^2}{2} \right] \quad (9.20)$$

for  $V_{ds} < V_{gs} - V_{th}$  (resistive region), where  $k'$  is the process transconductance (typically of value  $20 \mu\text{A}/\text{V}^2$ ),  $W$  and  $L$  are channel width and length, respectively, and  $V_{th}$  is the transistor threshold voltage (typically between 1 and 2.5 V for enhancement mode NMOS devices).

Figure 9.12(b) shows the output characteristics of an NMOS device operating in the resistive region. The characteristics depict  $I_{ds}(V_{ds})$  with the gate to



**Figure 9.12** Enhancement mode NMOS transistor as a voltage-controlled switch for weight tuning: (a) device symbol, (b) output characteristics in the resistive region, and (c) channel resistance versus controlling voltage.

source voltage  $V_{gs}$  as a parameter for each curve. Comparing the switch characteristics of Figure 9.10(b) with the transistor characteristics of Figure 9.12(b), the reader may notice that  $V_{C,OFF} = V_{th}$ . Voltage  $V_{C,ON}$  is not defined for the NMOS transistor configured as a switch, since  $R_{ds}$  is always of nonzero value. Noticeably, the first quadrant characteristics of the switch also extend into the third quadrant, which makes it possible to apply not only positive but also negative input voltages to the synaptic connection. If  $V_{ds} \ll V_{gs} - V_{th}$ , then the quadratic-type relation (9.20) becomes approximately linear in  $V_{ds}$  and we have

$$I_{ds} \cong k' \frac{W}{L} (V_{gs} - V_{th}) V_{ds} \quad (9.21a)$$

This linear approximation of (9.20) as in (9.21a) can be considered accurate for practical purposes for  $V_{ds} < 0.5(V_{gs} - V_{th})$ . For this condition to be fulfilled, the voltage-controlled resistance value  $R_{ds}$  equal to the ratio of  $V_{ds}$  to  $I_{ds}$  becomes from (9.21a)

$$R_{ds} \cong \frac{L}{k' W (V_{gs} - V_{th})} \quad (9.21b)$$

It can be seen that  $R_{ds}$  is controllable in a very wide range and  $V_{gs}$  can be used as the control voltage. As illustrated in Figure 9.12(c), the switch remains open for  $V_{gs} \leq V_{th}$ , and then with  $V_{gs}$  increasing above  $V_{th}$ , the  $R_{ds}$  value drops quickly. An additional advantage of this simple configuration is that the value of  $R_{ds}$  can be scaled by the choice of the channel length to width ratio,  $L/W$ . Obviously, for large  $L/W$  values, the channel resistance is limited to large values. This becomes the case for so-called “long channel” or “long transistors.”

When applying this controlled switch as a tunable resistance, keep in mind that, ideally, the linear resistances of values as in (9.21b) are only obtainable at the origin  $I_{ds} = V_{ds} = 0$ . Otherwise, slight nonlinearity due to the quadratic term in (9.20) is always present, and the weights can only be approximately considered as linear voltage-controlled resistances. This should not be a serious problem, however, since no great precision of weight values is usually required for a neural network operating in the recall mode. Also, only positive resistances can be produced by the discussed switch and thus only weights of one sign can be implemented by the circuit configuration of Figure 9.11.

One possible way of producing both positive and negative weights is to use an additional network of positive synaptic weights in the form of a voltage divider at the noninverting input of an operational amplifier. This concept was illustrated earlier in Figure 9.6 for discrete resistor circuitry. Another possible solution is to use complementary output operational amplifiers, which would provide both positive and negative weight values by producing output voltages of both signs.

At this point we have gained insight into the analysis and design of neurons' weights, which can be adapted electrically by means of a control voltage. In addition to this very desirable feature of tunability, however, voltage-controlled weight values should be stored to ensure proper synapse operation over a period of time. This issue will be covered following the example below which focuses on analysis of a neuron circuit with electrically controlled weights.

## EXAMPLE 9.2

Let us analyze the neuron circuit illustrated in Figure 9.11 with three weights. Following are the data on transistors, control voltages, and the circuit elements used to design this circuit.

*Transistors:*  $k' = 20 \mu\text{A}/\text{V}^2$  (for all transistors)

$V_{th} = 1 \text{ V}$  (for all transistors)

$$\left(\frac{L}{W}\right)_1 = 1, \left(\frac{L}{W}\right)_2 = \frac{1}{3}, \left(\frac{L}{W}\right)_3 = \frac{1}{3}$$

*Control Voltages:*  $V_{gs1} = 5 \text{ V}$ ,  $V_{gs2} = 3 \text{ V}$ ,  $V_{gs3} = 3.75 \text{ V}$

*Circuit:*  $R_F = 100 \text{ k}\Omega$

Our objective is to find the neuron's output voltage  $v_o$  as a function of  $\mathbf{x}$  where  $\mathbf{x} = [x_1 \ x_2 \ x_3]^T$ .

Let us initially assume that the drain-source resistances of NMOS transistors are exactly linear because  $V_{dsi} \ll V_{gsi} - V_{th}$ , for  $i = 1, 2, 3$ . This would require that a small range of input voltages be maintained. With

this assumption, the transistors are in resistive regions and their channel resistances  $R_{dsi}$ , for  $i = 1, 2, 3$ , can be determined from (9.21b) as follows

$$R_{dsi} = \frac{(L/W)_i}{20 \cdot 10^{-6}(V_{gsi} - 1)}, \quad \text{for } i = 1, 2, 3 \quad (9.22a)$$

Performing calculations as in (9.22a) for the specified network parameters yields the following channel resistance values

$$R_{ds1} = 12.5 \text{ k}\Omega$$

$$R_{ds2} \approx 8.33 \text{ k}\Omega$$

$$R_{ds3} \approx 6.06 \text{ k}\Omega$$

Considering that the feedback resistance  $R_F$  is 100 kΩ and that in the linear range of operation we have

$$v_o = -R_F \left( \frac{x_1}{R_{ds1}} + \frac{x_2}{R_{ds2}} + \frac{x_3}{R_{ds3}} \right) \quad (9.22b)$$

then the resulting weight values are as follows:

$$w_1 = -8, \quad w_2 = -12, \quad w_3 = -16.5$$

Let us now verify the accuracy of the computed weights and determine a reasonable operating voltage range for the circuit. Limiting the range of input voltages so that  $V_{dsi} \leq 0.5(V_{gsi} - V_{th})$ , the maximum values of input voltages for linear approximation of  $R_{ds}$  as in (9.21b) become equal:

$$V_{dsi} = \frac{1}{2}(V_{gsi} - 1) \quad (9.22c)$$

Given the specified control voltages,  $V_{gsi}$ , the values of maximum input voltage,  $V_{dsi}$ , are 2, 1, and 1.375 V for each transistor, respectively. It is interesting to compute the actual values of implemented weights considering the condition that the input voltage has reached the maximum level beyond which the channel resistance cannot be considered linear.

For such a case,  $R_{ds}$  cannot be computed from a closed-form expression (9.21b) since its value also depends on the drain current. Using expression (9.20) we thus have for the transistor  $M1$

$$I_{ds1} = 20 \cdot 10^{-6}(1) \left( 4 \cdot 2 - \frac{2^2}{2} \right) \text{ A}$$

which yields

$$I_{ds1} = 120 \mu\text{A}$$

For the maximum input voltage level of value  $V_{ds1} = 2 \text{ V}$ , the resulting actual channel resistance is  $R'_{ds1}$

$$R'_{ds1} \approx 16.6 \text{ k}\Omega$$

It can be seen that the discrepancy between the newly computed accurate value of the channel resistance  $R'_{ds1}$  and the original approximate value of  $R_{ds1} = 12.5 \text{ k}\Omega$  is 33.3%. This indicates that for the working region of input voltages  $V_{ds1} < 2 \text{ V}$ , the actual channel resistance is between 12.5 and  $16.6 \text{ k}\Omega$ , with the resistance error ranging between 0 and 33.3%. In fact, the precise value of the resistance depends on the input voltage value.

The reader can easily verify that similar analysis results in the following actual resistance values in the bordercase of input voltages:

$$R'_{ds2} \approx 11.1 \text{ k}\Omega \quad \text{for } V_{ds2} = 1 \text{ V}$$

$$R'_{ds3} \approx 8.08 \text{ k}\Omega \quad \text{for } V_{ds3} = 1.375 \text{ V}$$

The actual weight values implemented by the circuit for the maximum level of inputs are now resulting in fairly different values than originally specified for this neuron which are valid for very small input voltages. The output voltage would be now of the value

$$v_o = -R_F \left( \frac{x_1}{16.6 \text{ k}\Omega} + \frac{x_2}{11.1 \text{ k}\Omega} + \frac{x_3}{8.08 \text{ k}\Omega} \right)$$

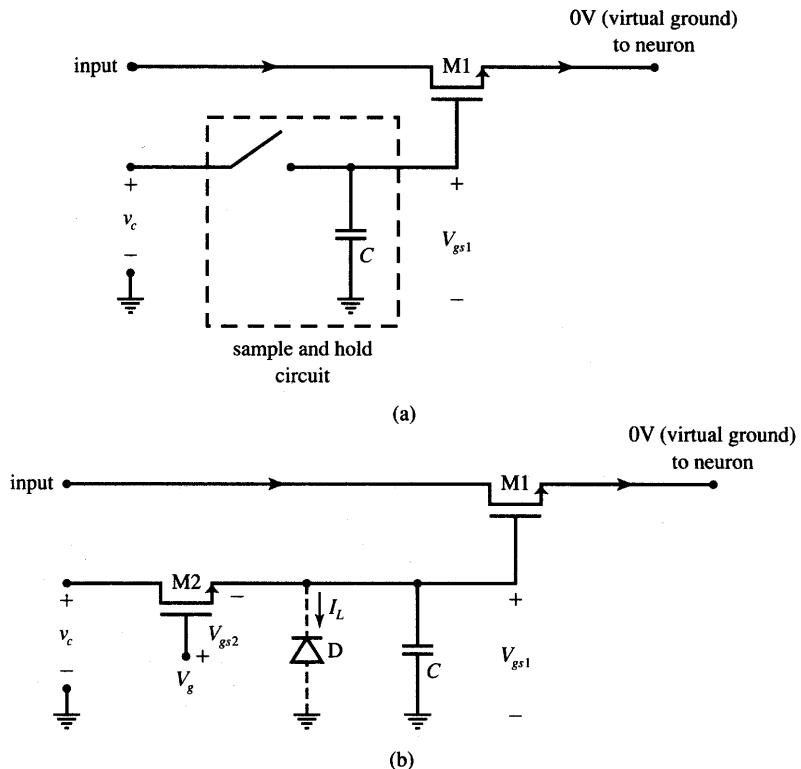
and the weights implemented by this network are  $w'_1 = -6$ ,  $w'_2 = -9$ , and  $w'_3 = -12.37$ . This example demonstrates that in order to preserve the linearity of transistor operation and to produce constant values of weights, the input voltages should be kept well below the upper limit of channel resistance linearity, which has been set at  $x_{imax} = 0.5(V_{gsi} - V_{th})$ , for  $i = 1, 2, 3$ . ■

### Analog Storage of Adjustable Weights

Storage of adjustable analog weights is one of the most important problems faced in analog implementation of artificial neural systems. The storage form can be analog; it would thus have the properties of an analog memory cell. Storage cells should also be as small as possible since one memory cell would be required per every weight (Vittoz et al. 1991) and the overall size of a neural network is determined primarily by its number of weights.

A commonly used storage technique is based on storing charge across a capacitive circuit element. Note that voltage-controlled weights outlined in the previous section use the gate of an NMOS transistor as the weight control terminal. Since no current is needed to drive the transistors' gates, the simple, natural method to store the weight value is to store it as a voltage across the gate capacitance (Tsividis and Anastassiou, 1987). The gate capacitance can be either intrinsic or intentionally added to the circuit to enhance its storage capability.

Figure 9.13(a) illustrates the concept of storing a weight as a voltage across a capacitor  $C$  at the gate of transistor  $M1$ . The weight control voltage  $v_c$  that needs to be stored is sampled by means of a sample-and-hold circuit. Transistor  $M2$  shown in Figure 9.13(b) functions as a sample-and-hold gate and provides



**Figure 9.13** Elementary sample-and-hold circuit for weight storage: (a) circuit with sample-and-hold switch and (b) all transistor implementation.

the ON/OFF operation of the switch. The voltage  $V_{gs1}$  directly controls the channel resistance of transistor  $M1$ .

For an ideal operation of switch  $M2$ ,  $V_{gs1}$  exactly tracks the weight control voltage  $v_c$  when the switch is at least partially ON. This corresponds to  $M2$  being conductive and  $V_{gs1} \approx v_c$ . To ensure the voltage tracking, the switch control voltage  $V_g$  must be well above the level of the weight control voltage  $v_c$ . This is due to the fact that the voltage actually controlling the gate of the switch,  $V_{gs2}$ , has to be in excess of  $V_{th2}$ . This condition ensures that the switch  $M2$  remains conductive between the drain and source. By inspection of Figure 9.13(b) we obtain

$$V_g = V_{gs1} + V_{gs2} \quad (9.23a)$$

For both  $M1$  and  $M2$  conducting, the following conditions must be met, respectively:

$$\begin{aligned} V_{gs1} &> V_{th1} \\ V_{gs2} &> V_{th2} \end{aligned} \quad (9.23b)$$

Assuming that threshold voltages of both transistors are identical and equal to  $V_{th}$ , we obtain from (9.23a)

$$V_g > 2V_{th} \quad (9.23c)$$

This conclusion supports an intuitively obvious statement that both transistors must operate with their gate-to-source voltages above  $V_{th}$ . In such a case,  $M2$  conducts the current (sampling mode) and adjusts the amount of charge that is to be stored in the capacitance  $C$ . The voltage across the capacitance,  $V_{gs1}$ , in turn controls the weight value. To ensure the analog storage of the weight value (hold mode), the open-circuit switch condition  $V_{gs2} < V_{th2}$  must subsequently be enforced to trap the charge at the capacitor.

In the absence of a positive pulse  $V_g$  at the gate of the switch, the capacitor  $C$  holds the charge, which has been stored during the most recent conductive (sampling) period of  $M2$ . When the switch is OFF and the drain and source of  $M2$  are disconnected, only a small leakage current  $I_L$  flows through a reverse-biased diffusion-to-bulk junction of  $M2$  (these regions are not shown in Figure 9.13(b)). As a result, the storage capacitor will slowly discharge. For today's conventional MOS technologies, the retention, or storage, time is of the order of several milliseconds. To store the weight control voltage for longer periods, the charge stored at the capacitance  $C$  needs to be refreshed once every few milliseconds. This leakage and refresh phenomena here are analogous to the phenomena encountered in the dynamical memory RAM cell, which also needs to be refreshed periodically.

For a number of weight values to be electrically tuned, one control voltage generator providing voltages  $v_c$  can be time-shared by many neurons. As each switch closes in turn so that sampling can take place,  $v_c$  assumes an appropriate value for charging the corresponding capacitor. Note that the capacitors of the various neurons and weights can be accessed using well-known addressing schemes from semiconductor memories. The basic difference is that the data stored now are in the form of analog control voltage rather than in the binary form (Tsividis and Anastassiou, 1987).

Once the appropriate control voltages have been stored on the gate capacitors, the network is ready to perform recall. Upon completion of recall, the storage capacitors can be accessed again, either to refresh the existing weight values or to impose new weight values that may result as an outcome of the incremental learning step.

The natural decay of a capacitor's charge that represents the learned weight value is one of the inherent limitations of the analog storage of weights. Essentially, this is the implementation problem shared by all analog nonvolatile memories. Advanced circuit techniques exist that extend the storage time of analog microelectronic memory through either additional special circuitry or special refresh signals. The reader is referred to the technical references for more details on special refresh circuitry (Vittoz et al. 1991). Other, alternative techniques for

analog memory for weight storage make use of electrically programmable non-volatile memory cells. Electrically programmable memory cells will be examined in Section 9.4.

---

### Digitally Programmable Weights

As stressed earlier in this chapter, one of the most desirable features of neuron's weights is their tunability. In addition to the tunability of weights with the use of an analog voltage control signal, weights implemented as integrated circuits can be controlled digitally. This technique for weight tuning allows for increased flexibility compared to the analog tuning mode due to the fact that weights can be both programmed and stored digitally. Weight control data, arranged in digital control words, are easily stored in digital registers. Understandably, the accuracy of the weight setting in this mode depends mainly on the length of the control word. This accuracy limitation is common for any digital-to-analog conversion task and is known as the *finite resolution and conversion error*. Moreover, the accuracy of the subsequent analog signal processing is also of importance. The key consideration here is that once the weight values have been set digitally, all subsequent network computation during the recall mode is performed with analog signals. This involves multiplication, addition, and nonlinear mapping by the neuron.

In addition to flexibility and accuracy, the digital tunability of weights provides a convenient interface between the training and recall mode of operation of the neural hardware. Since learning often requires considerable accuracy, it is usually performed off-line and with the use of high-precision digital computation. Weight values computed as a final result of simulated off-line training are subsequently programmed on the neural network chip using the digital-to-analog interface. In addition, digital weight storage in RAM cells or registers of a digital computer is easily available.

A block diagram of a circuit producing a single digitally programmable weight and employing a multiplying digital-to-analog converter (MDAC) is shown in Figure 9.14. MDAC circuits are used in many electronic circuits that require the use of digitally tunable resistances and a flexible digital-to-analog interface (Zurada 1981). The converter circuit shown in the figure performs the decomposition of an input voltage  $x_i$  into binary weighted currents  $i_1, i_2, \dots, i_n$ , which are proportional to  $x_i$ . The binary weighted currents are then summed and converted back into the output voltage in the current-to-voltage converter identical to the one depicted in Figure 9.9(b).

Assume that register bits of value 0 or 1 represent the entries of the binary control word  $b_N b_{N-1} \dots b_1$ , where the most-significant bit (MSB) is that corresponding to  $b_N$ , while the least-significant bit (LSB) corresponds to  $b_1$ . Although a digital word can be presented in a variety of codes, the one assumed

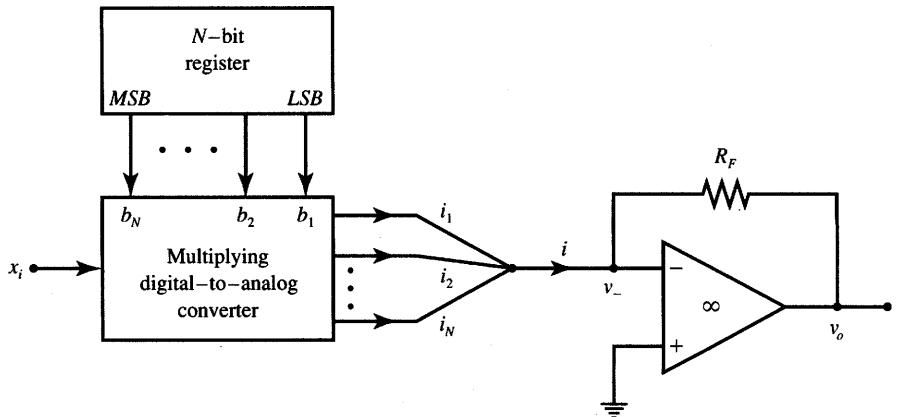


Figure 9.14 Block diagram of a multiplying digital-to-analog converter (MDAC).

here for expressing the weight value is simply its pure binary representation. Figure 9.15(a) shows the circuit diagram of a *weighted-resistor MDAC* (Sedra and Smith 1991). Successive resistances in the network are inversely proportional to the numerical significance of an appropriate bit. It can be seen that  $b_i$  of value 0 directs the weighted current to ground by positioning the respective switch  $S_i$  in its left-hand position. Switches for which  $b_i = 1$  direct the binary weighted currents to the summing node *B*. Let us note that the currents can be considered as binary weighted since node *B* serves as a virtual ground. The expression for current at the input of the summing current-to-voltage converter becomes

$$I = \frac{x_i}{R} b_N + \frac{x_i}{2R} b_{N-1} + \dots + \frac{x_i}{2^{N-1}R} b_1 \quad (9.24a)$$

This expression simplifies to

$$I = \frac{2}{R} x_i \left( \frac{b_N}{2} + \frac{b_{N-1}}{4} + \dots + \frac{b_1}{2^N} \right) \quad (9.24b)$$

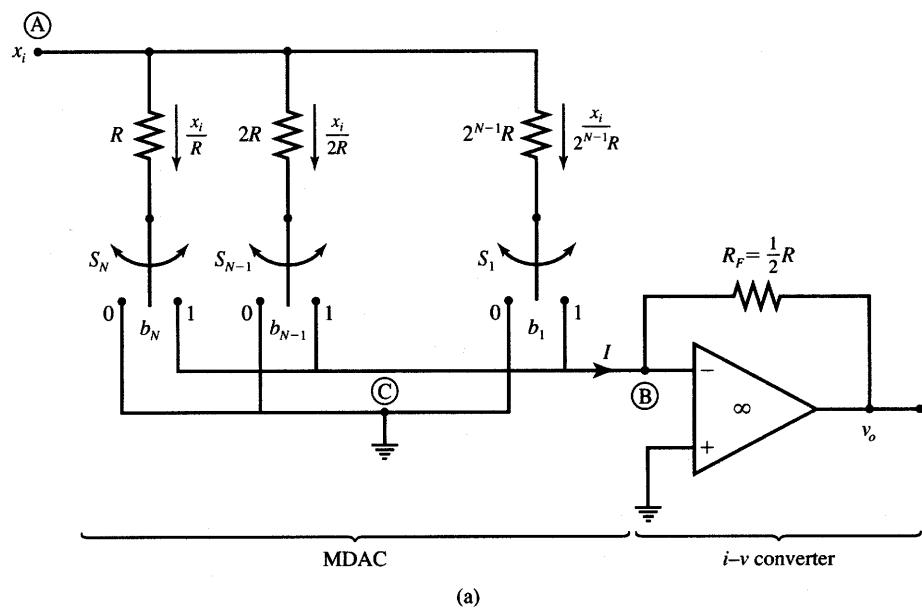
Let us define the digital control word *D* as

$$D \triangleq \frac{b_1}{2^N} + \frac{b_2}{2^{N-1}} + \dots + \frac{b_N}{2}$$

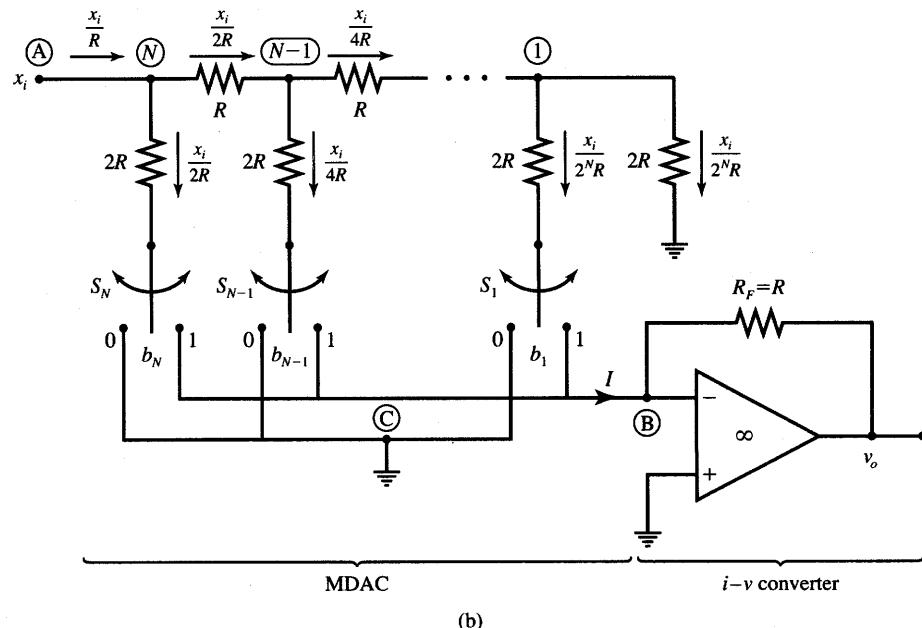
and rewrite (9.24) in a more concise form:

$$I = \frac{2}{R} x_i D \quad (9.25)$$

Note that *D* can take values  $0, 1/2^N, 1/2^{N-1}, \dots, 1 - 1/2^N$ . These values uniformly cover the range  $[0, 1 - 1/2^N]$  with the resolution  $2^{-N}$ . The resolution, for instance, is about 0.1% for a 10-bit converter, but it is only 6.25% for a four-bit converter. Depending on the scaling required, different *R<sub>F</sub>* values can be used in the feedback loop of the current-to-voltage converter. Assuming  $R_F = (1/2)R$ ,



(a)



(b)

**Figure 9.15** Circuit diagram of a digitally programmable weight: (a) using a weighted-resistor MDAC and (b) using a ladder-resistor MDAC.

we obtain from (9.25) and (9.19)

$$v_o = -Dv_i \quad (9.26)$$

The neuron's output voltage (9.26) is the product of the digital control word  $D$  representing the weight value and of the analog input voltage. That is why this particular configuration can be referred to as a *programmable attenuator*. However, by using appropriate scaling and selecting a suitable  $R_F$  value, the range of weight values can be adjusted and increased above the unity value according to specific design requirements. The value of  $R_F$  must be chosen in excess of  $R/2$  if the weights of magnitudes larger than unity are required. Similar to the circuits described earlier in this chapter, however, a large  $R_F$  causes saturation of  $v_o$  at low levels of currents  $I$ . In such cases, the neuron's response will approach the bipolar binary activation function and make it resemble the hard-limiting response of the threshold logic unit.

Figure 9.15(b) shows an alternative converter circuit called a *ladder-resistor MDAC* (Sedra and Smith 1991). The network operates on the same principle of producing binary weighted currents. Instead of binary weighted resistances, however, now a ladder network of  $R - 2R$  resistors is used as a current splitting structure. We observe from the figure that at any of the ladder nodes, the resistance is  $2R$  looking to the right from each node numbered  $1, 2, \dots, N$ . This is due to the fact that the horizontal resistance  $R$  of the ladder adds with a parallel combination of two resistances of value  $2R$ , each being connected to ground. Thus, such a series connection of  $R$ , with  $2R$  parallel to  $2R$ , yields the resistance  $2R$ . This particular property applies at each of the nodes  $2, 3, \dots, N$ .

As a result, the current entering each node from the left splits into two equal parts, one flowing down toward the switch, the other half toward the adjacent node to the right. Current is halved this way in every upper node of the ladder network yielding the total number of splits equal to  $N$ . However, only the currents for which switches are in the right-hand positions contribute to the current  $I$  entering the node  $B$  of the current-to-voltage converter. The output current of the MDAC thus has the value

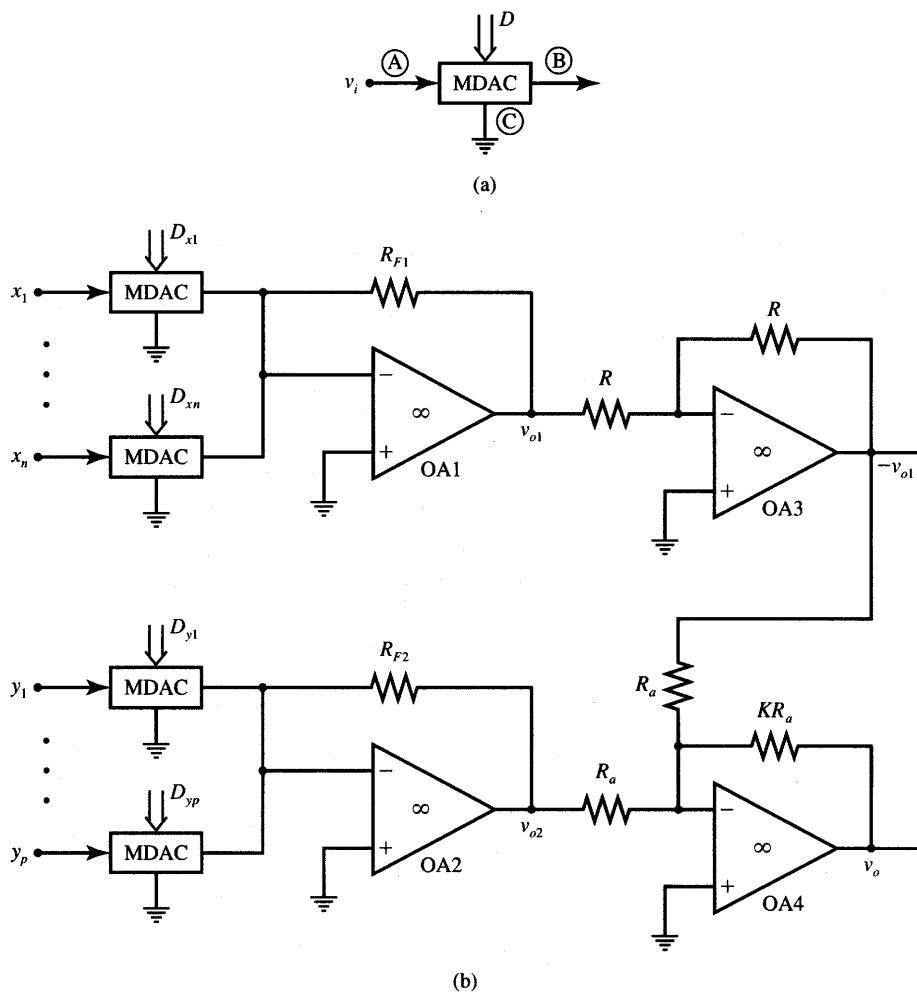
$$I = \frac{x_i}{R} \left( \frac{b_N}{2} + \frac{b_{N-1}}{4} + \dots + \frac{b_1}{2^N} \right) \quad (9.27a)$$

or

$$I = \frac{x_i D}{R} \quad (9.27b)$$

A formula identical to (9.26) for the output voltage in terms of input  $x_i$  can be obtained if we let the feedback resistance  $R_F$  of the operational amplifier be of value  $R$ .

Clearly, each MDAC configured as discussed and shown in Figure 9.15 can implement a single digitally controlled weight. The total of  $n + p$  MDACs needs to be connected to a current-to-voltage converter arrangement to implement the set of  $n$  negative and  $p$  positive digitally controlled weights. Assume that the



**Figure 9.16** Digitally controlled neuron with programmable bipolar weights: (a) MDAC symbol and (b) circuit diagram for  $n$  negative and  $p$  positive weights of a single neuron.

MDAC is represented in the form of a block diagram with the signal input voltage at node A, converter output at node B, and the ground terminal at node C as shown in Figure 9.16(a). An example neuron's circuit diagram can take the form of Figure 9.16(b). The neuron shown in the figure implements  $n$  negative and  $p$  positive weights. Operational amplifiers  $OA1$  and  $OA2$  produce output voltages as a function of both inputs and digital control words. We thus have for the output voltages

$$v_{o1} = -(x_1D_{x1} + x_2D_{x2} + \dots + x_nD_{xn}) \quad (9.28a)$$

and

$$v_{o2} = -(y_1 D_{y1} + y_2 D_{y2} + \dots + y_p D_{yp})$$

or, briefly

$$\begin{aligned} v_{o1} &= - \sum_{i=1}^n x_i D_{xi} \\ v_{o2} &= - \sum_{i=1}^p y_i D_{yi} \end{aligned} \quad (9.28b)$$

Operational amplifier *OA3* serves as a unity-gain inverter. Linear combination of the two inputs  $-v_{o1}$  and  $v_{o2}$  is subsequently computed by the inverting amplifier of gain  $-K$  based on the operational amplifier *OA4*. We thus have for the output voltage

$$v_o = K(v_{o1} - v_{o2}) \quad (9.29a)$$

The substitution of (9.28) into (9.29a) yields the neuron's output voltage determined by  $n$  negative and  $p$  positive weights as follows:

$$v_o = - \sum_{i=1}^n KD_{xi}x_i + \sum_{i=1}^p KD_{yi}y_i \quad (9.29b)$$

Again, this neuron implements a linear activation function of identity value with weights  $-KD_{xi}$ , for  $i = 1, 2, \dots, n$ , and  $KD_{yi}$ , for  $i = 1, 2, \dots, p$ . If, however, the output voltage  $v_o$  computed from (9.29) falls outside the linear output range ( $f_{\text{sat}-}, f_{\text{sat}+}$ ) of an operational amplifier, then the neuron's output saturates at one of the levels  $f_{\text{sat}-}$  or  $f_{\text{sat}+}$ .

Both weighted-resistor and ladder-resistor MDACs can be implemented in integrated circuit technologies. Weighted-resistor MDACs require only as many individual resistors as bits in contrast to the ladder-resistor MDAC, which requires twice as many resistors. However, the spread of resistance values, or the ratio of the largest to the smallest resistance for weighted-resistor MDACs, is  $2^N$ . Fabrication of integrated circuit resistors with extreme yet precise resistance values encounters technological difficulties. This limits the use of this type of MDAC to usually up to 5 or 6 bits. The ladder-type converter avoids this difficulty of extreme resistance values and is more suitable for high-accuracy resistance fabrication.

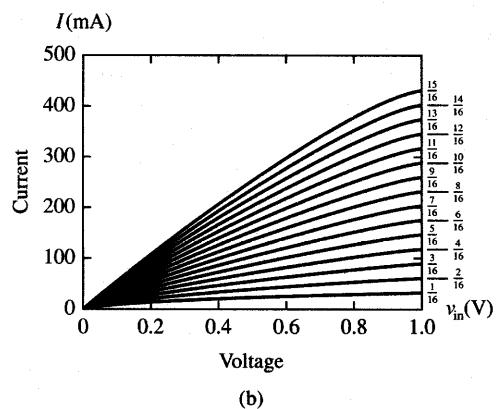
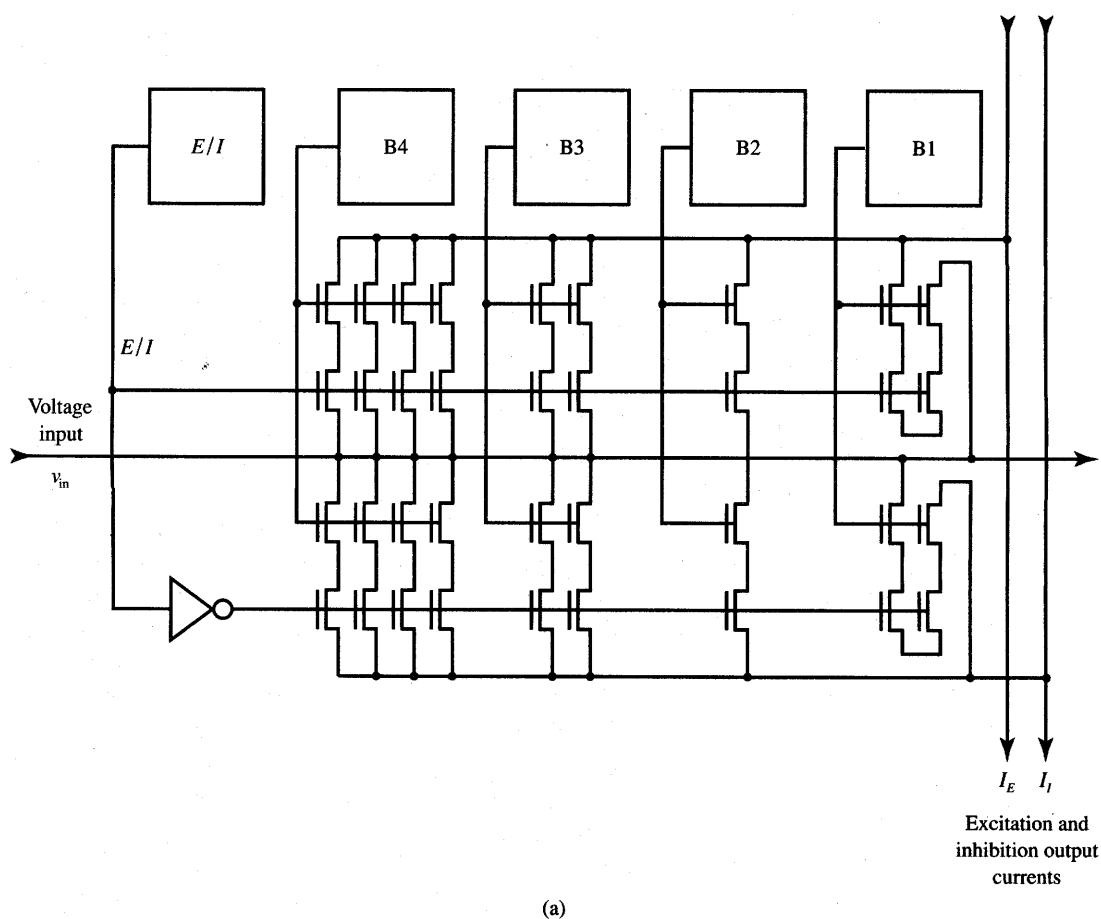
Let us look at an example of a neural network implementation that directly utilizes the described concepts. A neural network integrated circuit chip has been designed and fabricated using standard 3  $\mu\text{m}$  CMOS technology MDAC's (Raffel et al. 1989). The converters are 4-bit with the fifth bit controlling the sign of the implemented weight. Converters are addressed using a 10-bit address bus, thus allowing for handling of 1024 programmable weights. In addition, there are 32 analog signal inputs to the chip. Up to 16 neurons can be configured using the

chip circuitry when external amplifiers serving as current-to-voltage converters and neurons are added. The chip outputs 16 excitatory and 16 inhibitory currents to off-chip current-to-voltage converters. The outputs of the two converters are then fed into a differential amplifier-neuron providing analog levels compatible with the input levels on the chip. This particular feature enables building layered networks. The chip thus has a total of 79 analog and digital inputs and outputs, plus power supply lines. It is packaged in an 84-pin package. It occupies an active silicon area of 28 mm<sup>2</sup> and uses 28 500 transistors.

Let us look at some of the particularly interesting features of the chip. Figure 9.17(a) shows the circuit diagram of a pair of MDACs. The top converter produces an excitation current  $I_E$ , or positive weights; the other produces an inhibition current  $I_I$ , or negative weights. The  $E/I$  control bit determines which of the two converters is active. Weight voltages are encoded in 4-bit register  $B1$  through  $B4$ . Weighted resistors are built of NMOS transistors biased in the resistive mode and controlled at their gates by the binary voltage levels. This should ensure the resistors' linearity for the limited range of input voltages as expressed by formula (9.21). Here, instead of adjusting the  $R_{ds}$  of individual weighted resistances through design of a suitable ratio  $W/L$  of individual transistors, all transistors have been made of identical sizes. The resulting weighted resistance value is determined by the configuration of transistors controlled by each bit. It can be seen that  $B1$  controls two transistors in series; thus their total resistance is  $2R_{ds}$ . In contrast,  $B4$  controls four transistors connected in parallel so their total resistance is  $0.25R_{ds}$ . In series with the weight-producing transistors are switch transistors controlled by the  $E/I$  bit. This bit determines whether the excitation MDAC (top) or the inhibition MDAC (bottom) contributes the current to the output currents  $I_E$  and  $I_I$ , respectively.

Figure 9.17(b) depicts the family of measured output current characteristics of a single MDAC as a function of input voltage  $v_{in}$ . The values of the digital control word  $D$  serve as parameters for each curve shown. The lowest nonzero  $I(v_{in})$  characteristic is produced for  $D = 1/16$ ; the top one is for  $D = 15/16$ . To maintain the linearity of the weighted resistances of the MDAC, the converter input voltage  $v_{in}$  was kept below 1 V. The output current nonlinearity was found to be within 5% for this input range.

The described chip has been successfully tested in several applications (Rafel 1989). In the first experiment, a Hamming network has been designed using the approach outlined in Section 7.1. The network has been implemented for 16-dimensional input vectors representing six different classes ( $n = 16, p = 6$ ). Each of the six neurons in the Hamming network must be provided with bias input; therefore, fixed input voltages must be applied in addition to the weighted input voltages. The Hamming network requires weights with a single-bit resolution only since its binary weights are of values  $\pm 1$ . Six distinct prototype vectors labeled  $A$  through  $F$ , one of each class, have been stored in the weights of the tested network.



**Figure 9.17** MDACs pair for implementing positive or negative weights with 6.25% resolution: (a) circuit diagram and (b) output current of one of the MDACs. [©Lincoln Laboratory Journal, reprinted from Raffel et al. (1989) with permission.]

Figure 9.18(a) tabulates the results of the recall experiment performed by the network. Applied input vectors numbered 1 through 6 have been identical to the stored prototypes *A* through *F*. Each of the six output nodes labeled *A* through *F* is associated by the neural network with a stored vector bearing the same label. As expected, the corresponding output neuron's response *A* through *F* is the strongest for this set of inputs. This is highlighted on the diagonal of the matrix shown containing the highest responses circled and having measured values of 1130 or 1125 mV. The boxed numbers in the table of responses correspond to  $HD = 8$ , or a difference of 8 bits between an input and a stored prototype. For input vectors 7, 8, and 9 applied to the network, the detected prototypes at the smallest Hamming distance value are *A*, *C*, and *A* or *B*, respectively.

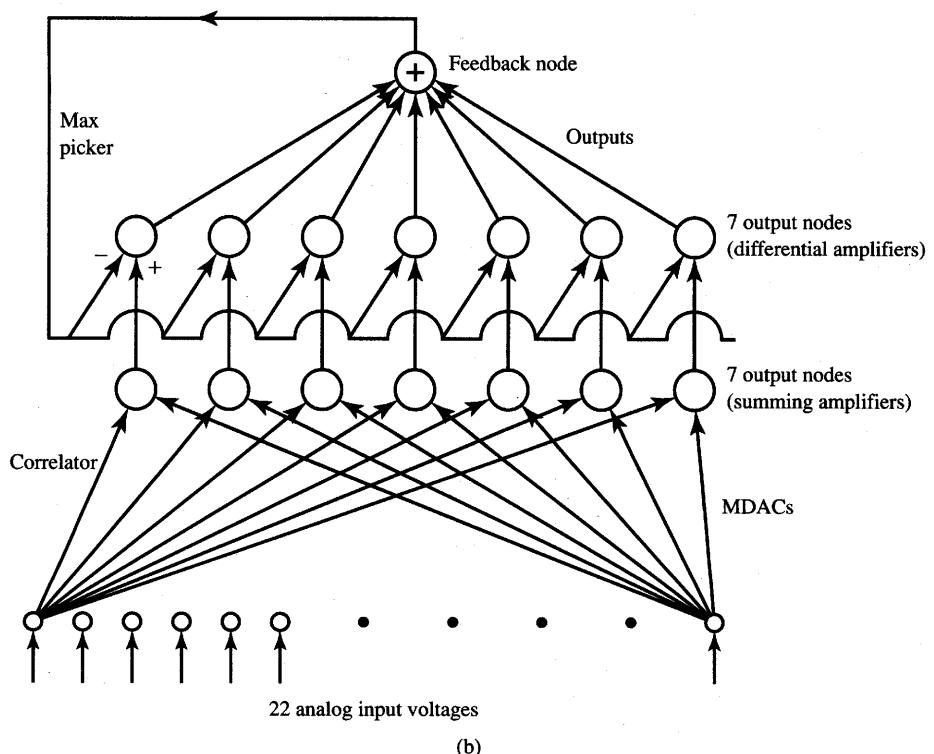
To test the chip operating with its full 4-bit resolution of weights, the network was configured as a 22-component vector correlator-classifier (first layer), followed by the MAXNET network (second layer and output node). The network diagram is depicted in Figure 9.18(b). Twenty two analog input voltages drive seven neurons, each of them representing one class, equivalent to a single cluster. This layer consists of 154 MDACs, one for each weight, connected to seven summing current-to-voltage converters. The second layer is configured as seven amplifiers, which amplify the difference of the input and feedback voltages. Inputs to this layer of neurons are from the first layer neurons, and from the single summing output node called here the feedback node. This node provides the output sum which is fed back to the inverting inputs of the differential amplifiers that constitute the second layer. The second layer of seven neurons and the output summing node operate jointly as MAXNET.

The discussed network configuration has been tested for classification of speech samples. The speech samples were the seven spoken numbers from "one" to "seven." The inputs to the network represent two concatenated 11-tuple vectors, each containing the spectral components from two 10-ms frames of the spoken numbers' samples. The network has been tested for 182 trials of different spoken digits. Seven misclassifications by the network were identified during the test.

Another implementation of programmable weights using MDACs is reported by Moopenn et al. (1990). A chip with 1024 weights utilizes 7-bit data latches to store the control words for the 7-bit weighted-resistance MDACs. The programmable weights implemented on the chip are of 6-bit resolution; the seventh bit is used for producing the weight polarity. To ensure the linearity of weights made of the drain-source resistances of NMOS transistors, long channel transistors are used in this design. Also, the high gate voltage of the switching transistor ensures operation within the linear range of drain-source characteristics of individual transistors serving as programmable resistances. The performance of the chip configured as a Hopfield network was successfully tested for generating the solution of the eight-city traveling salesman problem described in Section 5.6. The intercity distances were encoded in the programmable weight values.

Applied vector	Nearest stored vector	Output in millivolts					
		A	B	C	D	E	F
1	A	(1,130)	695	556	556	775	300
2	B	710	(1,125)	554	700	636	559
3	C	556	560	(1,125)	555	637	560
4	D	556	695	558	(1,125)	493	845
5	E	780	631	629	485	(1,130)	353
6	F	303	556	557	840	358	(1,130)
7	A	1,065	628	480	627	705	352
8	C	428	556	705	557	640	560
9	A & B	705	695	557	415	493	300

(a)



(b)

**Figure 9.18** MDAC-based neural network application: (a) Hamming network test results ( $n = 16$ ,  $p = 6$ ) and (b) binary vector classifier with MAXNET output. [©Lincoln Laboratory Journal, reprinted from Raffel et al. (1989) with permission.]

### Learning Weight Implementation

In our discussion so far, microelectronic analog weight, which transmits the signals between neurons, is typically represented by a channel conductance of an MOS transistor. The transistor's gate voltage is stored across the gate capacitance and performs the weight control function. Learning algorithms, however, have been considered so far to be running off-line. In this section we take a more challenging approach. Learning examined in this section is regarded as a continuous-time on-chip process, occurring while the network is responding to the present inputs. The learning signal is not switched on and off; thus, learning and recall are concurrent. However, the learning proceeds with its own dynamics, which is usually slower than the dynamics of the neural network recall.

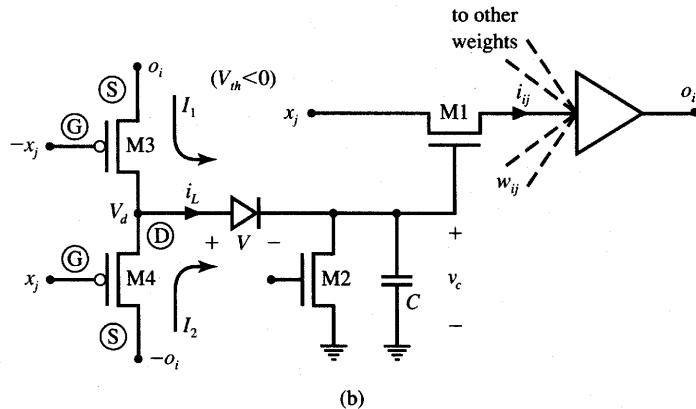
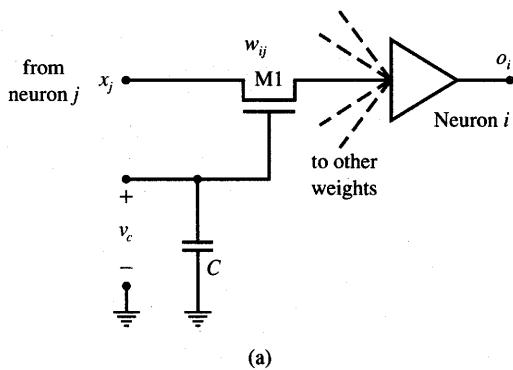
Figure 9.19(a) shows an integrated circuit weight implementation that makes use of this concept. Weight value  $w_{ij}$  is determined by the weight control voltage  $v_c$ , which is supplied by the learning algorithm. One of the most interesting learning modes is Hebbian learning for which the learning signal is simply available at the neuron's output. This section covers the design of weights that learn according to the Hebbian rule discussed earlier in Section 2.5. The weight adjustments according to this unsupervised learning rule are  $\Delta w_{ij} = co_i x_j$  as expressed in (2.32).

The learning weight performs the computation of weight adjustment locally at the synapse. The key component for weight adjustment is the product of analog input and output voltages,  $o_i x_j$ . This particular computation can be implemented by components called *analog multipliers*. A number of multiplier configurations is available for integrated circuit implementation (Geiger, Allen, and Strader 1990; Allen and Holberg 1987). As we will see below, the analog product computation can be implemented by a simple two-transistor circuit.

Figure 9.19(b) shows the learning synapse using the simple circuit of an analog multiplier consisting of a pair of identical transistors  $M3$ , and  $M4$  (Card, Schneider, and Moore 1991). Transistor  $M1$  performs functions identical to transistor  $M1$  on Figure 9.13. The synapse qualitatively approximates the multiplicative Hebbian correlation of input and output voltage levels,  $x_j$  and  $o_i$ . The learning current  $i_L$  is approximately proportional to the product  $o_i x_j$  when voltage  $v_c \approx 0$ . This approximation for current is valid conditionally for an existing correlation between the variables  $x_j$  and  $o_i$ . It is supported by the following discussion as well as by the measurements illustrated below. Assume that current  $i_L$  starts flowing at  $t = t_0$  into capacitor  $C$  due to the existing correlation between  $x_j$  and  $o_i$ . The weight control voltage  $v_c$  increases since

$$v_c(t) = \frac{1}{C} \int_{t_0}^t i_L(t) dt \quad (9.30a)$$

When continuous learning occurs and constant current  $i_L(t) = I_L$  is assumed,



**Figure 9.19** Electrically tunable weight  $w_{ij}$ : (a) without learning and (b) with learning embedded into the weight structure. [Part (b) adapted from Card, Schneider, and Moore (1991); © IEE, with permission.]

voltage  $v_c(t)$  increases linearly in time since

$$v_c(t) = \frac{1}{C} [v_c(t_0) + tI_L] \quad (9.30b)$$

Let us examine in more detail how the learning of the weight value is accomplished in this circuit. Assume that the levels of neuron outputs exceed the magnitude of the threshold voltage of the enhancement mode PMOS devices used, thus  $|x_j| > V_{th}$  and  $|o_i| > |V_{th}|$ . Three distinct learning conditions can be identified in this circuit as follows:

1. Both  $x_j$  and  $o_i$  are positive and therefore correlated. For  $x_j > 0$  and  $o_i > 0$ , assume that  $M3$  is conducting and  $M4$  is cut-off. For  $M3$ ,  $M4$  we have, respectively

$$\begin{aligned} V_{gs3} &= -x_j - o_i < V_{th} \\ V_{gs4} &= x_j + o_i > V_{th} \end{aligned}$$

Both inequalities are fulfilled unconditionally; our assumption has thus been correct; therefore,  $M3$  is indeed conductive and  $M4$  remains cut-off. Current  $I_1$  flows through  $M3$ , provided that the diode is forward biased. This, in turn, is only guaranteed as long as the voltage  $v_c$  remains small enough, or  $o_i$  large enough, to keep the diode conductive.

Under certain conditions,  $M3$  is found to be saturated and thus  $I_1$  remains constant in time and independent of  $v_c$ , which increases linearly in such circumstances. Since the condition for saturation of a PMOS device,  $V_{ds} < V_{gs} - V_{th}$ , is equivalent to  $V_d < V_g - V_{th}$ , we obtain the following condition for saturation of  $M3$ :

$$V_d < -x_j - V_{th}$$

or

$$v_c + V < -x_j - V_{th}$$

The above formula indicates that  $M3$  indeed remains saturated below a certain level of  $v_c$ . Otherwise, the condition is not fulfilled and  $M3$  is operating in the resistive range.

2. Both  $x_j$  and  $o_i$  are negative and therefore correlated. For  $x_j < 0$  and  $o_i < 0$ , assume that  $M3$  is cut-off and  $M4$  saturated. For  $M3$ ,  $M4$  we have, respectively

$$\begin{aligned} V_{gs3} &= -x_j - o_i > V_{th} \\ V_{gs4} &= x_j + o_i < V_{th} \end{aligned}$$

Again, this proves that the assumption about the operating regions of  $M3$  and  $M4$  is correct; thus,  $M3$  is indeed cut-off and  $M4$  is conductive. Current  $I_2$  flows through  $M4$  provided again that the diode is forward biased. Similarly as in case 1 the voltage  $v_c$  can be shown to increase as a result of this process due to the existing correlation of  $x_j$  and  $o_i$ . Again, as in the previous case,  $I_2$  remains constant in time for  $M4$  saturated. The reader can verify that  $M4$  is saturated when

$$v_c + V < x_j - V_{th}$$

Otherwise,  $M4$  operates in the resistive range if this condition is not met.

3. For  $x_j < 0$  and  $o_i > 0$  with an additional assumption of  $|x_j| \approx |o_i|$ , we obtain  $V_{gs3} = V_{gs4} \approx 0$ . Both transistors are thus cut-off. As required, no learning takes place in this case and  $I_L = 0$ , since  $x_j$  and  $o_i$  are obviously anticorrelated.

Let us summarize the learning progress in this circuit. As the capacitor charges due to the computed correlation of  $o_i$  and  $x_j$ , the weight  $w_{ij}$  increases. The voltage at the node  $D$  is equal to  $v_c$  plus the forward-biased diode voltage drop  $V$ . It can be noticed that for a certain level of  $v_c + V$ , the voltage  $v_c$  ceases to increase and  $i_L$  stops flowing even though the product  $o_i x_j$  is positive. Thus,

the learning signal represented by  $i_L$  becomes affected by the acquired weight value  $w_{ij}(t)$  and the weight learning process slows down, or even vanishes. It can therefore be seen that this circuit implements the Hebbian learning with saturation, because high weight values slow down further learning. Transistor  $M2$  is used to control the leakage current, which now produces the intentionally embedded weight forgetting property of this circuit. The decay of weights because of leakage current through  $M2$  occurs much slower than the learning so that the weight can memorize a large number of inputs/output pairs. The diode allows for decoupling of the learning and forgetting process so that their rates may be independently adjusted. The forgetting rate is usually several orders of magnitude slower than the learning rate.

The dimension of each synapse from Figure 9.19(b) implemented in 1.2- $\mu\text{m}$  CMOS technology is  $115 \times 10.9 \mu\text{m}$ . The smallest length of the CMOS transistor channel in this technology is  $1.2 \mu\text{m}$ . A synapse chip with an area of  $1 \text{ cm}^2$  could contain approximately 75 000 individual learning weights. SPICE simulations have been performed for the weight model discussed (Card, Schneider, and Moore 1991). Figure 9.20(a) depicts the current  $i_{ij}$  through the weight as a function of the weight input voltage  $x_j$  with  $v_c$  being the parameter. Figure 9.20(b) shows the learning current  $i_L$  versus  $o_i$ , with  $x_j$  being a parameter for each of the four characteristics shown. The figure corresponds to the case  $v_c = 0$  and therefore represents the maximum learning rate. It can be seen that the pair  $M3$  and  $M4$  performs well as an analog voltage multiplier, except for values  $o_i$  and  $x_j$  near the origin. The graphs reveal that small correlations existing between  $o_i$  and  $x_j$  are ignored by this multiplier.

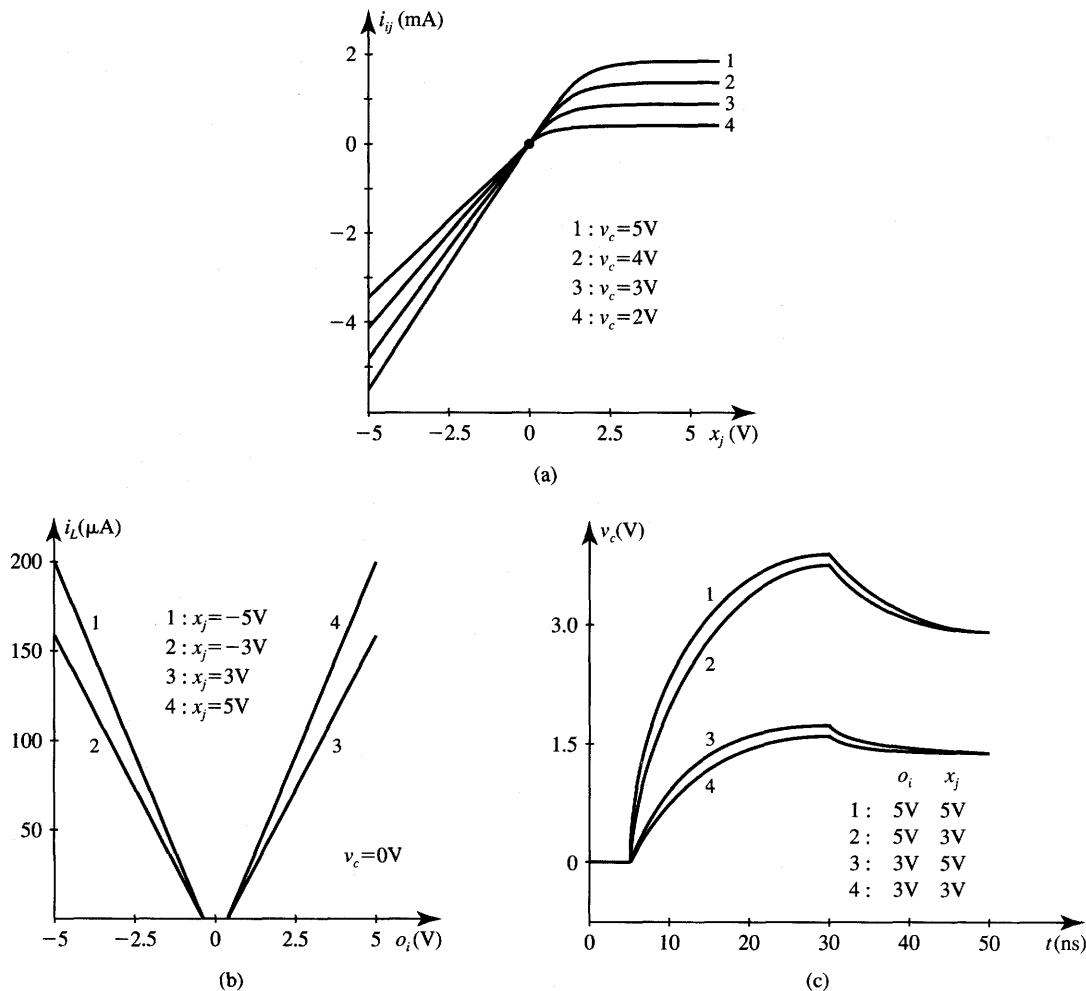
Figure 9.20(c) illustrates the weight control voltage curves,  $v_c$ , versus current  $i_L$ . The figure displays four different positive correlation learning cases between  $o_i$  and  $x_j$ : (5 V, 5 V), (5 V, 3 V), (3 V, 5 V), (3 V, 3 V). Note that the fastest learning is observed for (5 V, 5 V); however, faster learning occurs for case (5 V, 3 V) than for case (3 V, 5 V). The figure clearly illustrates the learning with saturation by this weight circuit. It can also be seen, that the weight decay starts at  $t = 30 \text{ ns}$  when the learning ceases and a slow weight value decay begins due to the discharge of  $C$  through  $M2$ .

## 9.3

---

### ACTIVE BUILDING BLOCKS OF NEURAL NETWORKS

Although the number of weights usually outnumbers the number of computing nodes, and the majority of the artificial neural network circuitry is composed of connections, an important part of the processing is performed outside synapses. Specifically, functions such as summing, subtracting, computing scalar products,



**Figure 9.20** Simulation results for learning weight of Figure 9.19(b): (a) weight current versus input voltage, (b) learning current versus output voltage, and (c) learning rate and saturation phenomena ( $t < 30$ ns); followed by forgetting through weight decay ( $t \geq 30$ ns). [Adapted from Card, Schneider, and Moore (1991); © IEE, with permission.]

or nonlinear mapping are for the most part performed independently of synaptic weighting. A number of electronic functional circuits such as current mirrors, inverters, differential amplifiers, transconductance amplifiers, analog multipliers, and scalar product circuits can perform these operations. Integrated circuits able to implement these functions and designed using CMOS technology are covered in this section. To avoid duplication and maintain our focus of study on neural

networks, the coverage of these building blocks is less comprehensive in this text than in more advanced microelectronic texts. Primary attention is devoted in this chapter to the circuits' applications in artificial neural systems rather than in their typical switching or amplifying configurations. For a more comprehensive study of building block circuit analysis and design, the reader may refer to a number of specialized microelectronic circuits texts available (Geiger, Allen, and Strader 1990; Allen and Holberg 1987; Millman and Grabel 1987; Sedra and Smith 1991).

### Current Mirrors

A *current mirror* is one of the most essential building blocks of many analog integrated circuits, with artificial neural systems being no exception. A current mirror takes its input current and creates one or more copies of it at one or more of its outputs. The term *current amplifier* is also used for a current mirror since an input current change results in an output current change.

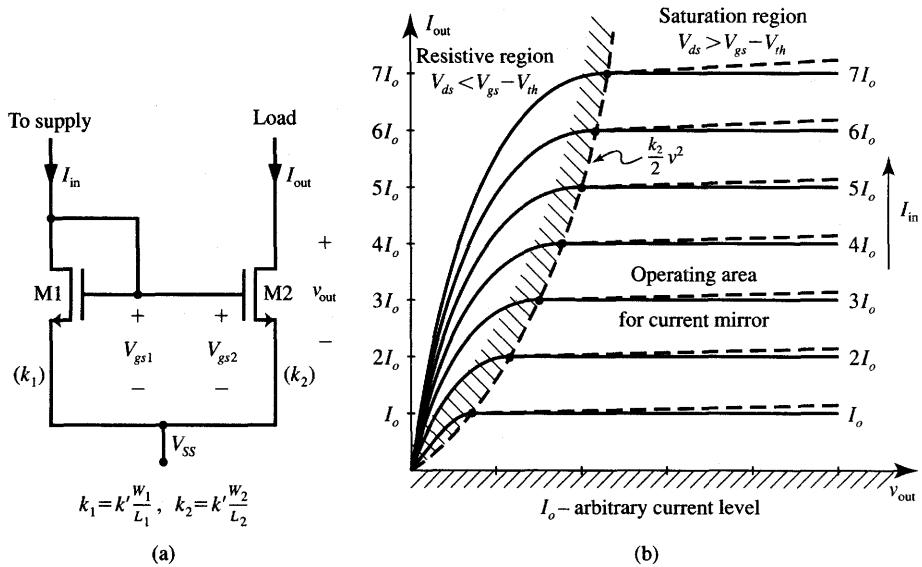
Figure 9.21(a) shows an example of a simple current mirror consisting of two matching NMOS transistors,  $M_1$  and  $M_2$ . The current mirror produces output current  $I_{\text{out}}$  proportional to the input current  $I_{\text{in}}$  so that  $I_{\text{out}} = \alpha I_{\text{in}}$ , where  $\alpha$  is the proportionality constant dependent on the transistor sizes and the fabrication process constants. For practical purposes, the relationship  $I_{\text{out}} = \alpha I_{\text{in}}$  should be maintained in large range of input current variations. Note that since  $V_{gs1} = V_{ds1}$ , then the condition  $V_{ds1} > V_{gs1} - V_{th1}$  is always fulfilled and, therefore,  $M_1$  is permanently saturated. Assuming that an additional condition is met,  $V_{ds2} = v_{\text{out}} > V_{gs2} - V_{th2}$ , both devices can be considered to be saturated. We thus obtain the mirror output/input current ratio (see Appendix):

$$\frac{I_{\text{out}}}{I_{\text{in}}} = \frac{k'(W_2/L_2)(V_{gs2} - V_{th2})^2(1 + \lambda V_{ds2})}{k'(W_1/L_1)(V_{gs1} - V_{th1})^2(1 + \lambda V_{ds1})} \quad (9.31a)$$

where  $\lambda$  stands for the channel length modulation effect. In our previous discussion of MOS transistor modeling,  $\lambda$  has been assumed to have a zero value without making a significant error. For the current mirror study, however, the simplifying assumption  $\lambda = 0$  may lead to essential inaccuracies of the analysis. Since  $V_{gs2} = V_{gs1}$  and  $V_{th1} = V_{th2}$  for matching transistors, we obtain from (9.31a)

$$\frac{I_{\text{out}}}{I_{\text{in}}} = \frac{W_2/L_2}{W_1/L_1} \left( \frac{1 + \lambda V_{ds2}}{1 + \lambda V_{ds1}} \right) \quad (9.31b)$$

Ideally,  $I_{\text{out}}/I_{\text{in}}$  should only be a function of the aspect ratios ( $W/L$ ) of both transistors. For identically sized transistors, and  $V_{ds1} = V_{ds2}$ , input and output current are then exactly equal. In such an ideal case, the current gain value  $\alpha$  is equal to unity. In the case of  $V_{ds2} \neq V_{ds1}$ , however, the channel length modulation



**Figure 9.21** Current mirror with NMOS transistors: (a) simple circuit diagram and (b) example  $I_{\text{out}} = f(I_{\text{in}})$  characteristics for unity-gain case [ $(W_1/L_1) = (W_2/L_2)$ ].

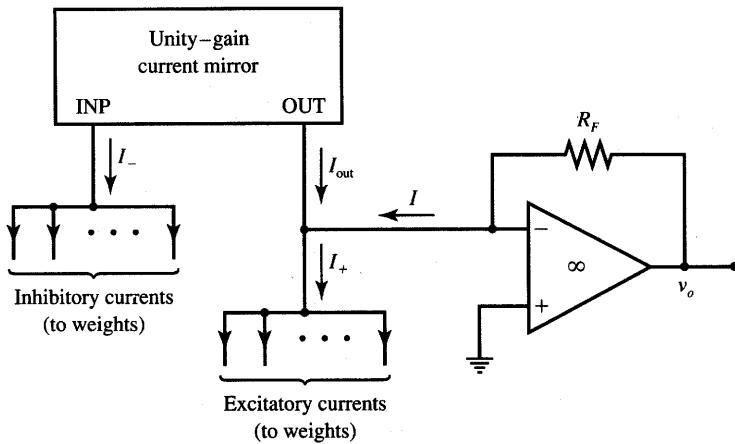
factor  $\lambda$  causes the output current to be slightly different than the input current. For  $\lambda = 10^{-2}V^{-1}$  (typically),  $I_{\text{out}}$  displays then a slight deviation from  $I_{\text{in}}$  that can be computed from (9.31b) for given values of drain-to-source voltages for both  $M1$  and  $M2$ .

Figure 9.21(b) depicts the unity-gain current mirror characteristics. The current mirror operating area should be maintained in the saturation region of transistor  $M2$ . Therefore,  $v_{\text{out}}$  must satisfy the condition  $v_{\text{out}} > V_{gs2} - V_{th2}$ , which easily translates into the following inequality

$$v_{\text{out}} > \sqrt{\frac{2I_{\text{out}}}{k_2}} \quad (9.32)$$

where  $k_2 = k'W_2/L_2$ .

The output characteristics of the current mirror are shown in the figure as horizontal lines for the idealized case when  $\lambda = 0$ , thus indicating that the device reproduces the input current at its output in a wide range of output voltages  $v_{\text{out}}$ . A current mirror therefore behaves as both current source and current replicator. For more realistic MOS transistor modeling with  $\lambda \neq 0$ , the output current characteristics in the saturation region have nonzero slopes. This is shown in Figure 9.21(b) with dashed lines. It can be seen that the output current produced by the current mirror also depends to some extent on its output voltage. As a result, the device can therefore be considered a nonideal current source.



**Figure 9.22** Current mirror use for computing the difference between the excitatory and inhibitory currents.

Current mirrors are ideally suited for detecting the differences of two currents. Specifically, they can be applied in artificial neural networks to compute the difference between an excitatory and an inhibitory current. This function can be performed by the unity-gain current mirror loaded with a current-to-voltage converter as shown in Figure 9.22. Noting that the current  $I$  to be converted is equal to the following current difference

$$I = I_+ - I_{\text{out}} \quad (9.33a)$$

and using the current mirror property of  $I_- = I_{\text{out}}$  we obtain

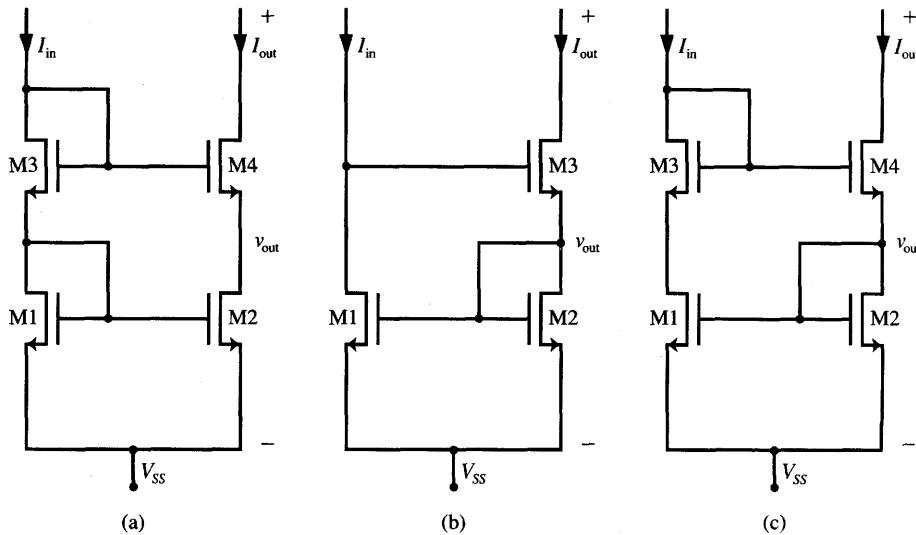
$$I = I_+ - I_- \quad (9.33b)$$

Thus, the converter output voltage is of value

$$v_o = (I_+ - I_-)R_F \quad (9.34)$$

This result is valid provided the operational amplifier remains in the linear region and does not saturate. Otherwise, output voltage  $v_o$  takes either the positive or negative saturation value,  $f_{\text{sat}+}$  or  $f_{\text{sat}-}$ , respectively.

Three main reasons make the actual current mirror perform as a nonideal device: the channel length modulation effect, threshold voltage differences between transistors used, and imperfect geometrical matching of the fabricated transistors. Figure 9.23 shows three circuit diagrams of improved current mirrors. They are cascode (a), Wilson (b), and modified Wilson (c) types (Millman and Grabel 1987; Geiger, Allen, and Strader 1990; Sedra and Smith 1991). All of the current mirrors shown have distinctly more constant current  $I_{\text{out}}$  versus  $v_{\text{out}}$ , and reduced channel length modulation effect compared to the original simple two-transistor circuit from Figure 9.21(a).



**Figure 9.23** Basic configurations of MOS current mirrors: (a) cascode, (b) Wilson, and (c) modified Wilson

### Inverter-based Neuron

Most neuron elements presented in this chapter have consisted of a suitably coupled operational amplifier circuit that simultaneously performed current summation, current-to-voltage conversion, and nonlinear mapping. We have noticed, however, that the activation functions of neurons designed so far with operational amplifiers are of limited form only. As discussed, they are (1) linear, (2) of finite gain, with the gain value determined by the slope of the linear part of the characteristics near the origin, and (3) beyond the linear region the characteristics saturated at  $f_{sat+}$  and  $f_{sat-}$  levels. In summary, all activation functions produced thus far have been either linear or piece-wise linear.

In numerous applications, however, the neurons' characteristics are required to have a sigmoidal shape of the activation function. "Sigmoidal" is used here in the sense of any smooth S-shaped curve, and it does not refer to any specific function. Such characteristics can be obtained as voltage transfer characteristics of a simple complementary circuit known in electronics as a "push-pull amplifier." This circuit behaves as a voltage inverter and is often used in logic circuits.

The CMOS inverter circuit is shown in Figure 9.24 and consists of complementary NMOS and PMOS transistors with drains tied together that also act as the output terminal. Gates are also connected to form the input, and sources are tied to two opposite supply voltages  $V_{DD}$  and  $V_{SS} = -V_{DD}$ , where  $V_{DD} > 0$ . The circuit performs as an inverting voltage amplifier with considerable voltage

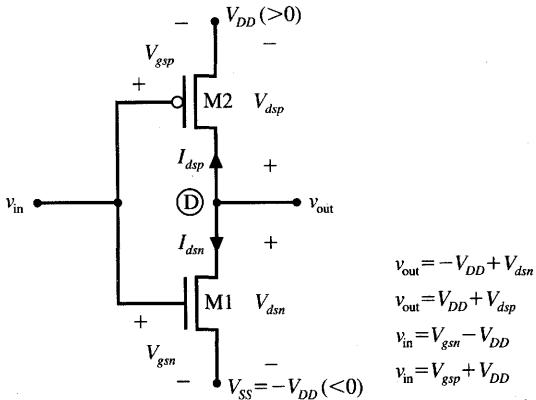


Figure 9.24 Inverter-based half-neuron circuit diagram.

gain near the origin. To derive the activation function for this circuit, we assume that there is full electrical symmetry between the NMOS and PMOS devices, or  $k_n = k_p$ . Symmetry also requires that both transistors' threshold voltages be of the same magnitude, which means  $V_{thn} = -V_{thp}$ .

To obtain the voltage transfer characteristics  $v_{out}(v_{in})$  for this circuit let us use the equation

$$I_{dsn} + I_{dsp} = 0 \quad (9.35)$$

valid for the case of open-circuit inverter output. It can be seen that each of the transistors can be in one of the three regions: cut-off, resistive, or in saturation (see Appendix). Although the number of combinations of regions for two transistors in the pair is nine, only five pairs of stable states can exist in the entire range of input and output voltages (Zurada, Yoo, and Bell 1989).

To construct the output characteristics, we start with large negative input voltages  $v_{in}$  such that  $v_{in} + V_{DD} < V_{thn}$ . Since by inspection of the figure we have for the NMOS transistor

$$V_{gsn} = v_{in} + V_{DD} \quad (9.36a)$$

thus  $M1$  remains as the cut-off. However, at the same time transistor  $M2$  is fully on since its gate-to-source voltage is below  $V_{thp}$ . Indeed, we have for  $M2$ ,

$$V_{gsp} = v_{in} - V_{DD} < V_{thp} \quad (9.36b)$$

However, since no current can flow through any of the two transistors connected in series due to the cut-off of  $M1$ , transistor  $M2$  behaves as a short circuit between the drain and source. Also, the following condition holds for the resistive region of  $M2$ :

$$V_{dsp} > V_{gsp} + V_{thp} \quad (9.37)$$

We thus have for this region

$$v_{\text{out}} = V_{DD}, \quad \text{for } v_{\text{in}} < -V_{DD} + V_{thn}$$

Figure 9.25(a) shows the voltage transfer characteristics of this circuit. The discussed part of the characteristics is marked *AB*. In this operating range of the neuron transistor, *M1* is the cut-off and *M2* is in the resistive region.

When the input voltage reaches the level slightly above  $-V_{DD} + V_{thn}$ , the NMOS device enters the saturation region, the PMOS device remains in the resistive region and the channel currents start flowing with the following values:

$$I_{dsn} = \frac{k}{2}(V_{gsn} - V_{thn})^2 \quad (9.38a)$$

$$I_{dsp} = -k \left[ (V_{gsp} - V_{thp})V_{dsp} - \frac{V_{dsp}^2}{2} \right] \quad (9.38b)$$

Let us now notice that (9.38a) and (9.38b) remain valid for the following conditions, respectively:

$$V_{dsn} > V_{gsn} - V_{thn} \quad (\text{M1 saturated}) \quad (9.39a)$$

and

$$V_{dsp} > V_{gsp} - V_{thp} \quad (\text{M2 resistive}) \quad (9.39b)$$

Using identities labeled on Figure 9.24 and expressing  $v_{\text{in}}$ , and  $v_{\text{out}}$  in terms of the transistors' terminal voltages and supply voltages allows for the expression of all voltages between transistor terminals as follows

$$V_{dsn} = v_{\text{out}} + V_{DD} \quad (9.40a)$$

$$V_{dsp} = v_{\text{out}} - V_{DD} \quad (9.40b)$$

$$V_{gsn} = v_{\text{in}} + V_{DD} \quad (9.40c)$$

$$V_{gsp} = v_{\text{in}} - V_{DD} \quad (9.40d)$$

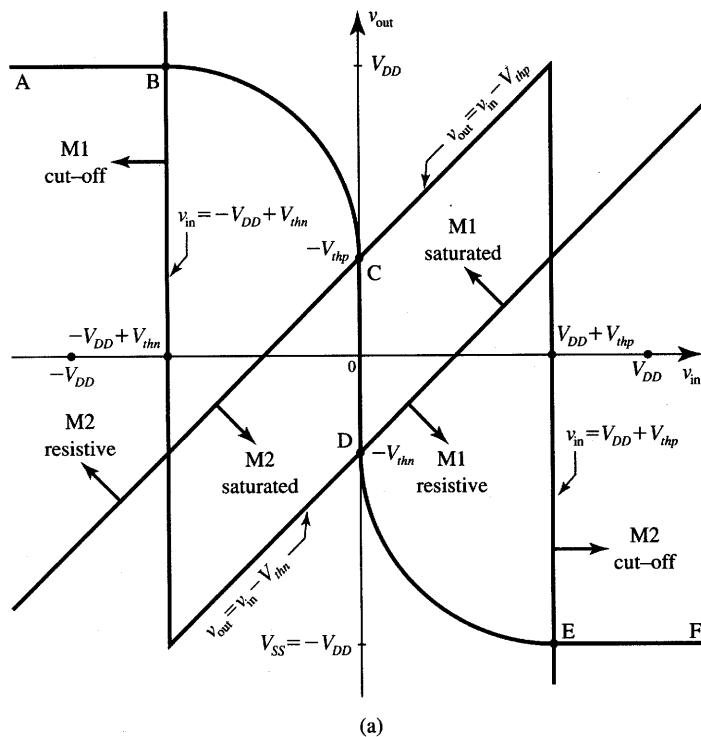
Conditions (9.39) for the transistors' operating regions can now be rewritten to the form

$$v_{\text{out}} > v_{\text{in}} - V_{thn} \quad (\text{M1 saturated}) \quad (9.41a)$$

$$v_{\text{out}} > v_{\text{in}} - V_{thp} \quad (\text{M2 resistive}) \quad (9.41b)$$

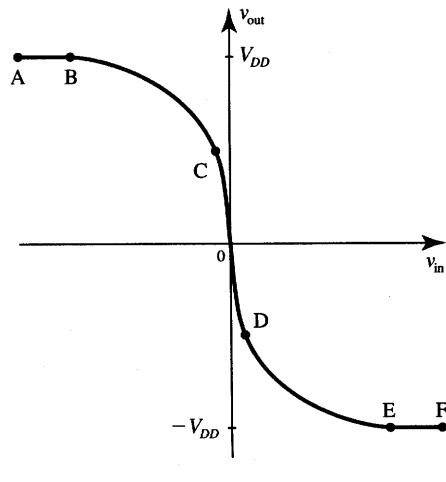
The line  $v_{\text{out}} = v_{\text{in}} - V_{thn}$  obtained from (9.41a) separates the resistive and saturation regions of *M2* as shown in Figure 9.25(a). A straight line  $v_{\text{out}} = v_{\text{in}} - V_{thp}$  obtained from (9.41b) is also shown in the figure. The line passes through point *C* and separates the saturation and resistive regions of *M1*.

Our objective in this step is to find  $v_{\text{out}}$  versus  $v_{\text{in}}$  for *M1* saturated and *M2* resistive, or beyond point *B* of the characteristics. This part of the input/output curve is responsible for the sigmoidal part of the neuron's activation function. Using relationships (9.40) again allows for rearranging expressions for currents



Region	M1	M2
AB	Cut-off	Resistive
BC	Saturated	Resistive
CD	Saturated	Saturated
DE	Resistive	Saturated
EF	Resistive	Cut-off

(b)



**Figure 9.25** Voltage transfer characteristics of an inverter-based neuron (device and power supply symmetry assumed): (a)  $v_{out}$  versus  $v_{in}$  ( $\lambda = 0$ ) (ideal transistors case), (b) regions of transistors operation, and (c)  $v_{out}$  versus  $v_{in}$ , ( $\lambda \neq 0$ ) (nonideal transistors case).

as in (9.38) to the form

$$I_{dsn} = \frac{k}{2}(v_{in} + V_{DD} - V_{thn})^2 \quad (9.42a)$$

$$I_{dsp} = -k \left( v_{in} - V_{thp} - \frac{v_{out} + V_{DD}}{2} \right) (v_{out} - V_{DD}) \quad (9.42b)$$

It now follows from (9.35) that (9.42) can be rewritten as:

$$\frac{1}{2}(v_{in} + V_{DD} - V_{thn})^2 = \left( v_{in} - V_{thp} - \frac{v_{out} + V_{DD}}{2} \right) (v_{out} - V_{DD}) \quad (9.43)$$

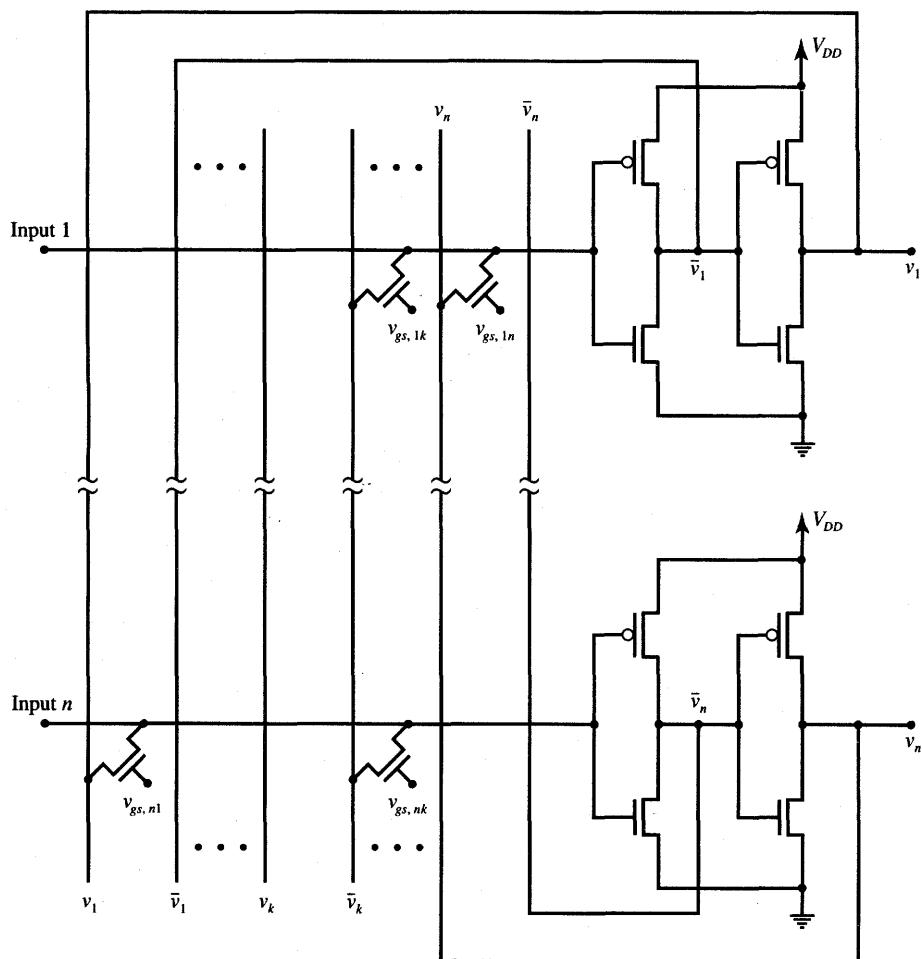
This relationship is somewhat unwieldy for hand calculation but does provide the desired relationship between  $v_{in}$  and  $v_{out}$  in the concerned region of operation of the neuron. The relationship is graphed as part *BC* of the characteristics in Figure 9.25(a). When  $v_{in}$  reaches 0 V, the characteristics at point *C* crosses the region borderline and transistor *M*2 enters the saturation region. This results in two transistors being saturated simultaneously. The output voltage remains undefined in this *CD* part of the characteristics and  $-V_{thn} < v_{out} < -V_{thp}$  for  $v_{in} = 0$ .

A summary of the operating regions for both transistors covering the complete characteristics is illustrated in Figure 9.25(b). The remaining part *DEF* of the characteristics can be obtained from a similar analysis using data in the two bottom rows of Figure 9.25(b). The discussion above makes use of the assumption that the channel length modulation effect is negligible in both transistors, or  $\lambda = 0$ . For a more realistic assumption  $\lambda \neq 0$  we obtain the inverter characteristics as shown in Figure 9.25(c). The main difference here is that in contrast to the ideal case, this curve is of finite slope in the region *CD* when both *M*1 and *M*2 are saturated.

It is obvious that connecting two inverting half-neurons from Figure 9.24 in cascade yields a complete neuron element. The resulting input/output characteristics of the cascade connection is of the noninverting type and has a sigmoidal shape. This neuron configuration is especially well suited for building associative memory circuits or continuous-time gradient-type networks. Figure 9.26 depicts an example of a circuit diagram of an  $n$ -bit single-layer recurrent autoassociative memory using a cascade connection of two push-pull inverters as neurons. Due to the availability of both  $v_i$  and inverted output  $\bar{v}_i$ , positive as well as negative weights can be implemented in this circuit. The weight matrix of the memory network shown contains weights made of single-transistor resistances and controlled by the gate voltages. The transistors of this circuit need to operate in their linear region to maintain constant weight values as has been discussed in Section 9.2.

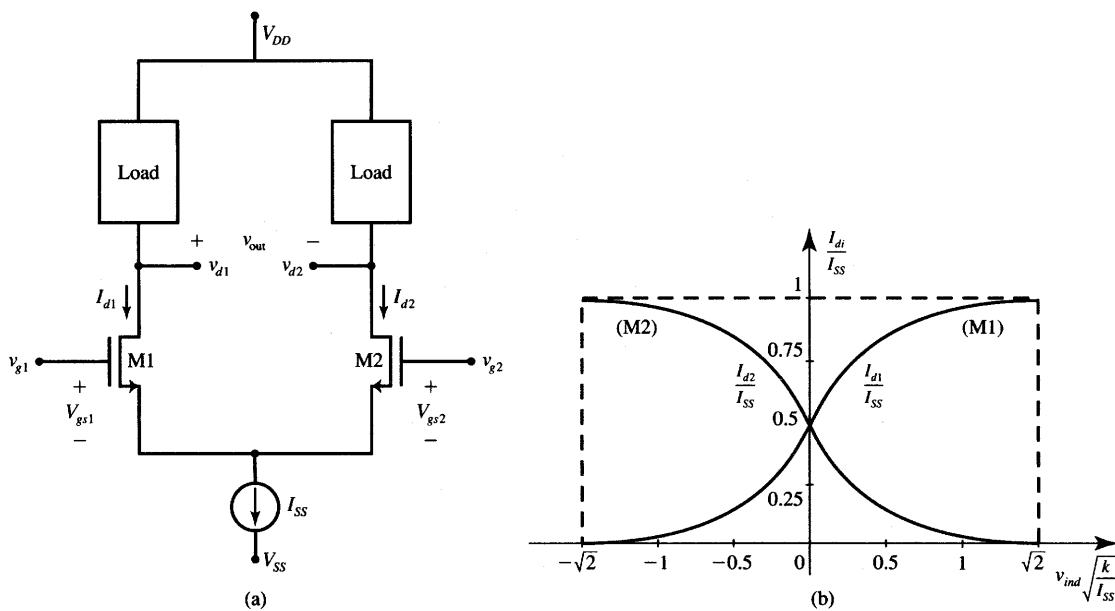
### Differential Voltage Amplifiers

Throughout this chapter we have most often used integrated circuit operational amplifiers as versatile building blocks of microelectronic neural systems.



**Figure 9.26** Recurrent autoassociative memory with neurons from Figure 9.25.

An operational amplifier is essentially a differential voltage amplifier of very high gain with voltage output. As we have seen, the negative feedback loop of the operational amplifier has to be closed to achieve its stable operation. However, differential voltage amplifiers exist that can operate without the negative feedback loop and their voltage gain is typically a hundred or more times lower than that of operational amplifiers. Such differential voltage amplifiers are very often used as input stages of operational amplifiers. This section is concerned with the properties of such *differential voltage amplifiers*. It also covers the basic concept of the transconductance amplifier. These two simple and closely related circuits have characteristics that make them desirable building blocks of microelectronic neural networks.



**Figure 9.27** Basic differential voltage amplifier with an NMOS pair: (a) circuit diagram and (b) normalized transconductance characteristics for normalized differential input voltage.

Figure 9.27(a) shows the schematic of an NMOS differential voltage amplifier called briefly the differential pair. The circuit consists of a pair of matching NMOS transistors  $M_1$  and  $M_2$ , two possibly symmetric loads, and a reference current source  $I_{ss}$ . Let us examine the pair's transfer characteristics, noting that the circuit should primarily serve as an amplifier of the differential voltage signal  $v_{ind}$  defined as follows:

$$v_{ind} \triangleq V_{gs1} - V_{gs2} \quad (9.44a)$$

which is equal to

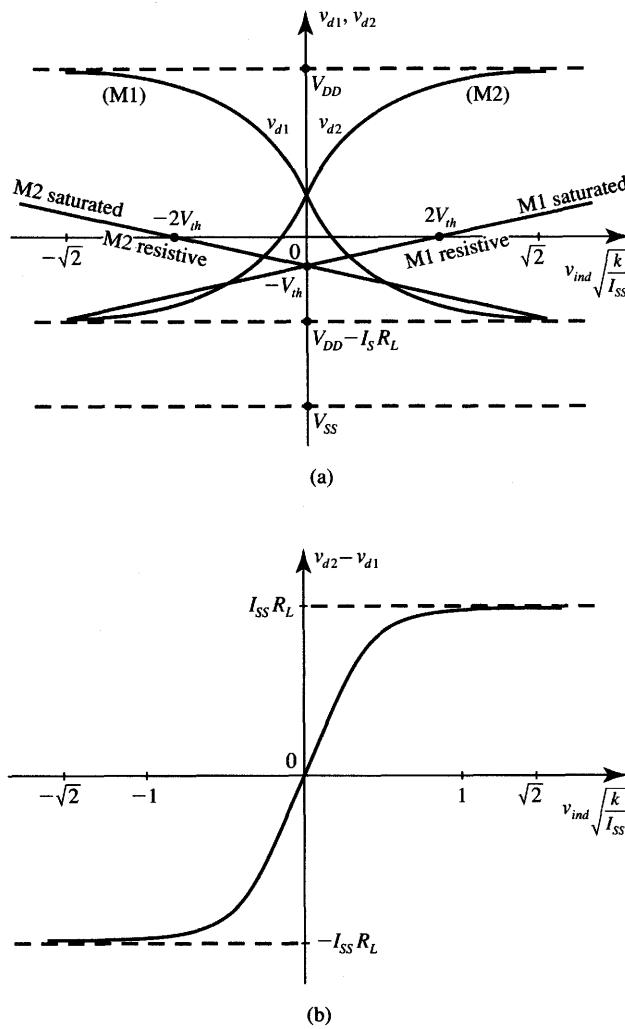
$$v_{ind} = v_{g1} - v_{g2} \quad (9.44b)$$

Assume that transistors  $M_1$  and  $M_2$  are always in saturation, which is a reasonable assumption as we will see later from subsequent discussion and from the lines separating operating regions on Figure 9.28(a). For the saturation condition, we have the drain-to-source current of both transistors:

$$I_{di} = \frac{k_i}{2}(V_{gsi} - V_{thi})^2, \quad \text{for } i = 1, 2 \quad (9.45a)$$

which leads to the following expression for  $V_{gs1}$  and  $V_{gs2}$ :

$$V_{gsi} = \sqrt{\frac{2I_{di}}{k_i}} + V_{thi}, \quad \text{for } i = 1, 2 \quad (9.45b)$$



**Figure 9.28** Voltage transfer characteristics of a differential input amplifier: (a) single-ended output configuration and (b) double-ended output (differential output) configuration.

This allows us to express  $v_{ind}$  from (9.44a) as a function of currents  $I_{d1}$  and  $I_{d2}$ :

$$v_{ind} = \sqrt{\frac{2I_{d1}}{k}} - \sqrt{\frac{2I_{d2}}{k}} \quad (9.46)$$

provided  $V_{th1} = V_{th2}$ , and  $k_1 = k_2 = k$  for matching transistors. Observing that the reference current source establishes the sum of both channel currents  $I_{ds1}$  and  $I_{ds2}$  as equal to  $I_{ss}$

$$I_{ss} = I_{d1} + I_{d2} \quad (9.47)$$

and forming a quadratic solution of (9.46) and (9.47) for  $I_{d1}$  and  $I_{d2}$  leads to

$$I_{d1} = \frac{I_{ss}}{2} + \frac{I_{ss}}{2} \left( \frac{k v_{ind}^2}{I_{ss}} - \frac{k^2 v_{ind}^4}{4 I_{ss}^2} \right)^{1/2} \quad (9.48a)$$

$$I_{d2} = \frac{I_{ss}}{2} - \frac{I_{ss}}{2} \left( \frac{k v_{ind}^2}{I_{ss}} - \frac{k^2 v_{ind}^4}{4 I_{ss}^2} \right)^{1/2} \quad (9.48b)$$

Due to the assumed saturation condition, expressions (9.48) are valid for the following limited range of differential input voltages:

$$-\sqrt{\frac{2I_{ss}}{k}} < v_{ind} < \sqrt{\frac{2I_{ss}}{k}} \quad (9.49)$$

The transfer characteristics  $I_{d1}$  versus  $v_{ind}$  and  $I_{d2}$  versus  $v_{ind}$  can now be sketched using (9.48) in the region of  $v_{ind}$  specified by (9.49). Figure 9.27(b) shows a plot of the normalized drain currents through transistors  $M1$  and  $M2$ .

Assume that the loads of the driving transistors  $M1$  and  $M2$  from Figure 9.27(a) are resistors of an arbitrary value  $R_L$ . It can be seen that the output voltages  $v_{d1}$  and  $v_{d2}$  can be expressed as

$$\begin{aligned} v_{d1} &= V_{DD} - I_{d1} R_L \\ v_{d2} &= V_{DD} - I_{d2} R_L \end{aligned} \quad (9.50)$$

Equations (9.50) are plotted in Figure 9.28(a). The graphs depict the voltage transfer characteristics of a single-ended output differential-input amplifier with outputs being either  $v_{d1}$  or  $v_{d2}$ . The saturation regions for both transistors are also graphed on the figure. The equations of lines separating the transistors' operating regions are

$$\begin{aligned} v_{d1} &= \frac{v_{ind}}{2} - V_{th} \\ v_{d2} &= -\frac{v_{ind}}{2} - V_{th} \end{aligned} \quad (9.51)$$

and this indicates that the initial assumption about the two transistors being saturated is fulfilled in a rather large range of the differential input voltages. Let us also notice that the amplifier output can also be configured in the differential mode so that  $v_{out} = v_{d2} - v_{d1}$ . Such a connection ensures that the range of the output voltage covered by this configuration is between  $-I_{ss}R_L$  and  $I_{ss}R_L$ . This can be seen from the double-ended differential-input characteristics shown in Figure 9.28(b).

Although the double-ended, or differential output, amplifier provides a desirable shape for the transfer characteristics and its gain is twofold compared to the single-ended stage, none of its two output terminals are grounded. To circumvent this rather impractical load configuration, a current mirror can be used to form the load. The advantage of such a configuration is that the differential output

signal in the form of drain currents in the circuit of Figure 9.27(a) is converted to a single-ended output with the load grounded at one terminal.

The circuit configuration implementing this concept is shown in Figure 9.29(a). In this circuit, the output voltage or current is taken from the connected drains of transistors  $M2$  and  $M4$ . The most essential feature of this arrangement is that by definition of the current mirror as in (9.33) we have in this circuit

$$I_{\text{out}} = I_{d1} - I_{d2} \quad (9.52)$$

This circuit providing the output variable of current  $I_{\text{out}}$  is also called the *transconductance amplifier*. It behaves like current source controlled by the differential input voltage  $v_{\text{ind}}$ .

Expressing currents  $I_{d1}$ ,  $I_{d2}$  as in (9.48) allows for rewriting (9.52) as follows:

$$I_{\text{out}} = I_{ss} \left( \frac{k v_{\text{ind}}^2}{I_{ss}} - \frac{k^2 v_{\text{ind}}^4}{4 I_{ss}^2} \right)^{1/2} \quad (9.53)$$

The transconductance of this circuit,  $g_m$ , is defined as

$$g_m = \left. \frac{d(I_{\text{out}})}{d(v_{\text{ind}})} \right|_{v_{\text{ind}}=0} \quad (9.54)$$

Computation of  $g_m$  from (9.54) using (9.53) indicates that the transconductance  $g_m$  of the amplifier is identical to the transconductance of a single transistor  $M1$  or  $M2$  of the pair (Geiger, Allen, and Strader 1990) and is equal to

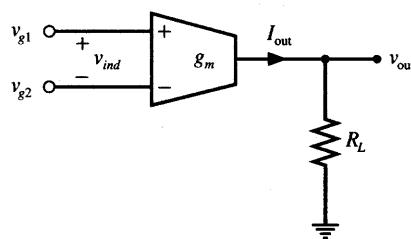
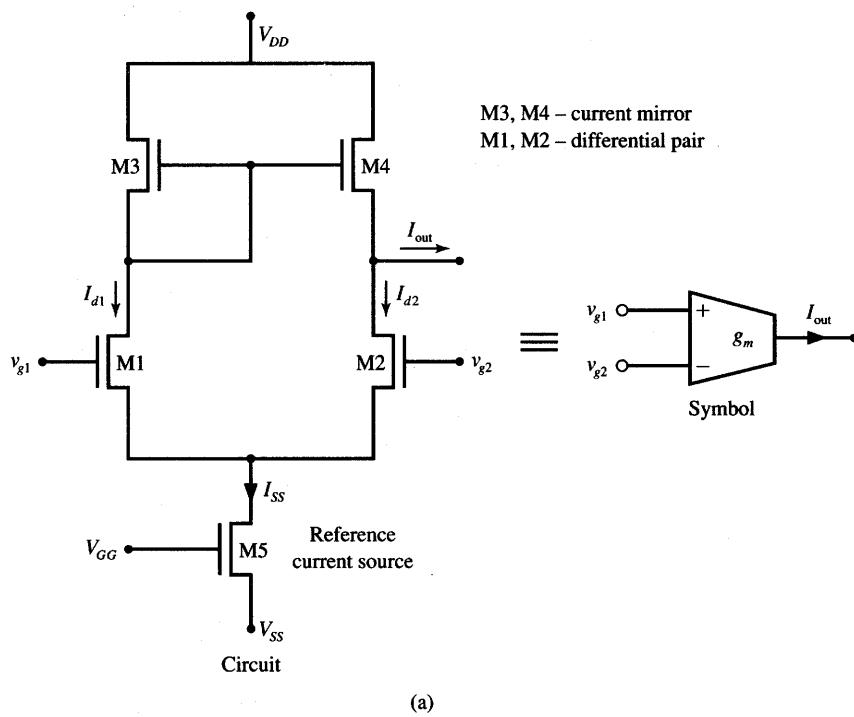
$$g_m = \sqrt{k I_{ss}} \quad (9.55)$$

If the load of the transconductance amplifier is resistive, then the circuit performs like an ordinary differential voltage amplifier with the voltage gain of value

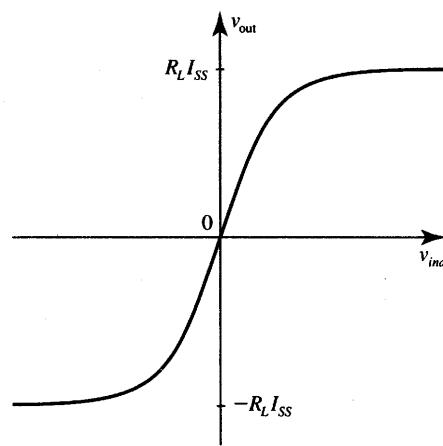
$$\frac{d(v_{\text{out}})}{d(v_{\text{ind}})} = g_m R_L \quad (9.56)$$

The symbol of the transconductance amplifier is shown in Figure 9.29(b). It can be seen in this figure that the amplifier load is resistive. The differential voltage gain expression of (9.56) is only valid near the origin  $v_{\text{ind}} = v_{\text{out}} = 0$ . Similar to the transfer characteristics from Figures 9.27(b) and 9.28(b), the characteristics of the transconductance amplifier with resistive load also becomes strongly nonlinear for larger differential inputs  $v_{\text{ind}}$  and eventually it saturates. Typical curve  $v_{\text{out}}$  versus  $v_{\text{ind}}$  is shown in Figure 9.29(c). Since the currents of the differential amplifier in saturation are limited to  $\pm I_{ss}$ , the output voltage of the loaded transconductance amplifier is limited to between  $-I_{ss} R_L$  and  $I_{ss} R_L$ .

The transconductance amplifier has several distinct characteristics different from the conventional high open-loop gain differential operational amplifier.



(b)



(c)

**Figure 9.29** Transconductance (voltage-to-current) amplifier: (a) circuit diagram, (b) symbol of the transconductance amplifier loaded by a resistance  $R_L$ , and (c) voltage transfer characteristics for the circuit of part (b).

They are:

1. The voltage gain level of an operational amplifier is controlled by the ratios of resistance in the feedback loop to the resistances between the signal input and the operational amplifier input. In the transconductance amplifier, the gain is proportional to  $g_m$ , which is controlled by the internal supply current  $I_{ss}$ , as can be seen from expressions (9.55) and (9.56).
2. The voltage gain of the operational amplifier does not depend on the load resistance  $R_L$  as is the case for the transconductance amplifier [see Equation (9.56)]. This difference is due to the fact that the operational amplifier performs as a voltage source, and it has output stages with low output resistance. The transconductance amplifier performs as a current source producing current of certain value; its output voltage is thus proportional to the load resistance value.
3. The input/output voltage or current transfer characteristics of the transconductance amplifier characteristics have a sigmoidal shape. The transconductance amplifier computes an activation function with a smooth transition from the linear behavior to the behavior that is saturated. The counterpart characteristics of the operational amplifier are, rather, linear next to the origin and then displaying flat saturation far from the origin.

The simplicity of the design and the sigmoidal shape of the produced transfer characteristics make the transconductance amplifier a versatile building block of neural networks as will be shown later. One additional desirable feature of the transconductance amplifier is its high input resistance.

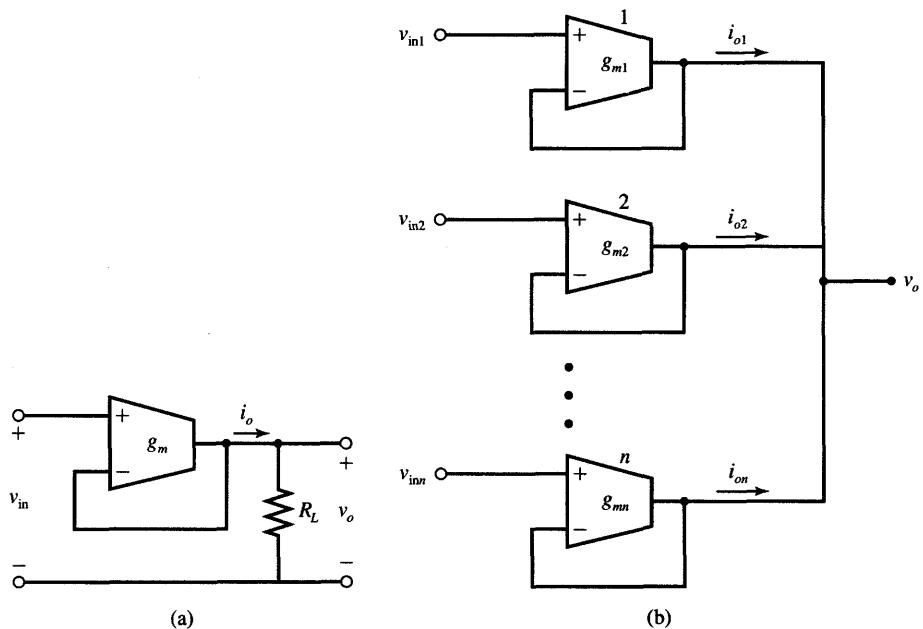
### **Scalar Product and Averaging Circuits with Transconductance Amplifiers**

Transconductance amplifiers can be used to process voltage input signals and are able to produce current or voltage outputs. The circuit can be configured in a number of ways depending on its intended function. Figure 9.30(a) shows the voltage follower circuit using the transconductance amplifier with a negative feedback loop closed. Its output current  $i_o$  is proportional to the difference between the output voltage and the input voltage and is expressed by the following formula:

$$i_o = g_m(v_{in} - v_o) \quad (9.57)$$

Substituting  $i_o = v_o/R$ , the expression for the voltage gain of the voltage follower circuit becomes

$$\frac{v_o}{v_{in}} = \frac{g_m R}{1 + g_m R} \quad (9.58a)$$



**Figure 9.30** Useful configurations of transconductance amplifiers: (a) voltage follower and (b) follower-aggregation circuit.

If the follower is connected to a high-resistance load and  $g_m R \gg 1$ , the output voltage approximately follows the input voltage since we have

$$\frac{v_o}{v_{in}} \cong 1 \quad (9.58b)$$

The most attractive property of the voltage follower as an electronic circuit building block is that the output voltage follows the input. However, no current is taken at its input, thus the circuitry that provides the signal  $v_{in}$  to the follower's input is not loaded due to the follower's very large input resistance value. Also, the follower's output voltage depends only on the input voltage and remains invariant under variable load conditions.

Several voltage followers can be connected to compute a scalar product or the average value of several input voltages (Mead 1989). The circuit diagram for such a computational function of interconnected transconductance amplifiers is depicted in Figure 9.30(b). The configuration is called the *follower-aggregation circuit*. Each amplifier  $i$  has a transconductance  $g_{mi}$ . Each voltage follower supplies an output current proportional to the difference between the input voltage and the output voltage. The contribution of each input  $v_{ini}$  is weighted by the transconductance of the associated amplifier.

For the output node of voltage  $v_o$  we can express the total current as a sum of contributions  $g_{mi}(v_{ini} - v_o)$ , for  $i = 1, 2, \dots, n$ . Since no current leaves the output node we have

$$\sum_{i=1}^n g_{mi}(v_{ini} - v_o) = 0 \quad (9.59)$$

Rearranging the terms of (9.59) and computing  $v_o$  yields

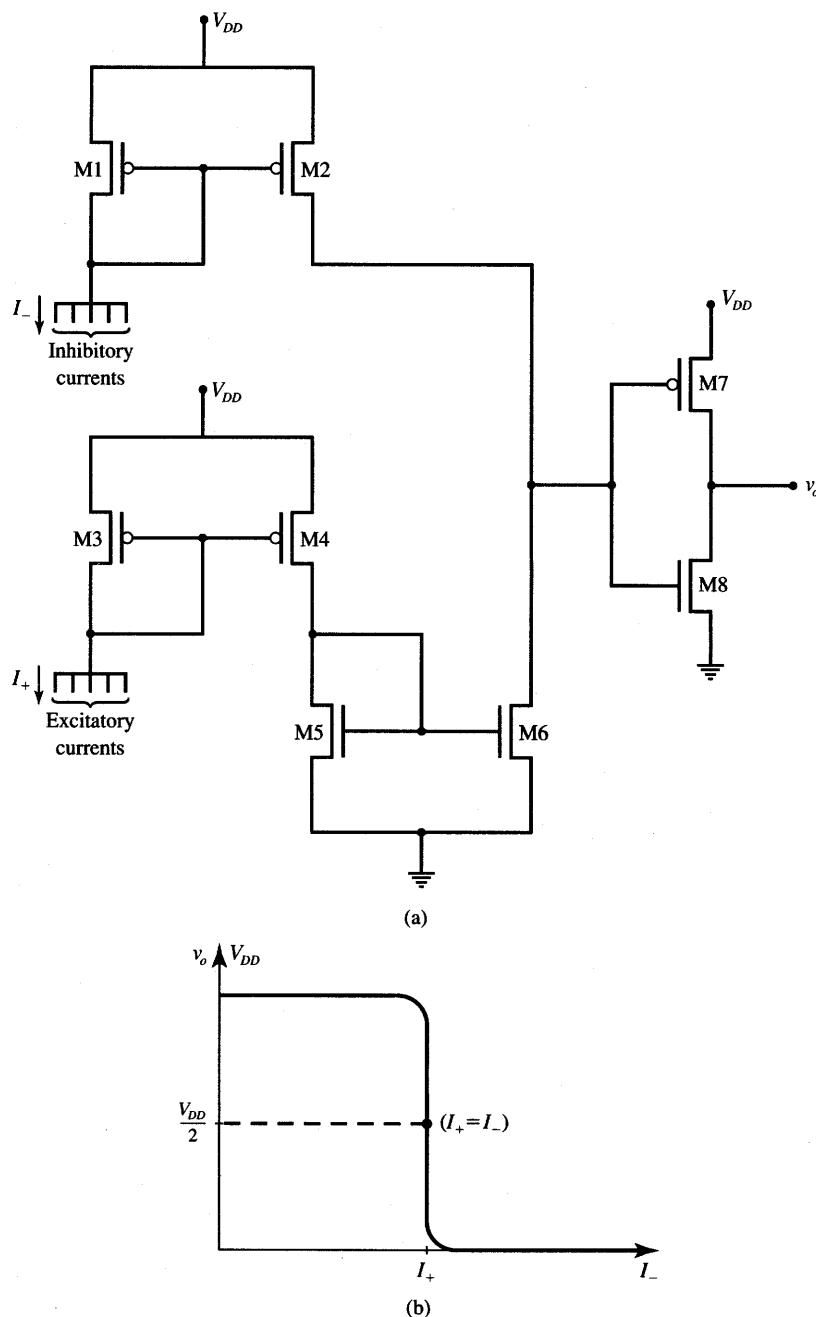
$$v_o = \frac{\sum_{i=1}^n g_{mi} v_{ini}}{\sum_{i=1}^n g_{mi}} \quad (9.60)$$

Equation (9.60) shows that the output voltage  $v_o$  is the weighted average of inputs  $v_{ini}$ , and that each input voltage is weighted by its transconductance. Thus, the follower-aggregation circuit shown in the figure computes the weighted sum of  $n$  input voltages, with transconductances  $g_{mi}$  serving as the weighting coefficients. This function can also be interpreted as computing the average value of an ensemble of inputs. In another interpretation, the output voltage  $v_o$  as in (9.60) can be understood to be a scalar product of weights and inputs, with the expression  $(\sum_{i=1}^n g_{mi})^{-1}$  serving as the proportionality constant.

Formula (9.60) is only valid for the range of input voltages  $v_{ini}$  that ensures the linear operation of each transconductance amplifier. If one of the input voltages  $v_{ini}$  is of excessive positive or negative value, the corresponding transconductance amplifier saturates and does not contribute beyond its bias current. This feature limits the effect of a possible faulty or out of bounds value in the averaging of a large number of signals. Thus, the voltage  $v_o$  will not follow a few off-scale inputs. The discussed implementation of the averaging circuit using voltage followers has great robustness against bad data points (Mead 1989).

### Current Comparator

A current comparator circuit offers another useful alternative solution to the task of summation of inhibitory and excitatory currents, which are produced in synaptic connections at the neuron's input. The comparator sums the inhibitory and excitatory currents flowing through the weights and performs the nonlinear mapping of their difference into the output voltage. The comparator's output voltage may be required to have sigmoidal characteristics as a function of a differential current  $I_+ - I_-$ . Figure 9.31 shows a typical circuit diagram for a CMOS current comparator. It uses two  $p$ -channel current mirrors consisting of transistors  $M1, M2$  and  $M3, M4$ , one  $n$ -channel current mirror consisting of transistors  $M5, M6$ , and one CMOS inverter as a thresholding element (Bibyk and Ismail 1990). The inverter, composed of  $M7$  and  $M8$ , should preferably be electrically symmetric.



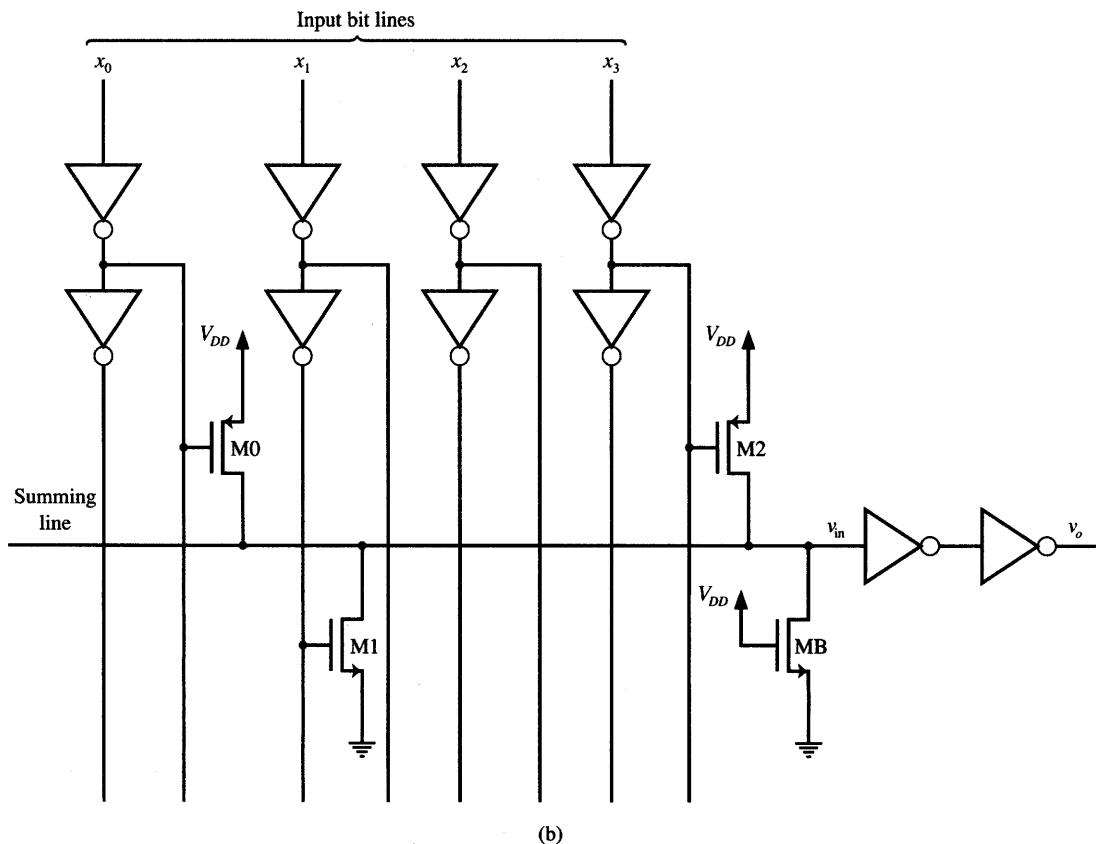
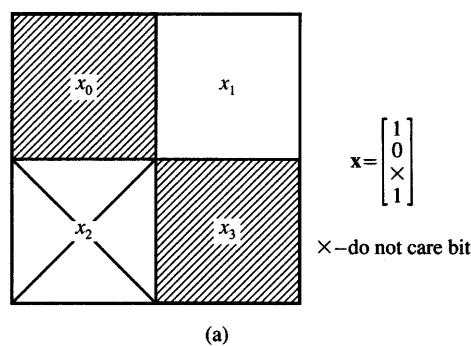
**Figure 9.31** Current comparator: (a) circuit diagram and (b) voltage versus inhibitory current characteristics.

The sum of all excitatory currents,  $I_+$ , is mirrored through the  $n$ -channel current mirror, and it flows through  $M6$ . At the same time the sum of all inhibitory currents flows out from transistor  $M2$ . If  $I_+ = I_-$ , then  $M2$  and  $M6$  will source and sink, respectively, the same amount of current. The output voltage in such a case is equal to  $V_{DD}/2$ . Any imbalance of currents such that  $I_+ < I_-$  causes the output voltage  $v_o$  to decrease from its quiescent, center value of  $V_{DD}/2$ . This is shown in Figure 9.31(b). When  $I_+ > I_-$ , a reverse situation occurs and the output voltage increases. For proper operation of this circuit, the sizing of transistors must be such that the current mirror transistor in each of the three mirrors must remain in saturation during the entire operating range of total excitatory and inhibitory currents. Each of the currents consists of  $n$  synaptic component currents produced by the input component signal weight. In addition, the current mirror design has to provide for accurate balance such that  $v_o = V_{DD}/2$  at  $I_- = I_+$ . This is ensured when  $M2$ ,  $M6$  and  $M7$ ,  $M8$  are electrically identical transistor pairs.

### Template Matching Network

As discussed in previous chapters, template matching represents one of the frequently used forms of neural processing. Template matching can be of particular importance for handling images in bit-map form, and is especially suitable for preprocessing of the binary data. Template matching networks are typically required to decode binary  $n$ -tuple vectors and output the logic level that is unique and different from any outputs that are produced for  $n$ -tuple inputs that do not match the specified template. From this standpoint, an output of the template matching network can be expressed as a combinational Boolean function. Perhaps the most suitable Boolean function for this task is called an equivalency function. It is defined as a complement of an XOR function. A number of techniques exist to design Boolean functions of interest. Although the operation of matching using these well-known techniques can be easily implemented using conventional digital logic gates, we will focus in this section on a design that is analog in nature and possibly involves only a single transistor per bit to be matched (Chung 1991). This ensures the minimum size of the matching network, with only a single transistor per weight, and it provides for analog operation and binary decision output.

An example bit-map template to be matched is shown in Figure 9.32(a). The template shown is represented by the vector  $\mathbf{x} = [1 \ 0 \ \times \ 1]^t$ , where  $\times$  stands for a “do not care” condition. The simple CMOS network of Figure 9.32(b) consists of up to four transistors  $M0$ – $MB$ , one per bit, plus a bias transistor  $MB$ , and it can realize the template bit-map matching function of interest. The transistors are sized in such a way that only the matching bit input pattern results in  $v_{in} > V_{DD}/2$ . This requires symmetrical transfer characteristics for the



**Figure 9.32** Template matching network: (a) example template bit map and (b) circuit diagram.

two output inverters, which produce the voltage  $v_o$  as a function of  $v_{in}$  such that  $v_o = V_{DD}/2$  when  $v_i = V_{DD}/2$ .

When the input matches the value of 1 stored in the template, this turns on the single PMOS bit transistor. This occurs since the gate voltage of the PMOS device is pulled down to zero; see transistors  $M0$  and  $M2$  as examples. The channel resistance of value  $R$  then connects the power supply to the summing line, as would be the case when  $x_0 = x_3 = 1$  for the example network. For a matching bit of zero value, the corresponding bit transistor of the NMOS type remains open. This will be the case for  $M1$  responding to  $x_1 = 0$ .

Assume now that the bias transistor  $MB$ , which is always conductive, also has a channel resistance value of  $R$ . Note that when  $M0$  and  $M2$  are turned on in parallel, they represent the total resistance of  $0.5R$  from the summing line to the  $V_{DD}$  node. Since the total resistance connected between the summing line and the ground is of value  $R$ , this yields a value of  $\frac{2}{3}$  for the voltage division ratio at the summing line. As a result,  $v_{in} = (2/3)V_{DD}$ , and the output voltage is high, thus indicating that a matching input vector is present at the input bit lines. The reader can easily verify that for a mismatch of one or more bits, the summing line voltage drops below the value  $V_{DD}/2$  and the output voltage  $v_o$  can be interpreted as assuming a logic low value.

To design the more general bit template matching network exemplified in Figure 9.32, all transistors must be electrically identical. This requires their device transconductances to be identical so as to yield the resistances of value  $R$  when the devices are conductive. In addition, the bias transistor  $MB$  needs to be sized properly so that the matching input vector yields the value of  $v_{in}$  above  $0.5V_{DD}$ . More detailed inspection of the network leads to the observation that the bias transistor resistance should be chosen equal to  $R/(n - 1)$ , where  $R$  is the resistance of each of the bit transistors and  $n$  is the number of bits equal to 1 in the template to be matched (see Problem 9.20).

## 9.4

### ANALOG MULTIPLIERS AND SCALAR PRODUCT CIRCUITS

Electrically tunable synapses make it possible to multiply the input signal  $x_i$  times the associated adjustable weight value. In a case for which the weight value  $w_i$  is a function of the control voltage, the scalar product of input and weight vector can be expressed in the following form:

$$\overline{net} = \sum_{i=1}^n w_i(v_{ci})x_i \quad (9.61a)$$

where the  $i$ 'th weight value is dependent on the controlling voltage  $v_{ci}$ , for  $i = 1, 2, \dots, n$ . This approach to computing the product of weight and the input signal

based on (9.61a) has been used in most discussions of the analog implementations of neural circuitry with tunable weights presented thus far in this chapter.

Alternatively, the scalar product of an electrically tunable weight vector and of the  $n$ -tuple input signal vector can be obtained as a sum of outputs  $v_i x_i$  of analog voltage multipliers as follows:

$$\text{net} = c \sum_{i=1}^n v_i x_i \quad (9.61b)$$

where  $v_i$  is the voltage directly proportional to the weight value  $w_i$ , and  $c$  is the proportionality constant common for all  $n$  weights under consideration. In contrast to the approach taken in (9.61a), now the weight value can be represented directly using the variable voltage  $v_i$ .

Essentially, the approach based on (9.61b) tends to be slightly more costly to use. Expression (9.61a) allows for implementation of the product of an input variable and of a weight value dependent on a weight control voltage considered to be a parameter. In (9.61b), weights are treated as direct voltage variables rather than as constants controlled by electrical variables. Thus, the products of input signals and weights are obtained through simple analog multiplication.

Below we look at several implementation techniques of analog multipliers. We will also cover scalar product circuits that make use of analog multipliers. Such circuits often prove useful for building artificial neural systems.

### Depletion MOSFET Circuit

A simple integrated circuit analog multiplier can be built from all MOS components (Khachab and Ismail 1989a). The circuit requires an operational amplifier and fully matched depletion mode NMOS transistors working in pairs. The multiplier can operate with both positive and negative input voltages. The basic multiplier circuit diagram is shown in Figure 9.33(a). The input voltage is  $x$ , and the gate voltage  $v$  of transistor  $M2$  is used to express the weight value.

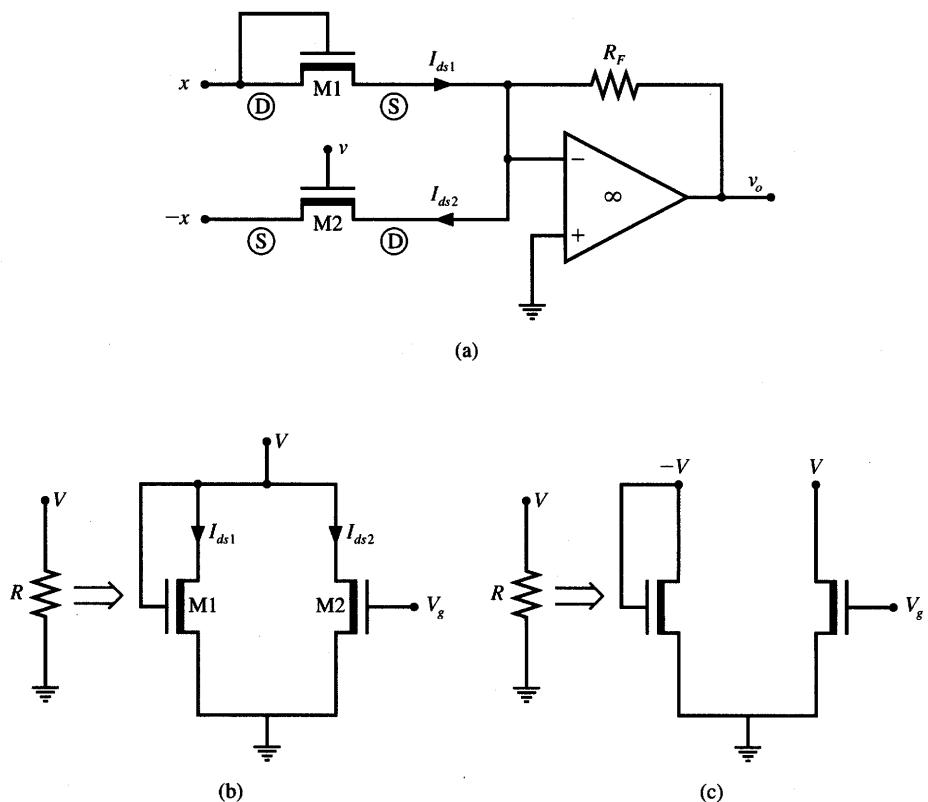
Since both transistors need to remain in the resistive region in this circuit arrangement, their drain currents for  $V_{gs} < V_{ds} - V_{dep}$  are expressed as

$$I_{ds} = k \left[ (V_{gs} - V_{dep})V_{ds} - \frac{V_{ds}^2}{2} \right] \quad (9.62)$$

where  $V_{dep}$  is the depletion voltage of each transistor, typically having a value between  $-2$  and  $-4$  V. Since for  $M1$  we have  $V_{ds} = V_{gs} = x$ , expression (9.62) can be rewritten for this device:

$$I_{ds1} = k \left[ (x - V_{dep})x - \frac{x^2}{2} \right] \quad (9.63a)$$

Note that for the transistor  $M2$  we have  $V_{ds} = x$  and  $V_{gs} = x + v$ ; thus, expression



**Figure 9.33** Analog multiplier with two depletion mode MOS transistors and an operational amplifier: (a) circuit diagram, (b) integrated circuit implementation of resistance, and (c) another version of part (b).

(9.62) for the drain current becomes

$$I_{ds2} = k \left[ (x + v - V_{dep})x - \frac{x^2}{2} \right] \quad (9.63b)$$

Figure 9.33(a) shows that the difference between currents  $I_{ds1}$  and  $I_{ds2}$  is converted to the output voltage  $v_o$  by the closed-loop operational amplifier configured as a current-to-voltage converter. We thus have

$$v_o = -(I_{ds1} - I_{ds2})R_F \quad (9.64a)$$

Plugging expressions (9.63) into (9.64a) results in the following output voltage for the analyzed analog multiplier when it is working in the linear region of the operational amplifier:

$$v_o = R_F k v x \quad (9.64b)$$

The nonlinear (quadratic) terms of  $I_{ds1}$  and  $I_{ds2}$  have canceled due to the subtraction of both currents. Expression (9.64b) therefore remains valid provided both depletion mode transistors operate in the resistive region. Thus, the following condition must be fulfilled:

$$|v + x| \leq |V_{dep}|$$

Formula (9.64b) implies that the circuit of Figure 9.33(a) is able to perform four-quadrant multiplication. The multiplication constant of the obtained multiplier is proportional to the transconductance value  $k$  of the transistors used, and to the user-selected feedback resistance  $R_F$ . Thus, the multiplication constant can be suitably adjusted by the choice of  $R_F$ .

An all-MOS realization of this multiplier can be accomplished by replacing the feedback resistor with two depletion mode transistors operating again in the resistive region. Two possible replacement circuit diagrams for resistors are shown in Figures 9.33(b) and 9.33(c). Drain currents for transistors  $M1$ , and  $M2$  can be obtained for the matching transistor pair  $M1, M2$  of Figure 9.33(b), respectively, as

$$I_{ds1} = kV \left( \frac{V}{2} - V_{dep} \right) \quad (9.65a)$$

$$I_{ds2} = k \left[ (V_g - V_{dep})V - \frac{V^2}{2} \right] \quad (9.65b)$$

Expression (9.65b) is in a standard form (see Appendix) with the condition  $V_{ds} = V$ . Expression (9.65a) is obtained from the same standard form with additional simplification such that  $V_{ds} = V = V_{gs}$ . Since the value of the equivalent resistance implemented by the two transistors connected in parallel can be expressed as follows

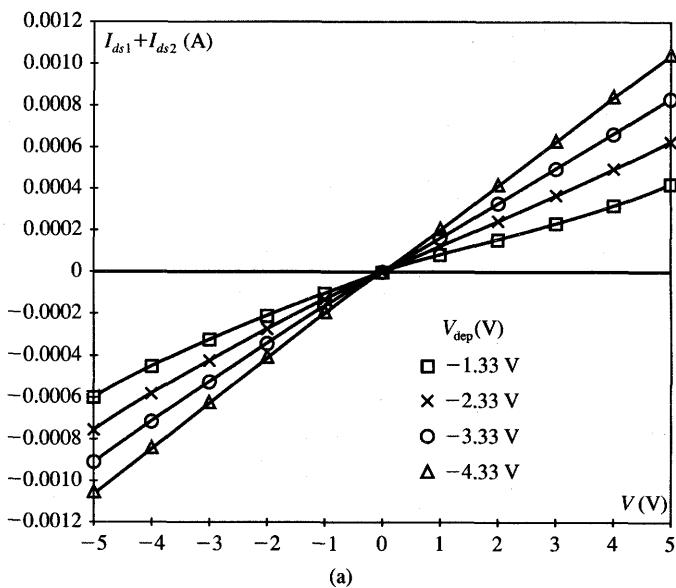
$$R = \frac{V}{I_{ds1} + I_{ds2}} \quad (9.66a)$$

substituting  $I_{ds1}$  and  $I_{ds2}$  from (9.65) into (9.66a) results in the following formula:

$$R = \frac{1}{k(V_g - 2V_{dep})} \quad (9.66b)$$

It can be seen that  $V_g$  is the control voltage used to tune the value of the equivalent feedback resistor. The resistance value also depends on the depletion voltage  $V_{dep}$  of the transistors used. Figure 9.34(a) shows the voltage/current characteristics of the all-MOS resistor of Figure 9.33(b). The curves shown were obtained from the circuit simulation using the SPICE2G.1 program for  $V_g = 2$  V,  $k = 20 \mu\text{A}/\text{V}^2$ , and for four values of  $V_{dep}$  as tabulated in Figure 9.34(b) (Rao 1988).

As stated before, the discussed circuit configuration functions properly only for the resistive mode of operation of both  $M1$  and  $M2$  and thus the following



$V_{\text{dep}}(\text{V})$	$R$ at $V = 0 \text{ V}$ (simulated) ( $\text{k}\Omega$ )	$R$ (from (9.66b)) ( $\text{k}\Omega$ )
-1.33	10.7	10.729
-2.33	7.5	7.507
-3.33	5.8	5.774
-4.33	4.5	4.609

(b)

**Figure 9.34** Characteristics of the all-depletion mode MOS resistor of Figure 9.33(b) ( $V_g = 2 \text{ V}$ ) for  $V_{\text{dep}}$  as a parameter: (a) voltage versus current characteristics and (b) equivalent resistance values.

condition must apply:

$$V + V_g \leq |V_{\text{dep}}| \quad (9.67a)$$

In this particular circuit application, the general condition (9.67a) takes the form

$$v_o + V_g \leq |V_{\text{dep}}| \quad (9.67b)$$

In contrast to the configuration discussed, the resistance produced by the configuration illustrated in Figure 9.33(c) does not depend on the value of the depletion voltage  $V_{\text{dep}}$  of the transistors used. Calculations similar to those performed for the resistor of Figure 9.33(b) yield the following equivalent resistance value

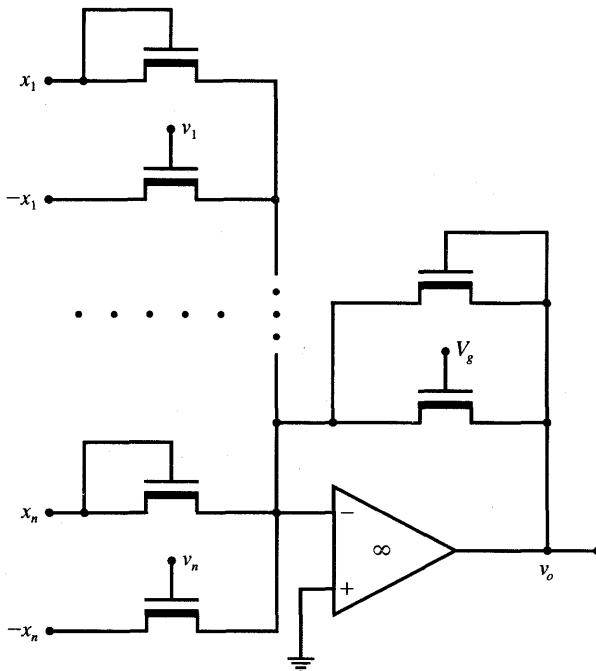


Figure 9.35 All-MOS scalar product circuit with saturation.

for this circuit:

$$R = \frac{1}{kV_g} \quad (9.68)$$

However, to achieve cancellation of both nonlinear terms in the current expressions (9.65) and of  $V_{dep}$ , the input voltages to the multiplier of opposite polarities must be produced and used as inputs for this configuration.

The discussed analog multiplier cell from Figure 9.33(a) can be replicated  $n$  times to implement the full scalar products as expressed in (9.61b). Weight coefficients can be either positive or negative—depending on the polarity of voltage  $v$  used to control the sign and weight value. Figure 9.35 shows the circuit diagram of the scalar product circuit. It computes the following expression, which is a linear combination of the terms of Equation (9.64b):

$$v_o = R_F(k_1v_1x_1 + k_2v_2x_2 + \dots + k_nv_nx_n) \quad (9.69)$$

where  $k_i$  is the transconductance coefficient of transistors in the  $i$ 'th input pair, for  $i = 1, 2, \dots, n$ .

Expression (9.69) becomes similar to (9.61b) for all transistors of identical electrical parameters when  $k_1 = k_2 = \dots = k_n$ . The saturation level, as well as the steepness of the activation function, are set by the voltage  $V_g$ , which controls

the feedback resistance  $R_F$  according to either (9.66b) or (9.68), depending on the resistor configuration used. Both voltage values  $x_i$  and  $-x_i$  are required to drive the inputs of a single synapse of this scalar product circuit. These double-polarity inputs are intrinsically available when neurons are comprised of cascaded inverting amplifiers, or push-pull inverters, as described earlier in this chapter (Bibyk and Ismail 1989).

### Enhancement Mode MOS Circuit

The basic concept of analog multipliers using all-MOS integrated circuitry presented in the last section can be extended to circuits with enhancement mode field-effect devices or other types of active devices. Again, the common-mode nonlinear terms of a transistor's nonlinear current characteristics can be eliminated, and the difference current signal can develop the product of two input voltages at the output of the analog multiplier. The circuit diagram of the analog multiplier cell is depicted in Figure 9.36 (Khachab and Ismail 1989b). The circuit comprises a single operational amplifier and four electrically identical, matched MOS input transistors that could be enhancement or depletion, and both either  $n$ - or  $p$ -type.

Using the model of the transistors in the resistive region and with reference to the notation of Figure 9.36, the current differences at the inputs of the operational amplifier can be expressed as follows (Khachab and Ismail 1989b):

$$I_1 - I_2 = k' \left( \frac{W}{L} \right)_i (x_1 - x_2)(y_1 - y_2) \quad (9.70a)$$

$$I'_1 - I'_2 = k' \left( \frac{W}{L} \right)_f v_o (z_1 - z_2) \quad (9.70b)$$

where  $(W/L)_i$  and  $(W/L)_f$  are width/length ratios of the transistors' channels in the input and feedback paths, respectively. Applying the KCL at each input node of the operational amplifier we obtain

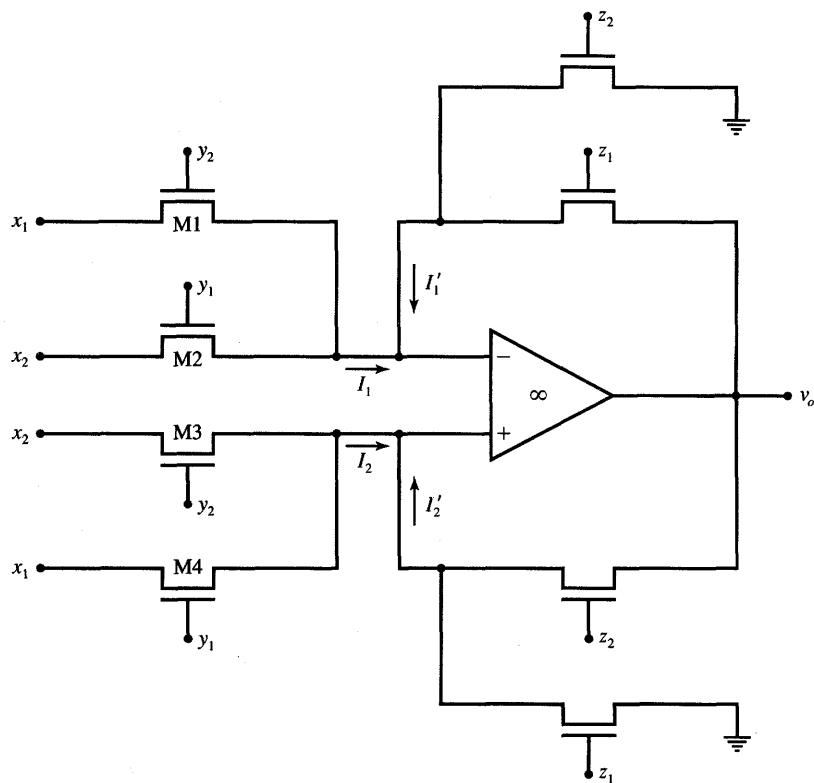
$$I_1 - I_2 = I'_2 - I'_1 \quad (9.71)$$

Hence, the output voltage  $v_o$  becomes from (9.70)

$$v_o = \frac{(W/L)_i}{(W/L)_f} \frac{(x_1 - x_2)(y_1 - y_2)}{z_1 - z_2} \quad (9.72)$$

To ensure the operation of all transistors in the resistive region, the following conditions must apply at the input and output, respectively, of the multiplier

$$\begin{aligned} x_1, x_2 &\leq \min(y_1 - V_{th}, y_2 - V_{th}) \\ v_o &\leq \min(z_1 - V_{th}, z_2 - V_{th}) \end{aligned} \quad (9.73)$$



**Figure 9.36** Analog multiplier cell with four enhancement mode transistors and a closed-loop operational amplifier.

where  $V_{th}$  is the threshold voltage of the matched MOS devices. As can be seen from (9.72), the circuit achieves the product computation of difference input voltages  $x_1 - x_2$  and  $y_1 - y_2$ . In addition, the necessary biasing levels must be provided at the feedback transistors' gates dependent on the type of transistors used. Enhancement mode  $n$ -type transistors require positive gate voltages exceeding  $V_{th}$  if they are to operate properly. Depletion transistors, however, can accept positive, zero, or negative bias voltages at their gates.

Let us require the four-quadrant multiplication to be performed by the multiplier cell with weight control voltage  $v$ , and for the input voltage  $x$  applied at gate and input terminals, respectively. Letting

$$\begin{aligned}x_1 &= -x_2 = x \\y_1 &= Q_o + v, \quad y_2 = Q_o - v \\z_1 &= V_{g1}, \quad z_2 = V_{g2}\end{aligned}\tag{9.74}$$

we obtain expression (9.72) for the multiplier's output voltage, which reduces to

the desired form containing a product of analog voltages  $x$  and  $v$ :

$$v_o = \frac{4(W/L)_i}{(W/L)_f(V_{g1} - V_{g2})} xv \quad (9.75)$$

Note that a scalar product of two  $n$ -tuple vectors can be achieved by a straightforward extension of the simple two-input multiplier just described. In such a case, a total of  $n$  four-transistor circuits (as in Figure 9.36) is needed to combine into products all  $n$  inputs and  $n$  weight control voltages applied at gates. Weights are set equal to the gate control voltages  $v_i$ , for  $i = 1, 2, \dots, n$ , and inputs are  $x_i$ , for  $i = 1, 2, \dots, n$ .

### Analog Multiplier with Weight Storage

An alternative analog multiplier circuit useful for neural network applications can be designed using a matched pair of enhancement type NMOS and PMOS transistors connected in a differential mode (Kub et al. 1990). An example of a circuit that uses two PMOS devices is shown in Figure 9.37. The weight values are represented by the analog voltage  $v$  at the output of the switch  $M_3$ , which serves as an access transistor. The short-term weight storage capacitor  $C$  is connected to the gate of transistor  $M_1$ . The capacitor is charged to the weight control voltage  $v$  for normal weight operation during recall. Capacitor  $C$  can be either an intrinsic gate capacitance of transistor  $M_1$  or a specially formed capacitor structure.

For transistors  $M_1$  and  $M_2$ , both in resistive regions, the drain-to-source currents can be expressed as

$$I_{ds1} = k \left[ (v - V_{th})x - \frac{x^2}{2} \right] \quad (9.76a)$$

$$I_{ds2} = k \left[ (v_R - V_{th})x - \frac{x^2}{2} \right] \quad (9.76b)$$

where  $v_R$  is the weight reference voltage applied at the gate of transistor  $M_2$  and assumed here to be constant. It is also assumed that  $V_{ds} > V_{gs} - V_{th}$  since both  $M_1$  and  $M_2$  must remain in the resistive region to ensure an appropriate operation of this multiplier. The difference current can be obtained using (9.76) as

$$I_{ds1} - I_{ds2} = k(v - v_R)x \quad (9.77)$$

Inspection of Figure 9.37 reveals that the output voltages  $v_{o1}$  and  $v_{o2}$  are equal, respectively,

$$\begin{aligned} v_{o1} &= I_{ds1}R_F \\ v_{o2} &= I_{ds2}R_F \end{aligned} \quad (9.78)$$

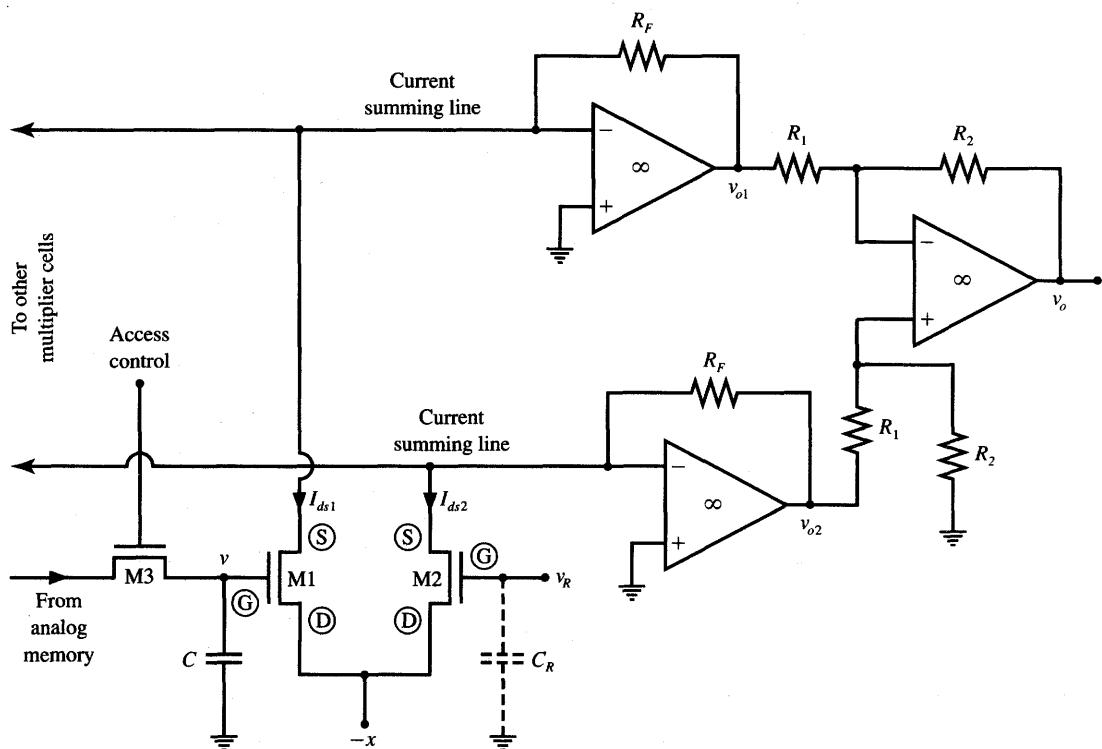


Figure 9.37 Analog multiplier with weight storage.

provided that both current-to-voltage converters work in the linear part of their operational amplifier characteristics. The output operational amplifier is connected as a difference amplifier so as to yield the amplified difference voltage of value

$$v_o = (v_{o2} - v_{o1}) \frac{R_2}{R_1} \quad (9.79a)$$

In the linear range of operation of the output operational amplifier its output voltage can be expressed from (9.77-9.78) as given:

$$v_o = k \frac{R_2}{R_1} R_F (v_R - v) x \quad (9.79b)$$

Note from (9.79b) that the resistances  $R_1$ ,  $R_2$ , and  $R_F$  can be used to control the steepness of the activation function in the linear region. As explained in the previous section, these values also set the range of the analog product value and its linear or nonlinear mapping to the output  $v_o$ . Obviously, if  $n$  multipliers are connected to each of the current summing lines, we obtain the total of  $I_{ds}$  currents of each line as  $n$ -tuple sums. In such cases, the output voltage  $v_o$  expressed for

a single weight by expression (9.79) yields the scalar product as follows:

$$v_o = k \frac{R_2}{R_1} R_F \sum_{i=1}^n (v_{R_i} - v_i) x_i \quad (9.80)$$

where  $v_{R_i}$  is the reference voltage of the  $i$ 'th weight.

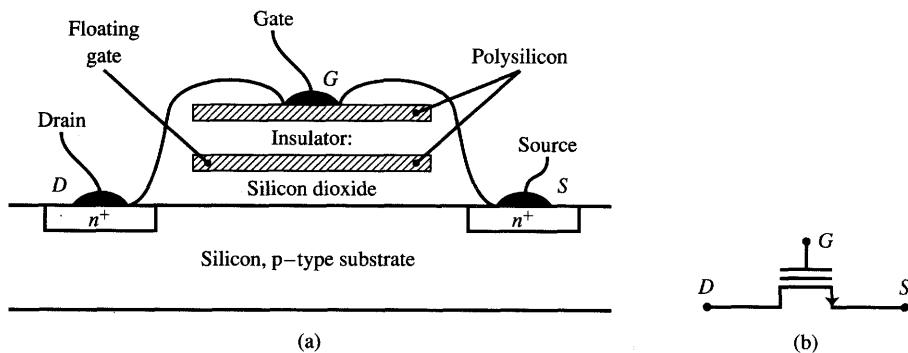
A  $11 \times 11$  matrix-vector multiplier based on the discussed concept has been implemented in CMOS  $p$ -well technology (Kub et al. 1990). Measurements of analog capacitive storage characteristics have shown that the weight decay rate due to the lack of weight refresh has been 30 mV/s at room temperature. The intrinsic gate capacitances with an estimated value of 90 fF have been used as capacitive storage elements. The weight values have been permanently stored in electrically erasable programmable read-only memory (EEPROM) or RAM in digital form. The analog weight voltage values have been provided at the inputs to the access transistors as output signals of digital-to-analog converters.

Substantial reduction of the weight control voltage decay rate has been achieved in the circuit of Figure 9.37 by adding a second capacitance  $C_R$  at the gate of transistor  $M2$ . Since the leakage currents of the gate capacitances of  $M1$  and  $M2$  are similar, they tend to cancel each other when the multiplier output currents are subtracted. In the measured integrated circuit with both  $C$  and  $C_R$  used, the decay rate has dropped to 0.6 mV/s. This configuration, however, has required the addition of another access transistor  $M4$  to store the weight reference voltage  $v_R$  (transistor  $M4$  not shown in the figure). The multiplier cell from Figure 9.36 with three transistors fabricated using the 3- $\mu\text{m}$   $p$ -well CMOS technology was of  $56 \mu\text{m}^2$ . The entire chip consisted of  $11 \times 11$  matrix-vector multiplier circuitry. It has been configured to compute 11 scalar products simultaneously, each with 11-tuple vectors.

### Floating-Gate Transistor Multipliers

Floating-gate transistors offer rather simple and efficient circuit implementation techniques for artificial neural systems. In particular, the technique is suitable for designing large networks fabricated with modern VLSI technologies (Bibyk and Ismail 1990). The most attractive feature of floating-gate MOS transistors used as analog multipliers for neural networks is that both the weight storage and analog multiplication of weights and inputs are implemented concurrently and by the same circuitry. The floating-gate transistors provide an adjustable, non-volatile analog memory, and, simultaneously, they behave as analog processors when appropriately connected.

The cross-section of a floating-gate MOS transistor is shown in Figure 9.38(a). The device is similar to the MOS transistor, but in addition to the regular gate shown as a top layer, it has one more layer of polysilicon to store a trapped charge. Once the charge is trapped in this layer called the floating gate,



**Figure 9.38** Floating-gate MOS transistor ( $n$ -type): (a) device cross-section and (b) symbol.

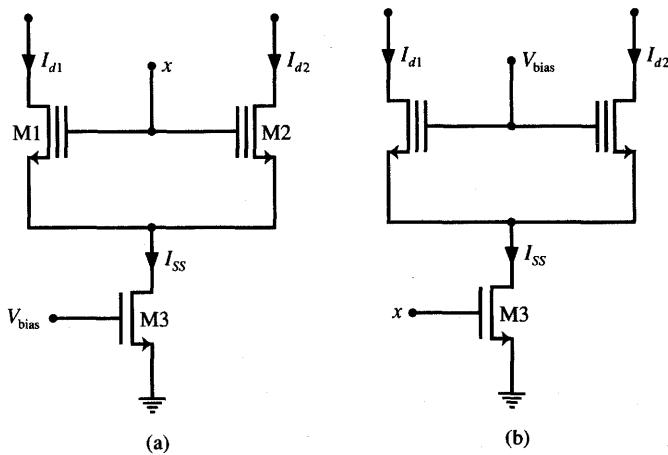
it produces a shift in the threshold voltage of the transistor. Figure 9.38(b) shows the symbol of the device. Note that the floating gate is not connected to any other part of the circuitry.

Let us briefly examine the operation of the floating-gate transistor. Before the device is programmed, it behaves as a regular  $n$ -channel enhancement mode MOSFET. In this case its threshold voltage is rather low. To program the floating-gate transistor, a large pulse voltage of about 20 V is applied between the gate and source terminals. Simultaneously, a large 25 V pulse of voltage is applied between the drain and source terminals. Due to the physical phenomena described in more detail elsewhere in the literature (Sedra and Smith 1991), the floating gate acquires charge. In this state, called the programmed state, the device threshold voltage is substantially higher than in the nonprogrammed state. Moreover, the value of the programmed threshold voltage can be controlled through the duration and level of voltage pulses applied. Once programmed, the device is able to retain the threshold value for years.

To return the floating-gate MOS transistor to the non-programmed state, the erasure process needs to take place. The erasure corresponds to the removal of the charge from the floating gate. It is accomplished by illuminating the floating gate with ultraviolet light of a suitable wavelength.

Let us now focus on the application of floating-gate devices for analog multipliers with simultaneous weight storage. The multiplier consists of a single-transistor current sink and two threshold-programmable transistors with gates tied together and connected as a differential pair as shown in Figure 9.39(a). Because transistors  $M_1$  and  $M_2$  are a perfectly matched pair, and thus nonprogrammed,  $I_{d1}$  and  $I_{d2}$  remain exactly the same. Through device programming, the threshold voltages are made unequal. The resulting difference of drain currents  $I_{d1}$  and  $I_{d2}$  becomes proportional to the difference of the programmed threshold voltages.

Assume that the current differences are sensed at the input of the neuron that produces the scalar product function and its sigmoidal mapping as discussed



**Figure 9.39** Floating-gate weight storage / multiplier configurations: (a) circuit diagram and (b) modified circuit.

earlier in this chapter. When input  $x$  is low, the circuit makes no current contribution to the neuron since  $M1$  and  $M2$  are cut-off and  $I_{d1} = I_{d2} \cong 0$ . For  $x$  larger and positive, the difference current  $I_{\text{diff}}$  becomes proportional to the threshold voltage difference of the two transistors.

An analysis of this circuit (Borgstrom, Ismail, and Bibyk 1990) shows that the differential current can be expressed:

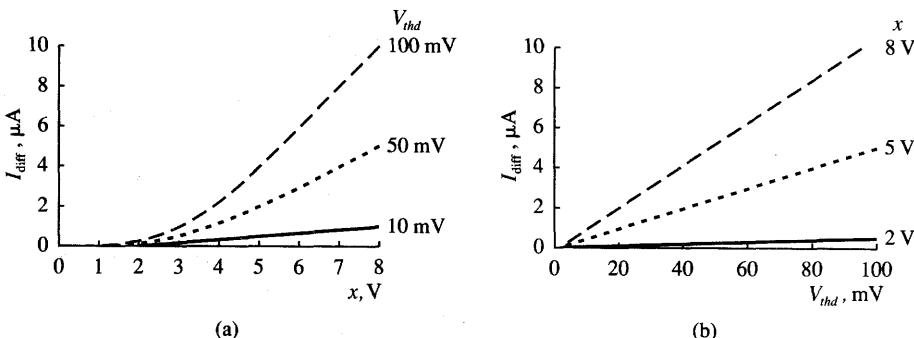
$$I_{\text{diff}} \stackrel{\Delta}{=} I_{d1} - I_{d2} \quad (9.81\text{a})$$

$$I_{\text{diff}} = V_{\text{thd}} \sqrt{\frac{kI_{ss}}{2}} \quad (9.81\text{b})$$

where the threshold voltage difference is  $V_{\text{thd}} = V_{\text{th2}} - V_{\text{th1}}$ , and  $k$  is the device transconductance common for both  $M1$  and  $M2$ . It is seen from (9.81b) that the current  $I_{\text{diff}}$  is proportional to  $V_{\text{thd}}$ , and also to the input voltage  $x$  in a narrow range of values. Although the latter relationship is not explicit in (9.81b), note that  $x$  establishes the level of the total current  $I_{ss}$ . To improve the performance of the circuit, simple modification of swapping  $x$  and  $V_{\text{bias}}$  as shown in Figure 9.39(b) can be undertaken. The current  $I_{\text{diff}}$  for this modified circuit is of approximate value

$$I_{\text{diff}} \cong xV_{\text{thd}} \sqrt{\frac{1}{2} k_{\text{inp}} k} \quad (9.82)$$

where  $k_{\text{inp}}$  is the device transconductance of the input transistor  $M3$ . Thus, the differential current  $I_{\text{diff}}$  in (9.82) is proportional to the product of both the input voltage  $x$  and of  $V_{\text{thd}}$ . The value of  $V_{\text{thd}}$ , however, is made through programming proportional to the weight value. Figure 9.40 shows the differential current



**Figure 9.40** Differential current characteristics for the modified circuit of Figure 9.39(b): (a) versus the input voltage and (b) versus the difference of threshold voltages. [© IEE; adapted from Borgstrom, Ismail, and Bibyk (1990) with permission.]

responses of the modified multiplier circuit from Figure 9.39(b). The graphs have been obtained through circuit simulations. They illustrate excellent linearity of  $I_{\text{diff}}$  with respect to  $V_{\text{thd}}$ , and good linearity with respect to  $x$ , in particular for large values of inputs. This can be seen from comparison of curves from Figures 9.40(a) and (b).

Floating-gate multipliers from Figure 9.39(a) can be used with a current comparator, example of which are shown in Figure 9.31(a). The inputs to the synapses,  $x$ , are single-ended voltages. The effective “weight” represented by the synapse is proportional to the difference of the two currents  $I_{d1}$  and  $I_{d2}$ , which is, in turn, proportional to the preprogrammed difference in the threshold voltages  $V_{\text{th}2}$  and  $V_{\text{th}1}$ . A single neuron with floating-gate programmable synapses can be obtained by connecting inputs  $I_{d1}$  of a number of synapses from Figure 9.39 to the input  $I_+$  of the current comparator of Figure 9.31(a), and by connecting inputs  $I_{d2}$  of the synapses to the input  $I_-$  of the current comparator.

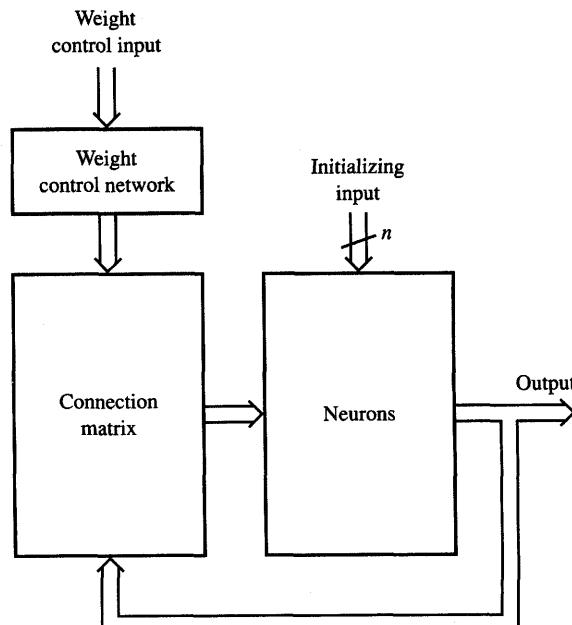
An experimental prototype of the neural network described above has been designed, fabricated, and tested (Borgstrom, Ismail, and Bibyk 1990). The chip was implemented with a 2- $\mu\text{m}$  CMOS double-poly p-well process. Poly 1 layer has been used to make the floating gate, and the poly 2 layer as the top gate. The transistors’ aspect ratio used in the multiplier has been 20  $\mu\text{m}/20 \mu\text{m}$ , and large  $W$  and  $L$  values have been selected to minimize mismatch due to the geometrical inaccuracies. The designed neural network uses the described current-mode technique and is especially attractive for modern VLSI technology. It offers programmable weights, excitatory and inhibitory current summation, the sigmoidal nonlinearity function of the current comparator, and the output in the form of voltage. The current-mode processing and current-mode building blocks used by the network are of special importance for microelectronic technologies where voltage signal handling is often limited for analog applications.

## 9.5

### ASSOCIATIVE MEMORY IMPLEMENTATIONS

This section briefly describes how to integrate electronic circuit components into an associative memory circuit. Since only modest precision is required for the nonlinear processors (neurons) and coupling elements (weights), associative memories are particularly convenient and attractive for microelectronic implementation. In a typical memory circuit, input and output data are in binary form. However, as explained in Chapter 6, analog computation is performed by the network throughout the recall phase. This feature makes it easy to integrate an associative memory circuit as a specialized module within large digital systems. Although feedforward networks can also be termed, at least in a certain respect, as associative memories, we have reserved the term "memories" for networks that are typically trained in the batch mode, and are dynamical systems with feedback, which is activated during the recall phase. Only such dynamical memory systems are discussed here.

Figure 9.41 shows a general schematic representation of a single-layer associative memory. The programming of the connection matrix is performed by a weight control network, which is responsible for storing appropriate weights at suitable locations within the connection matrix. The connection matrix contains



**Figure 9.41** Schematic diagram of a single-layer autoassociative memory network.

binary or, more accurately, ternary weights of values 0, +1, and -1; sometimes weights need to be set with higher resolutions. The weight control network can be designed either as a microprocessor-based controller or as a dedicated memory control network. This type of weight control network usually involves either a digital memory of RAM type, or registers, as well as address decoders, transmission gates, and buffers.

One of the first associative memory circuits involves implementation of binary weights of values 0, +1, and -1 (Graf and Vegvar 1987). The weight circuitry has been designed as shown in Figure 9.42. The figure depicts a single weight configuration and an associated weight control circuitry. The circuit from Figure 9.42(a) uses two RAM cells to program the single weight value. If DataOut1 = 0 and DataOut2 = 1, both transistors  $M2$  and  $M4$  are cut-off, thus allowing no current to or from the input line of the  $i$ 'th neuron. This programs a missing connection, or  $w_{ij} = 0$ .

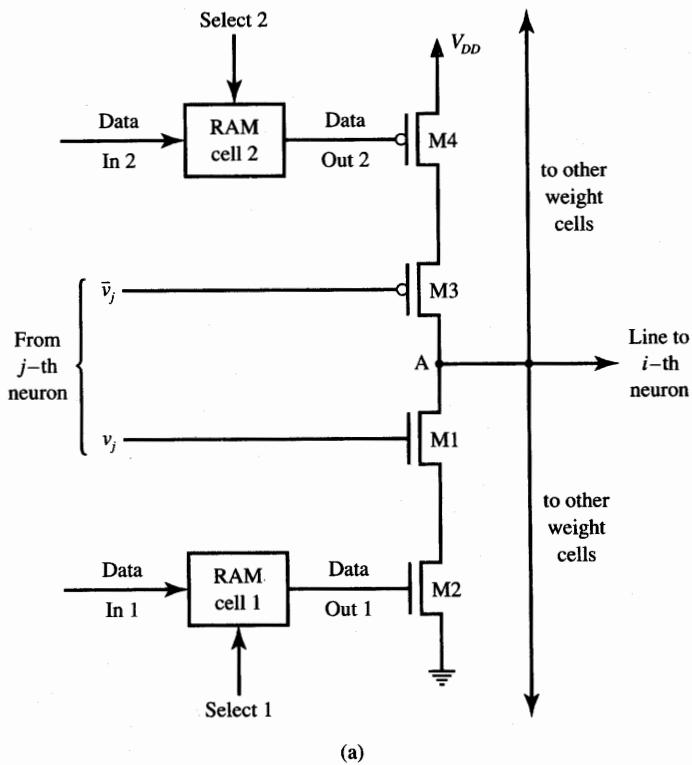
Assume now that  $v_j = V_{DD}$ . An excitatory current flowing toward the  $i$ 'th neuron can be produced by programming DataOut1 = DataOut2 = 0. The excitatory current  $i_{exc}$ , which is sourced to the line, can be denoted as shown on the diagram of Figure 9.42(b). Tied transistors  $M3$  and  $M4$  can be represented as their respective channel resistances  $R_{ds3}$  and  $R_{ds4}$ . This set of control signals from RAM cells implements  $w_{ij} = 1$ .

When the memory outputs DataOut1 = DataOut2 = 1, while  $v_j$  remains positive, the direction of current reverses as shown in Figure 9.42(c). In this way  $w_{ij} = -1$  is produced due to the sinking of current away from the current input line of the  $i$ 'th neuron. This circuit arrangement imitates the apparent negative conductance  $g_{ij} = w_{ij}$  of unity value between the output of the  $j$ 'th neuron and the input of the  $i$ 'th neuron. This negative conductance effect is due to the counterflow of sinking current  $i_{inh}$  with respect to the sourced current  $i_{exc}$ .

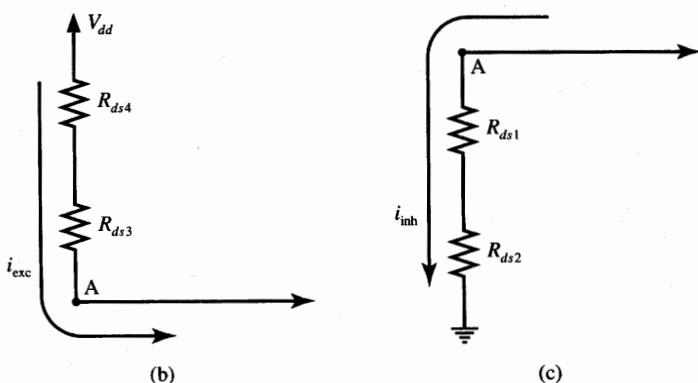
The circuitry of Figure 9.42 implements fixed and ternary weight values. It is obvious that by selecting weighted resistances of MOSFET devices, the amount of current sourced,  $i_{exc}$ , or current sunk,  $i_{inh}$ , could be controlled with better resolution. Several circuit techniques to achieve a higher resolution of resistances along with the principle of analog or digital tunability of weights that could be used for associative memory circuits were discussed earlier in this chapter.

An example auto-associative memory chip was fabricated with  $2.5 \mu\text{m}$  CMOS technology (Graf and Vegvar 1987). The chip contained about 75 000 transistors in an area  $6.7 \text{ mm} \times 6.7 \text{ mm}$ . Figure 9.43 shows the schematic of memory interface with the data input/output. There are 54 neuron-amplifier pairs used in the memory, each neuron-amplifier being a simple inverter. Data input and output are made through transmission gates  $T1$  and  $T2$ , respectively. Data are fed through a buffer containing one storage cell, each connected to the output neuron. The content of the buffer can be utilized to program synaptic connections using the approach shown in Figure 9.42.

To initialize the memory output, the READ/WRITE transistor needs to conduct in order to pass the signal to the neuron from the buffer. The memory



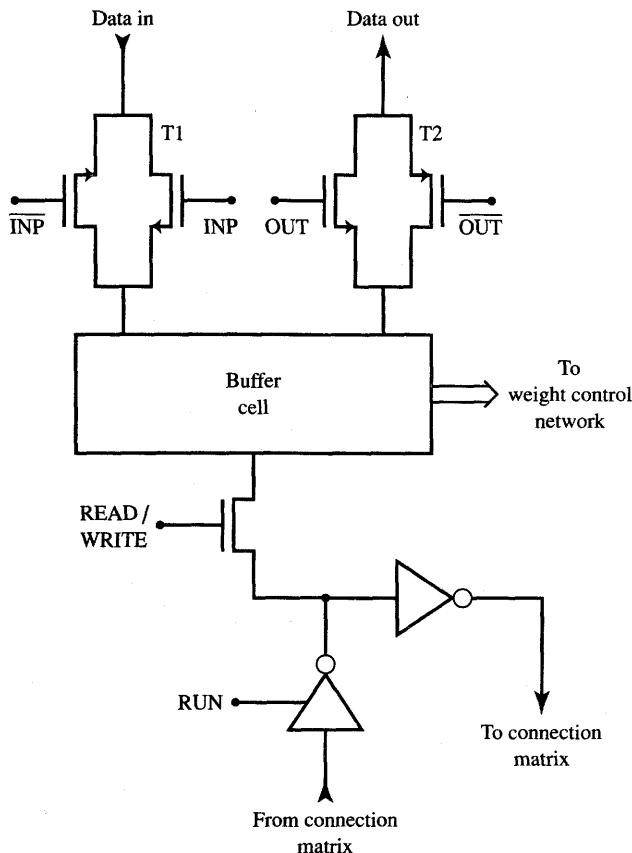
(a)



(b)

(c)

**Figure 9.42** Implementation of binary weights: (a)  $w_{ij}$  weight cell diagram, (b) current sourcing for  $w_{ij} = 1$ , and (c) current sinking for  $w_{ij} = -1$ .



**Figure 9.43** Neuron cell with buffer unit and controlling gates. [Adapted from Graf and Vegvar (1987); © IEEE; used with permission.]

feedback loop remains open during the initialization. Following initialization, the RUN control signal is turned to initiate recall, and the network evolves to an equilibrium state. After the circuit has reached a stable state, the READ/WRITE transistor passes the output voltage data to the buffer. These data can be read out as the components of the memory recalled binary vector.

The fabricated network consists of  $54^2 = 2916$  binary weights designed according to the concept of Figure 9.42(a) (Graf and Vegvar 1987). Tests on the chip have been run with up to 10 vectors programmed as stable states. Auto-associative recall has been performed reliably, and only the states programmed into the circuit and no spurious states have been retrieved during experiments. A large spread of convergence times has been observed dependent on the overlap of the input and memory vector. The measured recall times have been between 20 and 600 ns. In more general cases, the convergence time scale would depend

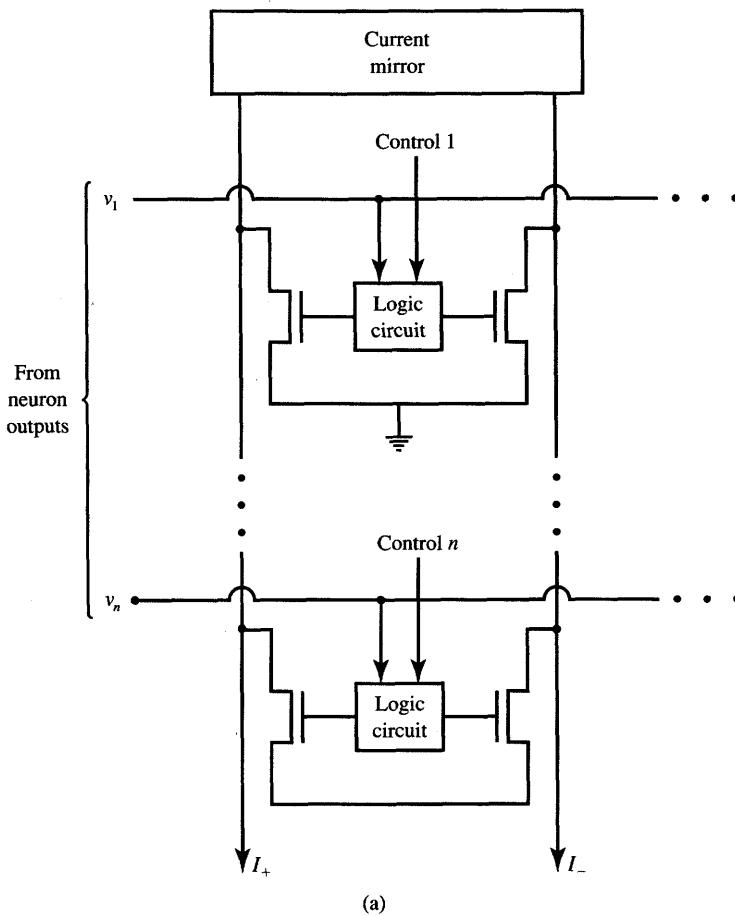
on the product of weight resistances times the intrinsic input capacitances of the neurons used. The power dissipation measured for the single convergence of the memory has been typically of the order of 200 mW, reaching 500 mW in the worst case.

The drawback of single input line current summation used in this memory design is that excitatory and inhibitory currents may not be the same due to the mismatching between the sourcing and sinking circuitry. Since the mismatches are subsequently summed, this can produce a substantial error in the total input current value (Verleysen and Jespers 1991). To compensate for potential lack of symmetry due to the mixture of *n*- and *p*-type MOSFETs in weight circuitry as in Figure 9.42, a two-line system can be used instead. Such arrangements were already discussed in more detail in Section 9.3 for neural network architectures other than associative memories.

In this technique, excitatory and inhibitory currents are summed on separate lines and weight currents are contributed to each of the lines by the same type transistors. Figure 9.44(a) illustrates a two-line current summation. Each synapse is a differential pair controlled by two RAM cells as shown in more detail in Figure 9.44(b) (Verleysen and Jespers 1991). If DataOut1 = 1, current is sunk through M0, provided either *M*1 or *M*2 is conductive. Transistor *M*1 conducts and contributes to the excitatory current when Out is high. At the same time *M*2 remains cut-off. The contribution to the inhibitory current is generated when Out is low, thus *M*2 is conducting and *M*1 is cut-off. The current mirror circuit needs to be used as described in previous sections in order to compare the excitatory and inhibitory currents and to produce the output voltage as a function of their difference.

As suggested earlier, associative memories can also utilize the channel resistances of MOSFET devices as tunable weights to achieve weight setting with higher resolution. The concept of using properly biased field-effect transistors to implement weights was discussed in more detail in Section 9.2. Let us now look at an example of a hardware implementation of a versatile and expandable connection matrix chip reported by Moopenn et al. (1988).

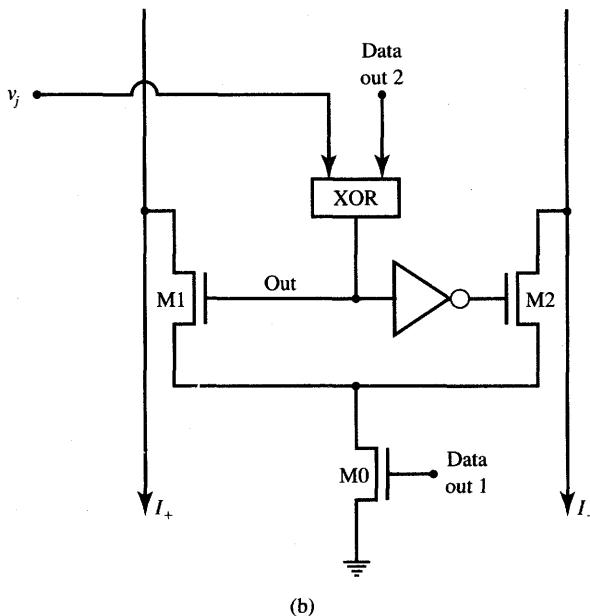
A single synaptic chip with 1024 weights has been implemented that contains an array of  $32 \times 32$  "long channel" NMOS transistors as illustrated in Figure 9.45. Since neurons are kept off-chip, the circuit remains purely "synaptic" and is thus easily cascadable with identical chips. Such a multichip architecture allows for expansion of the chip up to the size of  $512 \times 512$  weights, if chips are arranged in an array. In addition, the chips can be connected in parallel, thus allowing for higher than binary weight resolution. In this type of connection mode, the conductances are added since individual weights are connected in parallel. To ensure low-power dissipation within the connection matrix, weight resistances are selected from large values, thus taking small current levels. For this reason, large *L/W* (long channel) transistors are used on the chip (*L* = 244  $\mu\text{m}$ , *W* = 12  $\mu\text{m}$ ). This ensures typical channel resistance  $R_{ds}$  values greater than 200 k $\Omega$  in the



**Figure 9.44a** Two-line current summation: (a) circuit diagram.

entire operating range [see formula (9.21)]. Also, a large ratio  $L/W$  enhances the linearity of the controlled weights, for a given range of operating voltages, thus it results in increased weight accuracy. This also increases the precision of implemented weights, which becomes of particular importance when connection matrices need to be connected in parallel in order to achieve better resolution.

Figure 9.45(a) depicts the block diagram of the fabricated synaptic chip and also shows an expanded single weight cell. Transistor  $M_2$ , controlled by the gate with voltage  $Q$ , acts as an ON/OFF switch. The  $Q$  output is provided by the latch, which is externally addressed via the row and column address decoders. Transistor  $M_1$  operates as a long channel resistance  $R_{ds}$  controlled by the gate voltage  $V_G$ . Since  $V_G$  takes only one nonzero value in this circuit arrangement,

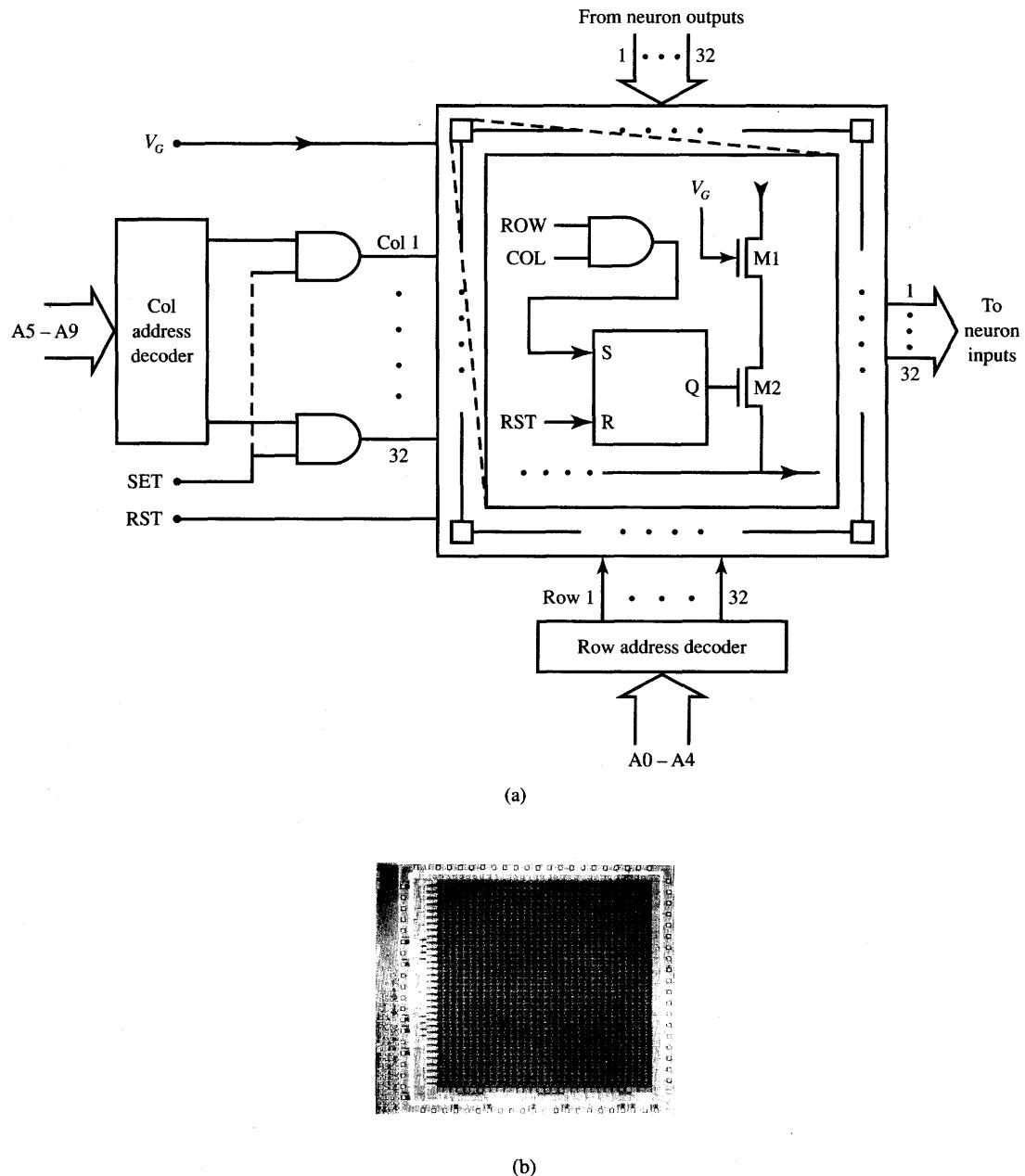


**Figure 9.44b** Two-line current summation (continued): (b) implementation of binary weight.

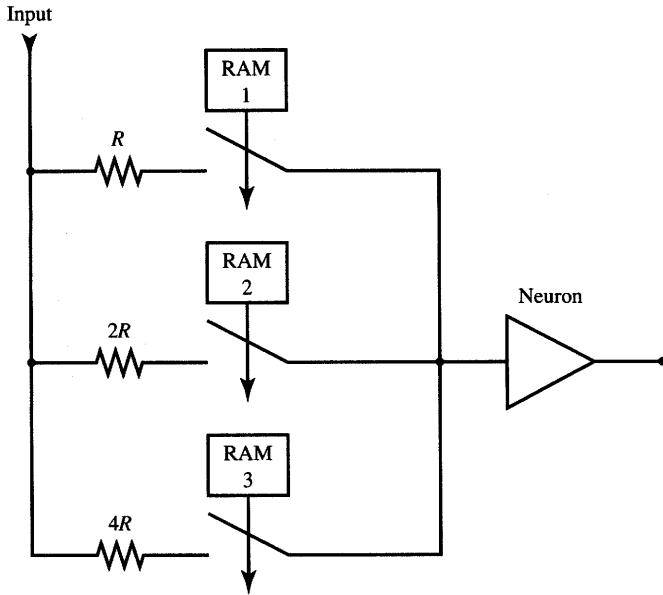
only binary unipolar weights can be produced in this circuit. With a weight control voltage  $V_G$  of 5 V, for example, values for  $R_{ds}$  of about 200 k $\Omega$  can be obtained over an input operating range. The weights' linearity of 5% has been measured for the input voltage in the range up to 1.5 V.

Figure 9.45(b) shows the microphotograph of the 32  $\times$  32 connection chip. The synaptic chip has been fabricated using 3  $\mu$ m bulk CMOS process. A microcomputer interface needs to be used in this project to program the weight values, to initialize the memory, and to provide read-out of the recalled vector. The chip has been used as a basic building block for a number of associative memory experiments. An average convergence time within the connection chip has been observed of 10  $\mu$ s, with some recalls taking up to 50  $\mu$ s. The discussed associative memory network has demonstrated an excellent stability and associative recall performance.

Another approach to building a reconfigurable memory connection matrix is shown schematically in Figure 9.46. The weight value can be adjusted by turning the binary weighted resistances on or off. The resistances are controlled by 1-bit RAM memory switches (Lee and Sheu 1991). The figure exemplifies weights that can be arranged externally for up to 3-bit resolution. Resistances can be realized either as channel resistances of MOS transistors or as current sinks/sources connected to current mirrors.



**Figure 9.45** Synaptic chip with long channel MOSFETs and binary weights: (a) chip architecture and single weight implementation and (b) microphotograph of a CMOS 3- $\mu\text{m}$  chip with 1024 weight cells. [© American Institute of Physics; adapted from Moopenn (1988) with permission.]



**Figure 9.46** Block diagram of a 3-bit resolution associative memory weight cell.

## 9.6

### ELECTRONIC NEURAL PROCESSORS

In this chapter we have formulated the basic requirements and outlined the operational parameters of artificial neural hardware processors. We also discussed numerous issues related to the design of special-purpose artificial neural systems. Much of the material presented to this point has been oriented toward the implementation of the basic building blocks of neural hardware. Synaptic connections, neuron elements, elementary current-mode processors, and scalar product circuits have been among the main building blocks covered. Our emphasis on the analog neural processing has been due to the fact that it offers size and speed advantages compared to conventional numerical simulations. Analog neural processing is also much more efficient than purely digital hardware computation since convergence times for parallel analog hardware are significantly smaller. We have also seen that some electronic implementations can be made as combinations of analog and digital processors. While recall and often also learning are carried out as analog processes, digital programmability and digital control and interface offer increased accuracy and flexibility.

This section discusses sample electronic neural network implementations. Rather than continue to focus on the principles and basic building blocks, here

we outline the main features of ready-to-use neural processors. Some of them appear experimental, some are available commercially—nevertheless, all of the discussed systems are reported to be complete and functional as described in the technical literature. Included in this section are Intel's ETANN chip (Intel 1991), the analog programmable neural processor (Fisher, Fujimoto, and Smithson 1991), a general-purpose neural computer (Mueller et al., 1989, 1989b), and an optically controlled neural network (Frye et al. 1989).

The Electrically Trainable Analog Neural Network (ETANN) 80170 integrated circuit chip is a commercial product suitable for real-time neurocomputing applications. Its applications include recognition of printed or handwritten characters, diagnosis of fault patterns, signal recognition, image processing and convolution, speech recognition, nonlinear process control, robotic motion control and tactile pattern recognition. The chip operates in the recall phase with a speed of  $2 \cdot 10^9$  CPS, which is well beyond the capabilities of both the neural network software and hardware accelerator boards.

The ETANN 80170 chip is trainable in two modes. Given a specific mapping task to be performed, synaptic weights can be trained for a selected neural network model. This task can be done off-chip either by the user's software or by the learning simulation software provided with the chip. Downloading the weights on the chip terminates this learning performed "off-the-chip" on a programmable host neurocomputer. Also, off-chip learning maximizes the flexibility of training approaches and of architectures potentially chosen for the network.

Alternatively, the chip can be subject to "chip-in-the-loop" learning. In this mode of training, the network itself performs the recall function and is trained on-line, while software is used for recomputing the weights during training. This mode of learning enables more realistic network development because the chip characteristics and imperfections are included and accounted for in each weight adaptation step. Because the chip uses EEPROM technology, the weights can be changed many times. The programmable weights are trainable with up to six bits of resolution.

The ETANN chip comes in a 208-pin package. It consists of 64 programmable neurons and a total of 10 240 programmable weights. The block diagram of the chip is shown in Figure 9.47(a). Each neuron is capable of computing the output for 128 input weights with variable inputs and 32 fixed-bias weights. Figure 9.47(b) illustrates programmable synapse operation. The weight value is programmed and stored as threshold voltages of a pair of floating-gate MOS transistors as described in detail in Section 9.4. The circuit shown in the figure performs the analog multiplication of the differential voltage  $\Delta V_{in}$  times the weight value, which, in turn, is proportional to the threshold voltage difference. The output from the synapse is in the form of the current difference  $\Delta I_{out}$ .

Families of the synaptic connections' current characteristics shown in Figure 9.48 illustrate the performance of programmable weights. Figure 9.48(a) shows the differential output current as a function of the weight input voltage with

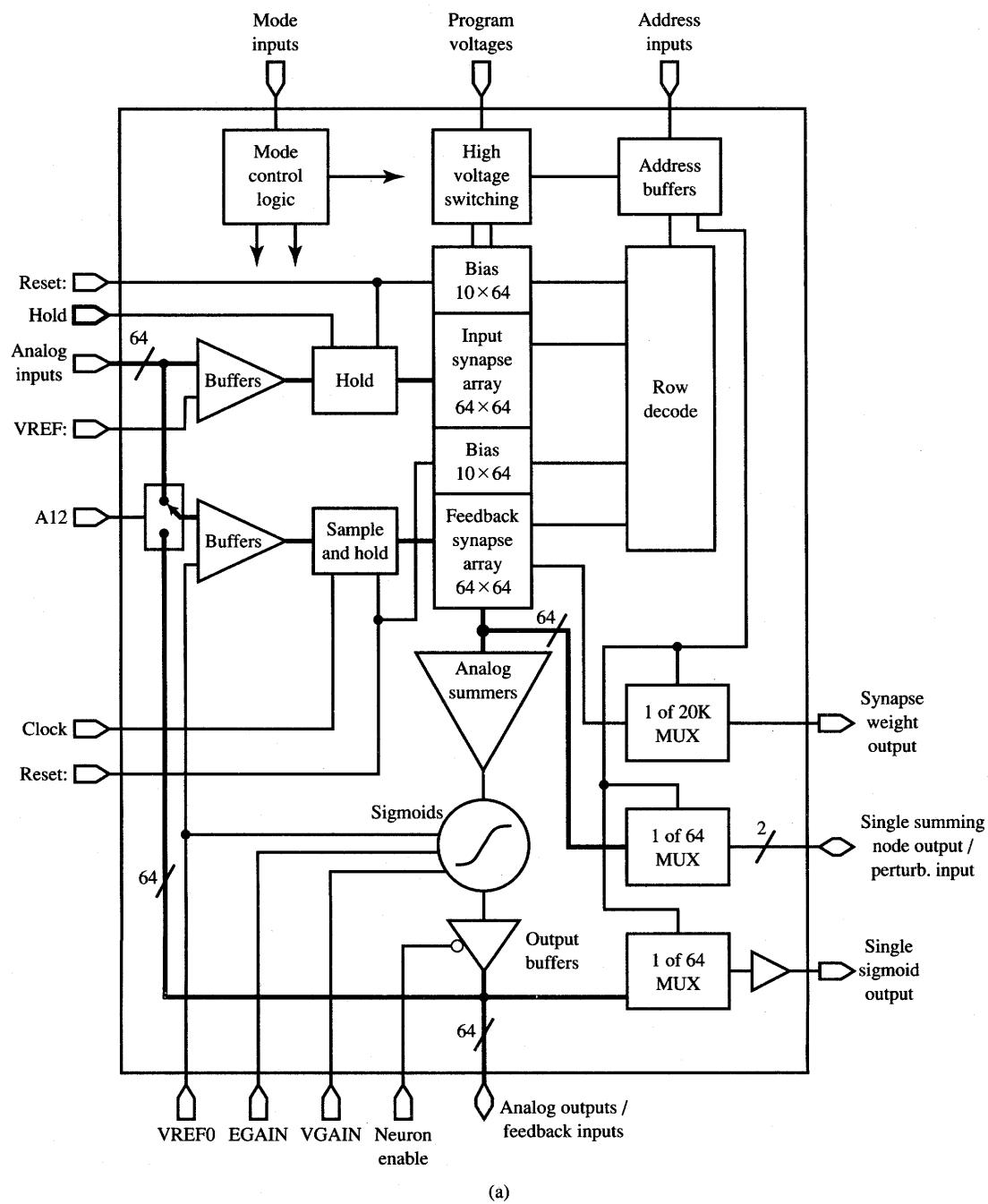
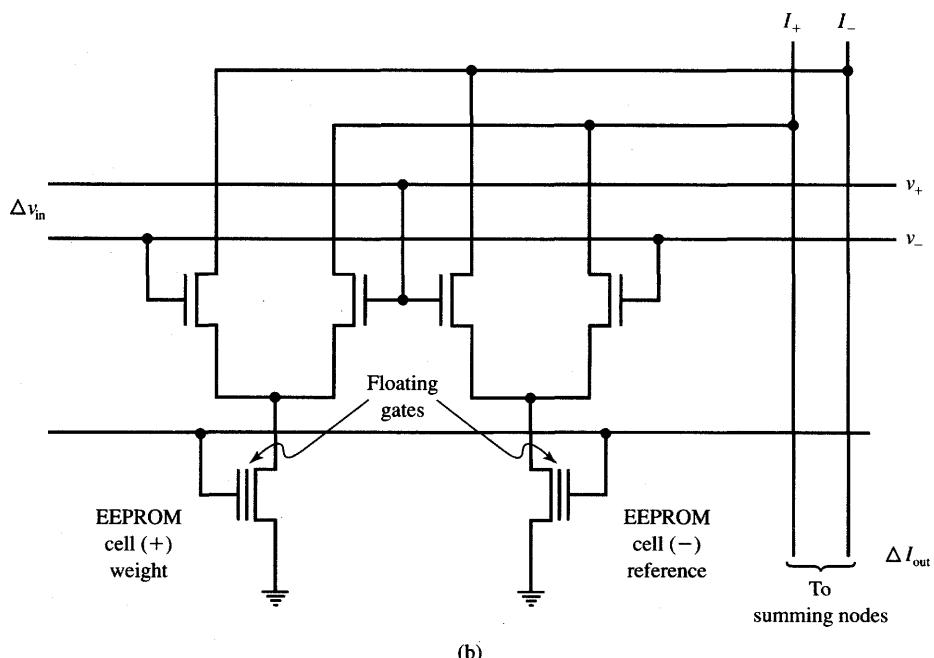


Figure 9.47a ETANN 80170 chip: (a) block diagram. (Courtesy of Intel Corporation)

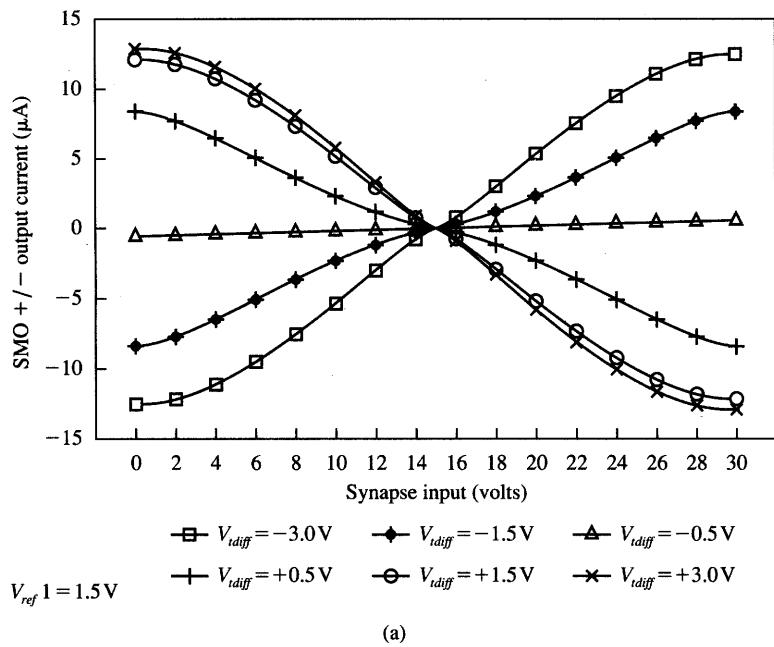


**Figure 9.47b** ETANN 80170 chip (continued): (b) synaptic connection implementation. (Courtesy of Intel Corporation)

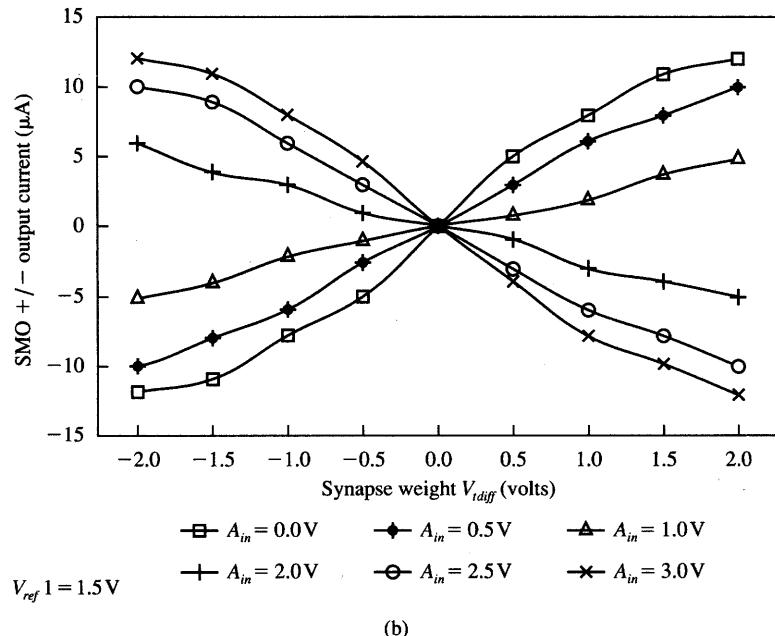
the threshold voltage difference  $V_{thd}$ , called here  $V_{t-Diff}$  as a parameter for each curve shown. The six different values of weights displayed in the figure have been obtained with the threshold voltage value of one of the transistors fixed as a reference,  $V_{ref1}$ . The graphs show that the weights can be set with fine accuracy and a considerable linearity of weights is achieved. Figure 9.48(b) shows the differential input current versus the difference of the threshold voltages  $V_{thd}$  with the input voltage as a parameter for the family of curves. Noticeably, the inputs and outputs of the weight cell are both differential. Weight changes are accomplished by applying high-voltage pulses to the ETANN chip and causing the floating gate to store the weight value as described earlier in Section 9.4.

Currents generated by products of up to 160 weights times the inputs are summed and converted to an output voltage. This is accomplished by the neuron characterized by an activation function as shown in Figure 9.49. The sigmoidal function takes a voltage-controlled shape. The neuron's output voltage corresponds to a "squashed" scalar product of input and weight vectors, and it depends on the value of the control voltage VGAIN.

The ETANN chip can be programmed and trained using a development system that plugs into a personal computer, which serves as a host. The development system contains converters, conditioners, multiplexers, and logic controllers,

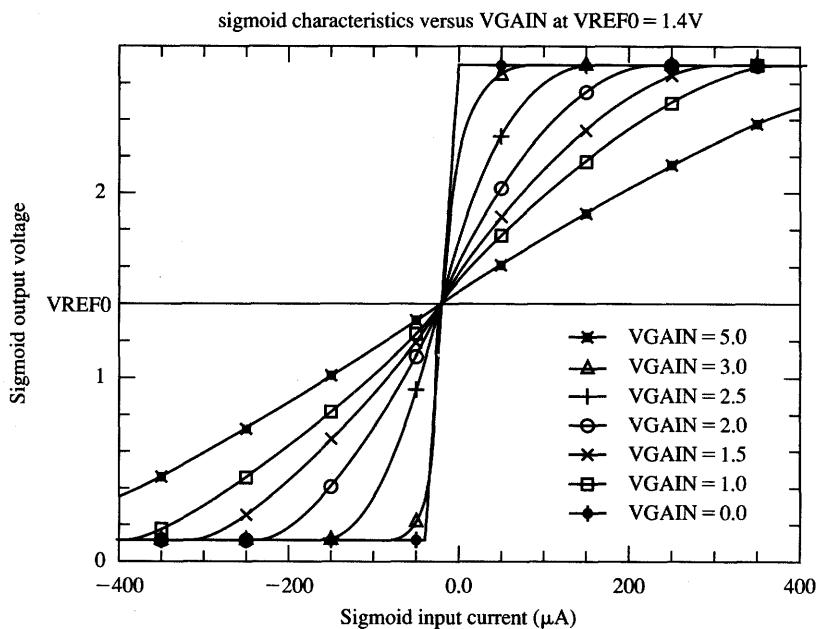


(a)



(b)

**Figure 9.48** Synapse differential output current characteristics: (a) versus differential input voltage,  $V_{tdiff}$  as a parameter and (b) versus  $V_{tdiff}$ , differential input voltage as a parameter. (Courtesy of Intel Corporation)



**Figure 9.49** Neuron activation function versus differential input current, VGAIN voltage as a parameter. (Courtesy of Intel Corporation)

which are needed to program the neural network chip. System software is available to either verify the downloaded weights or read weights out of a chip that has been previously trained in an on-line mode. The ETANN chip can be wire-wrapped into a variety of architectures. Either cascaded or parallel architectures can be developed that employ up to 1024 neurons and up to 81 920 weights in the interconnected network.

The analog neural network processor described by Fisher, Fujimoto, and Smithson (1991) and discussed below represents a somewhat different category of neural hardware. In contrast to the ETANN concept, the processor can be reprogrammed on the fly as if it were read/write memory. The processor's analog throughput does not need to be halted during the weight adaptation or reprogramming. Thus, while ETANN is best suited for ultrafast execution of problems that require rather infrequent retraining, the analog neural network processor can be modified while working. In addition, the ETANN chip uses current-mode signal processing rather than the voltage-mode processing employed in an analog neural network processor.

The processor can be connected so it functions as either a feedforward or feedback network. The 256 neurons are made of operational amplifiers. The processor weights are arranged in an array of custom programmable resistor chips assembled at the board level. The 2048 programmable resistors are physically

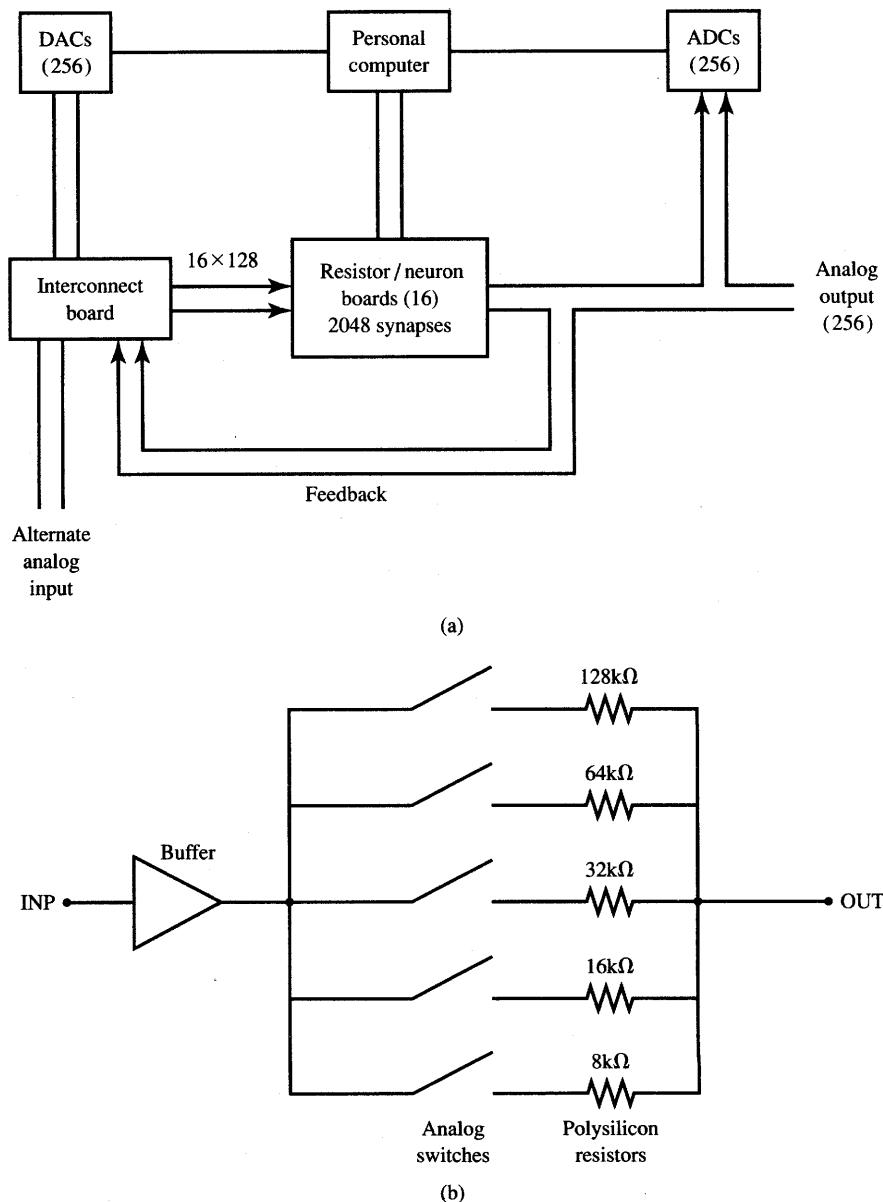
located on 16 boards. The overall architecture of the processor is shown in Figure 9.50(a). Input to the network can be simply analog, or from 12-bit digital sources converting the digital signal to analog form in the block of 256 digital-to-analog (D/A) converters.

The experimentally tested D/A conversion rate for the processor built has been 5 kHz. The processor input data can be provided externally or from the computer. The analog output data can be monitored after a conversion of up to 256 outputs by a set of 256 A/D converters. The interconnection boards used by the processor adjust it to the problem class and are architecture specific, while the rest of the processor is general-purpose. While the system requires more space than an integrated neural network chip, the use of the boards and wire-wrap interconnect boards makes it possible to reconfigure the network easily.

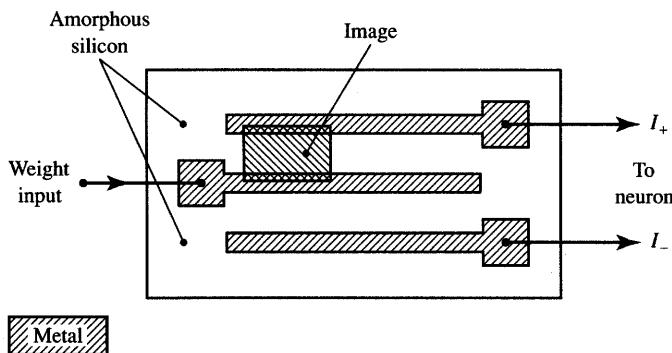
The heart of the analog network of the processor is made up of the digitally programmable analog resistors. A single programmable weight diagram is shown in Figure 9.50(b). The resistors are made on a gate array chip. This weight implementation concept is similar to that described in Section 9.2 for MDACs with binary weighted resistances. This processor, however, uses integrated circuit resistors rather than MOSFETs. Each weight consists of a buffer connected to five polysilicon resistors through analog bipolar switches. The resistors range from 8 to 128 k $\Omega$  and can be set up with an 8- k $\Omega$  step, thus providing 4-bit resolution. This resolution has been found adequate for the neural control applications studied by Fisher, Fujimoto, and Smithson (1991). The buffered design allows for the connection of many resistors to the neuron's input without placing excessive current load on the summing and amplifying neuron. Also, since the weight program data are stored in digital form within computer memory, or registers, no refresh signals are required to store their values.

The programmable neural network processor has been successfully tested for control of adaptive mirrors' actuators. The purpose of adaptive mirrors is to undo the distortions of images of solar or celestial objects. Such images are distorted when the light forming the image passes through the atmosphere. The light reflected from adaptive mirrors can produce corrected images. However, the mirror adaptation cycle must not take longer than 10 ms to keep up with atmospheric changes. The neural processor described has been able to control the mirrors significantly faster than any of the conventional digital control methods attempted.

Another example of an electronic neural network is described by Mueller et al. (1989a, 1989b). The network is designed and implemented in the form of a general-purpose analog neural computer. It consists of three types of interconnected modules. They are dedicated VLSI chips: neurons, modifiable synapses, and routing switches. The neurocomputer runs entirely in the analog mode but the architecture, weights, neuron parameters, and time constants are all set digitally from a host computer.



**Figure 9.50** Programmable analog neural network processor: (a) block diagram and (b) programmable resistor weight schematic. [© IEEE; part (a) reprinted, and part (b) adapted, from Fisher, Fujimoto, and Smithson (1991), with permission.]



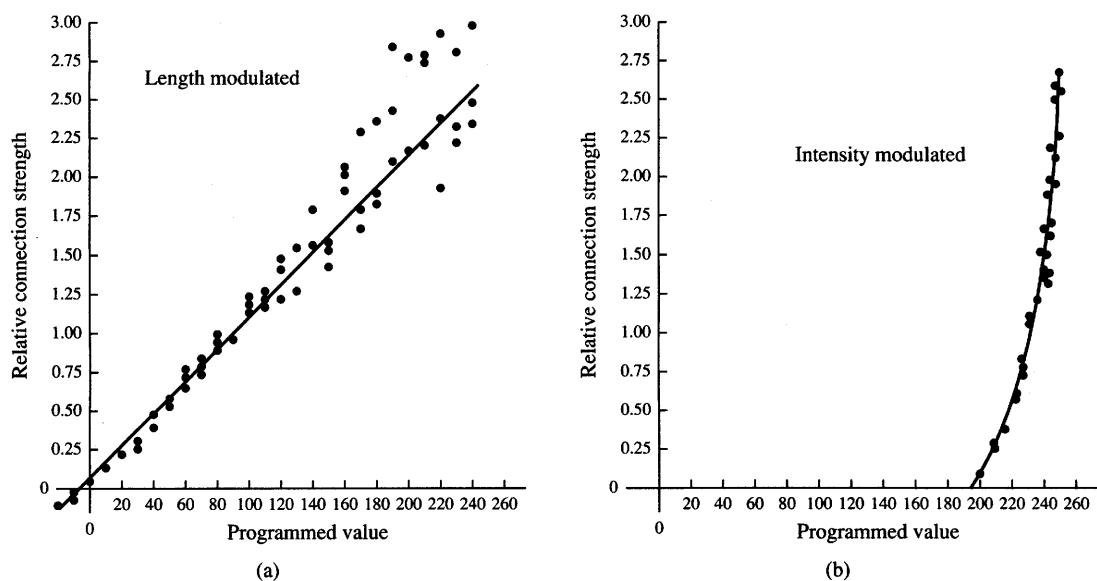
**Figure 9.51** Top view of the photosynapse.

An interesting feature of this neurocomputer is that the analog outputs of the neurons are read through analog multiplexers. Subsequently, these data are digitized and stored in digital memory. During training, the weights and threshold values of the neurons are computed by the host computer and set on the neuron and synaptic chips. The multiplexing operation does not interfere with the actual analog computation.

The synaptic weights are implemented by current mirrors that scale the currents with 5-bit resolution. Since the neurons' outputs are voltages, voltage-to-current converters have been used to provide appropriate current inputs to the current mirrors. The converters are followed by the current dividers consisting of current mirrors in series. To retain the weight value, synaptic cells are provided with local 6-bit weight control memory. Five bits determine the weight magnitude and the sixth bit is used to determine the sign of the weight. The memory consists of shift registers that read the data during the programming phase.

The neuron circuit consists of a rectified summing amplifier, comparator, and an output driver. Inputs to the neurons are currents; outputs are analog voltages. Digital control can be used to adjust the following neuron parameters: bias (threshold), minimum output, and the choice of a linear or sigmoidal transfer function. The sigmoidal transfer function is obtained by adding one or more non-linear devices in parallel with the feedback resistor of the summing operational amplifier. The neuron chip fabricated consisted of 16 neurons implemented in 2- $\mu\text{m}$  CMOS technology. The reader is encouraged to refer to the literature for more detail on this versatile circuit (Mueller et al. 1989a, 1989b).

An interesting example of a neural network with optically controlled hardware has been described by Frye et al. (1989). Synaptic weights can be built using amorphous silicon photoconductors. Such weights, which can be called "photosynapses," react to the amount of light energy projected onto them. A schematic illustration of a photosynapse (top view) is shown in Figure 9.51. The photosynapse is composed of three narrow strips of metal deposited on a layer

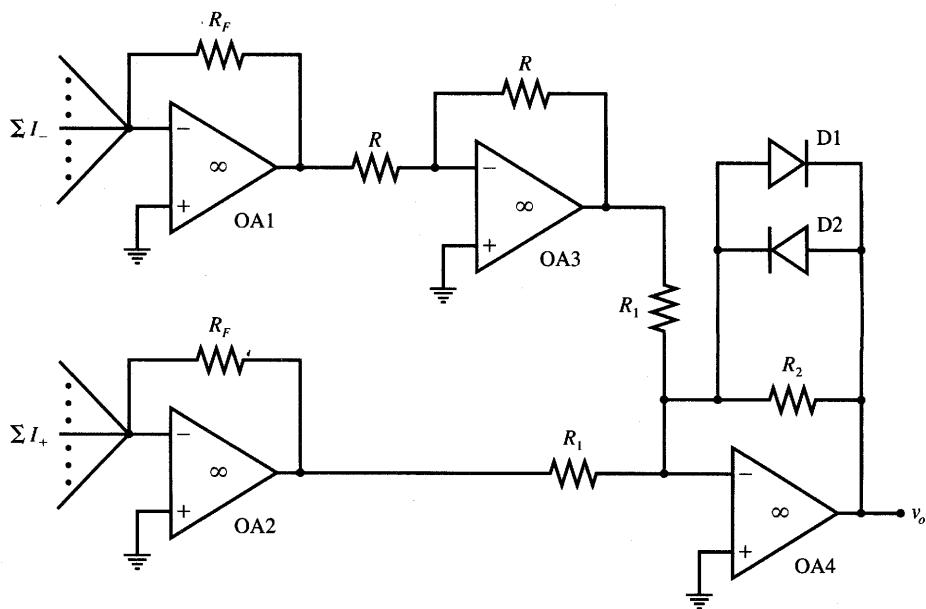


**Figure 9.52** Weight conductance (normalized) versus programmed value for the photosynapse: (a) image length modulation and (b) image intensity modulation, full bar length. [© IEEE; reprinted from Frye et al. (1989); with permission.]

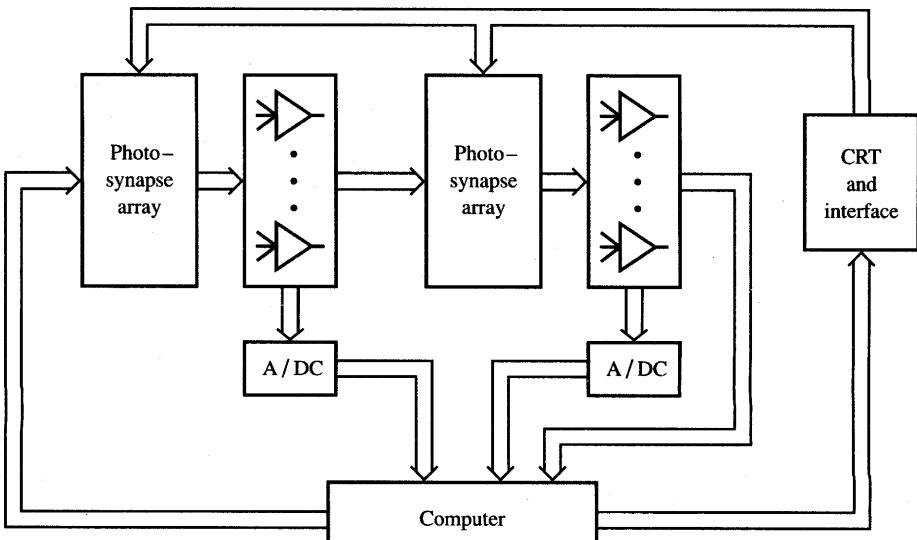
of glow-discharge amorphous silicon. The region between input and either the negative or positive output line forms a photoconductor whose electrical conductance can be controlled by an image projected on the synapse. The weight conductance is a function of the length of the bar of light projected and also of its intensity. In addition, the length of the bar is proportional to the length of the image.

The image projected on the matrix of synaptic connections has been generated by a high-intensity CRT. The length of the image bar spanned 240 pixels. The weight conductance has been coded accordingly into the integer values between 0 and 240. Alternatively, the intensity of light alone has been used to control the conductance encoded as an integer ranging from 0 to 255. This intensity has been projected on the photosynapse when using a full-length bar of 240 pixels.

Measured values for the normalized weight conductance are shown in Figure 9.52. It can be seen that the conductance modulated by the length of the bar at full intensity displays a linear relationship. The behavior of weight as a function of bar intensity, however, shows a very different functional dependence. Because of the nature of the CRT light and of the properties of the photoconducting material employed to make the photosynapse, the intensity-modulated conductance is a strongly nonlinear function of the programmed value, showing only a small value of conductance for programmed intensity values below 200 (Frye et al. 1989).



(a)



(b)

**Figure 9.53** Optically controlled multilayer feedforward network: (a) neuron circuit diagram and (b) block diagram of the system. [Adapted from Frye et al. (1989); © IEEE; with permission.]

As shown in Figure 9.51, the weight current can be either excitatory when  $I_+ > 0$ , or inhibitory when  $I_- > 0$ . A simple neuron circuit performing an aggregation of currents  $\sum I_+$  and  $\sum I_-$  is shown in Figure 9.53(a). The first stage consists of current-to-voltage converters, or transconductance amplifiers built with operational amplifiers  $OA1$  and  $OA2$ . A unity-gain inverter built with  $OA3$  provides the sign change for the voltage but without scaling of its magnitude. An inverting summer mode with  $OA4$  provides for the summation of the voltages of both channels. The two diodes in the feedback loop are provided to give the neuron sigmoidally-shaped response characteristics rather than piecewise linear-shaped characteristics.

The described network with photosynaptic connections has been used in a multilayer feedforward architecture trained in the error back-propagation mode. The block diagram of the overall error back-propagation training system is illustrated in Figure 9.53(b). The external computer is used to generate the training data and to evaluate the error produced by the actual system with the current setting of weights. Then, the new images are produced and sent to the photosynapses that correspond to the newly adjusted weight values. The output of the neural system and of the hidden neurons is monitored in each training step so that the new set of weights can be recomputed and translated into light images. The described network has been successfully trained to locate the target point of a ballistic trajectory. The system has proven very useful despite some disadvantages displayed by the actual hardware.

## 9.7

### CONCLUDING REMARKS

Throughout this book we have seen that a wide variety of problems can be solved by using the neural network computing framework. However, each of the classes of problems that needs to be solved usually requires a different architecture and training approach as well as different data stored in the network in the form of weights. The exposition in this chapter has focused on building working models of neural networks which we called neural processors. Among the numerous approaches to implementation of artificial neural systems we have stressed microelectronic hardware realizations. We have emphasized analog circuits to implement neural systems because they provide high synapse density and high computational speed (Satyanarayana, Tsividis, and Graf 1990). Once the algorithms are developed and the networks trained, such realizations provide an efficient way of solving a variety of technical problems. In most cases, neural analog processors offer improvement over both software simulations and digital circuits emulating analog operation.

Most real-life applications of artificial neural networks appear to typically require at least several hundred neurons. An essential aspect of neural hardware implementation is the mechanism used to represent and manipulate the weights. Although more difficult to accomplish than weight adjustment, learning also appears to be an essential ingredient for most neural network applications. A number of designs to accomplish these features within the neural network hardware rather than on the host computer have been presented in this chapter.

While some approaches use off-line learning to train an artificial neural system, which is both possible and practical, the inherent ability to learn on the chip as more information becomes available to the system is invaluable. However, the development of an artificial neural system chip with on-chip learning seems to be a nontrivial task. Even for limited learning capability, the network design becomes quite complex (Collins and Penz 1990). Development of standard digital application-specific integrated circuit hardware requires high expenditures to design, fabricate, and test the digital circuit chip. Development of neural network hardware seems to be even more costly. This is one of the reasons why the neural network chips currently available often involve technology demonstrations as opposed to the easily usable, perhaps even hand-held circuits for practical applications. The task of making a universal and usable neural network chip still remains a challenging one.

Let us consider what could be called an "ideal" neural network. Such a network should possess the following characteristics:

- Work for any neural processing algorithm
- Contain at least 1000 neurons, which can be interconnected globally
- Have programmable analog weights
- Be able to learn on-chip
- Consist of small-area neurons and interconnections
- Operate at low-power levels
- Be stable, reproducible, and extendable so that larger systems can be built through interconnecting neural network building blocks
- Be affordable.

Let us compare the above requirements with the present capabilities of integrated circuit technology. One of the most complex semiconductor chips currently available (early 1992) is the 1-Mbit DRAM. One can say that a 1-Mbit DRAM is roughly equivalent to a 1000-neuron network, which involves about one million weights. The DRAM network, however, stores binary information in contrast to the expected analog weight storage of the neural hardware. Therefore, just from the standpoint of granularity of data storage, it is much easier to fabricate a 1-Mbit DRAM chip than a 1000 neuron network chip. Additional expectations such as weight tunability, learning capability and required nonlinear processing by the

neuron bring even more complexity to the neural hardware picture (Collins and Penz 1989).

Neural network chips available at present, including those discussed in this chapter, meet somewhat relaxed criteria compared to the above list of desirable features and are of reduced complexity and size. The chips made thus far contain on the order of 100 processing nodes. We can state that the currently available neural hardware, including VLSI chips, still lags somewhat behind the neural networks' theoretical computational capabilities. Therefore, most large-scale neural computation is still done on serial computers, special-purpose accelerator boards, or by theoretical analysis.

Out of necessity, the exposition of neural network implementation in this chapter has been limited to a number of approaches and design techniques. Also, microelectronic VLSI technology has been emphasized for the most part. Other designs and approaches are available and are described in more specialized technical reports. The reader may refer to the literature for further study of the subject.

Below is a partial list of topics and related literature:

- Neurocomputing architectures (Bessiere et al. 1991; Goser et al. 1989; Jutten, Guerin, and Herault 1990; Morton 1991; Ouali, Saucier, and Thrile 1991; Ramacher 1991; Vidal 1988)
- Architecture of neural VLSI chips (Wasaki, Itario, and Nakamura 1990; Schwartz and Samalam 1990; Foo, Anderson, and Takefuji; 1990; Kaul et al. 1990; Mack et al. 1988; Mann 1990; Lambe, Moopenn, and Thakoor 1988)
- Silicon implementation of neural networks (Maher et al. 1989; Murray 1991; Paulos and Hollis 1988; Reed and Geiger 1990; Rossetto et al. 1989; Salam and Choi 1990; Verleysen and Jespers 1989; Wang and Salam 1990)
- Optoelectronic implementations of neural networks (Farhat 1989; Lu et al. 1989; Psaltis et al. 1989; Szu 1991; Wagner and Psaltis 1987).

In addition, a pioneering monograph on neurobiologically inspired neural networks is available (Mead 1989). The book is mainly devoted to microelectronic hardware emulation of a number of the powerful organizing principles found in biological neural systems. Integrated circuit design techniques are employed in the book to implement certain features of biological networks. Examples include imitation of human senses such as hearing, vision, and other perceptions. Most of the circuits developed by Mead (1989) operate in the subthreshold range of transistors. MOS transistors in the subthreshold range have beneficial properties such as extremely low power consumption and the transistor's current versus voltage characteristics is described with exponential relationships rather than quadratic with additional saturation.

As stated before, efficient neural processing hardware is crucially important for the success of neural network technology if we are going to take full advantage of its learning and processing capabilities. There is strong and growing activity in this area. Analog semiconductor technology, charge-coupled devices, and optical implementation are all among the explored technologies. Although semiconductor technology is presently the most widespread, the capabilities of the other technologies have not yet been fully evaluated for efficient implementation of neural network hardware.

## PROBLEMS

Please note that problems highlighted with an asterisk (\*) are typically computationally intensive and the use of programs is advisable to solve them.

*P9.1* Analyze the neuron circuit from Figure P9.1 and compute its weight values. Compute the neuron's response  $f(x)$  for the following inputs knowing that  $f_{\text{sat}+} = -f_{\text{sat}-} = 13 \text{ V}$ :

- (a)  $x = [-1 \ 3 \ 0.5]^t$
- (b)  $x = [0 \ 0.5 \ 1.5]^t$
- (c)  $x = [-1 \ -2 \ -0.5]^t$

(The input vector components are specified in Volts.)

*P9.2* Design the neuron circuit as in Figure 9.5 with weights of values  $-3$ ,  $-4$ , and  $-6$ . Use the feedback resistance value of  $1200 \Omega$ . By writing appropriate inequalities, determine in which segments of the input space,  $x$ , the neuron saturates and assumes the values  $f_{\text{sat}+}$  or  $f_{\text{sat}-}$ . Assume  $f_{\text{sat}+} = -f_{\text{sat}-} = 13 \text{ V}$ .

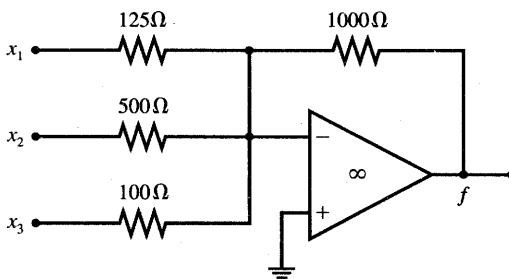
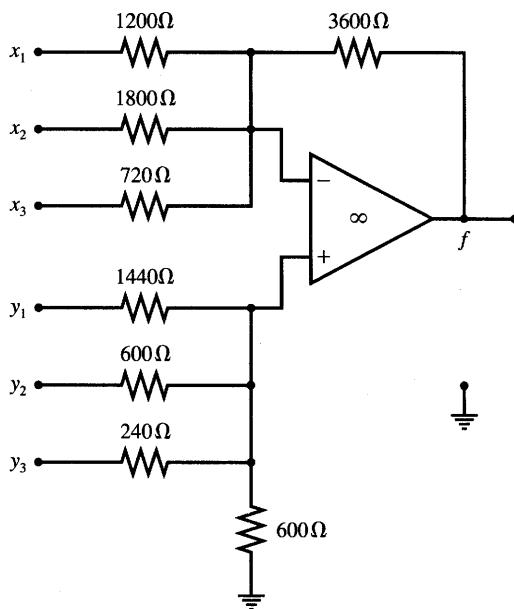


Figure P9.1 Neuron circuit for Problem P9.1.



**Figure P9.3** Resistor-operational amplifier implementation of neuron with weights for Problem P9.3.

**P9.3** The neuron circuit shown in Figure P9.3 implements the following mapping:

$$f(\mathbf{x}, \mathbf{y}) = [w_1 \ w_2 \ w_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + [v_1 \ v_2 \ v_3] \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

- (a) Find weights  $w_1, w_2, w_3$  and  $v_1, v_2, v_3$ .
- (b) Assume that input to this single neuron network is now the vector with entries being voltages

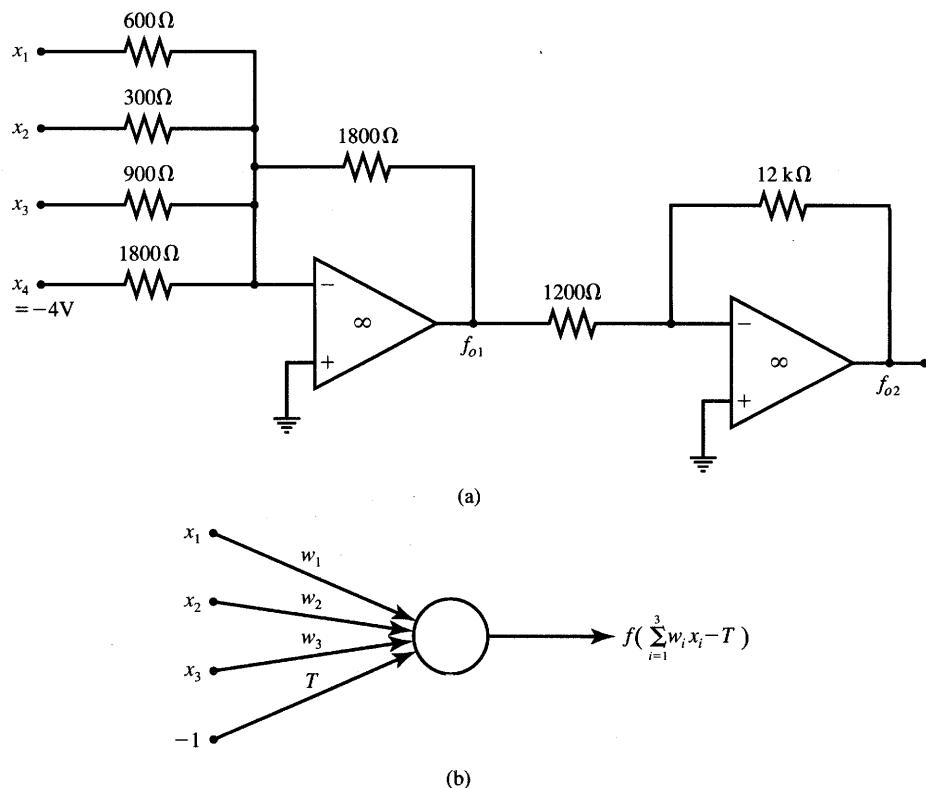
$$\mathbf{a} = [x_1 \ x_2 \ x_3 \ y_1 \ y_2 \ y_3]^t$$

such that (1)  $\|\mathbf{a}\| = 1$  and (2) the input exactly matches the total weight vector defined as

$$\mathbf{w}_{\text{tot}} \stackrel{\Delta}{=} [w_1 \ w_2 \ w_3 \ v_1 \ v_2 \ v_3]^t$$

so that  $\mathbf{w}_{\text{tot}}^t \mathbf{a} = \|\mathbf{w}_{\text{tot}}\|$ . Find the neuron's output voltage  $f(\mathbf{a}, \mathbf{w}_{\text{tot}})$  due to  $\mathbf{a}$ .

**P9.4** The operational amplifier-resistor neuron circuit from Figure P9.4(a) needs to be analyzed. Compute  $f_{o2}$  as a function of  $w_1, w_2$ , and  $w_3$ , and of the bias  $T$  as indicated on the neuron symbol of Figure 9.4(b). Assume for computation purposes that the two neurons are equivalent.



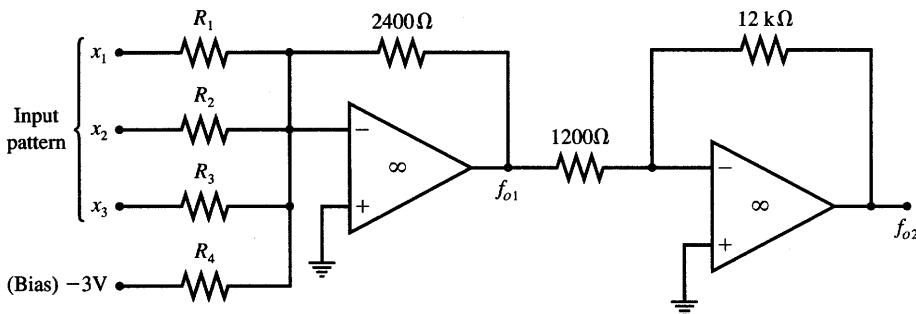
**Figure P9.4** High-gain neuron with weights and nonzero threshold: (a) circuit diagram and (b) symbol.

- (a) Draw the neuron's transfer characteristics as a function of  $z = \sum_{i=1}^3 w_i x_i$ .
- (b) Determine the range of activation  $z$  that allows for the linear operation of the neuron. Assume  $|f_{\text{sat}-}| = |f_{\text{sat}+}| = 14$  V.
- (c) Assume that the neuron needs to classify patterns of two classes and find its responses due to the eight individual input values, which are vertices of a three-dimensional  $[-1, 1]$  cube.

**P9.5** The perceptron shown in Figure P9.5 needs to classify input patterns into two classes. The centers of gravity for each class are known as follows:

$$\mathbf{x}_1 = \begin{bmatrix} 5 \\ 4 \\ 7 \end{bmatrix} : \text{class 1}$$

$$\mathbf{x}_2 = \begin{bmatrix} -1 \\ 0 \\ 4 \end{bmatrix} : \text{class 2}$$

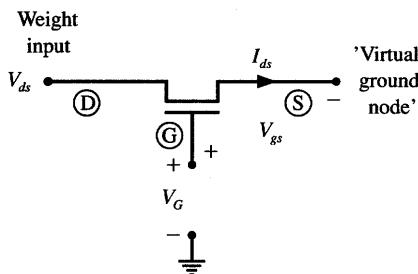


**Figure P9.5** Perceptron for classification task of Problem P9.5.

Design the minimum-distance dichotomizer that would best classify patterns from the two classes. Follow the design steps below:

- Compute the weights and threshold value of the bipolar binary perceptron.
- Find  $R_1$ ,  $R_2$ ,  $R_3$ , and  $R_4$  for the high-gain perceptron from Figure 9.5 that approximates the bipolar binary perceptron.
- Identify the region of linear response of the designed perceptron. This region contains inputs that are falling between the two distinct classes and produce response in the range between  $f_{\text{sat}-}$  and  $f_{\text{sat}+}$ . Assume for computations that  $|f_{\text{sat}-}| = |f_{\text{sat}+}| = 14$  V.

- P9.6** The enhancement-mode NMOS transistor shown in Figure P9.6 is configured to function as a voltage-controlled synaptic weight. Voltage  $V_G = V_{gs}$  is the weight control voltage. It sets the drain-to-source, or channel, resistance  $R_{ds}$  that determines the weight current,  $I_{ds}$ . Knowing that  $k' = 20 \mu\text{A}/\text{V}^2$ ,  $W/L = 10$ , and  $V_{th} = 1.5$  V, evaluate the value of  $R_{ds}$  as follows:
- Compute and sketch  $R_{ds}$  as a function of  $V_{gs}$  for  $1.5 \text{ V} < V_{gs} < 5 \text{ V}$ . Assume that the channel resistance is linear because of linearization



**Figure P9.6** NMOS transistor as electrically controlled weight resistance.

of  $I_{ds}(V_{ds})$  due to the limited input levels, which fulfill condition  $V_{ds} < 0.5(V_{gs} - V_{th})$ .

- (b) Compute and sketch  $R_{ds} = V_{ds}/I_{ds}$  using the following unabbreviated formula for  $I_{ds}$ :

$$I_{ds} = k' \frac{W}{L} \left[ (V_{gs} - V_{th})V_{ds} - \frac{V_{ds}^2}{2} \right]$$

Solve this part of the problem for the condition  $V_{ds} = 0.5(V_{gs} - V_{th})$ . Again,  $V_{gs}$  should be maintained in the range from 1.5 V to 5 V. Notice the increase of the computed resistance value compared to case (a) due to the quadratic term now present in the expression for  $I_{ds}$ .

- P9.7 The scalar product circuit with electrically tunable weights as illustrated in Figure 9.11 uses  $R_F = 10 \text{ k}\Omega$  and NMOS transistors with  $k' = 20 \mu\text{A/V}^2$ ,  $W/L = 10$ , and  $V_{th} = 2 \text{ V}$ . The circuit needs to implement the function

$$v_o = -2.5x_1 - 6x_2 - 4x_3$$

Complete the design of the circuit by computing

- (a) the gate control voltages  $V_{gs1}$ ,  $V_{gs2}$ , and  $V_{gs3}$  producing the specified scalar product  
 (b) the range of input voltages  $x_1$ ,  $x_2$ , and  $x_3$  ensuring the linearity of the channel resistance. Use the linearity condition  $V_{dsi} = V_i < 0.5(V_{gsi} - 2)$ , for  $i = 1, 2, 3$ .

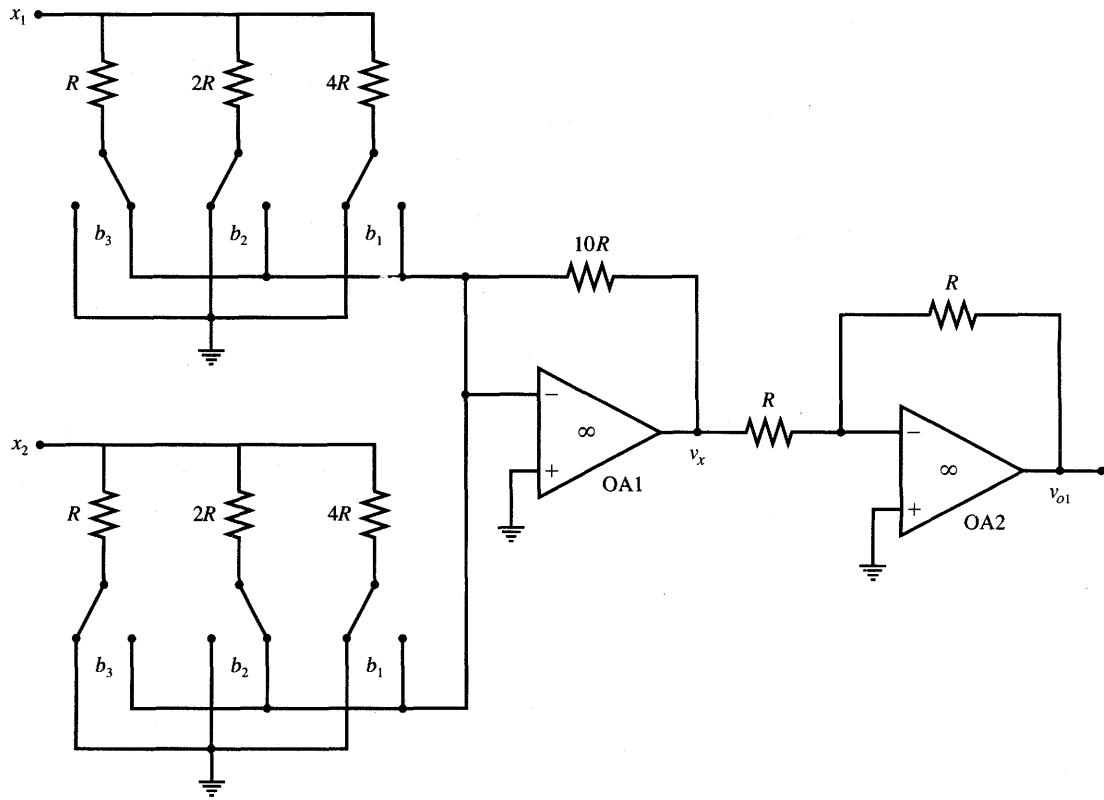
- P9.8 (a) Two 3-bit MDACs are connected in a circuit as shown in Figure P9.8(a). Calculate the output voltage  $v_{o1}$  assuming that  $x_1$  and  $x_2$  are input voltages.  
 (b) Calculate the output voltage  $v_{o2}$  for the input voltages  $y_1$  and  $y_2$  by analyzing the circuit shown in Figure P9.8(b).  
 (c) A difference voltage amplifier is used to combine excitatory voltage  $v_{o1}$  [from part (a)] with inhibitory voltage  $v_{o2}$  [from part (b)] as shown in Figure P9.8(c). Find

$$v_{o3}(\mathbf{w}) = [w_1 \ w_2 \ w_3 \ w_4] [x_1 \ x_2 \ y_1 \ y_2]^t$$

for this neuron circuit when all operational amplifiers operate in their linear regions.

- P9.9 A linear activation function neuron needs to be designed using three weighted-resistor MDACs as shown in Figure 9.15(a). MDACs are controlled by 4-bit words. The neuron's output voltage needs to be

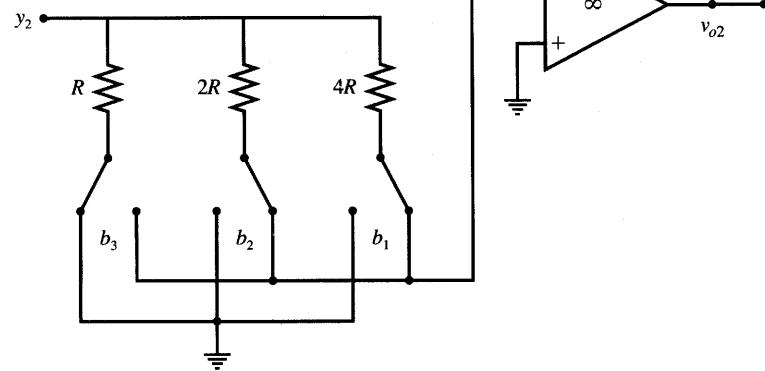
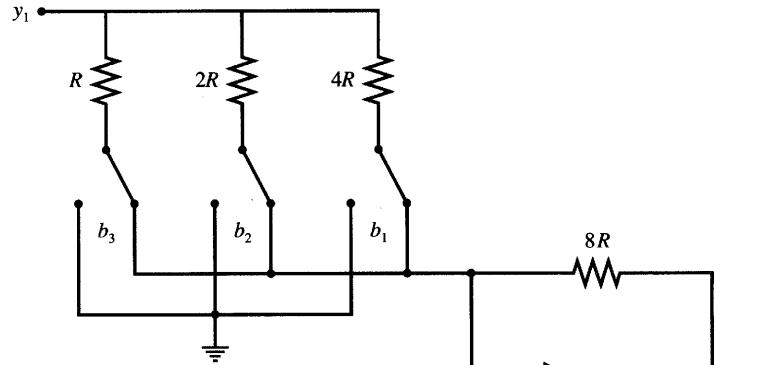
$$v_o = -8v_{i1} - 4.5v_{i2} - 2.5v_{i3}$$



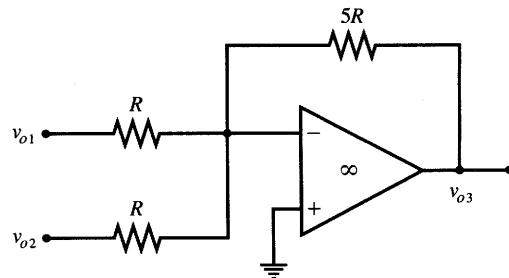
**Figure P9.8a** Neuron circuit using 3-bit MDACs: (a) positive weight circuit.

provided the operational amplifier operates in the linear region. Select  $R_F = 4.5R$ , where  $R$  is the resistance determining the MSB switch current, or  $S_4$  current. Calculate the control word settings  $D_1$ ,  $D_2$ , and  $D_3$  that best approximate the required output. Upon completion of the design, find:

- the percent errors in approximation of the specified weight coefficients  $-8$ ,  $-4.5$ , and  $-2.5$  through control words  $D_1$ ,  $D_2$ , and  $D_3$ , respectively
- the output voltage for  $v_{i1} = v_{i2} = v_{i3} = 100$  mV with control word settings as in part (a)
- the percent error of the output voltage computed in part (b) as compared with the accurate output voltage required.

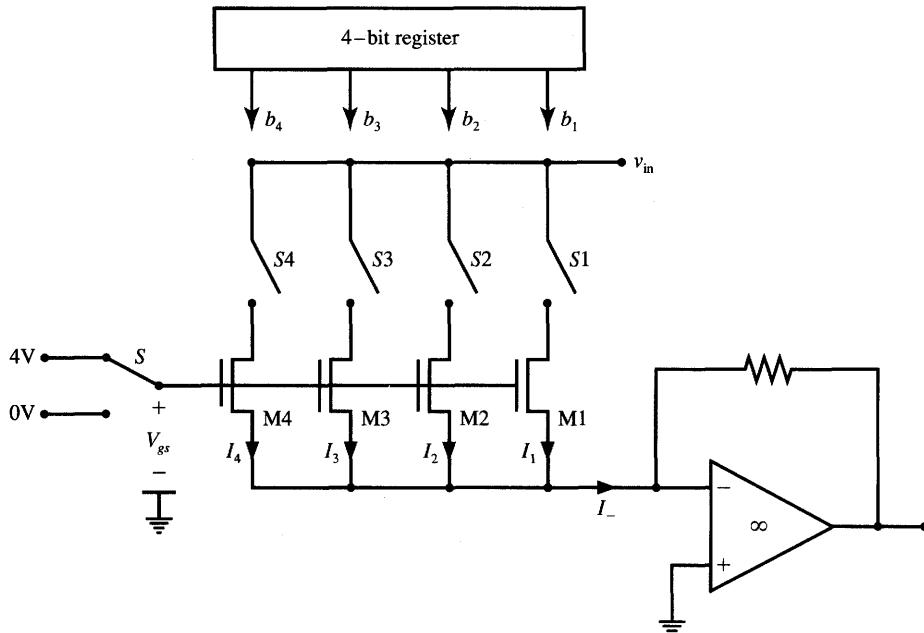


(b)



(c)

**Figure P9.8b,c** Neuron circuit using 3-bit MDACs (continued): (b) negative weight circuit, and (c) neuron circuit.



**Figure P9.10** Integrated circuit programmable weight configuration.

**P9.10** The circuit configuration shown in Figure P9.10 is used to produce a single digitally programmable weight. Weighted currents in vertical branches of the MDAC are obtained by proper sizing of enhancement mode NMOS transistors  $M_4$  through  $M_1$ . Each of the transistors is assumed conductive and behaves as an integrated circuit linear resistance between the drain and source,  $R_{ds}$ , for  $V_{gs} = 4$  V provided  $0 < v_{in} < 1$  V. The threshold voltage of transistors used is  $V_{th} = 1$  V, the process transconductance is  $k' = 20 \mu\text{A/V}^2$ .

Transistor current in the resistive region is approximated by the expression

$$I_{ds} \cong k' \frac{W}{L} (V_{gs} - V_{th}) V_{ds} \quad \text{for } V_{ds} < V_{gs} - V_{th}$$

The following design conditions need to be assumed:

- (1)  $v_{in,\max} = 1$  V
- (2)  $I_{\max} = 1.875 \mu\text{A}$  when  $b_1, b_2, b_3, b_4$  are of unity value and  $S_1, S_2, S_3$ , and  $S_4$  are short-circuited
- (3)  $V_{gs} = 4$  V

Complete the design by computing required value  $W/L$  of transistors  $M1$  through  $M4$  used to produce the weight.

- P9.11 Consider the current mirror as in Figure 9.21 with  $V_{ss} = 0$  V,  $I_{in} = 10 \mu\text{A}$  and assume transistors  $M1$  and  $M2$  to be identical, with  $V_{th} = 1$  V,  $k' = 20 \mu\text{A}/\text{V}^2$ ,  $L = 10 \mu\text{m}$ , and  $W = 40 \mu\text{m}$ . Find  $V_{gs1} = V_{gs2}$  and the lowest allowable output voltage  $v_{out}$  at the present current level  $I = 10 \mu\text{A}$  for proper operation of the current mirror.
- P9.12 The basic differential amplifier as shown in Figure 9.27(a) with differential input and single-ended output and resistive loads  $R_L$  has voltage transfer characteristics as shown in Figure 9.28(a). For this case, compute the voltage gains  $A_{v1}$ , and  $A_{v2}$  defined as

$$A_{v1} = \left( \frac{\partial v_{d1}}{\partial v_{ind}} \right) \Big|_{v_{ind}=0}$$

$$A_{v2} = \left( \frac{\partial v_{d2}}{\partial v_{ind}} \right) \Big|_{v_{ind}=0}$$

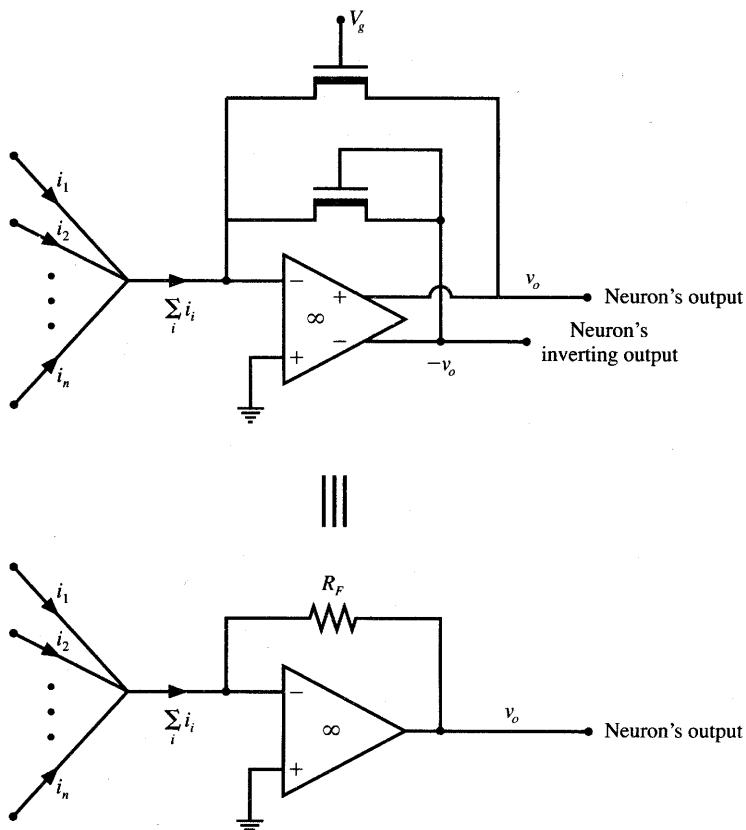
Express computed gains in terms of  $k$ , and  $I_{ss}$ .

- P9.13 Design a template matching network using the concept of one transistor per single bit matching element as illustrated in Figure 9.32(b). The input vector representing the template bit map to be matched is  $\mathbf{x} = [1 \ 0 \ \times \ 1 \ 0 \ 0]^t$ , where  $\times$  denotes a “do not care” bit.
- P9.14 The double-ended operational amplifier shown in Figure P9.14 that performs as a neuron element implements simultaneously summation and current-to-voltage conversion. It uses two matching depletion mode transistors with device transconductance and depletion voltage values  $k$  and  $V_{dep}$ , respectively. Show that both networks of Figure P9.14 are identical if (9.68) holds, or

$$R_F = \frac{1}{kV_g}$$

- P9.15 A simple way to reduce the nonlinearity of a weight consisting of an MOS transistor is to use a properly biased transmission gate as shown in Figure P9.15. Show that if both transmission gate transistors remain in the resistive region (NMOS:  $V_{ds} < V_{gs} - V_{thn}$ , PMOS:  $V_{ds} > V_{gs} - V_{thp}$ ), the resistance of the produced weight is equal

$$R \triangleq \frac{V_1 - V_2}{I} = \frac{1}{2k(V_g - V_{thn})}$$



**Figure P9.14** Integrated circuit feedback resistance for current summation and current-to-voltage conversion for Problem P9.14.

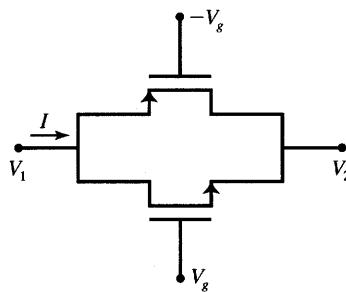
Assume:

$$I_{dsn} = k_n \left[ (V_{gs} - V_{thn})V_{ds} - \frac{V_{ds}^2}{2} \right] \quad \text{for NMOS device}$$

$$I_{dsp} = k_p \left[ (V_{gs} - V_{thp})V_{ds} - \frac{V_{ds}^2}{2} \right] \quad \text{for PMOS device}$$

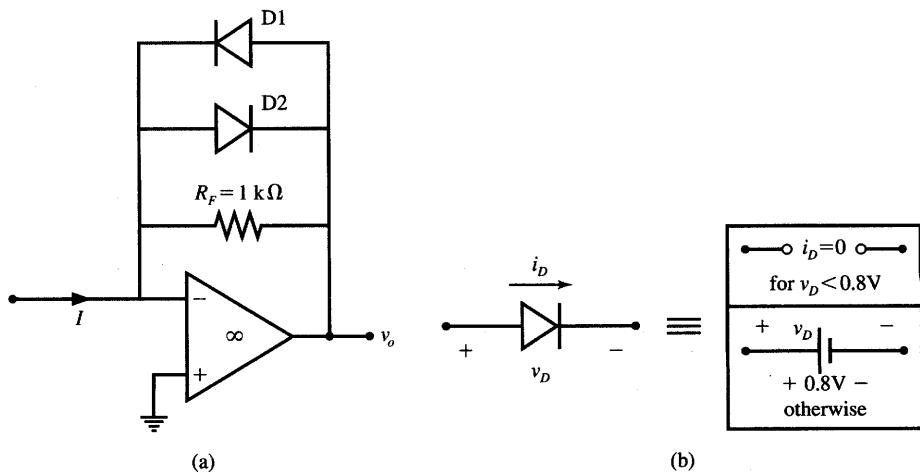
$$k = k_n = k_p, \quad V_{th} = V_{thn} = -V_{thp}$$

(Hint: Assume initially  $V_1 > V_2$  for derivation purposes. Since the weight circuitry is electrically symmetrical, the weight resistances remain identical when  $V_1 < V_2$  is subsequently assumed.)

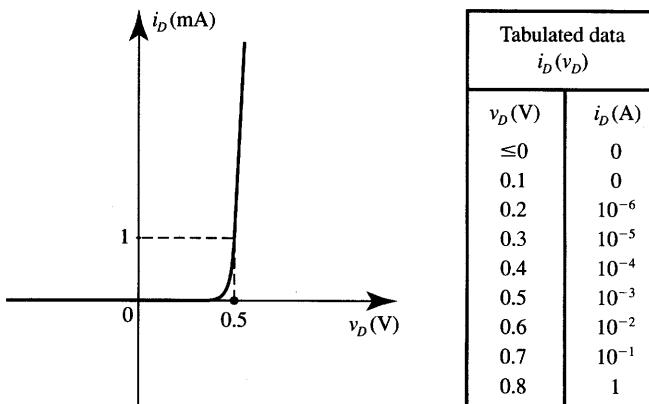


**Figure P9.15** Integrated circuit resistance using a transmission gate.

**P9.16** The output stage of a neuron with the sigmoidal transfer characteristics can be designed with two diodes connected in parallel to the operational amplifier feedback resistor. The diodes cause the approximate saturation voltages of the neuron, which are substantially lower than the saturation voltages produced without the diodes in place. The resulting current-to-voltage converter that needs to be analyzed in this problem is shown in Figure P9.16(a). Assume that  $D1$  and  $D2$  are identical and described by the model approximated as shown in the box of Figure P9.16(b). Provide the answer to the problem in the form of the graphical characteristics of  $v_o$  versus  $I$  for the current-to-voltage converter.



**Figure P9.16** Current-to-voltage converter with nonlinear feedback for saturation level control.



**Figure P9.17** The current versus voltage characteristics of diodes  $D_1$  and  $D_2$  for Problem P9.17. Values tabulated should be used for the piecewise approximation of the curve shown.

**P9.17** Repeat Problem P9.16 after replacing diodes  $D_1$  and  $D_2$  with the diode model characteristics sketched in Figure P9.17. The piecewise linear characteristics of the diodes as provided in the table can be used for obtaining the graphical solution.

[Hint: The parallel connection of  $R_F$ ,  $D_1$ , and  $D_2$  may be treated as a single nonlinear resistor carrying current  $i_{\text{tot}}$ . Current versus voltage characteristics  $i_{\text{tot}}(v_o)$  of this composite nonlinear resistor can be obtained graphically by adding the currents through  $R_F$ ,  $D_1$ , and  $D_2$ . The current-to-voltage conversion curve  $v_o(I)$  can then be found since  $i_{\text{tot}} = -I$ .]

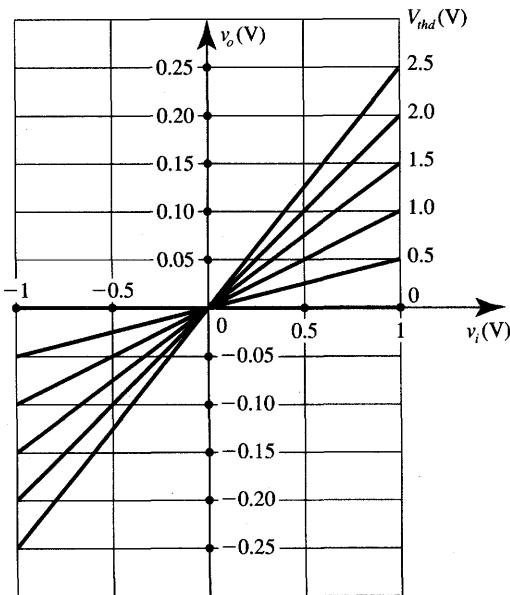
**P9.18** An analog voltage multiplier has the following transfer characteristics:

$$v_o = cv_i V_{\text{thd}}, \quad \text{where } V_{\text{thd}} > 0$$

where  $v_i$ , and  $v_o$  are input and output voltages, respectively,  $V_{\text{thd}}$  is the difference in threshold voltages of the MOSFET devices used, and  $c > 0$  is a constant. Measurements of the multiplier output voltage versus its input voltage have been taken and are illustrated in Figure P9.18. Based on the characteristics shown:

- (a) Compute the multiplication constant  $c$  of this multiplier.
- (b) Draw the characteristics  $v_o$  versus  $V_{\text{thd}}$  with the input voltage as a parameter taking values  $v_i = 0; 0.25; 0.5; 0.75; 1.0$  (V).

**P9.19** Analyze the neuron circuit shown in Figure 9.53(a) and obtain its transfer characteristics  $v_o$  versus the difference of total excitatory and



**Figure P9.18** Characteristics of an analog voltage multiplier for Problem P9.18.

inhibitory currents as follows:

$$v_o = f(\sum I_+ - \sum I_-)$$

Assume that the circuit components have the values  $R_F = 10 \text{ k}\Omega$ ,  $R_1 = 1 \text{ k}\Omega$ , and  $R_2 = 5 \text{ k}\Omega$ . Assume also that all operational amplifiers operate in their linear regions. Solve the problem for two separate cases as follows:

- (a) Diodes  $D1$  and  $D2$  are disconnected.
- (b) Diodes  $D1$  and  $D2$  are connected as shown and modeled by the equivalent circuit as depicted in Figure P9.16(b)

**P9.20** This problem requires the analysis of the template matching network shown in Figure 9.32(b), for which  $n$  denotes the number of bits equal to 1 in the original template vector to be matched. Assume that the bit transistor resistances are  $R$ , and that the bias transistor resistance is chosen to have the value  $R/(n-1)$ . Show that the following properties of the network apply for this choice of component values:

- (a)  $v_{in} = V_{DD}n/(2n-1) > V_{DD}/2$  for a perfect match
- (b)  $v_{in} = V_{DD}/2$  for the mismatch of a single input bit originally “0”
- (c)  $v_{in} = V_{DD}/2$  for the mismatch of a single input bit originally “1”

---

## REFERENCES

- Allen, P. E., and D. R. Holberg. 1987. *CMOS Analog Circuit Design*. New York: Holt, Rinehart and Winston.
- Alspector, J., B. Gupta, and R. B. Allen. 1989. "Performance of a Stochastic Learning Microchip," in *Advances in Neural Information Processing Systems*. San Mateo, Calif.: Morgan Kaufmann Publishers.
- Bessiere, P., A. Chams, A. Guerin, J. Herault, C. Jutten, and J. C. Lawson. 1991. "From Hardware to Software and Designing a Neurostation," in *VLSI Design of Neural Networks*, ed. U. Ramacher, U. Rueckert. Boston: Kluwer Academic Publishers.
- Bibyk, S., and M. Ismail. 1989. "Issues in Analog VLSI and MOS Techniques for Neural Computing," in *Analog Implementation of Neural Systems*, ed. C. Mead, M. Ismail. Boston: Kluwer Academic Publishers.
- Bibyk, S., and M. Ismail. 1990. "Neural Network Building Blocks for Analog MOS VLSI," in *Analogue IC Design: The Current Mode Approach*, ed. C. Toumazou, F. J. Lidgey, D. G. Haigh. London: Peter Peregrinus, Ltd.
- Borgstrom, T. H., M. Ismail, and S. B. Bibyk. 1990. "Programmable Current-Mode Neural Network for Implementation in Analogue MOS VLSI," *IEE Proc. Part G* 137(2): 175-184.
- Card, H. C., C. R. Schneider, and W. R. Moore. 1991. "Hebbian Plasticity in MOS Synapses," *IEE Proc. Part F* 138(1): 13-16.
- Chung, H. W. 1991. "CMOS VLSI Implementation of Neural Networks," in *Proc. '91 Neural Networks and Fuzzy Systems Application Workshop*, Seoul, South Korea, June 7-8, 1991. pp. 209-225.
- Collins, D. R., and P. A. Penz. 1989. "Considerations for Neural Network Hardware Implementations," in *Proc. 1989 IEEE Int. Symp. on Circuits and Systems*, Portland, Oregon, May 9-12, 1989. pp. 834-837.
- Collins, D. R., P. A. Penz, and J. B. Barton, "Neural Network Architectures and Implementations," in *Proc. 1990 IEEE Int. Symp. on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990. pp. 2437-2440.
- DARPA. 1988. *Neural Network Study*. Fairfax, Virginia: AFCEA Int. Press.
- Farhat, N. H. 1989. "Optoelectronic Neural Networks and Learning Machines," *IEEE Circuits and Devices Magazine* (September): 32-41.
- Fisher, W. A., R. J. Fujimoto, and R. C. Smithson. 1991. "A Programmable Analog Neural Network Processor," *IEEE Trans. Neural Networks* 2(2): 222-229.
- Foo, S. Y., L.R. Anderson, and Y. Takefuji. 1990. "Analog Components for the VLSI of Neural Networks," *IEEE Circuits and Devices Magazine* (July): 18-26.

- Frye, R. C., R. A. Rietman, C. C. Wong, B. L. Chin. 1989. "An Investigation of Adaptive Learning Implemented in an Optically Controlled Neural Network," in *Proc. Joint IEEE Int. Conf. on Neural Networks*, San Diego, California, pp. 457-463.
- Geiger, R. L., P. E. Allen, and N. R. Strader. 1990. *VLSI Design Techniques for Analog and Digital Circuits*. New York: McGraw Hill Book Co.
- Goser, K., U. Hilleringmann, U. Rueckert, and K. Schumacher. 1989. "VLSI Technologies for Artificial Neural Networks," *IEEE Micro* (December): 28-43.
- Graf, H. P., and P. D. Vegvar. 1987. "CMOS Associative Memory Chip Based on Neural Networks," in *Proc. 1987 IEEE Int. Solid State Conf.*, February 27, 1987. pp. 304-305, 347.
- Hopfield, J. J. 1990. "The Effectiveness of Analogue 'Neural Network' Hardware," *Network* 1: 27-40.
- Howard, R. E., L. D. Jackel, and H. P. Graf. 1988. "Electronic Neural Networks," *AT&T Technical J.* (May): 58-64.
- Intel Corporation. 1991. *Electrically Trainable Analog Neural Network ETANN Intel 80170NX*.
- Jutten, C., A. Guerin, and J. Herault. 1990. "Simulation Machine and Integrated Implementation of Neural Networks: A Review of Methods, Problems, and Realizations," in *Neural Networks, Proc. of EURASIP Workshop*, Sesimbra, Portugal, February 15-17, 1990, ed. L. B. Almeida, C. J. Wellekens.
- Kaul, R., S. Bibyk, M. Ismail, and M. Andro. 1990. "Adaptive Filtering Using Neural Network Integrated Circuits," in *Proc. 1990 IEEE Symp. on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990. pp. 2520-2523.
- Khachab, N. I., and M. Ismail. 1989a. "A New Continuous-Time MOS Implementation of Feedback Neural Networks," in *Proc. 32nd Midwest Symp. on Circuits and Systems*, Urbana, Illinois, August 1989. pp. 221-224.
- Khachab, N. I., and M. Ismail. 1989b. "MOS Multiplier/Divider Cell for Analogue VLSI," *Electron. Lett.* 25(23): 1550-1552.
- Kub, F. J., K. K. Moon, I. A. Mack, and F. M. Long. 1990. "Programmable Analog Vector-Matrix Multiplier," *IEEE J. of Solid-State Circuits* 25(1): 207-214.
- Lambe, J., A. Moopenn, and A. P. Thakoor. 1988. "Electronic Neural Networks," *IEEE Eng. Medicine and Biology Magazine* (December): 56-57.
- Lee, B. W., and B. J. Sheu. 1991. *Hardware Annealing in Analog VLSI Neurocomputing*. Boston: Kluwer Academic Publishers.
- Lu, T., S. Wu, X. Xu, and F. T. S. Yu. 1989. "Two-Dimensional Programmable Optical Neural Network," *Appl. Opt.* 28(22): 4908-4913.

- Mackie, S., H. P. Graf, D. B. Schwartz, and J. S. Denker. 1988. "Microelectronic Implementations of Connectionist Neural Networks," in *Neural Information Processing Systems*, ed. D. Z. Anderson. New York: American Institute of Physics.
- Maher, M. A. C., S. P. Deweerth, M. A. Mahowald, and C. A. Mead. 1989. "Implementing Neural Architectures Using Analog VLSI Circuits," *IEEE Trans. Circuits and Systems* 36(5): 643-652.
- Mann, J. 1990. "The Effects of Circuit Integration on a Feature Map Vector Quantizer," in *Advances in Neural Information Processing Systems*, ed. D. Touretzky, vol. 2. San Mateo, Calif.: Morgan Kaufmann Publishers.
- Mead, C. 1989. *Analog VLSI and Neural Systems*. Reading, Mass.: Addison-Wesley Publishing Co.
- Millman, J., and A. Grabel. 1987. *Microelectronics*, 2nd Ed. New York: McGraw Hill Book Co.
- Moopenn, A., H. Langenbacher, A. P. Thakoor, and S. K. Khanna. 1988. "Programmable Synaptic Chip for Electronic Neural Networks," in *Advances in Neural Information Processing Systems*, ed. D. Touretzky, vol. 2. San Mateo, Calif.: Morgan Kaufmann Publishers.
- Morton, S. G. 1991. "Electronic Hardware Implementation," in *Handbook of Neural Computing and Applications*. New York: Academic Press.
- Mueller, P., J. V. D. Spiegel, D. Blackman, T. Chiu, T. Clare, C. Donham, T. P. Hsieh, M. Loinaz. 1989a. "Design and Fabrication of VLSI Components for a General Purpose Analog Neural Computer," in *Analog Implementation of Neural Systems*, ed. C. Mead, M. Ismail. Boston: Kluwer Academic Publishers.
- Mueller, P., J. V. D. Spiegel, D. Blackman, T. Chiu, T. Clare, C. Donham, T. P. Hsieh, M. Loinaz. 1989b. "A Programmable Analog Neural Computer and Simulator," in *Advances in Neural Information Processing Systems*, ed. D. Touretzky, vol. 1. San Mateo, Calif.: Morgan Kaufmann Publishers.
- Murray, A. F. 1991. "Silicon Implementation of Neural Networks," *IEE Proc. Part F* 138(1): 3-12.
- Ouali, J., G. Saucier, and J. Thrile. 1991. "Fast Design of Dedicated Neuro-Chips," in *VLSI Design of Neural Networks*, ed. U. Ramacher, U. Rueckert. Boston: Kluwer Academic Publishers.
- Paulos, J. J., and P. W. Hollis. 1988. "Neural Networks Using Analog Multipliers," in *Proc. 1988 IEEE Int. Symp. on Circuits and Systems*, Helsinki, Finland. pp. 494-502.
- Psaltis, D., D. Brady, S. G. Gu, and K. Hsu. 1989. "Optical Implementation of Neural Computers," in *Optical Processing and Computing*. New York: Academic Press.

- Rao, S. 1988. "Design and Analysis of MOSFET Resistors in VLSI," MSEE thesis, University of Louisville, Kentucky.
- Raffel, J. I., J. R. Mann, R. Berger, A. M. Soares, and S. Gilbert. 1989. "A Generic Architecture for Wafer-Scale Neuromorphic Systems," *Lincoln Lab. J.* 2(1): 63-75.
- Ramacher, U. 1991. "Guidelines to VLSI Design of Neural Nets," in *VLSI Design of Neural Networks*, ed. U. Ramacher, U. Rueckert. Boston: Kluwer Academic Publishers.
- Recce, M., and P. C. Treleaven. 1988. "Parallel Architectures for Neural Computers," in *Neural Computers*, ed. R. Eckmiller, Ch. v. d. Malsburg. Berlin: Springer Verlag.
- Reed, R. D., and R. L. Geiger. 1989. "A Multiple-Input OTA Circuit for Neural Networks," *IEEE Trans. Circuits and Systems* 36(5): 767-770.
- Rossetto, O., C. Jutten, J. Herault, and I. Kreutzer. 1989. "Analog VLSI Synaptic Matrices as Building Blocks for Neural Networks," *IEEE Micro* (December): 56-63..
- Salam, F. M. A., and M. R. Choi. 1990. "An All-MOS Analog Feedforward Neural Circuit with Learning," in *Proc. 1990 IEEE Int. Symp. on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990. pp. 2508-2511.
- Satyanarayana, S., Y. Tsividis, and H. P. Graf. 1990. "A Reconfigurable Analog VLSI Neural Network Chip," in *Advances in Neural Information Processing Systems*, ed. D. Touretzky, vol. 2. San Mateo, Calif.: Morgan Kaufmann Publishers.
- Schwartz, D. B., and V. K. Samalam. 1990. "Learning, Function Approximation and Analog VLSI," in *Proc. 1990 IEEE Int. Symp. on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990. pp. 2441-2444.
- Sedra, A. S., and K. C. Smith. 1991. *Microelectronic Circuits*. Orlando, Fla.: Saunders College Publ.
- Szu, H. H. 1991. "Optical Neuro-Computing," in *Handbook of Neural Computing and Applications*. New York: Academic Press.
- Treleaven, P., M. Pacheco, and M. Vellasco. 1989. "VLSI Architectures for Neural Networks," *IEEE Micro* (December): 8-27.
- Tsividis, Y. P., and D. Anastassiou. 1987. "Switched-Capacitor Neural Networks," *Electron. Lett.* 23(18): 958-959.
- Tsividis, Y., and S. Satyanarayana. 1987. "Analogue Circuits for Variable-Synapse Electronic Neural Networks," *Electron. Lett.* 23(24): 1313-1314.
- Verleysen, M., and P. G. A. Jespers. 1989. "An Analog Implementation of Hopfield's Neural Network," *IEEE Micro* (December): 46-55.

- Verleysen, M., B. Sirletti, A. Vandemeulebroeke, and P. G. A. Jespers. 1989. "A High-Storage Capacity Content-Addressable Memory and Its Learning Algorithm," *IEEE Trans. Circuits and Systems* 36(5): 762-766.
- Verleysen, M., and P. Jespers. 1991. "Precision of Computations in Analog Neural Networks," in *VLSI Design of Neural Networks*, ed. U. Ramacher, U. Rueckert. Boston: Kluwer Academic Publishers.
- Vidal, J. J. 1988. "Implementing Neural Nets with Programmable Logic," *IEEE Trans. Acoustics, Speech, and Signal Proc.* 36(7): 1180-1190.
- Vittoz, E., H. Ougney, M. A. Maher, O. Nys, E. Dijkstra, and M. Chevroulet. 1991. "Analog Storage of Adjustable Synaptic Weights," in *VLSI Design of Neural Networks*, ed. U. Ramacher, U. Rueckert. Boston: Kluwer Academic Publishers.
- Wagner, K., and D. Psaltis. 1987. "Multilayer Optical Learning Networks," *Appl. Opt.* 26(23): 5061-5075.
- Wang, Y., and F. M. A. Salam. 1990. "Design of Neural Network Systems from Custom Analog VLSI Chips," in *Proc. 1990 IEEE Int. Symp. on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990. pp. 1098-1101.
- Wasaki, H., Y. Hario, and S. Nakamura. 1990. "A Localized Learning Rule for Analog VLSI Implementation of Neural Networks," in *Proc. 33rd Midwest Symp. Circuits and Systems*, Calgary, Alberta, Canada, August 12-14, 1990.
- Zurada, J. M. 1981. "Application of Multiplying Digital-to-Analogue Convertor to Digital Control of Active Filter Characteristics," *IEE Proc. Part G* 128(2): 91-92.
- Zurada, J. M. 1981. "Programmable State Variable Active Biquads," *J. Audio Eng. Soc.* 29(11): 786-793.
- Zurada, J. M., Y. S. Yoo, and S. V. Bell. 1989. "Dynamic Noise Margins of MOS Logic Gates," in *Proc. 1989 IEEE Int. Symp. on Circuits and Systems*, Portland, Oregon, May 9-11, 1989.



---

# APPENDIX

- A1** Vectors and Matrices
- A2** Quadratic Forms and Definite Matrices
- A3** Time-Varying and Gradient Vectors, Jacobian and Hessian Matrices
- A4** Solution of Optimization Problems
- A5** Stability of Nonlinear Dynamical Systems
- A6** Analytic Geometry in Euclidean Space in Cartesian Coordinates
- A7** Selected Neural Network Related Concepts
- A8** MOSFET Modeling for Neural Network Applications
- A9** Artificial Neural Systems (ANS) Program Listing

This appendix gives a brief introduction to basic mathematical concepts used throughout this text. It also contains the listing of the main procedures of the ANS program. Reading this appendix should give the student a good understanding of most of the definitions, interpretations, and operations. Reviewing the listing of the source code would provide the reader with an understanding of the programming of neural network computing algorithms in the Pascal language.

## A1

---

### VECTORS AND MATRICES

---

#### Definition of Vectors

Consider  $n$  real numbers (scalars)  $x_1, x_2, \dots, x_n$ . These numbers can be arranged so as to define a new object  $\mathbf{x}$ , called a *column vector*:

$$\mathbf{x} \triangleq \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (\text{A1.1})$$

A lowercase boldface letter will always denote a column vector in this text, for example,  $\mathbf{x}$ ,  $\mathbf{y}$ ,  $\mathbf{w}$ . The numbers  $x_1, x_2, \dots, x_n$  are the *components* of the vector. The vector itself can be called an  $n$ -component,  $n$ -tuple, or  $n$ -dimensional vector.

We can associate with the  $\mathbf{x}$  vector its *transpose vector*, denoted by  $\mathbf{x}^t$  and defined as follows:

$$\mathbf{x}^t \triangleq [x_1 \ x_2 \ \dots \ x_n] \quad (\text{A1.2})$$

where superscript  $t$  denotes the transposition. Note that  $\mathbf{x}^t$  is a row vector.

### Operations on Vectors

Consider the column vectors

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \mathbf{z} \triangleq \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix} \quad (\text{A1.3})$$

We define the *sum of the column vectors*  $\mathbf{x}$  and  $\mathbf{y}$ ,  $\mathbf{x} + \mathbf{y}$ , to be the column vector  $\mathbf{z}$  as in (A1.3). Note that adding two  $n$ -component vectors requires adding their respective components.

The *product of a column vector  $\mathbf{x}$  by a scalar  $c$*  is defined as a vector whose components are products  $cx_i$  such that

$$c\mathbf{x} \triangleq c \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} cx_1 \\ cx_2 \\ \vdots \\ cx_n \end{bmatrix} \quad (\text{A1.4})$$

### Linear Dependence of Vectors

Consider a set of  $n$ -dimensional vectors  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$  used to construct a vector  $\mathbf{x}$  such that

$$\mathbf{x} = c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \dots + c_m\mathbf{x}_m = \sum_{i=1}^m c_i\mathbf{x}_i \quad (\text{A1.5})$$

where  $c_1, c_2, \dots, c_m$  are scalars. The vector  $\mathbf{x}$  is a *linear combination* of vectors  $\mathbf{x}_i$ , for  $i = 1, 2, \dots, m$ . A set of vectors  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$  is said to be *linearly dependent* if numbers  $c_1, c_2, \dots, c_m$ , exist and are not all zero, such that

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \dots + c_m\mathbf{x}_m = \mathbf{0} \quad (\text{A1.6})$$

If vectors  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$  are not linearly dependent, we say that they are *linearly independent*.

**EXAMPLE A1.1**

Consider three vectors

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 0 \\ -2 \\ 1 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 2 \\ 4 \\ -3 \end{bmatrix}$$

Since  $2\mathbf{x}_1 + (-3)\mathbf{x}_2 + (-1)\mathbf{x}_3 = \mathbf{0}$ , the vectors are linearly dependent.

Consider three vectors

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

No constants,  $c_i$ ,  $i = 1, 2, 3$ , which are not all zero, can be found such that  $c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + c_3\mathbf{x}_3 = \mathbf{0}$ . Therefore, vectors  $\mathbf{x}_1$ ,  $\mathbf{x}_2$ , and  $\mathbf{x}_3$  are considered to be linearly independent. ■

**Definition of Matrices**

A rectangular array  $\mathbf{A}$  as in (A1.7) of  $n \times m$  numbers  $a_{ij}$ , where  $i = 1, 2, \dots, n$ , and  $j = 1, 2, \dots, m$ , arranged in  $n$  rows and  $m$  columns is called an  $n \times m$  matrix

$$\mathbf{A} \triangleq \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \quad (\text{A1.7})$$

We denote matrices by uppercase boldface letters, for example,  $\mathbf{A}$ ,  $\mathbf{W}$ ,  $\mathbf{V}$ .

We can associate with the  $\mathbf{A}$  matrix its *transpose matrix*, denoted by  $\mathbf{A}'$  and defined as follows:

$$\mathbf{A}' \triangleq \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1m} & a_{2m} & \cdots & a_{nm} \end{bmatrix} \quad (\text{A1.8})$$

Note that  $\mathbf{A}'$  is formed from the matrix  $\mathbf{A}$  by interchanging its rows with columns. Therefore, the first column of  $\mathbf{A}$  becomes the first row of  $\mathbf{A}'$ , the second column of  $\mathbf{A}$  becomes the second row of  $\mathbf{A}'$ , and so on. Thus,  $\mathbf{A}'$  is an  $m \times n$  matrix.

## Operations on Matrices

Consider two  $n \times m$  matrices,  $\mathbf{A}$  and  $\mathbf{B}$ , with their respective elements  $a_{ij}$  and  $b_{ij}$ , for  $i = 1, 2, \dots, n$ , and  $j = 1, 2, \dots, m$ . The *sum of matrices*  $\mathbf{A}$  and  $\mathbf{B}$  is the  $n \times m$  matrix whose elements are the sums of the corresponding elements:

$$\begin{aligned} \mathbf{A} + \mathbf{B} &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{bmatrix} \\ &= \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1m} + b_{1m} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2m} + b_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} + b_{n1} & a_{n2} + b_{n2} & \cdots & a_{nm} + b_{nm} \end{bmatrix} \end{aligned} \quad (\text{A1.9})$$

The *product of an  $n \times m$  matrix  $\mathbf{A}$  by a scalar  $c$*  is defined as an  $n \times m$  matrix whose elements are products  $ca_{ij}$  such that

$$c\mathbf{A} \triangleq c \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} = \begin{bmatrix} ca_{11} & ca_{12} & \cdots & ca_{1m} \\ ca_{21} & ca_{22} & \cdots & ca_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ ca_{n1} & ca_{n2} & \cdots & ca_{nm} \end{bmatrix} \quad (\text{A1.10})$$

The *product  $\mathbf{AB}$  of an  $n \times m$  matrix  $\mathbf{A}$  and of an  $m \times q$  matrix  $\mathbf{B}$*  is the  $n \times q$  matrix  $\mathbf{C}$  such that

$$c_{ik} = \sum_{j=1}^m a_{ij}b_{jk} \quad (\text{A1.11})$$

for  $i = 1, 2, \dots, n$ , and  $k = 1, 2, \dots, q$ . In matrix notation, the product is written

$$\mathbf{C} = \mathbf{AB} \quad (\text{A1.12})$$

Formula (A1.11) indicates that the element  $c_{ik}$  at the intersection of the  $i$ 'th row and  $k$ 'th column of the product matrix is computed by multiplying the  $i$ 'th row of  $\mathbf{A}$  by the  $k$ 'th column of  $\mathbf{B}$ . Note that for the product  $\mathbf{AB}$  to make sense, the number of columns of  $\mathbf{A}$  must be equal to the number of rows of  $\mathbf{B}$  [both are  $m$  in expression (A1.11)]. The reader should notice that matrix multiplication is not commutative.

**EXAMPLE A1.2**

Let us illustrate the use of (A1.11) by multiplying two matrices as shown below:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Matrix  $\mathbf{A}$  is  $3 \times 2$ , matrix  $\mathbf{B}$  is  $2 \times 2$ , thus the product matrix is  $3 \times 2$ :

$$\mathbf{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$

Note that, for example, element  $c_{12}$  of  $\mathbf{C}$  computed from (A1.11) is

$$c_{12} = \sum_{j=1}^2 a_{1j}b_{j2} = a_{11}b_{12} + a_{12}b_{22}$$

and the element  $c_{21}$  computed from (A1.11) is

$$c_{21} = \sum_{j=1}^2 a_{2j}b_{j1} = a_{21}b_{11} + a_{22}b_{21}$$

**Multiplication of Matrices and Vectors**

An  $m$ -tuple vector  $\mathbf{x}$  defined in (A1.1) can be interpreted as a matrix with  $m$  rows and one column. Let us consider the problem of *postmultiplying an  $n \times m$  A matrix with a vector  $\mathbf{x}$* , which is equivalent to an  $m \times 1$  matrix. If we let

$$\mathbf{y} \triangleq \mathbf{Ax} \tag{A1.13a}$$

or, in an expanded form

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \tag{A1.13b}$$

then we can compute components  $y_i$  of the vector  $\mathbf{y}$  using (A1.11) as follows:

$$y_i = \sum_{j=1}^m a_{ij}x_j, \quad \text{for } i = 1, 2, \dots, n \tag{A1.14}$$

Note that the vector  $\mathbf{y}$  resulting as a product is  $n$ -dimensional.

**EXAMPLE A1.3**

Let us illustrate the multiplication of a  $3 \times 2$  matrix  $\mathbf{A}$  by a two-component vector  $\mathbf{x}$ . In the general case, we have

$$\mathbf{y} = \mathbf{Ax} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \\ a_{31}x_1 + a_{32}x_2 \end{bmatrix}$$

**Symmetric Matrices and Their Properties**

Let us suppose that  $\mathbf{A}$  is an  $n \times n$  matrix. Such a matrix is called a *square matrix* since it has the same number of rows and columns. Let us note that a square matrix can be *symmetric*. This property applies when

$$\mathbf{A}' = \mathbf{A} \quad (\text{A1.15})$$

A square matrix  $\mathbf{A}$  whose elements not on the main diagonal are zero is called a *diagonal matrix*. The general form of a diagonal matrix is

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \quad (\text{A1.16})$$

The *unity* or *identity* matrix denoted as  $\mathbf{I}$  is a special case of the diagonal matrix in (A1.16) such that  $a_{ii} = 1$ , for  $i = 1, 2, \dots, n$ . We thus have

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \quad (\text{A1.17})$$

The *determinant* of a square matrix is a scalar and is denoted by

$$\det \mathbf{A} = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \quad (\text{A1.18})$$

(It is assumed that the reader is familiar with the computation of determinants.)

---

### Norms, Products, and Orthogonality of Vectors

The *Euclidean norm*, or *length*, of an  $n$ -tuple vector  $\mathbf{x}$  is denoted as  $\|\mathbf{x}\|$  and defined

$$\|\mathbf{x}\| \stackrel{\Delta}{=} (\mathbf{x}^t \mathbf{x})^{1/2} \quad (\text{A1.19})$$

Note that the Euclidean norm as in (A1.19) can be computed by using the concept of matrix multiplication. Indeed, for an  $n$ -component vector  $\mathbf{x}$  we have from (A1.19)

$$\|\mathbf{x}\| = \left( [x_1 \ x_2 \ \dots \ x_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \right)^{1/2} = \sqrt{\sum_{i=1}^n x_i^2} \quad (\text{A1.20})$$

This result is in agreement with the principles of matrix multiplication rules (A1.11) and (A1.12) for multiplication of a  $1 \times n$  matrix by an  $n \times 1$  matrix.

The *scalar (inner, dot) product* of two  $n$ -component vectors  $\mathbf{x}$  and  $\mathbf{y}$  is the scalar (number) defined as

$$\mathbf{x}^t \mathbf{y} = x_1 y_1 + x_2 y_2 + \dots + x_n y_n = \sum_{i=1}^n x_i y_i \quad (\text{A1.21})$$

Note that both vectors  $\mathbf{x}$  and  $\mathbf{y}$  are column vectors and that their scalar product is computed as a sum of  $n$  products of corresponding components.

An important property as below holds for the scalar product that links the Euclidean norms of vectors  $\mathbf{x}$  and  $\mathbf{y}$  and the angle  $\psi$  between them:

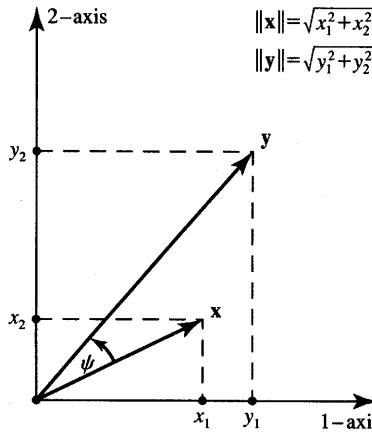
$$\mathbf{x}^t \mathbf{y} = \|\mathbf{x}\| \cdot \|\mathbf{y}\| \cos \psi \quad (\text{A1.22})$$

This property is illustrated in Figure A1 for the two-dimensional case of vectors  $\mathbf{x}$  and  $\mathbf{y}$ , where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad (\text{A1.23})$$

The reader should note that the scalar product operation on two vectors is commutative. We thus have  $\mathbf{x}^t \mathbf{y} = \mathbf{y}^t \mathbf{x}$ . In the text the scalar product of input and weight vectors is often denoted by *net*, which is the argument of the activation function.

Vectors  $\mathbf{x}$  and  $\mathbf{y}$  are said to be *orthogonal* if and only if their scalar product is zero. There is a simple connection between linear independence and orthogonality. If the nonzero vectors  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$  are mutually orthogonal (every vector is orthogonal to each other), then they are linearly independent. The three



**Figure A1** Illustration for the scalar product of two vectors,  $\mathbf{x}$  and  $\mathbf{y}$ .

vectors of Example A1.1

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

are examples of three mutually orthogonal vectors. They also are linearly independent.

#### EXAMPLE A1.4

Let us find the scalar product of two vectors  $\mathbf{x}$  and  $\mathbf{y}$ , their lengths, and the angle between them. The vectors are

$$\mathbf{x} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 1 \\ -1 \\ 4 \end{bmatrix}$$

The definition (A1.21) yields the following scalar product value

$$\mathbf{x}'\mathbf{y} = 2 \cdot 1 + 1 \cdot (-1) + 2 \cdot 4 = 9$$

The angle between the vectors can be found from expression (A1.22) as follows

$$\cos \psi = \frac{\mathbf{x}'\mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} \quad (\text{A1.24})$$

Since the lengths of the vectors  $\mathbf{x}$  and  $\mathbf{y}$  computed from (A1.20) result as

$$\|\mathbf{x}\| = \sqrt{2^2 + 1 + 2^2} = 3$$

$$\|\mathbf{y}\| = \sqrt{1 + 1 + 4^2} = \sqrt{18}$$

we obtain from (A1.24)

$$\cos \psi = \frac{9}{3\sqrt{18}} = \frac{1}{\sqrt{2}}$$

We thus conclude that  $\psi = 45^\circ$ . ■

The *outer (vector, cross) product* of two  $n$ -component vectors  $\mathbf{x}$  and  $\mathbf{y}$  is defined as

$$\mathbf{x} \times \mathbf{y} \stackrel{\Delta}{=} \mathbf{x}\mathbf{y}^t \quad (\text{A1.25})$$

Note that the outer product is an  $n \times n$  matrix as shown below:

$$\mathbf{x} \times \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & \dots & x_1y_n \\ x_2y_1 & x_2y_2 & \dots & x_2y_n \\ \vdots & \vdots & \dots & \vdots \\ x_ny_1 & x_ny_2 & \dots & x_ny_n \end{bmatrix} \quad (\text{A1.26})$$

Note that the outer product of two identical  $n$ -tuple vectors results in symmetrical  $n \times n$  matrix.

### Linear Systems, Inverse and Pseudoinverse Matrices

Consider a linear system of  $n$  equations with  $m$  unknowns  $x_1, x_2, \dots, x_m$ . The general form of the equation system is as in (A1.13)

$$\mathbf{y} = \mathbf{Ax} \quad (\text{A1.27})$$

In a case where  $m = n$  and  $\det \mathbf{A} \neq 0$ , the exact solution of (A1.27) is

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} \quad (\text{A1.28})$$

where  $\mathbf{A}^{-1}$  is the inverse matrix. In a case where  $m < n$ , equation (A1.27) does not have an accurate solution for  $\mathbf{x}$ . We may therefore be interested in obtaining the approximate solution,  $\bar{\mathbf{x}}$ , such that it minimizes the error  $\|\mathbf{Ax} - \mathbf{y}\|$ . This solution is

$$\bar{\mathbf{x}} = \mathbf{A}^+ \mathbf{y} \quad (\text{A1.29})$$

In the case when  $m > n$ , equation (A1.27) has an infinite number of solutions. Again, we may be interested in the solution,  $\bar{\mathbf{x}}$ , which has the smallest norm, i.e.,  $\|\bar{\mathbf{x}}\| \leq \|\mathbf{x}\|$  for each  $\mathbf{x}$  such that  $\mathbf{Ax} = \mathbf{y}$ . This solution is

$$\bar{\mathbf{x}} = \mathbf{A}^+ \mathbf{y} \quad (\text{A1.30})$$

In both cases (A1.29) and (A1.30) matrix  $\mathbf{A}^+$  is called the *pseudoinverse* of matrix  $\mathbf{A}$ .

When  $m < n$  we have the pseudoinverse matrix of value

$$\mathbf{A}^+ = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}' \quad (\text{A1.31})$$

When  $m > n$  we have the pseudoinverse matrix of value

$$\mathbf{A}^+ = \mathbf{A}'(\mathbf{A}\mathbf{A}')^{-1} \quad (\text{A1.32})$$

## A2

### QUADRATIC FORMS AND DEFINITE MATRICES

#### Definition of Quadratic Forms

Quadratic form  $E(\mathbf{x})$  of a vectorial variable,  $\mathbf{x}$ , is of fundamental importance for studying multi-input/multi-output physical systems. The *quadratic form* is a scalar-valued function of  $\mathbf{x}$  defined through a symmetric  $n \times n$  matrix  $\mathbf{A}$  as defined in expression (A1.7). The quadratic form is equal

$$E(\mathbf{x}) = \mathbf{x}'\mathbf{A}\mathbf{x} \quad (\text{A2.1})$$

which is equivalent to

$$E(\mathbf{x}) = [x_1 \ x_2 \ \dots \ x_n] \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (\text{A2.2})$$

Assuming the symmetric matrix  $\mathbf{A}$  and performing the multiplication from left to right of matrices  $1 \times n$  by  $n \times n$  by  $n \times 1$  yields the scalar function  $1 \times 1$ , which is the polynomial in the form

$$\begin{aligned} E(x_1, x_2, \dots, x_n) = & a_{11}x_1^2 + 2a_{12}x_1x_2 + 2a_{13}x_1x_3 + \dots + 2a_{1n}x_1x_n \\ & + a_{22}x_2^2 + 2a_{23}x_2x_3 + \dots + 2a_{2n}x_2x_n + \dots + a_{nn}x_n^2 \end{aligned} \quad (\text{A2.3})$$

Note that the assumption about the symmetry of matrix  $\mathbf{A}$  causes all terms without the squared variables  $x_i$ , for  $i = 1, 2, \dots, n$ , to be doubled in (A2.3) since  $a_{ij} = a_{ji}$ . If the matrix  $\mathbf{A}$  were not symmetric, we would have terms  $(a_{ij} + a_{ji})x_i x_j$  for all terms of (A2.3) without the squared variable. In short, the quadratic form of (A2.2) or (A2.3) can be rewritten as

$$E(\mathbf{x}) = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j \quad (\text{A2.4})$$

**EXAMPLE A2.1**

Let us compute the quadratic form  $E(\mathbf{x})$  defined as in (A2.1), where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

Using either of the formulas (A2.2), (A2.3), or (A2.4) yields the quadratic form as

$$E(\mathbf{x}) = x_1^2 + 4x_1x_2 + 3x_2^2$$

**Properties of Quadratic Forms**

Let us define the most important properties of quadratic forms related to their value in terms of the vectorial variable,  $\mathbf{x}$ . If for all nonzero  $\mathbf{x}$  ( $\mathbf{x} \neq \mathbf{0}$ )  $E(\mathbf{x}) = \mathbf{x}'\mathbf{A}\mathbf{x}$  is

- (a) non-negative, then  $E(\mathbf{x})$  is called a *positive semidefinite form* and  $\mathbf{A}$  is called a positive semidefinite matrix.
- (b) positive, then  $E(\mathbf{x})$  is called a *positive definite form* and  $\mathbf{A}$  is called a positive definite matrix.
- (c) non-positive, then  $E(\mathbf{x})$  is called a *negative semidefinite form* and  $\mathbf{A}$  is called a negative semidefinite matrix.
- (d) negative, then  $E(\mathbf{x})$  is called a *negative definite form* and  $\mathbf{A}$  is called a negative definite matrix.

Below we outline the procedure for the testing of quadratic forms for conditions (a) through (d). We begin with case (b).

The form  $E(\mathbf{x})$  is said to be positive definite if and only if, for the symmetric matrix  $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{bmatrix} \quad (\text{A2.5})$$

its principal determinants  $\det A_{11}$ ,  $\det A_{22}$ , ...,  $\det A_{nn}$ , defined as below fulfill the conditions

$$\begin{aligned} \det A_{11} &\stackrel{\Delta}{=} |a_{11}| > 0 \\ \det A_{22} &\stackrel{\Delta}{=} \begin{vmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{vmatrix} > 0 \\ &\vdots \\ \det A_{nn} &\stackrel{\Delta}{=} \det \mathbf{A} > 0 \end{aligned} \quad (\text{A2.6})$$

The test conditions are tabulated below for all relevant cases. Similar to (A2.6), the test involves evaluation of the signs of the principal determinants of matrix  $\mathbf{A}$ .

Case	Class	Test
(a)	Positive semidefinite	$\det A_{11} \geq 0, \det A_{22} \geq 0, \dots, \det A_{nn} \geq 0$
(b)	Positive definite	$\det A_{11} > 0, \det A_{22} > 0, \dots, \det A_{nn} > 0$
(c)	Negative semidefinite	$\det A_{11} \leq 0, \det A_{22} \geq 0 \dots$ etc. (note alternating signs) <span style="float: right;">(A2.7)</span>
(d)	Negative definite	$\det A_{11} < 0, \det A_{22} > 0, \dots$ , etc. (note alternating signs)
(e)	Indefinite	none of the above.

Note that the quadratic form  $E(\mathbf{x})$  is said to be *indefinite* if it is positive for some  $\mathbf{x}$  and negative for other  $\mathbf{x}$ . This points out that the sign of an indefinite quadratic form depends on  $\mathbf{x}$ .

### EXAMPLE A2.2

Let us look at an example of a quadratic form with a symmetric  $2 \times 2$  matrix  $\mathbf{A}$  such that its diagonal elements are zero:

$$\mathbf{x}' \mathbf{A} \mathbf{x} = [x_1 \ x_2] \begin{bmatrix} 0 & a \\ a & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Since we have

$$\det A_{11} = 0, \det A_{22} = -a^2 < 0$$

we can conclude that the evaluated quadratic form is indefinite. Indeed, multiplication of matrices or using expansion of the evaluated form as in (A2.3) yields the quadratic form

$$E(\mathbf{x}) = [ax_2 \ ax_1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 2ax_1x_2$$

The sign of the resulting function  $E(\mathbf{x})$  depends on  $\mathbf{x}$ . If  $x_1$  and  $x_2$  are of identical signs,  $E(\mathbf{x})$  remains positive, otherwise it is negative. Therefore, vectors  $\mathbf{x}$  from the first and third quadrant render the discussed form positive, otherwise it is negative.

**A3**


---

**TIME-VARYING AND GRADIENT  
VECTORS, JACOBIAN AND  
HESSIAN MATRICES**


---

**Time-Varying Vectors**

A *time-varying vector*  $\mathbf{x}(t)$  is defined as a column vector (A1.1), but its components are themselves functions of time, i.e.,

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} \quad (\text{A3.1})$$

The time derivative of the vector  $\mathbf{x}(t)$  is defined as

$$\frac{d\mathbf{x}(t)}{dt} = \begin{bmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \\ \vdots \\ \frac{dx_n(t)}{dt} \end{bmatrix} \quad (\text{A3.2a})$$

or in a shorthand notation as

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix} \quad (\text{A3.2b})$$

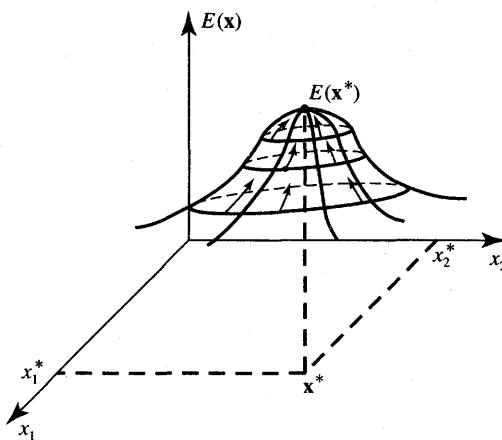

---

**Gradient Vectors**

Consider a scalar function  $E(\mathbf{x})$  of a vectorial variable  $\mathbf{x}$ , which is defined as an  $n$ -component column vector as in (A1.1). The *gradient vector* of  $E(\mathbf{x})$  with respect to the column vector  $\mathbf{x}$  is denoted as  $\nabla_{\mathbf{x}}E(\mathbf{x})$  and is equal

$$\nabla_{\mathbf{x}}E(\mathbf{x}) \triangleq \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \frac{\partial E}{\partial x_2} \\ \vdots \\ \frac{\partial E}{\partial x_n} \end{bmatrix} \quad (\text{A3.3})$$

Note that the gradient is also an  $n$ -tuple column vector.



**Figure A2** Illustration for the gradient vectors.

Notice that the immediate consequences of the definition of the gradient vector as in (A3.3) are

$$\nabla_x(x^t y) = y \quad (\text{A3.4a})$$

$$\nabla_x(x^t A y) = A y \quad (\text{A3.4b})$$

$$\nabla_x(x^t A x) = Ax + A^t x \quad (\text{A3.4c})$$

When dealing with quadratic forms and when  $A$  is a symmetric matrix, we have  $A = A^t$  and (A3.4c) simplifies to

$$\nabla_x(x^t A x) = 2Ax \quad (\text{A3.5})$$

The geometrical interpretation of a gradient is often useful. Figure A2 shows a function  $E(x)$  with  $x$  restricted to two components,  $x_1$  and  $x_2$ . Vector  $\nabla_x E(x)$  at a given point  $x_0$  specifies the direction of the maximum increase of  $E(x)$ . Note that at the maximum point,  $x^*$ , we have

$$\nabla_x E(x^*) = 0 \quad (\text{A3.6})$$

The property of the zero gradient also holds at a minimum point.

### Gradient of Time-Varying Vectors

Assume that  $x$  is a time-varying  $n$ -component vector and that a scalar function  $E[x(t)]$  (not necessarily a quadratic form) is defined for such a vectorial variable. The time derivative  $dE[x(t)]/dt$  for the evaluated scalar function can

be computed by the chain rule as follows

$$\frac{dE[\mathbf{x}(t)]}{dt} = \frac{\partial E}{\partial x_1} \cdot \frac{dx_1(t)}{dt} + \frac{\partial E}{\partial x_2} \cdot \frac{dx_2(t)}{dt} + \dots + \frac{\partial E}{\partial x_n} \cdot \frac{dx_n(t)}{dt} \quad (\text{A3.7a})$$

or, briefly

$$\frac{dE[\mathbf{x}(t)]}{dt} = \sum_{i=1}^n \frac{\partial E}{\partial x_i} \dot{x}_i(t) \quad (\text{A3.7b})$$

We may notice from (A3.3) that the derivative (A3.7) can be expressed succinctly using the gradient vector as follows:

$$\frac{dE[\mathbf{x}(t)]}{dt} = [\nabla_x E(\mathbf{x})]^t \dot{\mathbf{x}}(t) \quad (\text{A3.8})$$

Here we have used the notation of (A3.2b) for brevity.

### EXAMPLE A3.1

Consider that the following quadratic form with the matrix  $\mathbf{A}$  as in Example A2.1 is to be evaluated for changes in time:

$$E[\mathbf{x}(t)] = [e^{2t} \ t] \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} e^{2t} \\ t \end{bmatrix}$$

Notice that in our example we have

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} e^{2t} \\ t \end{bmatrix}$$

Using expression (A3.8) allows the result to be rewritten as a scalar product of the gradient vector

$$\nabla_x E(\mathbf{x}) = \begin{bmatrix} \frac{d}{dx_1} (x_1^2 + 4x_1x_2 + 3x_2^2) \\ \frac{d}{dx_2} (x_1^2 + 4x_1x_2 + 3x_2^2) \end{bmatrix} = \begin{bmatrix} 2x_1 + 4x_2 \\ 4x_1 + 6x_2 \end{bmatrix}$$

times the vector  $\dot{\mathbf{x}}(t)$  of value

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 2e^{2t} \\ 1 \end{bmatrix}$$

Combining the two results we obtain

$$\begin{aligned} \frac{dE[\mathbf{x}(t)]}{dt} &= [2e^{2t} + 4t \ 4e^{2t} + 6t] \begin{bmatrix} 2e^{2t} \\ 1 \end{bmatrix} \\ &= 2[2e^{4t} + (4t+2)e^{2t} + 3t] \end{aligned}$$

---

### Jacobian Matrix

Let us consider  $m$  functions  $E_i(\mathbf{x})$ , for  $i = 1, 2, \dots, m$ , of the vectorial variable  $\mathbf{x}$  defined as

$$\begin{aligned} E_1(\mathbf{x}) &= E_1(x_1, x_2, \dots, x_n) \\ E_2(\mathbf{x}) &= E_2(x_1, x_2, \dots, x_n) \\ &\vdots \\ E_m(\mathbf{x}) &= E_m(x_1, x_2, \dots, x_n) \end{aligned} \quad (\text{A3.9})$$

The matrix  $d\mathbf{E}/d\mathbf{x}$  called the *Jacobian matrix* and denoted  $\mathbf{J}(\mathbf{x})$  can be defined for (A3.9) as:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial E_1}{\partial x_1} & \frac{\partial E_1}{\partial x_2} & \dots & \frac{\partial E_1}{\partial x_n} \\ \frac{\partial E_2}{\partial x_1} & \frac{\partial E_2}{\partial x_2} & \dots & \frac{\partial E_2}{\partial x_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial E_m}{\partial x_1} & \frac{\partial E_m}{\partial x_2} & \dots & \frac{\partial E_m}{\partial x_n} \end{bmatrix} \quad (\text{A3.10})$$

### EXAMPLE A3.2

Let us compute the Jacobian matrix at  $\mathbf{x} = [2 \ 1]^t$  for the following functions of  $E_1(\mathbf{x})$  and  $E_2(\mathbf{x})$ :

$$\begin{aligned} E_1(\mathbf{x}) &= x_1^2 - x_1 x_2 \\ E_2(\mathbf{x}) &= x_1 x_2 + x_2^2 \end{aligned}$$

The Jacobian matrix expressed from (A3.10) is

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} 2x_1 - x_2 & -x_1 \\ x_2 & x_1 + 2x_2 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 1 & 4 \end{bmatrix}$$

---

### Hessian Matrix

For a scalar-valued function  $E(\mathbf{x})$ , a matrix of second derivatives called the *Hessian matrix* is defined as follows:

$$\nabla_x^2 E(\mathbf{x}) = \nabla_{\mathbf{x}} [\nabla_{\mathbf{x}} E(\mathbf{x})] \quad (\text{A3.11})$$

$$\nabla_x^2 E(\mathbf{x}) \triangleq \begin{bmatrix} \frac{\partial^2 E}{\partial x_1^2} & \frac{\partial^2 E}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 E}{\partial x_1 \partial x_n} \\ \frac{\partial^2 E}{\partial x_2 \partial x_1} & \frac{\partial^2 E}{\partial x_2^2} & \cdots & \frac{\partial^2 E}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 E}{\partial x_n \partial x_1} & \frac{\partial^2 E}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 E}{\partial x_n^2} \end{bmatrix} \quad (\text{A3.12})$$

Note that the Hessian matrix is of size  $n \times n$  and is symmetric. The matrix is often denoted by  $\mathbf{H}$ .

### EXAMPLE A3.3

Let us compute the Hessian matrix and determine whether it is positive definite for the function

$$E(\mathbf{x}) = (x_2 - x_1)^2 + (1 - x_1)^2$$

Computing first the gradient vector yields

$$\nabla_x E(\mathbf{x}) = 2 \begin{bmatrix} 2x_1 - x_2 - 1 \\ x_2 - x_1 \end{bmatrix}$$

and the Hessian matrix of size  $2 \times 2$  becomes

$$\nabla_x^2 E(\mathbf{x}) = 2 \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$$

Since the principal determinants of the matrix are 4 and 2, the Hessian matrix is positive definite. Note that the Hessian matrix is constant in this example and independent of  $\mathbf{x}$ . ■

## A4

### SOLUTION OF OPTIMIZATION PROBLEMS

#### Unconstrained Optimization Problem

An optimization problem is usually defined in terms of the *minimization of a scalar function* of a number of variables. If the variables are not constrained by inequalities or equality relationships, the optimization is said to be *unconstrained*. The function to be minimized is an *objective function*, sometimes also called

a cost function or criterion function. In neural network studies, the objective functions take the forms that follow:

1. *For feedforward networks:* The scalar error function  $E(\mathbf{w})$  in the weight space is the objective function.
2. *For recurrent networks:* The scalar energy function  $E(\mathbf{v})$  in the network output space is the objective function.

For a scalar valued objective function  $E(x)$  of a single variable  $x$ , the well-known conditions for a minimum at  $x = x^*$  are

$$\begin{aligned}\frac{dE(x^*)}{dx} &= 0 \\ \frac{d^2E(x^*)}{dx^2} &> 0\end{aligned}\quad (\text{A4.1})$$

For a scalar valued function  $E(\mathbf{x})$  of a vectorial variable  $\mathbf{x}$  we generalize conditions (A4.1) as given below:

$$\nabla_x E(\mathbf{x}^*) = \mathbf{0} \quad (\text{A4.2a})$$

$$\nabla_x^2 E(\mathbf{x}^*) \text{ is positive definite} \quad (\text{A4.2b})$$

Condition (A4.2a) requires the gradient vector at a minimum of  $E(\mathbf{x})$  to be a null vector. Condition (A4.2b) requires the Hessian matrix at a minimum of  $E(\mathbf{x})$  to be positive definite. The evaluation of conditions (A4.2) is possible only for continuous functions involved in expressions for  $E(\mathbf{x})$ , and in matrices  $\nabla_x E(\mathbf{x})$  and  $\nabla_x^2 E(\mathbf{x})$  for all values of  $\mathbf{x}$ .

To derive conditions (A4.2), let us inspect a two-variable function  $E(\mathbf{x})$  expanded in a Taylor series at  $\mathbf{x} = \mathbf{x}^*$  and retain the linear and quadratic term of the expansion:

$$\begin{aligned}E(\mathbf{x}) \cong E(\mathbf{x}^*) + (x_1 - x_1^*) \frac{\partial E(\mathbf{x}^*)}{\partial x_1} + (x_2 - x_2^*) \frac{\partial E(\mathbf{x}^*)}{\partial x_2} \\ + \frac{1}{2}(x_1 - x_1^*)^2 \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1^2} + \frac{1}{2}(x_2 - x_2^*)^2 \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_2^2} \\ + (x_1 - x_1^*)(x_2 - x_2^*) \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1 \partial x_2}\end{aligned}\quad (\text{A4.3a})$$

Equation (A4.3a) rewritten in the matrix form becomes for  $\Delta x_1 \triangleq x_1 - x_1^*$  and  $\Delta x_2 \triangleq x_2 - x_2^*$ :

$$\begin{aligned}E(\mathbf{x}) \cong E(\mathbf{x}^*) + \left[ \begin{array}{cc} \frac{\partial E(\mathbf{x}^*)}{\partial x_1} & \frac{\partial E(\mathbf{x}^*)}{\partial x_2} \end{array} \right] \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} \\ + \frac{1}{2} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix}^T \begin{bmatrix} \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1^2} & \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1 \partial x_2} \\ \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_2 \partial x_1} & \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_2^2} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix}\end{aligned}\quad (\text{A4.3b})$$

Using the gradient vector (A3.3), Hessian matrix (A3.12), and the notation

$$\Delta \mathbf{x} \triangleq \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix}$$

we obtain the following matrix form of (A4.3b):

$$E(\mathbf{x}) \cong E(\mathbf{x}^*) + [\nabla_{\mathbf{x}} E(\mathbf{x}^*)]^t \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^t [\nabla_{\mathbf{x}}^2 E(\mathbf{x}^*)] \Delta \mathbf{x} \quad (\text{A4.3c})$$

If  $\mathbf{x}^*$  is to be a minimum of  $E(\mathbf{x})$  as in (A4.3c), any infinitely small change  $\Delta \mathbf{x}$  at  $\mathbf{x}^*$  should result in  $E(\mathbf{x}^* + \Delta \mathbf{x}) > E(\mathbf{x}^*)$ . This requires the following:

1. The gradient vector at  $\mathbf{x}^*$  vanishes and makes the linear term in (A4.3c) of zero value, and
2. The Hessian matrix at  $\mathbf{x}^*$  is positive definite, which makes the last term in the quadratic form (A4.3c) positive independently of  $\Delta \mathbf{x}$ .

### Analytical Solution of a Selected Optimization Problem

Suppose that

$$E(\mathbf{x}) = \frac{1}{2} \mathbf{x}^t \mathbf{A} \mathbf{x} + \mathbf{b}^t \mathbf{x} \quad (\text{A4.4})$$

where  $\mathbf{x}$  is an  $n$ -dimensional column vector;  $\mathbf{A}$  is an  $n \times n$  symmetric, positive definite matrix; and  $\mathbf{b}$  is an  $n$ -dimensional constant vector.

The scalar multivariable function  $E(\mathbf{x})$  as defined in (A4.4) is one of the few functions for which an analytical solution for the unconstrained minimum exists. Computing the gradient vector and the Hessian matrix of  $E(\mathbf{x})$  yields:

$$\nabla_{\mathbf{x}} E(\mathbf{x}) = \mathbf{A} \mathbf{x} + \mathbf{b} \quad (\text{A4.5a})$$

$$\nabla_{\mathbf{x}}^2 E(\mathbf{x}) = \mathbf{A} \quad (\text{A4.5b})$$

Setting condition (A4.2a) for the minimum at  $\mathbf{x}^*$ , we obtain the equation for  $\mathbf{x}^*$ :

$$\mathbf{A} \mathbf{x}^* + \mathbf{b} = \mathbf{0} \quad (\text{A4.6a})$$

which has the solution

$$\mathbf{x}^* = -\mathbf{A}^{-1} \mathbf{b} \quad (\text{A4.6b})$$

Note that the Hessian matrix as in (A4.5b) has been assumed to be constant and positive definite; therefore, the solution found at  $\mathbf{x}^*$  in (A4.6b) is a minimum.

**EXAMPLE A4.1**

Let us evaluate the location of the minimum of the following function having the form of Equation (A4.4):

$$E(\mathbf{x}) = \frac{1}{2} [x_1 \ x_2] \begin{bmatrix} 2 & 2 \\ 2 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [1 \ 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

This quadratic form, supplemented by the linear term, is equivalent to the expanded polynominal of two variables:

$$E(\mathbf{x}) = x_1^2 + 2x_1x_2 + 4x_2^2 + x_1$$

Equation (A4.5a) takes the form:

$$\begin{bmatrix} 2 & 2 \\ 2 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \mathbf{0}$$

Solving it yields vector  $\mathbf{x} = \mathbf{x}^*$  such that

$$\begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = -\frac{1}{6} \begin{bmatrix} 4 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{2}{3} \\ \frac{1}{6} \end{bmatrix}$$

Checking the principal determinants of the Hessian matrix,  $\mathbf{A}$ , we obtain

$$\det A_{11} = 2 > 0$$

$$\det A_{22} = 2 \cdot 8 - 4 = 12 > 0$$

Therefore, because the Hessian matrix is positive definite, the point  $\mathbf{x}^*$  is the minimum of  $E(\mathbf{x})$ .

Note that selecting an arbitrary  $\Delta\mathbf{x}$  value yields a vanishing linear term and the quadratic term of the series expansion (A4.3c) is equal:

$$[\Delta x_1 \ \Delta x_2] \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = (\Delta x_1)^2 + 2\Delta x_1\Delta x_2 + 4(\Delta x_2)^2$$

This result is equivalent to the expression

$$(\Delta x_1 + \Delta x_2)^2 + 3(\Delta x_2)^2 > 0$$

which is always positive if  $\Delta x_1$  and  $\Delta x_2$  are not both equal to zero.

**Geometrical Interpretation**

Note that the gradient vector nullification at  $\mathbf{x} = \mathbf{x}^*$  as in (A4.2a) is necessary for the existence of the minimum at  $\mathbf{x}^*$ . The same property of gradient vector nullification, however, is also shared by maxima and saddle points. This

is illustrated for a function  $E(x_1, x_2)$  of two variables in Figure A3. The figure shows the shapes of  $E(x_1, x_2)$  for the case of a minimum (a), and of a saddle (b).

Assume that the functions represented by the surfaces shown are quadratic forms with the linear term in the form as in (A4.4). Matrix  $\mathbf{A}$  must be positive definite for the function shown in Figure A3(a). The figure illustrates conditions described in Example A4.1 with the minimum at  $\mathbf{x} = \mathbf{x}^* = [-2/3 \quad 1/6]^T$ . Figure A3(b) displays contour maps showing constant value lines for the function from Figure A3(a).

A quadratic form is, however, indefinite for the function depicted in Figure A3(c) because it can take either sign depending on  $\Delta\mathbf{x}$ . The function itself is shaped like a saddle. The function increases in one direction  $\Delta\mathbf{x}$ , but decreases in the other and the form is said to be *indefinite* at the saddle point. Also, matrix  $\mathbf{A}$  is indefinite at the saddle.

The reader may verify that if  $\mathbf{A}$  is chosen in Example A4.1 as

$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & -8 \end{bmatrix}$$

matrix  $\mathbf{A}$  becomes indefinite based on the test from Section A2.2 since

$$\det A_{11} = 2 > 0$$

$$\det A_{22} = -20 < 0$$

The saddle point is the solution of the equation

$$\begin{bmatrix} 2 & 2 \\ 2 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

which is

$$\mathbf{x}^* = \begin{bmatrix} -0.4 \\ -0.1 \end{bmatrix}$$

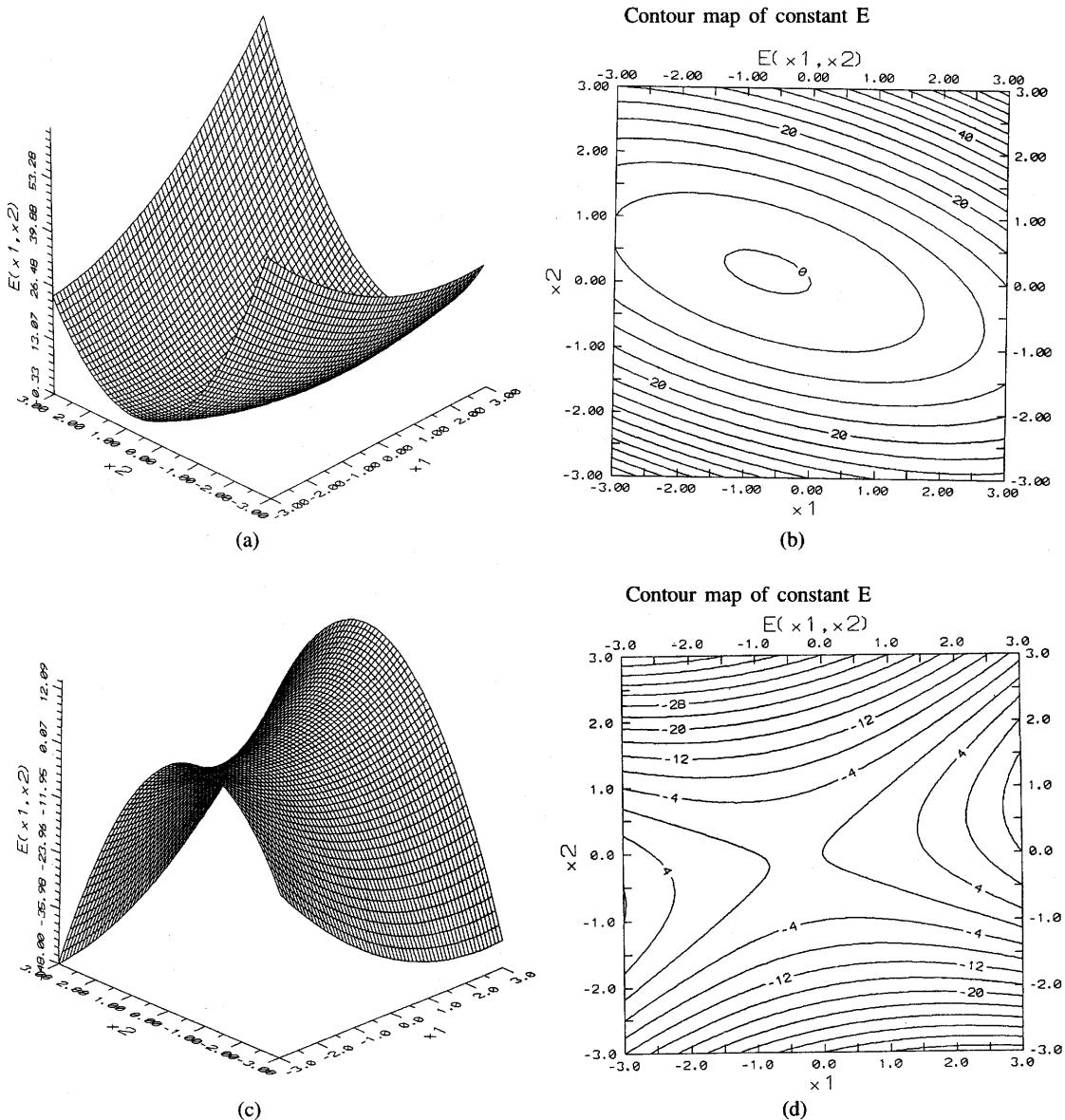
This case of a two-variable function is depicted in Figure A3(c). Figure A3(d) displays contour maps for the function from Figure A3(c) that possesses a saddle.

## A5

### STABILITY OF NONLINEAR DYNAMICAL SYSTEMS

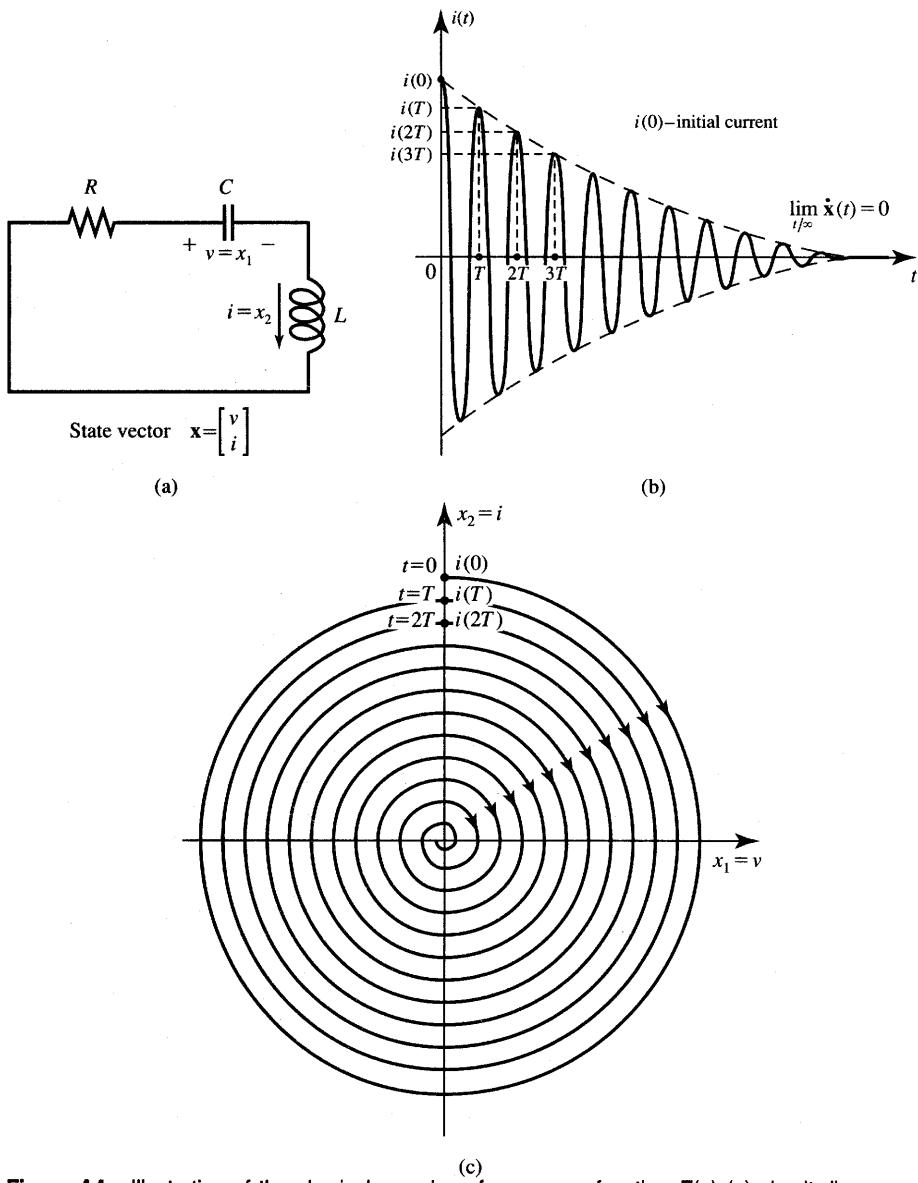
#### Preliminary Considerations

Late in the nineteenth century, the Russian mathematician A. M. Lyapunov developed an approach to stability analysis of dynamical systems. The unique feature of the approach is that only the form of the differential or difference



**Figure A3** Geometrical interpretation of minimum and saddle point: (a) function from Example A4.1 with minimum, (b) contour map for (a), (c) function with saddle point, and (d) contour map for (c).

equations needs to be known, not their solutions. The method, called *Lyapunov's method*, is based on the generalized energy concept. The method requires evaluation of a so-called Lyapunov function. Consider first an analogy between Lyapunov's function and the energy function of a physical system.



**Figure A4** Illustration of the physical meaning of an energy function  $E(\mathbf{x})$  (a) circuit diagram, (b) current waveform, and (c) state vector trajectory.

Figure A4(a) shows an autonomous (i.e., unforced) second-order electric system consisting of a series RLC connection. In this case, capacitor voltage  $v$  and inductor current  $i$  can be defined as state variables, and  $\mathbf{x}$  is the state vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \triangleq \begin{bmatrix} v \\ i \end{bmatrix} \quad (\text{A5.1})$$

The total physical energy stored in the system is known to be

$$E(\mathbf{x}) = \frac{1}{2} Cv^2 + \frac{1}{2} Li^2 = \frac{1}{2} [v \ i] \begin{bmatrix} C & 0 \\ 0 & L \end{bmatrix} \begin{bmatrix} v \\ i \end{bmatrix} \quad (\text{A5.2})$$

Thus the energy function is a quadratic function of state variables and

$$E(\mathbf{x}) > 0 \quad (\text{A5.3})$$

unless  $\mathbf{x} = \mathbf{0}$ . We can, therefore, say that the energy function is positive definite. When the energy function rate of change in time is evaluated we notice that

$$\mathbf{x}(t) = \begin{bmatrix} v(t) \\ i(t) \end{bmatrix} \quad (\text{A5.4a})$$

and from (A3.7a) we obtain

$$\frac{dE(\mathbf{x})}{dt} = \frac{\partial E(\mathbf{x})}{\partial x_1} \dot{x}_1 + \frac{\partial E(\mathbf{x})}{\partial x_2} \dot{x}_2 \quad (\text{A5.4b})$$

which is equivalent to

$$\frac{dE(\mathbf{x})}{dt} = [\nabla_x E(\mathbf{x})]^t \dot{\mathbf{x}} \quad (\text{A5.4c})$$

Using (A5.2) the derivative of the energy function versus time becomes

$$\frac{dE(\mathbf{x})}{dt} = [Cv \ Li] \begin{bmatrix} \dot{v} \\ \dot{i} \end{bmatrix} = Cv \dot{v} + Li \dot{i} \quad (\text{A5.5})$$

Knowing the form of the differential equations describing system  $\dot{x}_1 = f_1(\mathbf{x})$ ,  $\dot{x}_2 = f_2(\mathbf{x})$  is usually sufficient for evaluation of the expressions for  $E(\mathbf{x})$  and  $dE(\mathbf{x})/dt$ . For our sample circuit from Figure A4(a) the equation describing the system is

$$L \frac{di(t)}{dt} + \frac{1}{C} \int_{-\infty}^t i(t) dt + Ri(t) = 0 \quad (\text{A5.6a})$$

This integrodifferential equation can be rewritten as a system of two first-order equations using the state vector as defined in (A5.1):

$$\begin{aligned} \frac{dv(t)}{dt} &= \frac{i(t)}{C} \\ \frac{di(t)}{dt} &= -\frac{v(t)}{L} - \frac{Ri(t)}{L} \end{aligned} \quad (\text{A5.6b})$$

which is equivalent to

$$\begin{bmatrix} \dot{v} \\ \dot{i} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{C} \\ -\frac{1}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} v \\ i \end{bmatrix} \quad (\text{A5.6c})$$

Now we can evaluate the sign of the rate of change of  $E[\mathbf{x}(t)]$  versus time

using (A5.5):

$$\frac{dE(\mathbf{x})}{dt} = i(t)v(t) - i(t)v(t) - R i^2(t) \quad (\text{A5.7a})$$

which reduces to

$$\frac{dE(\mathbf{x})}{dt} = -R i^2(t) < 0 \quad (\text{A5.7b})$$

The discussed energy function always decreases in time unless  $i = 0$ . The result (A5.7b) can be easily interpreted physically. The circuit of Figure A4(a) represents a lossy resonant circuit. Once the circuit is energized with  $v(0) \neq 0$  or  $i(0) \neq 0$ , it starts producing damped oscillations as illustrated in Figure A4(b). The damping phenomena are a result of the nonzero resistance value in the circuit, which makes the system actually dissipative.

The evaluated solution for  $v(t)$  and  $i(t)$  can also be represented as a trajectory  $v(i)$ , with the time variable eliminated. The damped oscillations from Figure A4(b) take the form of a state trajectory as in Figure A4(c), which is drawn to the origin after a sufficiently long time has elapsed. Again, neither the time-domain solution nor knowledge of the trajectories is required to prove that the system is stable. Evaluation of the energy function alone allows the stability of the system to be proven or disproven.

### Lyapunov's Theorem

Consider the autonomous (i.e., unforced) system described with a system of  $n$  first-order linear or nonlinear differential equations:

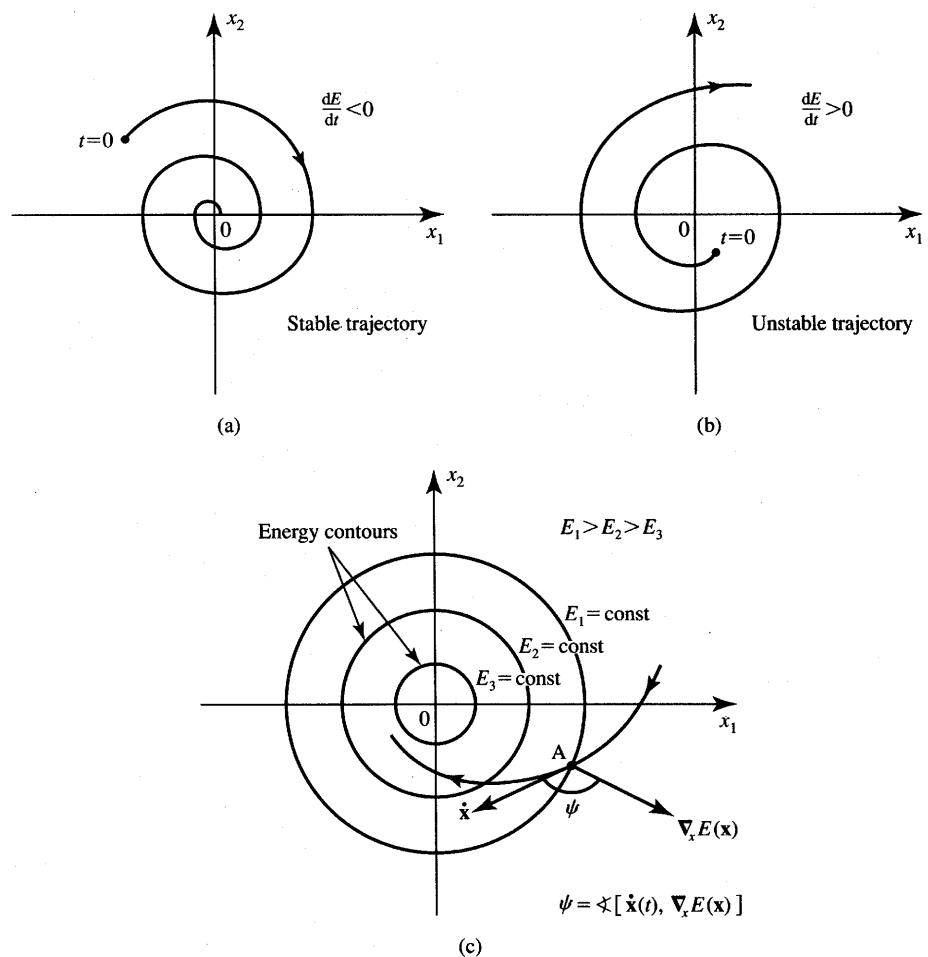
$$\begin{aligned} \dot{x}_1 &= f_1(\mathbf{x}) \\ \dot{x}_2 &= f_2(\mathbf{x}) \\ &\vdots \\ \dot{x}_n &= f_n(\mathbf{x}) \end{aligned} \quad (\text{A5.8a})$$

which is equivalent to

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \quad (\text{A5.8b})$$

We assume that the equations have been written so that  $\mathbf{x} = \mathbf{0}$  is an equilibrium point that satisfies equations  $\mathbf{f}(\mathbf{0}) = \mathbf{0}$ . We formulate a condition for the equilibrium  $\mathbf{x} = \mathbf{0}$  to be asymptotically stable which means that the state vector goes to zero as time goes to infinity. The argument goes like this: *If a positive definite function  $E(\mathbf{x})$  can be found such that*

1.  $E$  is continuous with respect to all components  $x_i$ , for  $i = 1, 2, \dots, n$ , and
2.  $dE[\mathbf{x}(t)]/dt < 0$ , which indicates the energy function decreasing in time, then the origin is asymptotically stable.



**Figure A5** Illustration of the Lyapunov theorem: (a) stable trajectory, (b) unstable trajectory, and (c) respective positions of velocity and gradient vectors.

Function  $E(\mathbf{x})$  satisfying the above requirements is called a *Lyapunov function*. The function is not unique; rather, many different Lyapunov functions can be found for a given system. If at least one function is known that meets conditions 1 and 2 given above, the system is asymptotically stable. This means that its attractor is a point rather than a bounded trajectory. Figure A5(a) illustrates asymptotical stability, which we simply call *stability* in our neural network studies. This trajectory is of the same type as that shown in Figure A4(c) representing the decaying oscillations from Figure A4(b). Figure A5(b) shows an unstable trajectory of a system with an increasing energy function such that  $dE[\mathbf{x}(t)]/dt > 0$ . This case would correspond to the negative resistance value in (A5.7b) and to the energy function increasing in time.

Note that modifying condition 2 so as not to have strictly negative but non-positive derivatives of the energy function would allow for so-called *limit cycles*, which correspond to limited but nondamped and steady oscillations. Indeed, for the example circuit from Figure A4(a) with  $R = 0$  we obtain from (A5.7b):

$$\frac{dE(\mathbf{x})}{dt} = 0 \quad (\text{A5.9})$$

In this case the energy function remains constant in time. This corresponds to the lossless LC resonant circuit, which produces nondecaying sinusoidal oscillations. Since the circuit does not dissipate its energy in the resistance, the oscillations are of constant amplitude. This case corresponds to a circular trajectory on the plane  $x_1, x_2$  rather than decreasing (stable) or increasing (unstable) spiral trajectories on the state plane.

Figure A5(c) provides an interesting observation relating the energy function  $E(\mathbf{x})$  and the trajectories of the stable system. Let us note that based on (A3.8) condition 2 is equivalent to

$$\frac{dE[\mathbf{x}(t)]}{dt} = [\nabla_{\mathbf{x}} E(\mathbf{x})]^t \dot{\mathbf{x}}(t) \quad (\text{A5.10})$$

which is the scalar product of the energy function gradient vector and of the velocity of the state vector  $\dot{\mathbf{x}}(t)$ . The velocity vector is obviously tangent to the trajectory. The gradient vector is normal to the energy contour line. These conditions are illustrated in Figure A5(c). Since stability condition 2 requires the scalar product (A5.10) to be negative, the angle between the two vectors

$$\psi \triangleq \measuredangle[\dot{\mathbf{x}}(t), \nabla_{\mathbf{x}} E(\mathbf{x})] \quad (\text{A5.11})$$

must be bounded as follows

$$-180^\circ < \psi < -90^\circ \quad (\text{A5.12})$$

This, in turn, means that the state velocity vector  $\dot{\mathbf{x}}(t)$  must have a negative projection on the gradient vector,  $\nabla_{\mathbf{x}} E(\mathbf{x})$ . This also indicates that the state velocity vector  $\dot{\mathbf{x}}(t)$  has a positive projection on the negative gradient vector,  $-\nabla_{\mathbf{x}} E(\mathbf{x})$ .

As a final note, the energy function having physical energy meaning often fits as a Lyapunov function. In many cases, however, in which a system model is described with differential equations, it may not be clear what “energy” of the system means. Therefore, the conditions that  $E(\mathbf{x})$  must satisfy to be a Lyapunov function are based on mathematical rather than physical considerations. For our neural network studies of fully coupled single-layer networks, the Lyapunov function is often called the *computational energy function* and has no physical energy relation. Note also that the inability to find a satisfactory Lyapunov function does not mean that the evaluated system is unstable.

No unique and best method exists to identify a Lyapunov function for a given system. A form  $E(\mathbf{x})$  can be assumed either as a pure guess or obtained by physical insight and energy-like considerations. Following the choice of the

hypothetical energy function which is positive definite,  $dE(\mathbf{x})/dt$  needs to be tested involving the equations  $\mathbf{f}(\mathbf{x}) = \dot{\mathbf{x}}$  in the process.

### EXAMPLE A5.1

Let us check the stability of the closed feedback loop control system shown in Figure A6. The system is described by the equation

$$U(s) - Q(s) \frac{K}{s(s + \alpha)} = Q(s)$$

where  $\alpha$  and  $K$  are positive constants. To evaluate the stability of the closed-loop system we use the equation for an autonomous system and assume  $U(s) = 0$ . This corresponds to the relationship

$$s^2 Q(s) + \alpha s Q(s) + K Q(s) = 0$$

In the time domain, the autonomous system is described with the equation

$$\ddot{q} + \alpha \dot{q} + Kq = 0$$

The following definitions of state variables can be introduced in order to identify the system of first-order differential equations:

$$\begin{aligned} x_1 &\triangleq q \\ x_2 &\triangleq \dot{q} \end{aligned}$$

Since  $x_2 = \dot{x}_1$ , we have  $\dot{x}_2 = \ddot{q}$ , and Equation (A5.8) can be written as follows:

$$\begin{aligned} \dot{x}_1 &= 0x_1 + 1x_2 \\ \dot{x}_2 &= -Kx_1 - \alpha x_2 \end{aligned}$$

We thus have the  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$  relationship as given below:

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -K & -\alpha \end{bmatrix} \mathbf{x}, \text{ where } \mathbf{x} \triangleq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Let our first guess for the energy function  $E(\mathbf{x})$  be

$$E(\mathbf{x}) = x_1^2 + x_2^2$$

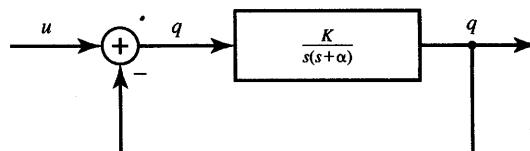


Figure A6 Illustration for Lyapunov function evaluation in Example A5.1.

Since the function is positive definite and fulfills condition 1, checking condition 2 yields:

$$\frac{dE[\mathbf{x}(t)]}{dt} = \begin{bmatrix} \frac{\partial E(\mathbf{x})}{\partial x_1} & \frac{\partial E(\mathbf{x})}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{dx_1}{dt} \\ \frac{dx_2}{dt} \end{bmatrix} = 2x_1 \dot{x}_1 + 2x_2 \dot{x}_2$$

Using the equations describing the system, the rate of change of the energy function in time becomes

$$\frac{dE[\mathbf{x}(t)]}{dt} = 2x_1x_2 + 2x_2(-Kx_1 - \alpha x_2) = 2x_1x_2(1 - K) - 2\alpha x_2^2$$

To check whether or not this result is always positive independent of  $\mathbf{x}$  we can make it equivalent to the following quadratic form:

$$2x_1x_2(1 - K) - 2\alpha x_2^2 = [x_1 \quad x_2] \begin{bmatrix} 0 & 1 - K \\ 1 - K & -2\alpha \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Since  $\det A_{11} = 0$ , the derivative of the energy function is not negative and the system cannot be asserted to be stable. In fact, it cannot be called an unstable system either. However, another choice of the Lyapunov function can be attempted. Assuming

$$E(\mathbf{x}) = Kx_1^2 + (x_2 + \alpha x_1)^2$$

we have selected a positive definite function satisfying condition 1. Checking condition 2 yields:

$$\frac{dE[\mathbf{x}(t)]}{dt} = [2x_1K + 2(x_2 + \alpha x_1)\alpha]x_2 + 2(x_2 + \alpha x_1)(-Kx_1 - \alpha x_2) = -2\alpha Kx_1^2$$

This concludes the proof that the chosen energy function has a negative definite derivative versus time and therefore the system is stable. ■

## A6

---

### ANALYTIC GEOMETRY IN EUCLIDEAN SPACE IN CARTESIAN COORDINATES

The point  $P_0(x_0, y_0)$  in the two-dimensional Euclidean space in Cartesian coordinates can be represented as a two-component column vector  $\mathbf{x}_0$  defined as

$$\mathbf{x}_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

## Equation Forms of a Straight Line on a Plane

For this section, refer to Figure A7.

### 1. The gradient form:

$$y = kx + q \quad (\text{A6.1})$$

### 2. The intercept form:

$$\frac{x}{p} + \frac{y}{q} = 1, p \neq 0, q \neq 0 \quad (\text{A6.2})$$

where  $p$  and  $q$  are intercept coordinates with the  $x$  and  $y$  axes, respectively.

### 3. The general form

$$ax + by + c = 0, \quad a^2 + b^2 > 0 \quad (\text{A6.3})$$

where constants  $a$  and  $b$  are coordinates of a vector  $\mathbf{n}$ :

$$\mathbf{n} = \begin{bmatrix} a \\ b \end{bmatrix}$$

which is normal to the line. The vector is called a *normal vector* and has the property that it points to the side of a line for which  $ax + by + c > 0$ , which is called the *positive half-plane*. This property is proven below.

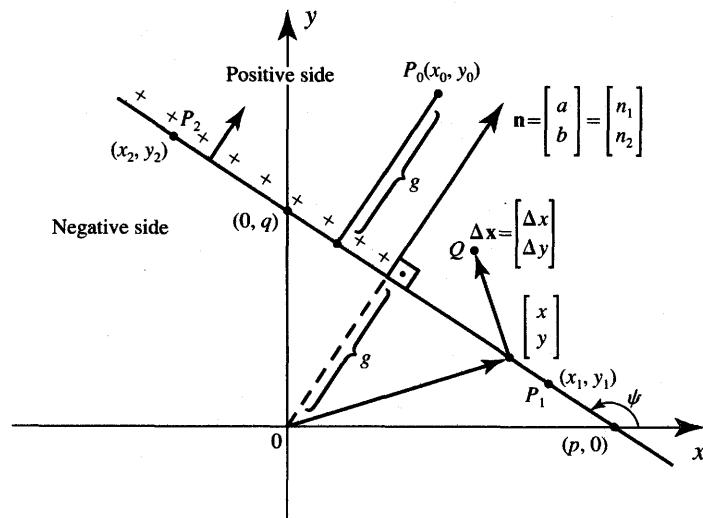


Figure A7 Illustration of equation forms for a straight line.

Indeed, for a point  $[x \ y]^t$  on the line we have from (A6.3)

$$\mathbf{n}^t \begin{bmatrix} x \\ y \end{bmatrix} + c = 0 \quad (\text{A6.4})$$

Assume that a vector  $\Delta\mathbf{x}$  has been added to  $\mathbf{x}$  so that point  $Q(x + \Delta x, y + \Delta y)$  is on the upper side of the line of Figure A7. Computing from (A6.4) yields

$$\begin{aligned} \mathbf{n}^t \begin{bmatrix} x + \Delta x \\ y + \Delta y \end{bmatrix} + c &= \left( \mathbf{n}^t \begin{bmatrix} x \\ y \end{bmatrix} + c \right) + \mathbf{n}^t \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\ &= \mathbf{n}^t \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \end{aligned} \quad (\text{A6.5})$$

Since the vectors  $\mathbf{n}$  and  $\Delta\mathbf{x} = [\Delta x \ \Delta y]^t$  are both nonzero and the angle between them is between  $-90^\circ$  and  $90^\circ$ , their scalar product (A6.5) is always positive. This proves that  $\mathbf{n}$  always points to the positive half-plane. Note that the positive half-plane is denoted by + signs, or by a small arrow pointing toward the positive side of the line.

Note that this equation form can be used with the unit normal vector  $\mathbf{r}$  rather than the normal vector.

#### 4. The two-point form:

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}, \text{ or} \quad (\text{A6.6a})$$

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0 \quad (\text{A6.6b})$$

The line passes through points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$

#### 5. The normal vector-point form:

$$\mathbf{n}^t(\mathbf{x} - \mathbf{x}_1) = 0, \text{ or} \quad (\text{A6.7a})$$

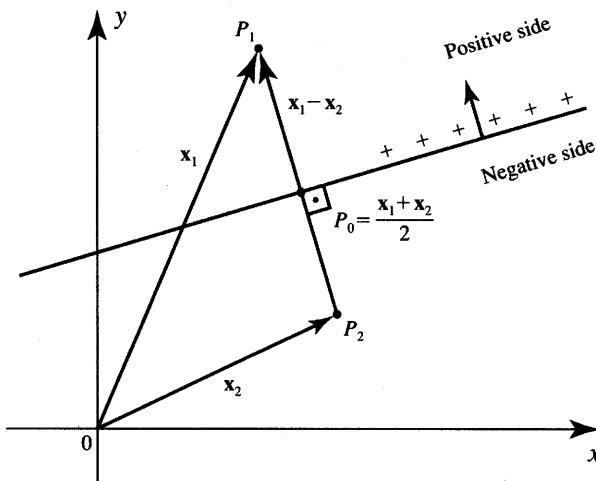
$$n_1(x - x_1) + n_2(y - y_1) = 0 \quad (\text{A6.7b})$$

The line is perpendicular to its normal vector  $\mathbf{n} = [n_1 \ n_2]^t$  and passes through point  $P_1(x_1, y_1)$ . This form of the equation remains valid also for the unit normal  $\mathbf{r}$  used rather than the normal vector. Indeed, dividing both sides of (A6.7a) by  $(n_1^2 + n_2^2)^{1/2}$  we obtain:

$$\frac{\mathbf{n}^t}{\|\mathbf{n}\|}(\mathbf{x} - \mathbf{x}_1) = 0 \quad (\text{A6.8})$$

Using the definition of the unit normal vector

$$\mathbf{r} \triangleq \frac{\mathbf{n}}{\|\mathbf{n}\|}$$



**Figure A8** Illustration of a line bisecting the segment between  $\mathbf{x}_1$  and  $\mathbf{x}_2$ .

we obtain an alternative normal vector-point form equation:

$$\mathbf{r}'(\mathbf{x} - \mathbf{x}_1) = 0 \quad (\text{A6.9})$$

6. The *form for bisecting a segment between  $\mathbf{x}_1$  and  $\mathbf{x}_2$* :

$$(\mathbf{x}_1 - \mathbf{x}_2)' \mathbf{x} + \frac{1}{2} (\|\mathbf{x}_2\|^2 - \|\mathbf{x}_1\|^2) = 0 \quad (\text{A6.10})$$

This line passes through the center of gravity of two points—point  $P_1(x_1, y_1)$  represented by the vector  $\mathbf{x}_1$ , and point  $P_2(x_2, y_2)$  represented by the vector  $\mathbf{x}_2$ —and is perpendicular to the line's normal vector  $\mathbf{x}_1 - \mathbf{x}_2$ . The coordinates of the center of gravity point  $P_0$  are  $0.5(\mathbf{x}_1 + \mathbf{x}_2)$ . This line is illustrated in Figure A8.

The derivation below of Equation (A6.10) is based on the Equation (A6.7a). Indeed, the line of interest is normal to the vector  $\mathbf{x}_1 - \mathbf{x}_2$  and passes through the center point  $P_0$ . We thus have from (A6.7a)

$$(\mathbf{x}_1 - \mathbf{x}_2)'(\mathbf{x} - \mathbf{x}_0) = 0 \quad (\text{A6.11a})$$

which is equivalent to

$$(\mathbf{x}_1 - \mathbf{x}_2)' \mathbf{x} - \frac{1}{2} (\mathbf{x}_1 - \mathbf{x}_2)' (\mathbf{x}_1 + \mathbf{x}_2) = 0 \quad (\text{A6.11b})$$

Performing multiplication and reducing similar terms we obtain

$$(\mathbf{x}_1 - \mathbf{x}_2)' \mathbf{x} + \frac{1}{2} (\mathbf{x}_2' \mathbf{x}_2 - \mathbf{x}_1' \mathbf{x}_1) = 0 \quad (\text{A6.11c})$$

Using the identity  $\mathbf{x}' \mathbf{x} = \|\mathbf{x}\|^2$  we notice that (A6.11c) becomes identical to (A6.10).

### Useful Relationships

For this section, refer to Figure A7. The *distance*  $g$  from the line  $ax + by + c = 0$  to the point  $P_0(x_0, y_0)$  is

$$g = \left| \frac{ax_0 + by_0 + c}{\sqrt{a^2 + b^2}} \right| \quad (\text{A6.12})$$

For special cases when the point  $P_0$  is the origin, the distance from the origin to the line equals

$$g = \left| \frac{c}{\sqrt{a^2 + b^2}} \right| \quad (\text{A6.13})$$

Note that the distance is always a non-negative quantity by definition.

### EXAMPLE A6.1

This example reviews the main forms of equations of a straight line and illustrates useful relationships. Let us begin by finding the gradient form of an equation for the line shown in Figure A9. Since

$$k = \tan \psi = \frac{3}{2}, \quad q = -3$$

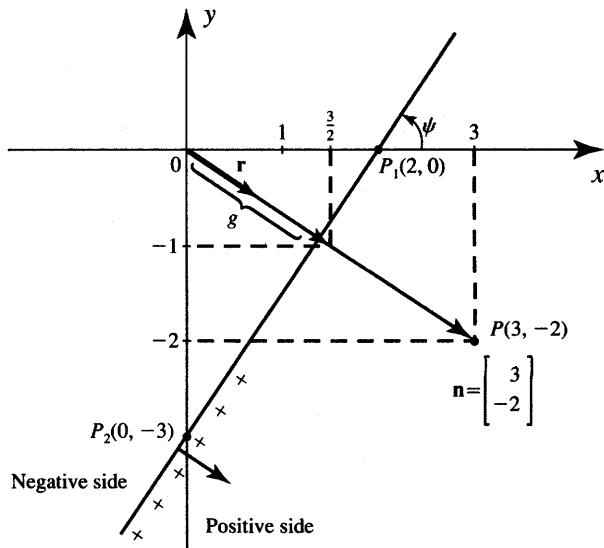


Figure A9 Illustration for Example A6.1.

then Equation (A6.1) has the form

$$y = \frac{3}{2}x - 3$$

Since  $p = 2$  and  $q = -3$ , the intercept form of equation is

$$\frac{x}{2} - \frac{y}{3} = 1$$

The general form of the equation can be obtained by rearranging either of the two forms above to the following form:

$$3x - 2y - 6 = 0$$

The resulting normal vector perpendicular to the line and pointing to the positive side of the line is  $[3 \quad -2]^t$

The unit normal vector  $\mathbf{r}$  is

$$\mathbf{r} = \frac{\begin{bmatrix} 3 \\ -2 \end{bmatrix}}{\sqrt{3^2 + 2^2}} = \frac{1}{\sqrt{13}} \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

Note that the length of the normal vector remains irrelevant for this form of line equation. Indeed,  $[3/2 \quad -1]^t$  is another normal vector to the line. The line's general equation for this normal vector is obtained by dividing the original line equation in the general form by 2 as given below:

$$\frac{3}{2}x - y - 3 = 0$$

The distance from the line to the origin is

$$g = \left| \frac{6}{\sqrt{3^2 + 2^2}} \right| = \frac{6}{\sqrt{13}}$$

The two-point form equation (A6.6) for the line passing through points  $(2, 0)$  and  $(0, -3)$  is

$$\frac{y}{x - 2} = \frac{-3}{-2}$$

Knowing the line normal vector and a single point  $P$  on the line, the equation as in (A6.7) can be obtained:

$$\begin{bmatrix} 3 \\ -2 \end{bmatrix}^t \left( \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right) = 0$$

or

$$3x - 2y - 6 = 0$$

This concludes the review of the example forms of straight line equations. ■

## Equation Forms of a Plane in Three-dimensional Space

**1. The intercept form:**

$$\frac{x}{p} + \frac{y}{q} + \frac{z}{s} = 1, p \neq 0, q \neq 0, s \neq 0 \quad (\text{A6.14})$$

where  $p, q$  and  $s$ , are plane intercept coordinates with the  $x, y$  and  $z$  axes, respectively.

**2. The general form:**

$$ax + by + cz + d = 0, a^2 + b^2 + c^2 > 0 \quad (\text{A6.15})$$

The constants  $a, b$  and  $c$  determine the normal vector  $\mathbf{n}$

$$\mathbf{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

which points to the *positive side* of the plane, usually denoted by an arrow. Equation (A6.15) can be rewritten in vectorial notation:

$$\mathbf{n}' \begin{bmatrix} x \\ y \\ z \end{bmatrix} + d = 0 \quad (\text{A6.16})$$

**3. The three-point form:**

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0 \quad (\text{A6.17})$$

The plane passes through three non-linear points  $P_1(x_1, y_1, z_1)$ ,  $P_2(x_2, y_2, z_2)$ , and  $P_3(x_3, y_3, z_3)$ .

**4. The normal vector-point form:**

$$a(x - x_1) + b(y - y_1) + c(z - z_1) = 0 \quad (\text{A6.18a})$$

or

$$\mathbf{n}'(\mathbf{x} - \mathbf{x}_1) = 0 \quad (\text{A6.18b})$$

where

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \text{ and } \mathbf{x}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$

The plane is perpendicular to the normal vector  $\mathbf{n}$  and passes through point

$P_1(x_1, y_1, z_1)$ . This form or equation also remains valid for the unit normal vector  $\mathbf{n}$  used rather than normal vector.

5. The form for bisecting a segment between  $\mathbf{x}_1$  and  $\mathbf{x}_2$ :

$$(\mathbf{x}_1 - \mathbf{x}_2)' \mathbf{x} + \frac{1}{2} (\|\mathbf{x}_2\|^2 - \|\mathbf{x}_1\|^2) = 0 \quad (\text{A6.19})$$

This plane passes through the center point between two points— $P_1(x_1, y_1, z_1)$  represented by the vector  $\mathbf{x}_1$ , and  $P_2(x_2, y_2, z_2)$  represented by the vector  $\mathbf{x}_2$ —and is perpendicular to the plane's normal vector  $\mathbf{x}_1 - \mathbf{x}_2$ . The coordinates of the center point  $P_0$  are  $0.5(\mathbf{x}_1 + \mathbf{x}_2)$ .

### Useful Relationships in Three-dimensional Euclidean Space

1. Distance of point  $P_0(x_0, y_0, z_0)$  from the plane  $ax + by + cz + d = 0$

$$g = \left| \frac{ax_0 + by_0 + cz_0 + d}{\sqrt{a^2 + b^2 + c^2}} \right| \quad (\text{A6.20})$$

When the point  $P_0$  is the origin, the distance (A6.20) reduces to

$$g = \left| \frac{d}{\sqrt{a^2 + b^2 + c^2}} \right| \quad (\text{A6.21})$$

2. The distance between two points  $P_1(x_1, y_1, z_1)$  and  $P_2(x_2, y_2, z_2)$  is equal to the length of vector  $\mathbf{x}_1 - \mathbf{x}_2$ :

$$\|\mathbf{x}_1 - \mathbf{x}_2\| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \quad (\text{A6.22})$$

where

$$\mathbf{x}_1 \triangleq \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \mathbf{x}_2 \triangleq \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

3. The mid-point of the line segment connecting points  $P_1$  and  $P_2$  as in (A6.22) corresponds to a vector  $\mathbf{x}_m$  such that

$$\mathbf{x}_m = \frac{\mathbf{x}_1 + \mathbf{x}_2}{2} \quad (\text{A6.23})$$

4. The center of gravity of a system with  $K$  masses  $m_k$  distributed at points  $\mathbf{x}_k$  is at  $\mathbf{x}_c$  such that

$$\mathbf{x}_c = \frac{\sum_{k=1}^K m_k \mathbf{x}_k}{\sum_{k=1}^K m_k} \quad (\text{A6.24})$$

Note that the center  $P(x, y, z)$  of mass of a triangle is given by

$$\mathbf{x}_c = \frac{\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3}{3}, \text{ i.e.,}$$

$$x = \frac{x_1 + x_2 + x_3}{3}, y = \frac{y_1 + y_2 + y_3}{3}, z = \frac{z_1 + z_2 + z_3}{3}$$

### EXAMPLE A6.2

This example reviews some forms of equations for a plane and illustrates useful relationships. Let us find the equations of a plane perpendicular to the vector  $[2 \ 3 \ 6]^t$  and passing through point  $(0, 7, 0)$ . The normal vector-point equation (A6.18) for the plane can be obtained:

$$2x + 3(y - 7) + 6z = 0, \text{ or}$$

$$2x + 3y + 6z - 21 = 0$$

To sketch the plane, it is convenient to rearrange the plane equation to the intercept form (A6.14) as follows:

$$\frac{x}{21/2} + \frac{y}{7} + \frac{z}{7/2} = 1$$

The plane is shown in Figure A10. The unit normal vector for the plane is

$$\mathbf{r} = \frac{1}{\sqrt{4+9+36}} \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} \frac{2}{7} \\ \frac{3}{7} \\ \frac{6}{7} \end{bmatrix}$$

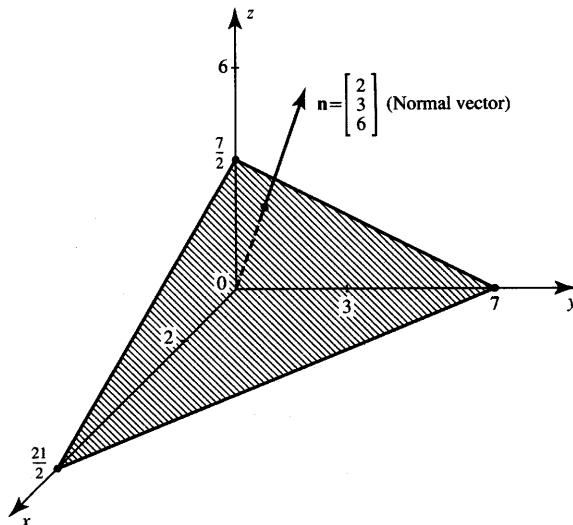


Figure A10 Plane for Example A6.2.

Note that the distance between the origin and the plane is

$$g = \left| \frac{21}{\sqrt{2^2 + 3^2 + 6^2}} \right| = 3$$

Since the normal vector points to the upper part of the plane, the upper part is the positive half-space; the lower part, including the origin, is the negative half-space.

## A7

### SELECTED NEURAL NETWORK RELATED CONCEPTS

#### Exclusive OR (XOR) Operation

The *Exclusive OR*, denoted with operator  $\oplus$ , is a binary operation on logic arguments  $x_1$  and  $x_2$ . It assigns the value 1 to the two arguments if and only if they have complementary values; that is,  $x_1 \oplus x_2 = 1$  if either  $x_1$  or  $x_2$  is 1, but not when both  $x_1$  and  $x_2$  are 0, or when both  $x_1$  and  $x_2 = 1$ . The truth table for Boolean variables 0, 1 is

$x_1$	$x_2$	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Let us note that the operation is associative since

$$(x_1 \oplus x_2) \oplus x_3 = x_1 \oplus (x_2 \oplus x_3) = x_1 \oplus x_2 \oplus x_3$$

This allows the extension of the XOR operation for  $n$  logic arguments as

$$\text{XOR}(x_1, x_2, \dots, x_n) = x_1 \oplus x_2 \oplus \dots \oplus x_n$$

The multivariable XOR function is called the *parity function* since it assigns the value 1 to  $n$  arguments if and only if the number of the arguments of value 1 is odd.

---

# Index

A/D conversion. *See* Analog-to-digital conversion  
Absolute correction rule, 122  
dichotomizer training and, 128  
Activation functions. *See also*  
    *specific type*  
biological neuron model, 32–36  
ETANN 80170 chip, 655, 657  
feedforward network, 37, 38,  
    39–42  
steepness of, 209–10  
Activation value, calculation of,  
    575  
Acute myocardial infarction diagnosis, expert system for,  
    537–39  
ADALINE (ADaptive LINEar combiner), 19

Adaptation, learning and, 55–58  
Adaptive mapping system, supervised learning, 151–52  
Adaptive mirrors, processor for,  
    658  
Adaptive resonance networks, 20,  
    76, 432–44  
Adaptive resonance theory 1  
    (ART1) network,  
        432–44  
    algorithm for, 432–37  
Addition, binary, 287–94  
Address-addressable memory,  
    316  
Adjustable weights, 586. *See also*  
    Weight(s); Weight adjustment(s)  
analog storage of, 592–95  
Algorithms  
    ART1 network, 434–37  
    error back-propagation, 187,  
        188–90  
    Ho-Kashyap, 377  
    implementation of, 565–66.  
        *See also* Hardware; *specific algorithms*  
    multicategory continuous perceptron training, 180–81  
    Newton-Raphson, 286  
    R-category discrete perceptron training, 146  
recording (storage), 315  
recurrent associative memory storage and retrieval,  
    335–36  
recursive update, 258

- relaxation, 284–86  
 retrieval, 327–28  
 selection of, 569  
 self-organizing feature map, 424–32  
 single continuous perceptron training, 141–42  
 single discrete perceptron training, 131–32  
 storage, 328–29  
 synchronous state updating, 261–62  
 All-MOS multiplier, 633–36  
 Allen, P. E., 605, 610, 612, 622  
 Allen, R. B., 254  
 Almeida, L. B., 213  
 Alspector, J., 254  
**ALVINN** (Autonomous Land Vehicle In a Neural Network), 8–10, 76  
 Amari, S. I., 19  
 Amit, D. J., 376  
**Amplifiers**  
 constraint, 459  
 current, 610–13  
 differential voltage, 617–24  
 operational. *See Operational amplifiers*  
 push-pull, 613  
 transconductance, 622–26  
**Analog electronic neurocomputing circuits**, 575–80  
**Analog multipliers**, 630–43. *See also* Scalar product circuits  
 all-MOS realization of, 633–36  
 floating-gate transistor, 640–43  
 learning weights and, 605–8  
 with weight storage, 638–40  
**Analog neural network processor**, 657–58, 659  
**Analog storage of adjustable weights**, 592–95  
**Analog-to-digital (A/D) conversion**  
 continuous-time network and, 273–76, 278–83, 287–94  
 optimization and, 14–15  
 Anastassiou, D., 592, 594  
 Anderson, C. W., 499, 547, 548  
 Anderson, D., 547  
 Anderson, J. A., 19, 547  
 Anderson, L. R., 665  
 Andro, M., 665  
 Antsaklis, P. J., 244  
 Apple Computer, 465  
 Applied input, instantaneous response to, 4–10  
**Approximation**, 7, 8  
 continuous function, 204, 206, 414  
 learning as, 55–56  
 multilayer feedforward networks in, 196–205, 206  
 staircase, 197–200  
 successive, 284–86  
**Approximation quality**, measure of, 56  
**Approximation theory**, 55–56  
**Approximators**, 7, 8  
 universal, 196–205, 206  
 Arbib, M. A., 26, 29  
**Architectures.** *See also specific type*  
 classification of, 74–75  
 data representation vs., 214–16  
 for forward kinematics problem, 519–23  
 optimization of, 218–20  
 Aronhime, P. B., 464, 466, 470  
**ART1** (adaptive resonance theory)  
 1) network, 432–44  
 Arteaga-Bravo, F. J., 518, 519  
**Artificial neural systems development**  
 future of, 21–22  
 history of, 17–20  
 working models of, 566–85  
**Artificial neurons**, 2. *See also Neurons*  
**Associative memories**, 313–86.  
*See also* Autoassociative memory; Heteroassociative memory  
 applications of, 377  
 basic concepts relating to, 314–20  
 bidirectional, 354–70. *See also* Bidirectional associative
 memory
- memory capabilities of, 313  
 implementations of, 644–52  
 linear, 76, 320–25  
 mode of addressing in, 317  
 models of, 317  
 prototype vectors and, 315–16  
 quality criteria for design of, 376  
 research, 19–20  
 of spatio-temporal patterns, 370–75  
**Asynchronous stochastic recursion**, 258–61  
**Asynchronous updating**, 257–61  
 flip-flop network and, 334  
 storage algorithm and, 329  
**Attenuator**, programmable, 598  
**Attraction**  
 basin of, 253  
 radius of, 344  
**Attractor**, 44, 253  
**Attributes**, semantic maps and, 541–42, 543, 544  
**Augmentation**, dummy, 366–68  
**Augmented pattern vector**, 109–10  
**Autoassociative memory**, 53, 76, 316. *See also* Associative memories  
**Hopfield**, 325–27. *See also* Hopfield autoassociative memory  
**recurrent**, 325–54. *See also* Recurrent autoassociative memory  
**Autoassociative memory chip**, 645, 647–48  
**Autocorrelation matrix**, 324, 328  
**Autocorrelator**, 324  
**Automaton**, 43  
**Automobile engine fault diagnosis**, connectionist expert system for, 228–29  
**Autonomous drivers**, 8–10, 76  
**Autonomous Land Vehicle In a Neural Network (ALVINN)**, 8–10  
**Axon**, 27, 29

- Back pain diagnosis, expert system for, 532–36
- Back-propagation training. *See* Error back-propagation training
- Baird, H. S., 476
- Barto, A. G., 490, 499, 511, 513, 548
- Basin of attraction, 253
- Batch learning, 56
- Baxt, W. G., 537, 538
- Bayesian approach to automated diagnosis systems, 527–28
- Bekey, G. A., 547
- Bell, S. B., 614
- Bengio, Y. R., 220
- Berger, R., 600, 601, 602
- Bergstresser, P. R., 528, 531
- Bessiere, P., 572, 665
- Biased neuron, 143
- Bibyk, S., 626, 636, 640, 642, 643, 665
- Bidirectional associative memory, 76, 354–70  
architecture of, 355–57, 358  
capacity of, 363  
encoding and decoding in, 357–59  
generalization to enable multiple associations, 368–70  
improved coding with, 363–68  
spatio-temporal patterns and, 370–75  
stability considerations with, 359–60
- Bidirectionally stable memory, 359–60
- Binary activation functions, 36  
bipolar, 33, 39  
Hebbian learning rule and, 61–63  
perceptron learning rule and, 64–66  
unipolar, 33–34
- Binary adder network, 287–94
- Binary patterns, restoration of, 13
- Binary perceptrons, 35, 36
- Binary weights, in associative memory implementation, 645–48
- Biological neurons, 26–36  
firing of, 29  
McCulloch-Pitts model of, 30–31  
modeling for artificial neural systems, 31–36  
structure of, 27–28  
threshold of, 29  
time units for modeling of, 29
- Biomedical signal processing, classifiers and, 6–7
- Bipolar activation functions  
biological neuron model, 33, 34, 36  
feedforward network, 39  
Hebbian learning rule and, 63  
perceptron learning rule and, 64
- Bipolar heteroassociative memory, 367
- Bipolar threshold function, bidirectional associative memory and, 355
- Bistable flip-flop, 330–36
- Blackman, D., 653, 658, 660
- Boolean functions. *See also* XOR function  
template matching network and, 628
- Borgstrom, T. H., 642, 643
- Boser, B., 476, 477, 482, 483, 484
- Bounds, D. G., 532, 533, 534, 535, 536
- BPOS network, forward kinematics problem and, 520–21, 522
- Brady, D., 665
- Brain  
capacity of, 375–76  
neurons in, 26–36  
“Brain language,” 18
- Braun, L., 487
- Brobst, R. W., 528, 531
- Broomstick balancing problem, 499–503
- Bryson, A. E., 20
- Buffer unit, neuron cell with, 645, 647
- Building block circuits, 609–30
- Burr, D., 221
- Calibration, 399
- Calvert, T. W., 153
- Campagna, D. S., 504, 508, 509, 510, 512
- Cao, W., 217
- Capacitance, in electrical model of Hopfield network, 266
- Capacitance plant, transfer function for, 486, 487
- Capacitive circuit element, storage of charge across, 592–95
- Capacitor, natural decay of charge, 594
- Card, H. C., 605, 606
- Cardio, R., 220
- Carpenter, G., 20, 432
- Cart-pendulum controller, 499–503
- Cartesian space partitioning, three-dimensional, 4–5
- Cascode current mirror, 612, 613
- Categories, and counterparts in languages, 540–41
- Category membership, classifiers and, 4–7. *See also* Classification; Classifiers
- Central processing unit (CPU), 569
- Cerebellar model articulation controller (CMAC), 489, 504–11, 512
- Chams, A., 572, 665
- Chandra, V., 464
- Channel length modulation factor, current mirrors and, 610–11, 612
- Channel resistance, 588
- Chapman, 418
- Character recognition networks, 221–25, 464–85  
based on error back-propagation training, 482–85  
based on handwritten character skeletonization, 478–82  
multilayer feedforward, 221–25, 464–76

- problem with handwritten digits in, 476–78
- Characteristic surface, 504–6, 507, 508
- Characteristics, invariant vs. variable, 569–70
- Chen, H. H., 377
- Chevroulet, M., 592, 594
- Chin, B. L., 653, 660, 661, 663
- Chin, C., 299
- Chip circuitry. *See also* Integrated circuits
- associative memory, 645, 647–51
  - ETANN 80170, 653–57
  - learning and, 664
  - MDAC, 600–603, 604
- Chip-in-the-loop learning, 653
- Chiu, T., 653, 658, 660
- Choi, M. R., 665
- Chua, L. O., 271, 299
- Chung, H. W., 628
- Circuit(s)
- associative memory, 644–52.
  - See also* Associative memories
  - capacitive, 592–95
  - electronic, 575–85
  - integrated, 585–608, 609. *See also* Integrated circuits
  - MDAC, 595–604
  - sample-and-hold, 593–95
  - scalar product. *See* Scalar product circuits
- Clare, T., 653, 658, 660
- Classes, 94
- Classical conditioning, 76, 77
- Classification, 53–54, 94–95. *See also* Categories; Clustering
- correlation, 109
  - discrete perceptron in, 120–32
  - geometric description of, 97–98
  - linearly nonseparable, 165–75
  - minimum distance, 108–14
  - multilayer feedforward networks in, 220–25
  - unsupervised, 58
- Classification error, minimization of, 133–43
- Classification response, 54
- Classifiers, 4–7, 391–98. *See also* Classification; specific type
- Closed-loop feedback system, in target position learning, 524
- Cluster discovery network, 432–44
- Clustering, similarity measures and, 399–401
- Clustering networks, 16, 76, 399–410
- design of neural network for, 581–85
  - initialization of weights in, 406–7
  - Kohonen, 401–5
  - recall mode and, 406
  - separability limitations of, 409–10
- Clusters, unsupervised learning of, 58, 399–410
- CMAC (cerebellar model articulation controller), 489, 504–11, 512
- CMOS current comparator, 626–28
- CMOS inverter circuit, 613–17, 618
- CMOS technology, 608, 609
- analog multiplier with weight storage and, 640
  - associative memory and, 645, 650, 651
- Cocontent function, 271
- Coding. *See* Encoding and decoding
- Collins, D. R., 664, 665
- Combinational optimization tasks, 16, 294–99
- “Committee” networks, 173. *See also* Multilayer feedforward networks
- Communication traffic, routing of, 298
- Competitive learning. *See also* Clustering networks
- leaky, 405
- winner-take-all learning rule and, 70–71
- Complement, of stored memory, 327
- Computation
- analog, 575–80
  - conventional vs. neural network work, 77
  - mixed, 567
  - by neural networks, 2. *See also* Neural computation
- Computational bandwidth, 571, 572
- Computational energy function. *See* Energy function
- Computers. *See* Hardware
- Computing nodes, 569
- Conductance
- in electrical model of Hopfield network, 266
  - photosynapse and, 661
- Conductance matrix, 289
- Connection matrix. *See* Weight matrix
- Connection strength, 2
- Connectionist expert systems, 226–29, 527–39
- for coronary occlusion diagnosis, 537–39
  - for low back pain diagnosis, 532–36
  - for skin disease diagnosis, 528–32
- Connections per second (CPS), 571, 572–73
- Constrained freedom pattern data, 415–16
- Constraint amplifiers, 459
- Content-addressable memories, 317
- Context, semantic maps and, 543, 545–46
- Continuous activation functions
- biological neuron model, 33–34
  - bipolar, 33, 34
  - delta learning rule and, 66–69
- feedforward network, 41, 42

- Hebbian learning rule and, 63  
steepness factor of, 209–10  
unipolar, 33–34
- Continuous function approximation, 204, 206  
counterpropagation in, 414
- Continuous function approximation error, 203
- Continuous perceptrons, 35, 36, 76  
multilayer networks for linearly nonseparable classifications, 170, 173  
R-category linear classifier using, 151  
single-layer networks for linearly separable classifications, 132–42, 175  
training rule of, 136
- Continuous-time feedback network, 47, 252, 264–76. *See also* Single-layer feedback networks  
electric, 49–52  
elementary synaptic connection within, 48–49  
optimization problems and, 287–99  
transient response of, 276–83
- Continuous-time learning, 59
- Continuous units. *See also* Neurons  
classifiers and, 5–7  
function approximation and, 7, 8
- Continuous-valued function, staircase approximation of, 197–200
- Control, inverse, 487–88
- Control configuration, open-loop, 487–88
- Control systems  
applications of, 485–513  
CMAC, 504–11, 512  
dynamic back-propagation and, 513  
Intellexer 605T and, 524  
inverted pendulum neurocontroller, 499–503
- learning, 494–99. *See also* Learning control systems  
process identification in, 489–94
- Conventional computers, parallel distributed processing hardware vs., 568–69
- Converter output voltage, 586
- Convex combination, 406–7
- Coprocessor accelerator boards, 566
- Coronary occlusion diagnosis, expert system for, 537–39
- Correction increment, 118, 120–23
- Correction rules, 122–23
- Correlation classification, 109
- Correlation learning rule, 69–70, 73
- Correlation term, Hebbian learning rule, 61
- Cosi, P., 220
- Cost function, linear programming and, 457, 463
- Cotter, N., 547
- Counterpropagation network, 76, 410–14  
trajectory mapping with, 521–23
- CPS units, 571, 572–73
- CPU (central processing unit), 569
- Craig, J. J., 514, 515
- Cross-correlation matrix, 322–24
- Cross-talk noise, 323
- Cruz, J. B., 365, 366, 367, 368
- Cumulative error, 195–96
- Cumulative weight adjustment, incremental updating vs., 208–9
- Current comparator, 626–28
- Current mirrors, 610–13  
differential voltage amplifiers and, 621–22, 623
- Current summation, in associative memory design, 648, 649
- Current-to-voltage converter, 586
- ladder-resistor MDAC, 598  
weighted-resistor MDAC, 596
- Cybenko, G., 220, 222
- Cybernetics, 19
- D/A conversion. *See* Digital-to-analog conversion
- DARPA, 571
- Data, dimensionality reduction of, 16–17
- Data clusters. *See* Cluster discovery network; Clustering networks; Clusters
- Data representation, network architectures vs., 214–16
- Data transfer, speed of, 572
- Davenport, M. R., 377
- Dayhoff, J. E., 28, 464
- Dayhoff, R. E., 464
- Decision error, 196
- Decision hyperplane  
linearly nonseparable pattern classification, 165–66  
single-layer perceptron classifier, 116–17
- Decision regions, 98
- Decision surfaces, 98, 101  
discrete perceptron classifier training, 125, 126  
discriminant functions and, 105
- Decoding. *See* Encoding and decoding
- Dedicated VLSI chips, 658, 660
- Delay network, 47–49. *See also* Time-delay network
- Delta, error signal term, 177, 178
- Delta function, Kronecker, 297, 328
- Delta learning rule, 66–69, 73, 77  
continuous perceptron and, 137–42  
generalized, 181–85  
multiperceptron layer and, 175–81  
Widrow-Hoff learning rule as special case of, 69
- Delta training/learning weight adjustments, 178

- DeMori, R., 220  
 Dendrites, 27  
 Denker, J. S., 476, 477, 478, 479, 480, 481, 482, 483, 484, 547, 569, 665  
 Depletion MOSFET circuit, 631–36  
 Dermatological diagnosis, expert system for, 528–32  
 Desai, M. S., 345, 464, 466, 470  
 DESKNET expert system, 528–32  
 Deweerth, S. P., 665  
 Dichotomization, triple, 150  
 Dichotomizer, 104–5. *See also* Single-layer perceptron classifiers  
     linear discriminant function and, 107–8  
     TLU-based perceptron, 115  
     weight adjustments of, 117–20  
 Dichotomizer training  
     absolute correction rule and, 128  
     fixed correction rule and, 126–27  
 Differential current  
     ETANN 80170 chip, 655, 656, 657  
     floating-gate transistor multipliers, 642–43  
 Differential voltage amplifiers, 617–24  
 Digital electronic neurocomputing circuits, 575–80  
 Digital outputs, summing network with, 287–94  
 Digital-to-analog (D/A) conversion  
     analog processor and, 658  
     multiplying converter, 595–604  
 Digitally programmable weights, 595–604  
 Digits, handwritten recognition, 476–78  
 Dijkstra, E., 592, 594  
 Dimensionality reduction, feature detection and, 16–17, 415–16  
 Discrete perceptrons, 35, 36, 76  
     layered networks of, 170, 172, 173  
     mapping in linearly nonseparable classification, 170, 172  
     R-category classifier training, 144–46  
     training and classification using, 120–32  
 Discrete-time bidirectional associative memory, 358. *See also* Bidirectional associative memory  
 Discrete-time feedback networks, 42, 43, 44–49, 252, 254–64  
 Discrete-time Hopfield networks, mathematical foundations of, 254–64  
 Discriminant functions, 99–106  
     contour map of, 102  
     linear, 107, 108, 111–13  
     multicategory classifiers and, 147  
 Disease diagnosis. *See* Connectionist expert systems  
 Distance function, as approximation quality measure, 56  
 Distributed processing hardware, efficiency of, 567–69  
 Distributed representation, 216  
     R-category discrete perceptron classifier training and, 145–46  
 Dobbins, R. W., 6, 547  
 Donham, C., 653, 658, 660  
 Dosdall, J., 228  
 Double-ended output differential-input amplifier, 620, 621–22  
 Dow, R. J. F., 220  
 DRAM network, 664–65  
 Dreyfus, G., 353  
 Dreyfus, S., 20  
 Drivers, autonomous, 8–10, 76  
 Duda, R., 401  
 Dummy augmentation, 366–68  
 Dutta, S., 548  
 Dynamic error back-propagation, 513  
 Dynamic memory networks, 317.  
     *See also* Associative memories  
 Dynamical plants  
     learning control architecture and, 495  
     transfer functions of, 486–87  
 Dynamical systems, 251–310.  
     *See also* Single-layer feedback networks  
     basic concepts of, 253–54  
 Dynamics, 2  
     of continuous-time two-bit A/D converter, 278–83  
 Eberhart, R. C., 6, 547  
 EBPTA (error back-propagation training algorithm), 187, 188–90  
 EEG. *See* Electroencephalogram  
 Efficiency measures, 572–73  
 Electric continuous-time network, 49–52  
 Electrical components, gradient-type network using, 265–67  
 Electrically controlled weights, 587–92, 605–8  
 Electroencephalogram (EEG), 76  
     classifiers and, 6–7  
 Electronic networks  
     flip-flop, 330–36  
     implementations of, 652–62, 663  
     linear programming, 461–63  
 Electronic neurocomputing circuits, 575–85  
 Electronically Trainable Analog Neural Network (ETANN)  
     80170 chip, 653–57  
 Elementary delay network, 47–49  
 Elementary nerve cell, 27. *See also* Biological neurons  
 Elementary synaptic connection, feedback network, 48–49  
 Embedded generalization, learning with, 506–7

- Encoding and decoding  
bidirectional associative memory, 357–59, 363–68  
multiple training, 365–68
- Energy function  
discrete-time Hopfield network, 258–60  
flip-flop network, 332, 333  
in four-bit A/D conversion problem, 288–89  
gradient-type Hopfield network, 268–76
- Hopfield autoassociative memory, 327–28
- recurrent autoassociative memory, 327–28, 340, 342–43
- temporal memory, 373–74
- time derivative of, 461
- in traveling salesman problem, 296–99
- in two-bit A/D conversion problem, 15, 278–83
- Energy minimization, 15  
bidirectional associative memory and, 359
- continuous-time network and, 278–83
- gradient-type Hopfield network, 269–70
- linear programming and, 457–64
- recurrent autoassociative memory and, 342–43
- Engineering, optimization in, 14
- Enhancement mode MOS circuit, 636–38
- Equilibria encoding, 55–56
- Error(s)  
continuous function approximation, 203  
cumulative, 195  
decision, 196  
finite resolution and conversion, 595  
root-mean-square normalized, 196  
training, 195–96
- Error back-propagation training, 175, 186–95
- in character recognition, 222–25, 466–76, 482–85
- dynamic, 513
- example of, 190–95
- expert systems using, 226–29
- forward kinematics problem and, 520–21, 522
- learning control systems and, 497–98
- with output splitting, 520–21, 522
- photosynapses and, 662, 663
- target position learning and, 523–27
- training errors and, 195–96
- Error back-propagation training algorithm (EBPTA), 187, 188–90
- convergence of, 207–13
- difficulties with implementation of, 205, 207
- learning constant and, 210–11
- Error minimization, 15  
continuous perceptron training using, 133–42
- local minimum, 205, 207
- Error signal term  $\delta$ , 177, 178
- error back-propagation and, 193
- ETANN 80170 chip, 653–57
- Excitatory impulses, 29
- Expert systems, 21  
connectionist, 226–29, 527–39. *See also* Connectionist expert systems  
multilayer feedforward networks and, 225–29
- Explanation capability  
in DESKNET expert system, 529–32
- limitations in dynamical systems, 254
- Farhat, N. H., 665
- Farrell, J. A., 353
- Feature array, 416
- Feature extraction, 95–96. *See also* Character recognition networks
- self-organizing semantic maps and, 539–46
- Feature mapping, 16–17, 19–20, 414–32
- self-organization of, 423–32
- Features, 95–96
- Feedback  
closed-loop, 524
- learning supervision and, 56–58
- Feedback connections, 2
- Feedback control. *See also* Control systems  
inverse, 488
- Feedback networks, 42–52  
automaton, 43
- continuous-time, 47, 48–52
- discrete-time, 42, 43, 44–49
- electric continuous-time, 49–52
- interconnection scheme of, 42–43
- learning in, 56
- recurrent, 42, 43, 44–49
- single-layer, 251–310. *See also* Single-layer feedback networks
- state transitions, 43–44
- Feedforward control. *See also* Control systems  
inverse, 488
- Feedforward learning, Hebbian learning rule and, 60
- Feedforward networks, 37–42  
associative memory, 318
- CMAC, 508–9. *See also* Cerebellar model articulation controller (CMAC)
- continuous activation function, 41
- counterpropagation, 410–14
- CPS vs. FLOPS units for, 573
- discrete activation function, 40
- forward kinematics problem and, 519–23
- Hamming, 391–98
- interconnection scheme of, 37, 38

multilayer, 163–248. *See also* Multilayer feedforward networks  
 two-layer, 39–42  
 Feedforward recall, 185–86  
 Financial risk evaluation, 571,  
 572  
 Finite resolution and conversion  
 error, 595  
 Firing rule, for McCulloch-Pitts  
 neuron model, 30–31  
 Fisher, W. A., 653, 657, 658, 659  
 Fixed correction rule, 123,  
 124–26  
 dichotomizer training and,  
 126–27  
 Fixed point  
 attractor, 253  
 recurrent autoassociative mem-  
 ory and, 349–54  
 Flip-flop network, 330–36  
 Floating-gate transistor multipli-  
 ers, 640–43  
 Floating-point representation, 572  
 FLOPS (floating-point operations  
 per second) measure, CPS  
 measure vs., 573  
 Follower-aggregation circuit,  
 625–26  
 Foo, S. Y., 665  
 Foo, Y. P. S., 16, 298  
 Forward kinematic equation, 515  
 Forward kinematics problem, 515  
 architecture comparison for,  
 519–23  
 solution of, 516–19  
 Forward plant identification,  
 489–90  
 Four-dimensional hypercube,  
 44–45  
 Fractional correction rule, 122  
 Frye, R. C., 653, 660, 661, 663  
 Fu, K. S., 502  
 Fujimoto, R. J., 653, 657, 658,  
 659  
 Fukushima, K., 19  
 Funanashi, K. I., 200  
 Function approximation, 7, 8. *See*  
*also* Approximation

Functional link networks,  
 230–33, 234  
 forward kinematics problem  
 and, 520–21, 522  
 functional model of, 232–33  
 hidden layer networks vs., 234  
 Gallant, S. I., 226  
 Gate capacitance, weight storage  
 as voltage across, 592–95  
 Geiger, R. L., 605, 610, 612,  
 622, 665  
 General learning rule, 59  
 General neuron symbol, 32–33  
 General vector field expression,  
 277  
 Generalization, 54  
 CMAC and, 506–7  
 Generalized delta learning rule,  
 181–85  
 Gilbert, S., 600, 601, 602  
 Girosi, F., 55, 56, 200, 548  
 Gonzalez, R. C., 153, 401  
 Gorman, R., 220, 547  
 Goser, K., 575, 665  
 Grabel, A., 610, 612  
 Gradient descent-based train-  
 ing, 132–42. *See also* Error  
 back-propagation training  
 single-layer continuous percep-  
 trons and, 151  
 Gradient-type Hopfield networks,  
 mathematical foundations of,  
 264–76  
 Gradient-type networks, 76, 264–  
 76. *See also* Single-layer  
 feedback networks  
 Gradient vector, 135–36, A13–14  
 Graf, H. P., 254, 476, 478, 479,  
 480, 481, 569, 645, 647,  
 662, 665  
 Grant, P. M., 377  
 Grossberg, S., 20, 71, 412  
 Grossberg layer of counterprop-  
 agation network, 412, 413,  
 521  
 Gruber, S., 548  
 Gu, S. G., 665  
 Guerin, A., 572, 574, 665  
 Guyon, I., 353  
 Hagiwara, M., 368, 370  
 Hall, 418  
 Hamming distance, 339, 341, A39  
 bidirectional associative mem-  
 ory and, 359, 362–63  
 dummy augmentation and, 366  
 energy function and, 342  
 MDAC and, 601–3, 604  
 memory capacity and, 344  
 memory convergence and, 345,  
 346, 347  
 Hamming network, 76, 391–98  
 Handwritten characters. *See also*  
 Character recognition net-  
 works  
 digits, 476–78  
 recognition of, 221–25, 482–  
 85  
 skeletonization of, 478–82  
 Harashima, F., 547  
 Hard-limiting activation func-  
 tions, 34, 35  
 Hardware, 565–678  
 active network building blocks,  
 608–30  
 actual models of artificial neu-  
 ral systems, 566–85  
 analog multipliers and scalar  
 product circuits, 630–43  
 associative memory implemen-  
 tation, 644–52  
 complexity related to node  
 numbers, 567–69, 570  
 electronic neural processors,  
 652–62, 663  
 electronic neurocomputing cir-  
 cuits, 575–85  
 integrated circuit synaptic con-  
 nections, 585–608, 609. *See*  
*also* Weight(s)  
 requirements for, 569–75, 664  
 two approaches to, 565–66,  
 570  
 weights. *See* Weight(s)  
 Hario, Y., 665  
 Hart, P., 401

Hashimoto, H., 547  
 Hasler, M., 261  
 Hassoun, M. H., 376, 377  
 Heart disease diagnosis, expert system for, 537–39  
 Hebb, D. O., 18, 60, 61  
 Hebbian learning rule, 18, 60–63, 73, 77  
 autocorrelation and, 324  
 bidirectional associative memory and, 357–58  
 correlation learning rule as special case of, 70  
 example of, 61–63  
 linear associative memory and, 322, 324  
 recurrent autoassociative memory and, 352–54  
 weights learning according to, 605–8, 609  
 Hecht-Nielsen, R., 36, 70, 406, 407, 410, 414, 513  
 Henderson, D., 476, 477, 478, 479, 480, 481, 482, 483, 484  
 Herault, J., 572, 574, 665  
 Hessian matrix, 272, 280, A16–17  
 Heteroassociative memory, 53, 316. *See also* Associative memories  
 bipolar, 367  
 classification and, 54  
 recurrent, 318  
 Heuristic capacity measure, bidirectional associative memory, 363  
 Hidden layer(s), 182–83, 193  
 forward kinematics problem and, 520  
 size of, 216–18, 224, 225  
 Hidden layer network  
 character recognition and, 467–76  
 functional link network vs., 234  
 Hilleringmann, U., 575, 665  
 History, of artificial neural systems development, 17–20

History sensitivity, networks with, 390–452. *See also* specific type  
 Ho, Y. C., 20  
 Ho-Kashyap algorithm, 377  
 Hoff, M. E., Jr., 19  
 Hoffman, G. W., 377  
 Holberg, D. R., 605, 610  
 Hollis, P. W., 665  
 Hopfield, J. J., 20, 253, 255, 266, 273, 295, 296, 297, 298, 325, 457, 458, 459, 461, 463, 586  
 Hopfield autoassociative memory, 317, 318, 325–27. *See also* Recurrent autoassociative memory  
 capacity of, 344–45  
 convergence vs. corruption of, 345–48  
 energy function for, 327–28  
 output update rule for, 327  
 Hopfield networks  
 discrete-time, 254–64  
 gradient-type, 264–76  
 Hornik, K., 200  
 Howard, R. E., 254, 476, 477, 482, 483, 484  
 Hripcsak, G., 226  
 Hsieh, T. P., 653, 658, 660  
 Hsu, K., 665  
 Hu, V., 254  
 Hubbard, W., 476, 477, 478, 479, 480, 481, 482, 483, 484  
 Hypercube, four-dimensional, 44–45  
 Iberall, T., 547  
*IEEE Computer*, 547  
*IEEE Transactions on Circuits and Systems*, 547  
 Image space, mapping pattern space into, 165, 168–70, 171  
 Images  
 autonomous driver and, 8–9  
 pattern restoration and, 13  
 photosynapse and, 661  
 Implicit knowledge, connectionist expert systems and, 528  
 Incremental updating, cumulative weight adjustment vs., 208–9  
 Indirect learning architecture, 495–99  
 Inhibitory impulses, 29  
 Initial weights, 208  
 clustering and, 406–7  
 Input  
 differential, 617–24  
 functional link networks and, 230–33  
 instantaneous response to, 4–10  
 Input currents, 289  
 current mirrors and, 610–13  
 differential. *See* Differential current  
 Input/output bandwidth, 571, 572, 573  
 Input pattern. *See* Input vectors  
 Input vectors  
 associations of, 53  
 biological neuron model, 32  
 feedforward network, 37  
 Input voltage. *See* Voltage  
 Integrated circuits. *See also* specific type  
 amplifiers. *See* Amplifiers  
 building block, 608–30  
 current comparator, 626–28  
 current mirrors, 610–13  
 ETANN 80170 chip, 653–57  
 follower-aggregation, 625–26  
 inverter, 613–17, 618  
 present capabilities of, 664–65  
 synaptic connections, 585–608, 609. *See also* Weight(s)  
 voltage follower, 624–25  
 Intel Corporation, ETANN 80170 chip, 653–57  
 Intelledex 605T, 524  
 Interconnection network, optimal routing of calls through, 298  
 Internal layers. *See* Hidden layer(s)  
 Invariant characteristics, 569  
 Inverse control, 487–88

- Inverse kinematics problem, 514, 515  
solution of, 516–19
- Inverse plant identification, 490–94
- Inverted pendulum neurocontroller, 499–503
- Inverter-based neuron, 613–17, 618
- Inverting neurons, 276
- Ismail, M., 626, 631, 636, 640, 642, 643, 665
- Isobe, M., 547
- Isolated words recognition, 571
- Jackel, L. D., 254, 476, 477, 478, 479, 480, 481, 482, 483, 484
- Jacobian matrix, 498–99, A16
- Jacobs, R. A., 213
- James, J., 228
- Jespers, P. G. A., 648, 665
- Job-shop scheduling, 16, 298
- Joint angles, in robot control, 514, 515. *See also* Robot kinematics
- Jones, D., 539
- Jones, R. S., 19
- Jutten, C. A., 572, 574, 665
- Kamp, Y., 261
- Kang, M. J., 289
- Kang, S. M., 299
- Karnin, E. D., 218
- Kaul, R., 665
- Kawato, M., 547
- Kelley, H. J., 20
- Kennedy, M. P., 271, 299
- Kenney, J. B., 298
- Key pattern, 313. *See also* Stored memory
- Khachab, N. L., 631, 636
- Khanna, S. K., 603, 648, 651
- Khorasani, K., 519, 520, 521, 522
- Kinematics, robot, 513–27
- Knowledge, implicit, 528
- Kohonen, T., 17, 19, 20, 322, 324, 401, 418, 420, 423, 424, 425, 426, 429, 430, 431, 540, 542, 544, 545, 546
- Kohonen layer of counterpropagation network, 411–14, 521
- Kohonen network, 401–5  
separability and, 410
- Komlos, J., 343
- Kosko, B., 354, 363, 374
- Kraft, L. G., 504, 508, 509, 510, 512
- Kreutzer, I., 665
- Kronecker delta function, 297, 328
- Kub, F. J., 638, 640
- Kubota, T., 547
- Kudon, M., 547
- Ladder-resistor MDAC, 597, 598, 600
- Lambe, J., 665
- Langenbacher, H., 603, 648, 651
- Language. *See also* Speech features  
semantic relationships within, 539–46
- Laplace transform, 487
- Laser range finder, in ALVINN, 8
- Latent summation period, 28
- Lateral interaction, Mexican hat type of, 419, 425
- Lateral interaction coefficient, MAXNET, 394
- Lawson, J. C., 572, 665
- Layered networks, 19, 20, 164.  
*See also* Multilayer feedforward networks
- Leaky competitive learning, 405
- Learning. *See also* Training  
adaptation and, 55–58  
as approximation, 55–56  
batch, 56  
chip design and, 664  
chip-in-the-loop, 653  
CMAC, 506–11  
competitive, 70–71  
continuous-time, 59  
of dynamical systems, 254
- generalization of. *See* Generalization
- leaky competitive, 405
- off-chip, 653
- supervised, 56–57. *See also* Supervised learning  
target position, 523–27  
theory of, 76–77  
unsupervised, 57–58, 399–410, 506
- Learning constant, 59, 210–11  
gradient procedure and, 133
- Learning control systems, 489
- error back-propagation and, 497–98  
nondynamic, 494–99
- Learning dichotomizer. *See* Dichotomizer
- Learning factors, 205, 207–20  
cumulative weight adjustment vs. incremental updating, 208–9
- initial weights and, 208
- momentum method and, 211–13
- network architectures vs. data representation, 214–16
- steepness of activation function, 209–10
- Learning profiles  
hidden layer size and, 224, 225. *See also* Hidden layer(s)
- for inverted pendulum task, 502, 503
- for learning control architectures, 496–97
- Learning rate, of functional link network vs. hidden layer network, 232, 234
- Learning rules, 2–3, 20, 59–74, 77. *See also* specific rules  
correlation, 69–70  
delta, 66–69  
general, 59  
Hebbian, 18, 60–63  
outstar, 71–72, 413  
perceptron, 64–66  
projection, 353–54  
Widrow-Hoff, 18, 69

- winner-take-all, 70–71, 401–5
- Learning signal, 59  
delta learning rule, 66  
Hebbian learning rule, 60  
perceptron learning rule, 64  
Widrow-Hoff learning rule, 69
- Learning time sequences, 229–30
- Learning weights, 586. *See also* Weight(s)  
implementation of, 605–8, 609
- Least mean square (LMS) learning rule, 19, 69, 73
- LeCun, Y., 476, 477, 482, 483, 484
- Lee, B. W., 650
- Lee, H. Y., 377
- Lee, Y. C., 377
- Leung, H. C., 220
- Liapunov function, 259, 460, A25–29
- Limit cycle, 253
- Linear associative memory, 76, 320–25
- Linear classifiers, 106–14  
continuous perceptron and, 144–46  
R-category, 151. *See also* R-category entries
- Linear discriminant function, 107, 108, 111–13
- Linear machines, 109
- Linear mapping, nonlinear mapping followed by, 200
- Linear programming, single-layer networks in, 299
- Linear programming modeling network, 456–64
- Linear programming problem, 457
- Linearly nonseparable pattern classification, 165–75
- Linearly separable patterns, 113–14  
single-layer continuous perceptron networks for, 132–42
- Lippmann, R. P., 391, 435, 441, 547
- Liu, H., 547
- Lloyd, P. J., 532, 533, 534, hfill
- 535, 536
- LMS (least mean square) learning rule, 19, 69, 73
- Local minimum, 205, 207  
recurrent autoassociative memory and, 342
- Local representation, R-category discrete perceptron classifier training and, 145
- Loinaz, M., 653, 658, 660
- Long, F. M., 638, 640
- Long channel, 589
- Long-term network memory, ART1, 437
- Low back pain diagnosis, expert system for, 532–36
- Low level vision, 571, 572
- Lu, J. J., 523, 524, 525, 526, 527
- Lu, T., 665
- Maa, C. Y., 299
- Mack, I. A., 638, 640
- Mackie, S., 569, 665
- Maher, M. A., 592, 594, 665
- Mahowald, M. A., 665
- Mann, J. R., 600, 601, 602, 665
- Mapping(s)  
approximators and, 7, 8  
classifiers and, 4–7  
feature, 414–32. *See also* Feature mapping  
feedforward, 38–39  
multilayer feedforward network, 164–65, 170, 171. *See also* Multilayer feedforward networks  
noise-free, 322  
nonlinear followed by linear, 200  
one-to-one vector-to-cluster, 406  
of plant inverse, 490–94  
semantic, 539–46  
sequence of, 523  
supervised learning system, 151–52  
trajectory, 521–23  
transition, 332–33  
triangular, 350, 351, 352
- Marko, K. A., 228
- Matching, ART1 and, 435. *See also* Cluster discovery network
- Mathematical modeling, function approximation in, 7
- Mathew, B., 532, 533, 534, 535, 536
- Maximum error, character recognition networks and, 466
- Maximum selector, replacement with discrete perceptrons, 144–46
- MAXNET, 76, 391–98  
ART1 and, 433–34  
MDAC chip and, 603, 604
- McClelland, T. L., 20, 68, 168, 208
- McCulloch, W. S., 18, 30, 31
- McCulloch-Pitts neuron model, 30–31
- McEliece, R. J., 343, 344, 345
- MCPTA (multicategory continuous perceptron training algorithm), 180–81
- MDAC. *See* Multiplying digital-to-analog converter
- Mead, C., 625, 626, 665
- Medical diagnosis, connectionist expert systems for, 226–27, 527–39. *See also* Connectionist expert systems
- Melsa, P. J., 298
- Memory  
associative, 313–86. *See also* Associative memories  
autoassociative. *See* Autoassociative memory  
human, 375–76  
long-term, 437  
simple, 10–13  
size of, 571, 572
- Memory cell, McCulloch-Pitts neuron model and, 30, 31
- Mendel, J. M., 547
- Mexican hat type of lateral interaction, 419, 425
- MFLOPS measure, 571, 572, 574

- Michandini, G., 217  
 Michel, A. N., 353  
 Miller, W. T., III, 547  
 Million of floating-point operations per second (MFLOPS), 571, 572, 574  
 Million of instructions per second (MIPS), 572  
 Millman, J., 610, 612  
 Minimum distance classification, 108–14  
     maximum selector replacement with discrete perceptrons and, 144–46  
 Minimum Hamming distance classifier, 391. *See also* Hamming network  
 Minimum spanning tree, 426–27  
 Minsky, M., 18, 19  
 MIPS measure, 572  
 Mirror output/input current ratio, 610  
 Mirrors  
     adaptive, 658  
     current. *See* Current mirrors  
 Mishkin, E., 487  
 Miyaka, S., 19  
 Models, 37–52. *See also specific models*  
     working, 566–85  
 Modified Wilson current mirror, 612, 613  
 Momentum method, 211–13  
 Momentum term, 211  
 Monotone increasing function, 461  
 Moon, K. K., 638, 640  
 Moopenn, A., 603, 648, 651, 665  
 Moore, W. R., 605, 606  
 Moore-Penrose pseudoinverse matrix, 353  
 Morton, S. G., 665  
 MOS circuit, enhancement mode, 636–38  
 MOS transistors. *See also* CMOS entries; NMOS transistors; PMOS transistors  
     depletion mode, 631–36  
     floating-gate, 640–43
- MOSFET circuit  
     associative memory and, 648, 651  
     depletion, 631–36  
     modeling, A39–43  
 Mueller, P. J., 653, 658, 660  
 Mulligan, J. H., 365, 366, 367, 368  
 Multicategory continuous perceptron training algorithm (MCPTA), 180–81  
 Multicategory single-layer perceptron networks, 142–52  
 Multidirectional associative memory, 76, 368–70  
 Multilayer feedforward networks, 163–248. *See also* Feed-forward networks; Layered networks  
     character recognition application of, 221–25, 464–76  
     classifying, 220–25  
     computational requirements for, 573–74  
     CPS unit and, 573  
     delta learning rule and, 175–81  
     error back-propagation training and, 186–95  
     expert, 225–29, 537–39  
     feedforward recall and, 185–86  
     in forward and inverse kinematics problems, 516–19  
     functional link, 230–33, 234, 520–21, 522  
     generalized delta learning rule and, 181–85  
     learning factors and, 205, 207–20  
     learning time sequences in, 229–30  
     linearly nonseparable pattern classification and, 165–75  
     photosynaptic connections in, 662, 663  
     pruning of, 218–20  
     in target position learning, 523–27  
     as universal approximators, 196–205, 206
- Multiperceptron layer, delta learning rule for, 175  
 Multiple association memory, 76, 368–70  
 Multiple encoding strategy, 365–68  
 Multiple-winner unsupervised learning, 407  
 Multipliers, analog, 605–8, 630–43  
 Multiply-accumulate sequence, 575  
 Multiplying digital-to-analog converter (MDAC), 595–604  
     ladder-resistor, 597, 598  
     weighted-resistor, 596–98  
 Murphy, J., 228  
 Murray, A. F., 665  
 Myocardial infarction diagnosis, expert system for, 537–39
- Nakamura, S., 665  
 NAND gate, using McCulloch-Pitts neuron model, 30, 31  
 Narendra, K. S., 186, 513  
 Natural similarity, 415  
 Negative gradient descent formula, for hidden layer, 183  
 Neighborhood generalization, CMAC and, 507  
 Neocognitrons, 19  
 Nerve cell, elementary, 27. *See also* Biological neurons  
 Nervous system, information flow in, 27  
 Nested nonlinear scheme, 200  
 NESTOR NDS-100 computer, connectionist expert system on, 229  
 Net signal, continuous perceptron and, 137  
 Networks. *See* Neural networks; specific type  
 Neural computation, 2–3  
     applications of, 3–17, 21–22  
     conventional computation vs., 77  
     examples of, 3–17

- Neural control, 485. *See also* Control systems
- Neural hardware, 565–678. *See also* Hardware; specific type
- Neural networks. *See also* specific type
- active building blocks of, 608–30
  - applications of, 546–48. *See also* specific applications
  - architectures of. *See* Architectures; specific type
  - classification of, 74–75, 76
  - clustering, 16, 76
  - defined, 37
  - feature detecting, 16–17, 19–20
  - feedback, 42–52
  - feedforward, 37–42
  - with history sensitivity, 390–452
  - ideal characteristics of, 664
  - implementation of. *See* Hardware
  - instantaneously responding to input, 4–10
  - literature on, 665
  - models of, 37–52. *See also* specific models
  - optimization of, 14–16
  - potential of, 1–2, 3, 21–22
  - recall times in, 571, 572
  - sizes of, 571, 572
- Neural processing, 53–55
- Neurocomputers, 565–66. *See also* Hardware
- simulated, 22
- Neurocontrol, 485. *See also* Control systems
- Neuroemulators. *See also* Hardware
- required features of, 570–75
- Neurology, EEG in, 6–7
- Neurons, 2. *See also* Continuous units
- biased, 143
  - biological, 26–36
  - general symbol of, 32–33
  - hidden. *See* Hidden layer(s)
  - inverter-based, 613–17, 618
  - inverting, 276
  - numbers of, and complexity of hardware, 567–69, 570
  - output signal of, 32
  - processing unit of, 32, 575.
  - See also* Processing nodes
- Newton-Raphson algorithm, 286
- Nguyen, D. H., 513, 547
- Nguyen, L., 519, 520, 521, 522
- Nilsson, N. J., 19, 130, 153, 164, 173
- NMOS transistors, 289
- associative memory and, 648–50, 652
  - in CMOS inverter circuit, 613–17
  - current mirrors with, 610–13
  - in depletion MOSFET circuit and, 631–36
  - differential voltage amplifier, 619–24
  - voltage-controlled weights and, 588–89
  - weight value storage across gate capacitance, 592–95
- Nodes, 2, 32. *See also* Neurons; Processing nodes
- Noise
- ART1 network and, 441–44
  - cross-talk, 323
- Noise-free mapping, 322
- Noise term vector, 321, 373
- Non-recurrent networks, 317
- Nonlinear mapping, linear mapping followed by, 200
- Nonlinear programming, single-layer networks in, 299
- Nonparametric training concept, 114–20
- discrete perceptron and, 120–32
- NOR gate, using McCulloch-Pitts neuron model, 30, 31
- Norris, D. E., 533
- Nys, O., 592, 594
- Off-chip learning, 653
- Omidvar, O. M., 547
- One-to-one vector-to-cluster mapping, 406
- Open-loop control configuration, 487–88
- Open-loop operational amplifier, 576, 577–80
- Operant conditioning, 76, 77
- Operational amplifier. *See also*
- Differential voltage amplifiers
  - analog implementation of processing node and, 576–85
  - depletion MOSFET circuit and, 632
  - MDAC, 599–600
  - voltage gain of, 624
- Optimization, 14–16, 20, 218–20, A17–21. *See also* specific type
- combinational, 16, 294–99
  - single-layer feedback networks in, 287–99, 300, 301
- Orthogonality, 337–38, A7
- bidirectional associative memory and, 358, 359
  - projection learning rule and, 353
- Orthonormality, 322, 324
- Ouali, J., 665
- Ougney, H., 592, 594
- Output, differential, 617–24
- Output current
- current mirrors and, 610–13
  - differential. *See* Differential current
- Output error, 497–98
- Output neurons, number of, 215–16
- Output pattern. *See* Output vectors
- Output signal, 32
- Output splitting, error back-propagation training with, 520–21, 522
- Output update rule, for Hopfield autoassociative memory, 327
- Output vectors
- biological neuron model, 32, 36
  - discrete-time Hopfield network, 256
  - feedback network, 43, 45, 256

- feedforward network, 37  
recall and, 53
- Output voltage.** *See* Voltage
- Outstar learning rule,** 71–72, 73  
counterpropagation network  
and, 413
- p-well CMOS technology,** analog multiplier with weight storage and, 640
- Pacheco, M., 568
- Page, E. W., 298
- Pairs**  
coding and retrieval with bidirectional associative memory, 360–63, 364  
dummy augmentation of, 366–68
- Pao, Y. H., 230, 232, 435, 523, 524, 525, 526, 527
- Papert, S., 19
- Parallel distributed processing,** 20
- Parallel distributed processing hardware,** efficiency of, 567–69
- Parameter changes,** learning and, 2
- Parameter selection,** for dynamical systems, 254
- Parity function,** 113–14
- Park, S., 272
- Parthasarathy, K., 186, 513
- Partitioning**  
in classification of linearly nonseparable patterns, 166, 168, 169  
classifiers in, 4–5
- Passino, K. M., 244
- Patel, R. V., 519, 520, 521, 522
- Pattern(s)**  
autoassociation of, 53  
classification of, 94–95. *See also* Classification  
coding and retrieval of pairs, 360–63, 364  
input. *See* Input vectors  
linearly nonseparable, 165–75  
linearly separable, 113–14  
natural similarity of, 415
- number vs. size of training patterns, 214–16
- output.** *See* Output vectors
- restoration of, 13  
spatio-temporal, 370–75  
temporal, 371
- Pattern classifier, discriminant functions** and, 104, 105
- Pattern display,** 101
- Pattern space,** 97–98  
mapping into image space, 165, 168–70, 171
- Pattern vectors**  
augmented, 109–10  
generation of, 97  
size vs. number of training patterns, 214–16
- Paturi, R., 343
- Paulos, J. J., 665
- Pendulum, inverted,** 499–503
- Penz, P. A., 664, 665
- Perceptron classifiers,** single-layer, 93–153. *See also* Single-layer perceptron classifiers
- Perceptron convergence theorem,** 129
- Perceptron learning rule,** 64–66, 73, 77
- Perceptrons**  
continuous, 35, 36, 76  
discrete (binary), 35, 36, 76  
invention of, 18–19
- Personnaz, L., 353
- Peterson, L. L., 528, 531
- Petsche, T., 342
- Phonemes,** feature detection and, 17, 430–31
- Phonotopic map,** 17
- Photosynapses,** 660–62, 663
- Physical systems,** simulation of, 21
- Pitts, W. H., 18, 30, 31
- Placement problems,** 299
- Planar pattern classification,** 171–72, 174
- Plant**  
characteristic surface of, 504–6, 507, 508
- CMAC-generated response of,** 511, 512
- Jacobian matrix of,** 498–99
- transfer functions or characteristics of,** 486–87
- Plant identification,** 489–94
- Plant output error,** 497–98
- PMOS transistors**  
analog multiplier with weight storage and, 638–40  
in CMOS inverter circuit, 613–17  
template matching network, 630
- Poggio, T., 55, 56, 200, 548
- Pomerleau, D. A., 8, 9, 547
- Posner, E. C., 343, 344, 345
- Problem-algorithm-model flowchart,** 569, 570
- Proceedings of the IEEE,* 547
- Process identification,** 489–94
- Processing,** 53–55
- Processing nodes,** 575  
analog, 575–85  
virtual ground, 577–78
- Processing speed,** 571, 572
- Processing unit, of neuron,** 32, 569
- Processors,** 567–69. *See also* Hardware; specific type
- Programmable analog neural network processor,** 657–58, 659
- Programmable attenuator,** 598
- Programmable neurocomputers,** 565–66. *See also* Hardware  
mixed computations in, 567
- Programmable weights,** 595–604
- Projection learning rule,** 353–54
- Prototype,** 313
- Prototype vectors,** 315–16
- Pruning, multilayer feedforward network,** 218–20
- Psaltis, D., 492, 493, 494, 496, 665
- Pseudoinverse matrix,** 353
- Pulse train,** 28
- Push-pull amplifier,** 613

- R-category continuous perceptron classifier training, 151
- R-category discrete perceptron classifier training, 144–46
- R-category discrete perceptron training algorithm (RDPTA), 146
- Radar pulse identification, 571–72
- Radius of attraction, recurrent autoassociative memory, 344
- Raffel, J. I., 600, 601, 602
- RAM cell, 568, 569
- Ramacher, U., 665
- Ramp, unipolar, 34, 35
- RAMSRA (recurrent associative memory storage and retrieval algorithm), 335–36
- Range finder, in ALVINN, 8
- Rauch, H. E., 298
- RDPTA (R-category discrete perceptron training algorithm), 146
- Recall, 53  
feedforward, 185–86
- Recall mode, clustering and, 406
- Recall times, 571, 572
- Recognition, classification and, 54, 94, 95, 220–25. *See also* Classification
- Recording (batch learning), 56
- Recording algorithm, 315
- Recurrent associative memory storage and retrieval algorithm (RAMSRA), 335–36
- Recurrent autoassociative memory, 318, 319, 325–38  
advantages of, 354  
asymptotic capacity of, 344  
capacity of, 343–45  
convergence of, 345–51, 352  
convergence vs. corruption of, 345–48  
convergent toward fixed points, 351–54  
energy function reduction in, 342–43  
fixed point concept and, 349–51, 352
- flip-flop network and, 330–36  
with inverter-based neurons, 617, 618
- limitations of, 354
- performance analysis of, 339–54
- performance considerations with, 336–38
- retrieval algorithm, 327–28
- spurious, 341–42
- storage algorithm, 328–29
- Recurrent feedback network, 42, 43, 44–49, 252. *See also* Discrete-time feedback network
- Recurrent heteroassociative memory, 318
- Recurrent memory  
autoassociative, 318, 319, 325–38  
heteroassociative, 318
- Recursion error, 349–50
- Recursive asynchronous updating, 258–61
- Recursive update algorithm, 258
- Reece, M., 569
- Reed, R. D., 665
- Refractory period, 29
- Relaxation algorithm, 284–86
- Relaxation modeling, in single-layer feedback networks, 283–86
- Representation  
distributed, 145–46, 216  
floating-point, 572  
local, 145
- Representation problem, 55–56
- Resistance, NMOS transistor channel, 588–90
- Resistive synaptic connection circuit, 586, 587
- Resistor, MDAC, 596–604
- Resistor-operational amplifier circuits, 576–85
- Resource allocation, job-shop scheduling and, 16
- Restoration of patterns, simple memory and, 13
- Retrieval, 315. *See also* Memory
- Retrieval algorithm, recurrent autoassociative memory, 327–28
- Rietman, R. A., 653, 660, 661, 663
- Rite, S. A., 19
- Ritter, H., 540, 542, 544, 545, 546
- Road images, autonomous driver and, 8–9
- Robot kinematics, 513–27, 571  
forward problem, 515, 516–23  
inverse problem, 514, 515, 516–19  
target position control in, 514, 523–27
- Rodemich, E. R., 343, 344, 345
- Rohrs, C. E., 298
- Root-mean-square normalized error, 196
- Rosenberg, C., 220, 547
- Rosenblatt, F., 18, 19, 36, 64
- Rosenfeld, E., 547
- Roska, T., 260
- Rossetto, O., 665
- Rueckert, U., 575, 665
- Rule-based automated diagnosis systems, 527
- Rumelhart, D. E., 20, 68, 168, 208
- Saddle points, 271, 279, 280, 283
- Saerens, M., 496
- Sage, J. P., 377
- Salam, F. M. A., 665
- Samalam, V. K., 665
- Sample-and-hold circuit, for weight storage, 593–95
- Sampling rate, 572
- Sartori, M. A., 244
- Saturation, 619, 621  
all-MOS scalar product circuit with, 635–36
- Satyana Rayana, S., 254, 662
- Saucier, G., 665
- Scalar averaging circuit, with transconductance amplifiers, 625–26

- Scalar criterion function, minimization of, 132–33
- Scalar product circuits, 630–43. *See also* Analog multipliers
- depletion MOSFET, 631–36
  - with electrically tunable weights, 588
  - enhancement mode MOS, 636–38
  - with transconductance amplifiers, 624–25
- Scheduling, job-shop, 16, 298
- Schneider, C. R., 605, 606
- Schumacher, K., 575, 665
- Schwartz, D. B., 569, 665
- SCPTA (single continuous perceptron training algorithm), 141–42
- SDPTA (single discrete perceptron training algorithm), 131–32
- Search argument, 313
- Sedra, A. S., 596, 598, 610, 612, 641
- Sejnowski, T., 220, 547
- Self-organization, 76
- clustering and feature detecting networks and, 16–17, 423–32, 539–46
  - unsupervised learning and, 58
- Self-organizing feature maps, 423–32
- Self-organizing semantic maps, 539–46
- Semantic maps, self-organizing, 539–46
- Separability limitations, clusters and, 409–10
- Sequence of mappings, 523
- Serial calculation, of activation value, 575
- Serra, R., 350, 352
- sgn function, squashed, 5, 6, 9
- Shanblatt, M. A., 299
- Shekhar, S., 548
- Shen, W., 284
- Sheu, B. J., 650
- Sideris, A., 492, 493, 494, 496
- Sietsma, J., 218
- Sigmoidal activation functions, feedforward network, 41–42
- Sigmoidal characteristics, 34, 613
- transconductance amplifier and, 624
- Signal flow, error back-propagation and, 186, 188
- Signal processors, 21
- classifiers and, 6–7
- Silicon-based neural hardware, 566. *See also* Hardware
- Silva, F. M., 213
- Silverstein, J. W., 19
- Similarity, natural, 415
- Similarity measures, clustering and, 399–401
- Simmons, D. M., 464
- Simple memory network, 10–13
- applications of, 12–13
  - pattern restoration and, 13
- Simpson, P. I., 405
- Single continuous perceptron training algorithm (SCPTA), 141–42
- Single discrete perceptron training algorithm (SDPTA), 131–32
- Single-ended output differential-input amplifier, 620, 621
- Single-layer feedback networks, 251–310
- basic concepts of, 253–54
  - capacity of, 300
  - continuous-time, 264–83
  - discrete-time Hopfield networks, 254–64
  - fully coupled, 298
  - gradient-time Hopfield networks, 264–76
  - limitations of, 300
  - linear programming, 456–64
  - optimization problems using, 287–99, 300, 301
  - recurrent, 252
  - relaxation modeling in, 283–86
  - transient response of, 276–83
- Single-layer perceptron classifiers, 93–153
- classification model and, 94–98
- continuous perceptron networks for linearly separable classifications, 132–42, 175
- decision regions and, 98
- discrete perceptron in training and classification, 120–32
- discriminant functions and, 99–106
- features and, 95–96
- linear, 106–14
- minimum distance classification and, 108–10
- multicategory networks, 142–52
- nonparametric training concept and, 114–20
- Siram, M., 299
- Skeletonization, of handwritten characters, 478–82
- Skin disease diagnosis, expert system for, 528–32
- Sklansky, J., 153
- Smith, K. C., 596, 598, 610, 612, 641
- Smithson, R. C., 653, 657, 658, 659
- Soares, A. M., 600, 601, 602
- Sobajic, D. J., 523, 524, 525, 526, 527
- Soft-limiting neuron, 35
- Soma, 27
- Soquet, A., 496
- Space transformations, sequence of, 523
- Spatial objects, pattern vector for, 97
- Spatio-temporal patterns, associative memory of, 370–75
- Special refresh signals, 594
- Speech features, mapping of, 16–17, 430–31
- SPICE2G1.6 program, 289–92, 633
- Spiegel, V. D., 653, 658, 660
- Spurious memories, 341–42
- Square error minimization, continuous perceptron training using, 133–42

- Squashed sgn function, 5, 6  
ALVINN and, 9
- Staircase approximation, 197–200
- State transitions, feedback network, 43–49
- Static networks, 317
- Static plants  
learning control architecture and, 495–96  
transfer characteristics of, 486, 487
- Stationary points  
continuous-time network, 278–83  
gradient-type Hopfield network, 271–73
- Steepest descent procedure. *See also* Gradient descent-based training
- Stepwise training, batch learning vs., 56
- Stimuli, linear associative memory and, 321
- Stinchcombe, M., 200
- Stochastic approximation, 207
- Stochastic recursion, asynchronous, 258–61
- Storage, analog, 592–95
- Storage algorithm, 315  
recurrent autoassociative memory, 328–29
- Stored memory, 313. *See also* Memory  
complement of, 327
- Strader, N. R., 605, 610, 612, 622
- Successive approximation method, 284–86
- Sudhakar, R., 464
- Summing network, with digital outputs, 287–94
- Sun, G. Z., 377
- Supervised learning, 56–57  
adaptive mapping system, 151–52  
counterpropagation network and, 413  
robot kinematics and, 516, 517
- Sutton, R. S., 499
- Suzuki, R., 547
- Switch, voltage-controlled, 586, 587–92, 593–95
- Symbol vectors, semantic maps and, 541–45
- Synapse, 27–28
- Synaptic connections. *See also* Weight(s)  
ETANN 80170 chip, 653–55  
feedback network, 48–49  
integrated circuit, 585–608, 609  
neuron models with, 35  
photosynaptic, 660–62, 663
- Synchronous state updating algorithm, 261–62
- Szu, H. H., 665
- Tagliarini, G. A., 298
- Takefuji, Y., 16, 298, 665
- Tank, D. W., 253, 273, 295, 296, 297, 298, 457, 458, 459, 461, 463
- Target position control, in robot kinematics, 514, 523–27
- Tattershall, G., 417
- Template matching network, 628–30
- Temporal associative memory, 76, 371–74
- Temporal objects, pattern vector for, 97
- Temporal patterns, 371
- Temporal sequences, 229–30
- Tensor model, 231
- Terminology, 2
- Thakoor, A. P., 603, 648, 651, 665
- Three-dimensional Cartesian space partitioning, 4–5
- Threshold function, bipolar, 355
- Threshold logic unit (TLU), 34  
dichotomizer and, 104–5, 115  
multilayer feedforward network and, 170, 172, 198, 199  
window implementation using, 198, 199
- Threshold logic unit (TLU)-based classifier, continuous perceptron element, 137
- Threshold logic unit (TLU)-based perceptron, 115
- Threshold of neuron, 29
- Threshold vector  
discrete-time Hopfield network, 256  
gradient-type Hopfield network, 268
- Thrile, J., 665
- Time-delay network, 229–30. *See also* Delay network
- Time-domain behavior. *See* Dynamics
- TLU. *See* Threshold logic unit
- Tolat, V. V., 499, 500, 501, 503
- Topological neighborhoods, semantic maps and, 540, 541
- Tou, J. T., 153, 401
- Touretzky, D., 547
- Trained network, classifiers and, 5–7
- Training. *See also* Learning  
back-propagation. *See* Error back-propagation training  
convergence of, 207–8  
discrete perceptron in, 120–32  
error, 195–96  
ETANN 80170 chip, 653–57  
improvements in efficiency of, 574  
multiple, 365–68  
nonparametric, 114–20  
randomness to, 207
- Training patterns, necessary number vs. pattern vector size, 214–16
- Training rule, continuous perceptron, 136
- Training sequence, 115
- Training set, in supervised learning, 57
- Trajectory control, of robotic manipulators, 514. *See also* Robot kinematics
- Trajectory mapping, 521–23

- Transconductance amplifiers, 622–24  
scalar product and averaging circuits with, 624–26
- Transfer characteristics, 486, 487  
of operational amplifier, 576, 577–80
- Transfer function, 486–87. *See also* Voltage transfer functions
- Transient response, of continuous-time networks, 276–83
- Transient simulations, 291–94
- Transistors. *See also* MOS transistors; NMOS transistors; PMOS transistors  
floating-gate, 640–43
- Transition map, bistable flip-flop, 332–33
- Traveling salesman problem, 294–99
- Treleaven, P. C., 568, 569
- Triangular map, 350, 351, 352
- Triple dichotomization, 150
- Tsividis, Y. P., 592, 594, 662
- Tsyplkin, Ya. Z., 58, 207, 548
- Two-bit A/D conversion problem. *See also* Analog-to-digital (A/D) conversion optimization and, 14–15
- Two-layer feedforward network, 39–42. *See also* Multilayer feedforward networks
- Unipolar activation functions, 33–34, 35, 36  
robot kinematics and, 519
- Unit step function, 198
- Unity-gain current mirrors, 611–13
- Universal approximators, multi-layer feedforward networks as, 196–205, 206
- Uno, Y., 547
- Unsupervised learning, 57–58, 399–410  
CMAC and, 506  
multiple-winner, 407
- Update cycle, memory convergence vs. corruption and, 347–48
- Updating  
asynchronous, 257–61, 329, 334
- Hopfield autoassociative memory and, 327
- incremental, 208–9
- self-organizing feature maps and, 425
- simple memory and, 10, 11–12
- stability with bidirectional associative memory, 359–60
- synchronous, 261–62
- temporal memory and, 373–75
- Variable characteristics, 569–70
- Vector field method, 276–78, 281–83
- Vector matching, counterpropagation and, 414–15
- Vector-to-cluster mapping, one-to-one, 406
- Vectors, 32, 36. *See also* specific type
- Vegvar, P. D., 645, 647
- Vehicle driver, autonomous, 8–10, 76
- Vellasco, M., 568
- Venkatesh, S. V., 343, 344, 345
- Verleysen, M., 648, 665
- Very-large-scale integrated (VLSI) chips, dedicated, 658, 660
- Very-large-scale integrated (VLSI) networks, 22  
floating-gate transistors and, 640
- VGAIN voltage, ETANN 80170 chip, 655, 657
- Vidal, J. J., 665
- Video input, in ALVINN, 8–9
- Vigilance levels, 442–43
- Vigilance test, 437, 439, 440
- Villalobos, L., 548
- Virtual ground, 577–78
- Vision, low level, 571, 572
- Vittoz, E., 592, 594
- VLSI networks. *See* Very-large-scale integrated entries
- Voltage  
current conversion to, 586–87.  
*See also* Current-to-voltage converter
- differential, 617–24
- VGAIN, 655, 657
- weight storage as, 592–95
- Voltage-controlled weights, 587–92  
with learning, 605–8  
voltage tracking and, 593
- Voltage decay rate, reduction of, 640
- Voltage follower circuit, 624–25
- Voltage transfer functions, 461–63  
differential input amplifier, 620, 621  
inverter-based neuron, 615, 617
- open-loop operational amplifier, 576, 577–80
- sigmoidal characteristics, 613
- von Neumann, J., 18, 376
- Wadell, G., 532, 533, 534, 535, 536
- Wagner, K., 665
- Waibel, A., 220
- Wang, L., 547
- Wang, Y., 665
- Wang, Y. F., 365, 366, 367, 368
- Wasaki, H., 665
- Wassel, G. N., 153
- Wasserman, P. D., 413
- Weight(s), 585–608, 609  
adjustable, 586, 592–95  
binary, 645–48  
connection strength expression by, 2  
digitally programmable, 595–604  
initial, 208  
initialization of, 406–7  
learning, 586, 605–8, 609  
photosynapses, 660–62, 663

- requirements for, 585–86  
voltage-controlled, 587–92
- Weight adjustment(s).** *See also*  
Single-layer perceptron classifiers  
CMAC and, 506  
cumulative, 208–9  
delta training /learning, 178  
hidden layers and, 193  
of learning dichotomizer,  
117–20
- Weight adjustment rule,** continuous perceptron and, 137
- Weight adjustment vector,** length of, 121
- Weight control voltage decay rate,** reduction of, 640
- Weight correction formula,** 120
- Weight matrix**  
ART1, 434  
discrete-time Hopfield network, 256  
feedback network, 45, 256, 288  
feedforward network, 38  
linear memory, 322–24  
multiple training, 365–66  
recurrent autoassociative memory, 339, 353–54
- storage algorithm for calculation of, 328–29
- Weight storage,** analog multiplier with, 638–40
- Weight vector,** 32  
learning rules and, 59–60. *See also* Learning rules
- Weighted-resistor MDAC,** 596–98, 600
- Werbos, P. J.**, 20, 168
- White, H.**, 200, 207
- Widrow, B.**, 19, 69, 499, 500, 501, 503, 513, 547
- Widrow-Hoff learning rule,** 19, 69, 73
- Wilhelmsen, K.**, 547
- Wilson current mirror,** 612, 613
- Winarske, T.**, 298
- Window implementation,** 198–200
- Window templates,** in handwritten character skeletonization, 481
- Winner-take-all learning rule,** 70–71, 73  
CMAC and, 506  
Kohonen network and, 401–5
- Winning neurons,** multiple, 407
- Wong, B. L.**, 660, 661, 663
- Wong, C. C.**, 653
- Word recognition,** 571
- Wu, S.**, 665
- XOR function,** 114, A38–39  
architectures and, 218–20  
error back-propagation and, 194  
functional link network and, 232–33, 234  
layered classifier in implementation of, 168  
template matching network expression as, 628
- Xu, X.**, 665
- Yamamura, A.**, 492, 493, 494, 496
- Yoo, Y. S.**, 614
- Yoon, Y. O.**, 528, 531
- Young, T. Y.**, 153
- Youssef, A. M.**, 376, 377
- Yu, F. T. S.**, 665
- Zanarini, G.**, 350, 352
- Zigoris, D. M.**, 464, 466, 470
- Zue, V. W.**, 220
- Zurada, J. M.**, 284, 289, 464, 466, 470, 595, 614

