One of the nice features of human memory is its ability to learn many new things without necessarily forgetting things learned in the past. A frequently cited example is the ability to recognize your parents even if you have not seen them for some time and have learned many new faces in the interim. It would be highly desirable if we could impart this same capability to an ANS. Most networks that we have discussed in previous chapters will tend to forget old information if we attempt to add new information incrementally.

When developing an ANS to perform a particular pattern-classification operation, we typically proceed by gathering a set of exemplars, or training patterns, then using these exemplars to train the system. During the training, information is encoded in the system by the adjustment of weight values. Once the training is deemed to be adequate, the system is ready to be put into production, and no additional weight modification is permitted.

This operational scenario is acceptable provided the problem domain has well-defined boundaries and is stable. Under such conditions, it is usually possible to define an adequate set of training inputs for whatever problem is being solved. Unfortunately, in many realistic situations, the environment is neither bounded nor stable.

Consider a simple example. Suppose you intend to train a BPN to recognize the silhouettes of a certain class of aircraft. The appropriate images can be collected and used to train the network, which is potentially a time-consuming task depending on the size of the network required. After the network has been trained and deemed to recognize all of the aircraft, the training period is ended and no further modification of the weights is allowed.

If, at some future time, another aircraft in the same class becomes operational, you may wish to add its silhouette to the store of knowledge in your network. To do this, you would have to retrain the network with the new pattern plus all of the previous patterns. Training on only the new silhouette could result in the network learning that pattern quite well, but forgetting previously learned patterns. Although retraining may not take as long as the initial training, it still could require a significant investment.

Moreover, if an ANS is presented with a previously unseen input pattern, there is generally no built-in mechanism for the network to be able to recognize the novelty of the input. The ANS doesn't know that it doesn't know the input pattern.

We have been describing what Stephen Grossberg calls the stability-plasticity dilemma [5]. This dilemma can be stated as a series of questions [6]:

- How can a learning system remain adaptive (plastic) in response to significant input, yet remain stable in response to irrelevant input?
- How does the system know to switch between its plastic and its stable modes?
- How can the system retain previously learned information while continuing to learn new things?

In response to such questions, Grossberg, Carpenter, and numerous colleagues developed adaptive resonance theory (ART), which seeks to provide answers. ART is an extension of the competitive-learning schemes that have been discussed in Chapters 6 and 7. The material in Section 6.1 especially, should be considered a prerequisite to the current chapter. We will draw heavily from those results, so you should review the material, if necessary, before proceeding.

In the competitive systems discussed in Chapter 6, nodes compete with one another, based on some specified criteria, and the winner is said to classify the input pattern. Certain instabilities can arise in these networks such that different nodes might respond to the same input pattern on different occasions. Moreover, later learning can wash away earlier learning if the environment is not statistically stationary or if novel inputs arise.

A key to solving the stability-plasticity dilemma is to add a feedback mechanism between the competitive layer and the input layer of a network. This feedback mechanism facilitates the learning of new information without destroying old information, automatic switching between stable and plastic modes, and stabilization of the encoding of the classes done by the nodes. The results from this approach are two neural-network architectures that are particularly suited for pattern-classification problems in realistic environments. These network architectures are referred to as ART1 and ART2. ART1 and ART2 differ in the nature of their input patterns. ART1 networks require that the input vectors be binary. ART2 networks are suitable for processing analog, or gray-scale, patterns.

ART gets its name from the particular way in which learning and recall interplay in the network. In physics, resonance occurs when a small-amplitude vibration of the proper frequency causes a large-amplitude vibration in an electrical or mechanical system. In an ART network, information in the form of processing-element outputs reverberates back and forth between layers. If the proper patterns develop, a stable oscillation ensues, which is the neural-network
equivalent of resonance. During this resonant period, learning, or adaptation, can occur. Before the network has achieved a resonant state, no learning takes place, because the time required for changes in the processing-element weights is much longer than the time that it takes the network to achieve resonance.

A resonant state can be attained in one of two ways. If the network has learned previously to recognize an input vector, then a resonant state will be achieved quickly when that input vector is presented. During resonance, the adaptation process will reinforce the memory of the stored pattern. If the input vector is not immediately recognized, the network will rapidly search through its stored patterns looking for a match. If no match is found, the network will enter a resonant state whereupon the new pattern will be stored for the first time. Thus, the network responds quickly to previously learned data, yet remains able to learn when novel data are presented.

Much of Grossberg's work has been concerned with modeling actual macroscopic processes that occur within the brain in terms of the average properties of collections of the microscopic components of the brain (neurons). Thus, a Grossberg processing element may represent one or more actual neurons. In keeping with our practice, we shall not dwell on the neurological implications of the theory. There exists a vast body of literature available concerning this work. Work with these theories has led to predictions about neurophysiological processes, even down to the chemical-ion level, which have subsequently been proven true through research by numerous references listed at the end of this chapter.

The equations that govern the operation of the ART networks are quite complicated. It is easy to lose sight of the forest while examining the trees closely. For that reason, we first present a qualitative description of the processing in ART networks. Once that foundation is laid, we shall return to a detailed discussion of the equations.

8.1 ART NETWORK DESCRIPTION

The basic features of the ART architecture are shown in Figure 8.1. Patterns of activity that develop over the nodes in the two layers of the attentional subsystem are called short-term memory (STM) traces because they exist only in association with a single application of an input vector. The weights associated with the bottom-up and top-down connections between $F_1$ and $F_2$ are called long-term memory (LTM) traces because they encode information that remains a part of the network for an extended period.

8.1.1 Pattern Matching in ART

To illustrate the processing that takes place, we shall describe a hypothetical sequence of events that might occur in an ART network. The scenario is a simple pattern-matching operation during which an ART network tries to determine whether an input pattern is among the patterns previously stored in the network. Figure 8.2 illustrates the operation.

In Figure 8.2(a), an input pattern, $I$, is presented to the units on $F_1$ in the same manner as in other networks: one vector component goes to each node. A pattern of activation, $X$, is produced across $F_1$. The processing done by the units on this layer is a somewhat more complicated form of that done by the input layer of the CPN (see Section 6.1). The same input pattern excites both the orienting subsystem, $A$, and the gain control, $G$ (the connections to $G$ are not shown on the drawings). The output pattern, $S$, results in an inhibitory signal that is also sent to $A$. The network is structured such that this inhibitory signal exactly cancels the excitatory effect of the signal from $I$, so that $A$ remains inactive. Notice that $G$ supplies an excitatory signal to $F_1$. The same signal is applied to each node on the layer and is therefore known as a nonspecific signal. The need for this signal will be made clear later.

The appearance of $X$ on $F_1$ results in an output pattern, $S$, which is sent through connections to $F_2$. Each $F_2$ unit receives the entire output vector, $S$,
Figure 8.2 A pattern-matching cycle in an ART network is shown. The process evolves from the initial presentation of the input pattern in (a) to a pattern-matching attempt in (b), to reset in (c), to the final recognition in (d). Details of the cycle are discussed in the text.

From $F_1$, $F_2$ units calculate their net-input values in the usual manner by summing the products of the input values and the connection weights. In response to inputs from a pattern of activity, $Y$, develops across the nodes of $F_2$. $F_2$ is a competitive layer that performs a contrast enhancement on the input signal like the competitive layer described in Section 6.1. The gain control signals to $F_2$ are omitted here for simplicity.

In Figure 8.2(b), the pattern of activity, $Y$, results in an output pattern, $I$. This output pattern is sent as an inhibitory signal to the gain control system. The gain control is configured such that if it receives any inhibitory signal from $F_2$, it ceases activity. $I$ also becomes a second input pattern for the $F_1$ units. $U$ is transformed by LTM traces on the top-down connections from $F_1$ to $F_2$. We shall call this transformed pattern $V$.

Notice that there are three possible sources of input to $F_1$, but that only two appear to be used at any one time. The units on $F_1$ (and $F_2$ as well) are constructed so that they can become active only if two out of the possible three sources of input are active. This feature is called the 2/3 rule and it plays an important role in ART, which we shall discuss more fully later in this section.

Because of the 2/3 rule, only those $F_1$ nodes receiving signals from both $I$ and $V$ will remain active. The pattern that remains on $F_1$ is $I \cap V$, the intersection of $I$ and $V$. In Figure 8.2(b), the patterns mismatch and a new activity pattern, $X^*$, develops on $F_1$. Since the new output pattern, $X^*$, is different from the original pattern, $S$, the inhibitory signal to $A$ no longer cancels the excitation coming from $I$.

In Figure 8.2(c), $A$ has become active in response to the mismatch of patterns on $F_1$. $A$ sends a nonspecific reset signal to all of the nodes on $F_2$. These nodes respond according to their present state. If they are inactive, they do not respond. If they are active, they become inactive and they stay that way for an extended period of time. This sustained inhibition is necessary to prevent the same node from winning the competition during the next matching cycle. Since $Y$ no longer appears, the top-down output and the inhibitory signal to the gain control also disappear.

In Figure 8.2(d), the original pattern, $X$, is reinstated on $F_1$, and a new cycle of pattern matching begins. This time a new pattern, $Y^*$, appears on $F_2$. The nodes participating in the original pattern, $Y$, remain inactive due to the long term effects of the reset signal from $A$.

This cycle of pattern matching will continue until a match is found, or until $F_1$ runs out of previously stored patterns. If no match is found, the network will assign some uncommitted node or nodes on $F_1$ and will begin to learn the new pattern. Learning takes place through the modification of the weights, or the LTM traces. It is important to understand that this learning process does not start or stop, but rather continues even while the pattern matching process takes place. Anytime signals are sent over connections, the weights associated with those connections are subject to modification. Why then do the mismatches not result in loss of knowledge or the learning of incorrect associations? The reason is that the time required for significant changes to occur in the weights is very long with respect to the time required for a complete matching cycle. The connections participating in mismatches are not active long enough to affect the associated weights seriously.

When a match does occur, there is no reset signal and the network settles down into a resonant state as described earlier. During this stable state, connections remain active for a sufficiently long time so that the weights are strengthened. This resonant state can arise only when a pattern match occurs, or during the enlistment of new units on $F_1$ in order to store a previously unknown pattern.

In an actual ART network, the pattern-matching cycle may not visit all previously stored patterns before an uncommitted $F_1$ node is chosen.
8.1 ART Network Description

8.1.2 Gain Control in ART
Before continuing with a look at the dynamics of ART networks, we want to examine more closely the need for the gain-control mechanism. In the simple example discussed in the previous section, the existence of a gain control and the 2/3 rule appear to be superfluous. They are, however, quite important features of the system, as the following example illustrates.

Suppose the ART network of the previous section was only one in a hierarchy of networks in a much larger system. The $F_2$ layer might be receiving inputs from a layer above it as well as from the $F_1$ layer below. This hierarchical structure is thought to be a common one in biological neural systems. If the $F_2$ layer were stimulated by an upper layer, it could produce a top-down output and send signals back to the $F_1$ layer. It is possible that this top-down signal would arrive at $P_1$ before an input signal, $I$, arrived at $P_2$ from below. A premature signal from $F_2$ could be the result of an expectation arising from a higher level in the hierarchy. In other words, $F_2$ is indicating what it expects the next input pattern to be, before the pattern actually arrives at $F_1$. Biological examples of this expectation phenomenon abound. For example, how often have you anticipated the next word that a person was going to say during a conversation?

The appearance of an early top-down signal from $F_2$ presents us with a small dilemma. Suppose $F_2$ produced an output in response to any single input vector, no matter what the source. Then, the expectation signal arriving from $F_2$ would elicit an output from $F_1$ and the pattern-matching cycle would ensue without ever having an input vector to $F_1$ from below. Now let's add in the gain control and the 2/3 rule.

According to the discussion in the previous section, if $G$ exists, any signal coming from $F_2$ results in an inhibition of $G$. Recall that $G$ nonspecifically arouses every $F_1$ unit. With the 2/3 rule in effect, inhibition of $G$ means that a top-down signal from $F_2$ cannot, by itself, elicit an output from $F_1$. Instead, the $F_1$ units become preconditioned, or sensitized, by the top-down pattern. In the biological language of Grossberg, the $F_1$ units have received a subliminal stimulation from $F_2$. If the expected input pattern is received on $F_1$ from below, this preconditioning results in an immediate resonance in the network. Even if the input pattern is not the expected one, $F_1$ will still provide an output, since it is receiving inputs from two out of the three possible sources, $I$, $G$, and $F_2$.

If there is no expectation signal from $F_2$, then $F_1$ remains completely quiescent until it receives an input vector from below. Then, since $G$ is not inhibited, $F_1$ units are again receiving inputs from two sources and $F_1$ will send an output up to $F_2$ to begin the matching cycle.

$G$ and the 2/3 rule combine to permit the $F_1$ layer to distinguish between an expectation signal from above, and an input signal from below. In the former case, $F_1$'s response is subliminal; in the latter case, it is supraliminal—that is, it generates a nonzero output.

Exercise 8.1: Based on the discussion in this section, describe how the gain control signal on the $F_1$ layer would function.

8.2 ART1
The ARTI architecture shares the same overall structure as that shown in Fig. 8.1. Recall that all inputs to ARTI must be binary vectors; that is, they must have components that are elements of the set $\{0, 1\}$. This restriction may appear to limit the utility of the network, but there are many problems having data that can be cast into binary format. The principles of operation of ART1 are similar to those of ART2, where analog inputs are allowed. Moreover, the restrictions and assumptions that we make for ARTI will simplify the mathematics a bit. We shall examine the attentional subsystem first, including the STM layers $P_1$ and $F_1$, and the gain-control mechanism, $G$.

8.2.1 The Attentional Subsystem
The dynamic equations for the activities of the processing elements on layers $P_1$ and $F_1$ both have the form

$$ ez_j = -z_j + (1 - Az_j) w_j^* + (\beta + Cz_j) y_j^* \quad (8.1) $$

where $z_j^*$ is an excitatory input to the $j$th unit and $y_j^*$ is an inhibitory input. The precise definitions of these terms, as well as those of the parameters $A$, $\beta$, and $C$, depend on which layer is being discussed, but all terms are assumed to be greater than zero. Henceforth we shall use $x_1$ to refer to activities on the $F_1$ layer, and $y_1$ for activities on the $F_2$ layer. Similarly, we shall add numbers to the parameter names to identify the layer to which they refer; for example, $B_2$, $A_2$. For convenience, we shall label the nodes on $F_1$ with the symbol $v_1$ and those on $F_2$ with $v_2$. The subscripts $j$ and $j'$ will be used exclusively to refer to the layers $F_1$ and $F_2$, respectively.

The factor $e$ requires some explanation. Recall from the previous section that pattern-matching activities between layers $F_1$ and $F_2$ must occur much faster than the time required for the connection weights to change significantly. The $e$ factor in Eq. (8.1) embodies that requirement. If we insist that $0 < e < 1$, then $F_1$ will be a fairly large value; that is, $F_1$ will reach its equilibrium value quickly. Since $x_1$ spends most of its time near its equilibrium value, we shall not have to concern ourselves with the time evolution of the activity values: We shall automatically set the activities equal to the asymptotic values. Under these conditions, the $e$ factor becomes superfluous, and we shall drop it from the equations that follow.
Exercise 8.2: Show that the activities of the processing elements described by Eq. (8.1) have their values bounded within the interval \([-\beta C, A]\), no matter how large the excitatory or inhibitory inputs may become.

**Processing on \(F_1\).** Figure 8.3 illustrates an \(F_1\) processing element with its various inputs and weight vectors. The units calculate a net-input value coming from \(F_2\) in the usual way:\(^2\)

\[
V_i = \sum_j u_{ij} z_{ij} \tag{8.2}
\]

We assume the unit output function quickly rises to 1 for nonzero activities. Thus, we can approximate the unit output, \(s_i\), with a binary step function:

\[
s_i = h(x_i) = \begin{cases} 
1 & x_i > 0 \\
0 & x_i \leq 0 
\end{cases} \tag{8.3}
\]

The total excitatory input, \(J^+_E\), is given by

\[
J^+_E = I_i + D_i V_i + B_i G \tag{8.4}
\]

where \(D_i\) and \(B_i\) are constants.\(^3\) The inhibitory term, \(J^-_I\), we shall set identically equal to 1. With these definitions, the equations for the \(F_1\) processing elements are

\[
x_i = -x_i + (1 - A_i x_i) I_i + D_i V_i + B_i G - (B_i + C_i x_i) \tag{8.5}
\]

The output, \(G\), of the gain-control system depends on the activities on other parts of the network. We can describe \(G\) succinctly with the equation

\[
G = \begin{cases} 
1 & \text{if } I = 0 \text{ and } F_2 = 0 \\
0 & \text{otherwise}
\end{cases} \tag{8.6}
\]

In other words, if there is an input vector, and \(F_2\) is not actively producing an output vector, then \(G = 1\). Any other combination of activity on \(I\) and \(F_2\) effectively inhibits the gain control from producing its nonspecific excitation to the units on \(F_1\).\(^4\)

---

\(^2\) The convention on weight indices that we have utilized consistently throughout this text is opposite to that used by Carpenter and Grossberg. In our notation, \(z_{ij}\) refers to the weight on the connection from the \(j\)th unit to the \(i\)th unit. In Carpenter and Grossberg's notation, \(z_{ij}\) would refer to the weight on the connection from the \(i\)th node to the \(j\)th node.

\(^3\) Carpenter and Grossberg include the parameter, \(D_i\), in their calculation of \(V_i\). In their notation, \(V_i = D_i \sum_j u_{ij} z_{ij}\); you should bear this difference in mind when reading their papers.

\(^4\) In the original paper by Carpenter and Grossberg, the authors described four different ways of implementing the gain control system. The method that we have shown here was chosen to be consistent with the general description of ART given in the previous sections. [5]

---

**Figure 8.3** This diagram shows a processing element, \(V_i\), on the \(F_1\) layer of an ART1 network. The activity of the unit is \(x_i\). It receives a binary input value, \(I_i\), from below, and an excitatory signal, \(G\), from the gain control. In addition, the top-down signals, \(u_{ij}\), from \(F_2\) are gated (multiplied by) weights, \(Z_{ij}\). Outputs, \(s_i\), from the processing element go up to \(F_2\) and across to the orienting subsystem, \(A\).

Let's examine Eq. (8.5) for the four possible combinations of activity on \(I\) and \(F_2\). First, consider the case where there is no input vector and \(F_2\) is inactive. Equation (8.5) reduces to

\[
x_{ij} = -x_{ij} - (B_i + C_i x_{ij}) \tag{8.7}
\]

In equilibrium \(x_{ij} = 0\).

\[
x_{ij} = -B_i \frac{1}{1 + C_i} \tag{8.7}
\]

Thus, units with no inputs are held in a negative activity state.

Now apply an input vector, \(I\), but keep \(F_2\) inactive for the moment. In this case, both \(F_1\) and the gain control receive input signals from below. Since \(F_2\)
is inactive, \( G \) is not inhibited. The equations for the \( F \) units become

\[
x_{1i} = -x_{1i} + (1 - A_1)x_{1i} (I_i + B_i G) - (B_1 + C_1 x_{1i})
\]

When the units reach their equilibrium activities,

\[
x_{1i} = \frac{I_i}{1 + A_1 (I_i + B_i) + C_1}
\]

(8.8)

where we have used \( G = 1 \). In this case, units that receive a nonzero input value from below also generate an activity value greater than zero and have a nonzero output value according to Eq. (8.3). Units that do not receive a nonzero input nevertheless have their activities raised to the zero level through the nonspecific excitation signal from \( G \).

For the third scenario, we examine the case where an input pattern, \( I \), and a top-down pattern \( V \), from \( F_2 \) are coincident on the \( F_1 \) layer. The equations for the unit activities are

\[
x_{1i} = -x_{1i} + (1 - A_1)x_{1i} (I_i + D_i V_i) - (B_1 + C_1 x_{1i})
\]

Here, the equilibrium value

\[
x_{1i} = \frac{I_i + D_i V_i - B_1}{1 + A_1 (I_i + D_i V_i) + C_1}
\]

(8.9)

requires a little effort to interpret. Whether \( x_{1i} \) is greater than, equal to, or less than zero depends on the relative values of the quantities in the numerator in Eq. (8.9). We can distinguish three cases of interest, which are determined by application of the 2/3 rule.

If a unit has a positive input value, \( I_i \), and a large positive net input from above, \( V_i \), then the 2/3 rule says that the unit activity should be greater than zero. For this to be the case, it must be true that \( I_i + D_i V_i - B_1 > 0 \) in Eq. (8.9). To analyze this relation further, we shall anticipate some results from the discussion of processing on the \( F_2 \) layer. Specifically, we shall assume that only a single node on \( F_1 \) has a nonzero output at any given time, the maximum output of a node is 1, and the maximum weight on any top-down connection is also 1. Since \( V = \sum \psi_{ij} \), and one of the \( \psi_{ij} \) is nonzero, then \( V \) is less than 1. Then, in the most extreme case with \( V = 1 \), and \( I_i = 1 \), we must have \( 1 + D_i \cdot B_1 > 0 \), or

\[
B_1 < D_1 + 1
\]

(8.10)

Any unit that does not receive a top-down signal from \( F_2 \) must have a negative activity, even if it receives an input from below. In this case, we must have \( I_i + B_1 < 0 \), or

\[
B_1 > 1
\]

(8.11)

Suppose \( F_2 \) is producing a top-down output (perhaps as a result of inputs from higher levels), but there is not yet an input vector \( I \). \( G \) is still inhibited in this case. The equilibrium state is

\[
x_{1i} = \frac{D_i V_i - B_1}{1 + A_1 D_i V_i + C_1}
\]

(8.12)

If a unit receives no net input from \( F_2 \), \( V_i = 0 \), then it remains at its most negative activity level, as in Eq. (8.7). If \( V_i > 0 \), then the unit’s activity rises to some value above that of Eq. (8.7), but it must remain negative because we do not want the unit to have a nonzero output based on top-down inputs alone. Then, from the numerator of Eq. (8.12), we must have \( D_i - B_1 < 0 \), or

\[
B_1 > D_1
\]

(8.13)

Equations (8.10), (8.11), and (8.13) combine to give the overall condition,

\[
\max (D_1, 1) < B_1 < D_1 + 1
\]

(8.14)

The ART1 parameters must satisfy the constraint of Eq. (8.14) to implement the 2/3 rule successfully and to distinguish between top-down and bottom-up input patterns.

Satisfying the constraints of Eq. (8.14) does not guarantee that a unit that receives both an input from below, \( I_i \), and one from above, \( V_i \), will have a positive value of activation. We must consider the case where \( V_i \) is less than its maximum value of 1 (even though \( I_i \) + 1, \( z_{ij} \) may be less than 1, resulting in \( V_i < 1 \)). In such a case, the condition that Eq. (8.9) gives a positive value is

\[
I_i + D_i V_i - B_1 > 0
\]

Since \( I_i = 1 \), this relation defines a condition on \( V_i \):

\[
V_i > \frac{B_1 - 1}{D_i}
\]

(8.15)

Equation (8.15) tells us that the input to unit \( V_i \) due to a top-down signal from \( F_2 \) must meet a certain threshold condition to ensure a positive activation of \( V_i \), even if \( V_i \) receives a strong input, \( I_i \), from below. We shall return to this result when we discuss the weights, or LTM traces, from \( F_2 \) to \( F_1 \).

**Processing on \( F_2 \)** Figure 8.4 shows a typical processing element on the \( F_2 \) layer. The gain-control input and the connection from the orienting subsystem are shown but we shall not include them explicitly in the following analysis.

The unit activations are described by an equation of the form of Eq. (8.1). We can specify the various terms as follows.

The net input received from the \( F_1 \) layer is calculated as usual:

\[
T_i = \text{net}_i = \sum_j h_{ij} z_{ij}
\]

(8.16)
To all units

From Figure 8.4

A processing element, \( V_j \), is shown on the layer of an ART1 network. The activity of the unit is \( x_j \). The unit \( V_j \) receives inputs from the \( F_j \) layer, the gain-control system, \( G_j \), and the orienting subsystem, \( A \). Bottom-up signals, \( s_j \), from \( F_j \) are gated by the weights, \( z_j \). Outputs, \( u_j \), are sent back down to \( F_j \). In addition, each unit receives a positive-feedback term from itself, \( g(x_j) \), and sends an identical signal through an inhibitory connection to all other units on the layer.

The total excitatory input to \( V_j \) is

\[
J^x_{ij} = D_j T_j + g(x_j)
\]  (8.17)

The inhibitory input to each unit is

\[
J^{-}_{ij} = \sum_{k \neq j} g(x_{kj})
\]  (8.18)

Substituting these values into Eq. (8.1) yields

\[
x_{2j} = -x_{2j} + \left( 1 - A_j T_j \right) \left( D_j T_j + g(x_j) \right) - \left( B_j + C_j x_j \right) \sum_{k \neq j} g(x_{kj}) \quad (8.19)
\]

Note the similarity between Eq. (8.19) and Eq. (6.13) from Section 6.1.3. Both equations describe a competitive layer with on-center off-surrond interactions. In Section 6.1.3, we showed how the choice of the functional form of \( g(x) \) influenced the evolution of the activities of the units on the layer.

Without repeating the analysis, we assume that the values of the various parameters in Eq. (8.19) and the functional form of \( g(x) \) have been chosen to enhance the activity of the single \( F_i \) node with the largest net-input value from \( A \), according to Eq. (8.16). The activities of all other nodes are suppressed to zero. The output of this winning node is given a value of one. We can therefore express the output values of \( F_i \) nodes as

\[
u_j = f(z_{j2}) = \begin{cases} 1 & T_j = \max(T_k) \forall k \\ 0 & \text{otherwise} \end{cases}
\]  (8.20)

We need to clarify one final point concerning Figure 8.4. The processing element in that figure appears to violate our standard of a single output per node: The node sends an output of \( g(x_j) \) to the \( F_j \) units, and an output of \( f(z_{j2}) \) to the \( F_i \) units. We can reconcile this discrepancy by allowing Figure 8.4 to represent a composite structure. We can arrange for unit \( V_j \) to have the single output value \( x_{2j} \). This output can then be sent to two other processing elements; one that gives an output of \( g(x_j) \) and one that gives an output of \( f(z_{j2}) \). By assuming the existence of these intermediate nodes, or interneurons, we can avoid violating the single-output standard. The node in Figure 8.4 then represents a composite of the \( V_j \) nodes and the two intermediate nodes.

**Top-Down LTM Traces.** The equations that describe the top-down LTM traces (weights on connections from \( F_j \) units to \( F_i \) units) should be somewhat familiar from the study of Chapter 6:

\[
\dot{x}_{2j} = (-z_j + b(x_j)) f(z_{j2})
\]  (8.21)

Since \( f(z_{j2}) \) is nonzero only for one value \( \epsilon_0 \) (one \( F_i \) node, \( V_j \)), Eq. (8.21) is nonzero only for connections leading down from that winning unit. If the \( j \)th \( F_i \) node is active and the \( j \)th \( F_j \) node is also active, then \( z_j = -z_j + 1 \) and \( z_j \) asymptotically approaches one. If the \( j \)th \( F_i \) node is active and the \( j \)th \( F_j \) node is not active, then \( z_j = -z_j \) and \( z_j \) decays toward zero. We can summarize the behavior of \( z_j \) as follows:

\[
\dot{z}_j = \begin{cases} 1 & V_j \text{active and } V_i \text{ active} \\ 0 & V_j \text{ active and } V_i \text{ inactive} \end{cases}
\]  (8.22)

Recall from Eq. (8.15) that, if \( F_i \) is active, then \( v_i \) can be active only if it is receiving an input, \( C_i \), from below and a sufficient large net input, \( V_i \).
8.2 ART1

from the $F_1$ layer. Since only one $F_2$ unit is active at a time, $v_j = u_j z_j = z_j^+ v_j$.
Equation (8.15) now becomes a condition on the weight on the connection from $v_j$ to $v_j$:

$$z_{ij} > \frac{B_j - 1}{D_j}$$  \hspace{1cm} (8.23)

Unless $z_{ij}$ has a minimum value given by Eq. (8.23), it will decay toward zero even if $v_j$ is active and $v_j$ is receiving an input, $J_j$, from below.

From a practical standpoint, all top-down connection weights must be initialized to a value greater than the minimum given by Eq. (8.23) in order for any learning to take place on the network. Otherwise, any time a $v_j$ is active on $F_2$, all connections from it to any unit on $F_1$ will decay toward zero; eventually, all the connection weights will be zero and the system will be useless.

If a resonant condition is allowed to continue for an extended period, top-down weights will approach their asymptotic values of 1 or 0, according to Eq. (8.22). For the purpose of our digital simulations, we shall assume that input patterns are maintained on $F_1$ for a sufficient time to permit top-down weights to equilibrate. Thus, as soon as a resonant state is detected, we can immediately set the appropriate top-down weights to their asymptotic values. If $v_j$ is the winning $F_1$ node, then

$$z_{ij} = \begin{cases} 1 & v_j \text{ active} \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (8.24)

Weights on connections from other nodes, $v_i \neq F_1$, are not changed. We refer to this model as fast learning. Weights on bottom-up connections also have a fast-learning model, which we shall discuss next.

**Bottom-Up LTM Traces.** The equations that describe the bottom-up LTM traces are slightly more complicated than were those for the top-down LTM traces. The weight on the connection from $v_i$ on $F_1$ to $v_j$ on $F_2$ is determined by

$$z_{ij} = K f(x_{ij}) \left(1 + \frac{h(x_{ij})}{z_j^+} \right)$$  \hspace{1cm} (8.25)

where $K$ and $L$ are constants, $f(x_{ij})$ is the output of $v_j$, and $h(x_{ij})$ is the output of $v_j$. As was the case with the top-down LTM traces, the factor $f(x_{ij})$ ensures that only weights on the winning node are allowed to change. Other than that, Eq. (8.25) is an equation for a competitive system with on-center off-surround interactions. In this case, however, it is the individual weights that are competing among one another, rather than individual units.

Let’s assume $v_j$ is the winning unit on $F_2$. There are two cases to consider. If $v_j$ is active on $F_2$, then $h(x_{ij}) = 1$; otherwise, $h(x_{ij}) = 0$. Weights on connections to other $F_2$ units do not change since, for them, $f(x_{ij}) = 0$, $k \neq j$.

Before going further with Eq. (8.25), we wish to introduce a new notation. If the input pattern is I, then we shall define the magnitude of I as $|I| = \sum_i I_i$. Since $I$ is either 0 or 1, the magnitude of I is equal to the number of nonzero inputs. The output of $F_1$ is the pattern $S$. Its magnitude is $|S| = \sum_i h(x_{ij})$, which is also just the sum of nonzero outputs on $F_1$. The actual output pattern, $S$, depends on conditions in the network:

$$S = \begin{cases} I & I_1 \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (8.26)

where the superscript on $V^j$ means that $v_j$ was the winning node on $F_2$. We can interpret $V^j$ as meaning the binary pattern with a 1 at those positions where the input, $V_j$, from above is large enough to support the activation of $v_j$ whenever $v_j$ receives an input $I_j$ from below.

Since $|S| = \sum_i h(x_{ij})$, then $\sum_i h(x_{ij}) = \sum_i h(x_{ij})$, which will be equal to either $|S| \cdot 1$ or $|S|$, depending on whether $v_j$ is active. We can now summarize the three cases of Eq. (8.25) as follows:

$$z_{ij} = \begin{cases} K f(x_{ij}) & v_j \text{ is active and } x_{ij} \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (8.27)

Remember, $v_j$ can remain active only if it receives an input $I_j$ from below and a sufficiently large input $V_j$ from above. In the fast-learning case, weights on the winning $F_1$ node, $v_j$, take on the asymptotic values given by

$$z_{ij} = \begin{cases} L & v_j \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (8.28)

where we have $L > 1$ in order to keep $L - 1 > 0$.

By defining the bottom-up weights as we have in this section, we impart to the ART1 network an important characteristic that we can elucidate with the following scenario. Suppose one $F_2$ node, $v_j$, wins and learns during the presentation of an input pattern $I_1$, and another node, $v_j$, learns input pattern $I_2$. Further, let input pattern $I_1$ be a subset of pattern $I_2$ that is, $I_1 \subseteq I_2$.

When next we present either $I_1$ or $I_2$, we would like the appropriate node on $F_2$ to win. Equation (8.28) ensures that the proper node will win by keeping the weights on the node that learned the subset pattern sufficiently larger than the weights on the node that learned the superset pattern.

For the subset pattern, $I_1$, weights on connections from active $F_1$ nodes all take on the value

$$z_{ij} = L - \frac{1}{L - 1}$$  \hspace{1cm} (8.29)

where we have used the result $|S| = |I_1| \cdot V_j = |I_1|$. We have not yet shown that this result holds, but we will see shortly that it is true because of the way that we initialize the top-down weights on the $F_1$ layer.
on connections from those active \( F \) nodes. Now let's calculate the net input to the two nodes for each of the input patterns to verify that node \( v_{j1} \) wins for pattern \( I_1 \) and node \( v_{j2} \) wins for pattern \( I_2 \).

The net inputs to \( v_{j1} \) and \( v_{j2} \), for input pattern \( I \), are

\[
T_{j1} = \sum \frac{z_{j1,k} h(x_k)}{L - 1 + |I|}
\]

and

\[
T_{j2} = \sum \frac{z_{j2,k} h(x_k)}{L - 1 + |I|}
\]

respectively, where the extra subscript on \( T \) and \( h(x_k) \) refers to the number of the pattern being presented. Since \( T_{j1} \) is greater than \( T_{j2} \), node \( v_{j1} \) wins as desired. When \( I_2 \) is presented, the net inputs are

\[
T_{j1} = \sum \frac{z_{j1,k} h(x_k)}{L - 1 + |I|}
\]

and

\[
T_{j2} = \sum \frac{z_{j2,k} h(x_k)}{L - 1 + |I|}
\]

Notice that \( z_{j1,k} \) appears in the numerator in the expression for \( T_{j1} \), instead of \( z_{j2,k} \). Recall that \( v_{j2} \) has learned only the subset pattern. Therefore, bottom-up weights on that node are nonzero only for \( F \) units that represent the subset. This time, since \( T_{j1} > T_{j2} \), so \( v_{j2} \) wins as desired.

Exercise 8.3: The expression for the net-input values to the winning \( F \) units has the form:

\[
T(I) = \frac{a I}{b + |I|}
\]

where \( a \) and \( b \) are both positive. Show that this function, called a Weber function, is an increasing function of \( |I| \) for \( |I| > 1 \).

On a network with uncommitted nodes in \( F \) (i.e., nodes that have not yet participated in any learning), we must also ensure that their weights due to the initialization scheme are not so great that they accidentally win over a node that has learned the pattern. Therefore, we must keep all initial weights below a certain value. Since all patterns are a subset of the pattern containing all \( 1_k \), we should keep all initial weight values within the range

\[
0 < z_{j,k}(0) < \frac{L}{L - 1 + |I|}
\]

where \( M \) is the number of nodes on \( F \), and \( \alpha \) is the number of bottom-up connections to each \( F \) node. This condition ensures that some uncommitted node does not accidentally win over a node that has learned a particular input pattern.

Exercise 8.4: Let node \( v_{j} \) learn an input pattern \( I \) according to Eq. (8.28). Assume that all other nodes have their weights initialized according to Eq. (8.29). Prove that presentation of \( I \) will activate rather than any other uncommitted node, \( v_{j} \neq f \).

### 8.2.2 The Orienting Subsystem

The orienting subsystem in an ART network is responsible for sensing mismatches between bottom-up and top-down patterns on the \( F \) layer. Its operation can be modeled by the addition of terms to the dynamic equations that describe the activities of the \( F \) processing element. Since our discussion has evolved from the dynamic equations to their asymptotic solutions and the fast-learning case, we shall not return to the dynamic equations at this point. There are many ways to model the dynamics of the orienting subsystem. One example is the development by Ryan and Winter, and by Ryan, Winter, and Turner, listed in the Suggested Readings section at the end of this chapter. Our approach here will be to describe the details of the matching and reset process and the effects on the \( F \) units.

We can model the orienting subsystem as a single processing element, \( A \), with an output to each unit on the \( F \) layer. The inputs to \( A \) are the outputs of the \( F \) units, \( S \), and the input vector, \( I \). The weights on the connections from the input vector are all equal to a value \( P \); those on the connections from \( F \) are all equal to a value \( Q \). The net input to \( A \) is then \( PA - QS \). The output of \( A \) switches on if the net input becomes nonzero:

\[
PA - QS > 0
\]

or

\[
P|I| > Q|S| \quad \frac{P}{Q} > \frac{|S|}{|I|}
\]

The quantity \( P/Q \) is given the name vigilance parameter and is usually identified by the symbol, \( p \). Thus, activation of the orienting subsystem is prevented if

\[
\frac{|S|}{|I|} > p
\]
Recall that \( |S| = |I| \) when \( F_2 \) is inactive. The orienting subsystem must not send a reset signal to \( F_2 \) at that time. From Eq. (8.30), we get a condition on the vigilance parameter:

\[
P \leq 1
\]

We also obtain a subsequent condition on \( P \) and \( Q_c \):

\[
P \leq Q_c
\]

The value of the vigilance parameter measures the degree to which the system discriminates between different classes of input patterns. Because of the way \( p \) is defined, it implements a self-scaling pattern match. By self-scaling, we mean that the presence or absence of a certain feature in two patterns may or may not cause a reset depending on the overall importance of that feature in defining the pattern class. Figure 8.5 illustrates an example of this self-scaling.

Stated another way, the value of \( p \) determines the granularity with which input patterns are classified by the network. For a given set of patterns to be classified, a large value of \( p \) will result in finer discrimination between classes than will a smaller value of \( p \).

![Input pattern and Top-down template](image1)

(a) For a value of \( p = 0.8 \), the existence of the extra feature in the center of the top-down pattern on the right is ignored by the orienting subsystem, which considers both patterns to be of the same class. (b) For the same value of \( p \), these bottom-up and top-down patterns will cause the orienting subsystem to send a reset to \( F_2 \).

![Input pattern and Top-down template](image2)

Figure 8.5 These figures illustrate the self-scaling property of the ART networks. (a) For a value of \( p = 0.8 \), the existence of the extra feature in the center of the top-down pattern on the right is ignored by the orienting subsystem, which considers both patterns to be of the same class. (b) For the same value of \( p \), these bottom-up and top-down patterns will cause the orienting subsystem to send a reset to \( F_2 \).

Having a value of \( p \) that is less than one also permits the possibility that the top-down pattern that is coded by \( F_2 \) to represent a particular class may change as new input vectors are presented to the network. We can see this by reference back to Figure 8.5(a). We implicitly assumed that the top-down pattern on the right (with the extra dot) had been previously encoded (learned) by one of the \( F_2 \) nodes. The appearance of the bottom-up pattern on the left (without the extra dot) does not cause a reset, so a resonance is established between the \( F_1 \) layer and the winning node on \( F_2 \) that produced the top-down pattern. During this resonance, weights can be changed. The \( F_1 \) node corresponding to the feature of the center dot is not active, since that feature is missing from the input vector. According to Eq. (8.22), the top-down weight on that connection will decay away; in the fast-learning mode, we simply set it equal to zero. Similarly, the bottom-up weight on the \( F_2 \) node will decay according to Eq. (8.27).

The recoding of top-down template patterns described in the previous paragraph could lead to instabilities in the learning process. These instabilities could manifest themselves as a continual change in the class of input vectors recognized, or encoded, by each \( F_2 \) unit. Fortunately, ART was developed to combat this instability. We shall not prove it here, but it can be shown that category learning will stabilize in an ART network, after at most a few recodings. We shall demonstrate this result in the next section, where we look at a numerical example of ART processing. The stability of the learning is a direct result of the use of the 2/3 rule described earlier in this chapter.

To complete the model of the orienting subsystem, we must consider its effects on the \( F_1 \) units. When a pattern mismatch occurs, the orienting subsystem should inhibit the \( F_2 \) unit that resulted in the nonmatching pattern, and should maintain that inhibition throughout the remainder of the matching cycle.

We have now concluded our presentation of the ART network. Before proceeding on to ART2, we shall summarize the entire ART model for the asymptotic and fast-learning case.

Exercise 8.5 Assume that an input pattern has resulted in an unsuccessful search for a match through all previously encoded \( F_2 \) templates. On the next matching cycle, one of the previously uncommitted \( F_2 \) nodes will win by default. Show that this situation will not result in a reset signal from \( A \). This proof demonstrates that ART will automatically enlist a new \( F_2 \) node if the input pattern cannot be categorized with previously learned patterns.

8.2.3 ART1 Processing Summary

For this summary we shall employ the asymptotic solutions to the dynamic equations and the fast-learning case for the weights. We shall also present a step-by-step calculation showing how the ART network learns and recalls patterns.

To begin with, we must determine the size of the \( F_1 \) and \( F_2 \) layers, and the values of the various parameters in the system. Let \( M \) be the number of units...
on $F_1$ and $N$ be the number of units on $F_2$. Other parameters must be chosen according to the following constraints:

- $A_1 > 0$
- $C_1 > 0$
- $D_k > 0$
- $\max\{D_1, 1\} < B_1 < D_1 + 1$
- $L > 1$
- $0 < p < 1$

Top-down weights ($f_j \rightarrow v_i$) are initialized according to

$$z_{ji}(0) = \frac{B_1 - 1}{D_1},$$

and bottom-up weights ($v_i \rightarrow y_j$) are initialized according to

$$0 < z_j(0) = \frac{L}{L - 1 + M}.$$

The activities on $F_2$ are initialized to zero, but, according to our chosen model, $F_1$ activities are initialized to

$$x_{1i}(0) = \frac{-B_1}{1 + C_1}.$$

All input patterns must be binary: $I \in \{0, 1\}$. The magnitude of a vector is equal to the sum of the components: for example, $|I| = \sum_i I_i$. Since we will be interested in the magnitude of only binary vectors, this sum will be equal to the number of nonzero components of the vector.

We are now ready to process data on the network. We proceed according to the following algorithm:

1. Apply an input vector $I$ to $F_1$. $F_1$ activities are calculated according to

$$T_1 = I + A_1(I_1 + B_1) + C_1.$$

2. Calculate the output vector for $F_1$,

$$s_j = h(\Delta x_j) \begin{cases} T_1 & \text{if } x_{1i} > 0 \\ 0 & \text{otherwise} \end{cases} \text{ if } O(x_{1i}) < 0.$$

3. Propagate $S$ forward to $F_2$ and calculate the activities according to

$$T_2 = \sum_{i=1}^M S^i z_j.$$

4. Only the winning $F_2$ node has a nonzero output:

$$u_j = \begin{cases} 1 & T_j = \max \{T_k\} \\ 0 & \text{otherwise} \end{cases}$$

We shall assume that the winning node is $v_j$. Propagate the output from $F_2$ back to $F_1$. Calculate the net inputs from $F_2$ to the $F_1$ units:

$$V_i = \sum_{j=1}^M u_j z_{ji}.$$

6. Calculate the new activities according to

$$x_{1i} = \frac{I_1 + B_1 V_i - B_1}{1 + A_1(L_1 + D_1 V_i) + C_1}.$$

7. Determine the new output values, $S_j$, as in step 2.

8. Determine the degree of match between the input pattern and the top-down template:

$$\left|\frac{S_j}{|I|}\right| < p \text{ mark } v_j \text{ as inactive, zero the outputs of } F_2, \text{ and return to step 1 using the original input pattern. If } \left|\frac{S_j}{|I|}\right| > p, \text{ continue.}$$

9. Update bottom-up weights on $v_j$ only

$$z_{ji} = \begin{cases} \frac{1}{x_{1i}} & \text{if } v_i \text{ is active} \\ 0 & \text{if } v_i \text{ is inactive} \end{cases}$$

10. Update the top-down weights coming from $v_j$ only to all $F_1$ units:

$$z_{ji} = \begin{cases} 1 & \text{if } v_i \text{ is active} \\ 0 & \text{if } v_i \text{ is inactive} \end{cases}$$

11. Remove the input pattern. Restore all inactive $F_2$ units. Return to step 1 with a new input pattern.

To see this algorithm in operation, let's perform a step-by-step calculation for a small example problem. We will use this example to see how subset and superset patterns are handled by the network.

We shall choose the dimensions of $F_1$ and $F_2$ as $M = 5$ and $N = 6$, respectively. Other parameters in the system are $A_1 = 1, B_1 = 1.5, C_1 = 5, D_1 = 0.9, L = 1, p = 0.9$.

We initialize weights on $F_1$ units by adding a small, positive value (0.2, in this case) to $(B_1 - 1)/D_1$. Each unit on $F_1$ then has the weight vector

$$z_i = \{0.756, 0.756, 0.756, 0.756, 0.756, 0.756\}.$$
Since $L = 3$ and $M = 5$, weights on $F_3$ units are all initialized to slightly less than $L/(L - 1 + M)$. Each weight is given the value $0.429 - 0.1 = 0.329$. Each weight vector is then

$$Z_j = (0.329, 0.329, 0.329, 0.329, 0.329)^T$$

All $F_2$ units are initialized to zero activity. $F_2$ activities are initialized to

$$X(0) = (-0.25, -0.25, -0.25, -0.25, -0.25)^T$$

We can now begin actual processing. We shall start with the simple input vector,

$$I = (0,0,0,1,0)^T$$

Now, we follow the sequence of the algorithm:

1. After the input vector is applied, the $F_1$ activities become

$$X = (0,0,0,0.118,0)^T$$

2. The output vector is $S = (0,0,0,1,0)^T$.

3. Propagating this output vector to $F_2$, the net inputs to all $F_2$ units will be identical:

$$T_j = z_j \cdot S$$

Then, $T = (0.329, 0.329, 0.329, 0.329, 0.329)^T$.

4. Since all unit activities are equal, simply take the first unit as our winner. Then, the output from $F_2$ is $U = (0,0,0,0,0)^T$.

5. Propagate back to $F_1$:

$$V = z \cdot U$$

Then, $V = (0,0,0,0.756,0.756,0.756,0.756)^T$.

6. Calculate the new activity values on $F_1$ according to Eq. (8.9):

$$X = (-0.123, -0.123, 0.023, -0.123)^T$$

7. Only unit 4 has a positive activity, so the new outputs are $S = (0,0,0,1,0)^T$.

8. $|S|/|I| > 1$. Resonance has been reached.

9. Update bottom-up weights on $F_1$ unit $v$ according to the fast-learning rule.

Since $L = 3$, $F_3$ becomes $(0,0,0,1,0)^T$. Thus, unit $v$ encodes the input vector exactly. It is actually in the pattern of nonzero weights that we are interested in. The fact that the single nonzero weight matches the input value is purely coincidental, as we shall see shortly.

II. Update weights on top-down connections. Only the first weight on each $F_1$ unit changes. If each row describes the weights on one unit, then the weight matrix for $F_1$ units is

$$0 0.756 0.756 0.756 0.756 0.756 \\
0 0.756 0.756 0.756 0.756 0.756 \\
0 0.756 0.756 0.756 0.756 0.756 \\
1 0.756 0.756 0.756 0.756 0.756 \\
0 0.756 0.756 0.756 0.756 0.756$$

That completes the cycle for the first input pattern. Now let's apply a second pattern that is orthogonal to the first—namely, $I_2 = (0,0,1,0,1)^T$. We shall abbreviate the calculation somewhat.

When the input pattern is propagated to $F_3$, the resultant activities are $T = (0.000, 0.657, 0.657, 0.657, 0.657, 0.657)^T$. Unit 1 definitely loses. We select unit 2 as the winner. The output of $F_2$ is $U = (0,1,0,0,0,0)^T$. Propagating back to $F_1$, we get $V = (0.756, 0.756, 0.756, 0.756, 0.756)^T$ and $X = (-0.123, -0.123, -0.023, -0.123)^T$. The resulting output matches the input vector, $(0,0,1,0,1)$ so there is no reset.

Weights on the winning $F_2$ unit are set to $(0,0,0.756,0.756,0.756)^T$. The second weight on each $F_1$ unit is adjusted such that the new weight matrix on $F_1$ is

$$0 0.756 0.756 0.756 0.756 \\
0 0.756 0.756 0.756 0.756 \\
0 0.756 0.756 0.756 0.756 \\
0 1 0.756 0.756 0.756 0.756 \\
0 1 0.756 0.756 0.756 0.756$$

For reference, the weight matrix on $F_3$ looks like

$$0 0 0 0.75 0 0.75 \\
0 0.329 0.329 0.329 0.329 \\
0.329 0.329 0.329 0.329 0.329 \\
0.329 0.329 0.329 0.329 0.329$$

Now let's see what happens when we apply a vector that is a subset of $I_2$—namely, $I_3 = (0,0,0,1,0,1)^T$. When this pattern is propagated forward, the activities on $F_2$ become $(0.075, 0.329, 0.329, 0.329, 0.329, 0.329)^T$. Notice that the second unit wins the competition in this case so $F_2$ output is $(0,1,0,0,0,0)^T$. Going back to $F_1$, the net inputs from the top-down pattern are $V = (0,0,1,0,1)^T$. 
In this case, the equilibrium activities are \( X = (-0.25, -0.25, -0.071) \), with only one positive activity. The new output pattern is \((0, 0, 0, 0, 1)^T\) which exactly matches the input pattern, so no reset occurs.

Even though unit 2 on \( F_1 \) had previously encoded an input pattern, it gets reencoded now to match the new input pattern that is a subset of the original pattern. The new weight matrices appear as follows. For \( F_1 \),

\[
\begin{bmatrix}
0 & 0 & 0.756 & 0.756 & 0.756 & 0.756 \\
0 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\
0 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\
1 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\
0 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756 \\
0 & 0.756 & 0.756 & 0.756 & 0.756 & 0.756
\end{bmatrix}
\]

For \( F_2 \),

\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0.329 & 0.329 & 0.329 & 0.329 & 0.329 \\
0.329 & 0.329 & 0.329 & 0.329 & 0.329 \\
0.329 & 0.329 & 0.329 & 0.329 & 0.329
\end{bmatrix}
\]

If we return to the superset vector, \((0, 0, 1, 0, 1)^T\) the initial forward propagation to \( F_2 \) yields activities of \((0.000, 0.000, 0.657, 0.657, 0.657, 0.657)^T\), so unit 2 wins again. Going back to \( F_1 \), \( V = (0, 0, 0, 0, 1)^T \), and the equilibrium activities are \((-0.25, -0.25, -0.071, -0.25, -0.071, -0.25)^T\). The new outputs are \((0, 0, 0, 0, 1)^T\). This time, we get a reset signal, since \( S_{\text{reset}} / 1_{\text{reset}} = 0.5 < \rho \). Thus, unit 2 on \( F_1 \) is removed from competition, and the matching cycle is repeated with the original input vector restored on \( F_1 \).

Propagating forward a second time results in activities on \( F_2 \) of \((0.000, 0.000, 0.657, 0.657, 0.657, 0.657)^T\), where we have forced unit 2’s activity to zero as a result of sustained inhibition from the orienting subsystem. We choose unit 3 as the winner this time, and it codes the input vector. On subsequent presentations of subset and superset vectors, each will access the appropriate unit directly without the need of a search. This result can be verified by direct calculation with the example presented in this section.

**Exercise 8.6:** Verify the statement made in the previous paragraph that the presentation of \((0, 0, 1, 0, 1)^T\) and \((0, 0, 0, 0, 1)^T\) result in immediate access to the corresponding nodes on \( F_2 \) without reset by performing the appropriate calculations.

**Exercise 8.7:** What does the weight matrix on \( F_2 \) look like after unit 3 encodes the superset vector given in the example in this section?

**Exercise 8.8:** What do you expect will happen if we apply \((0, 1, 1, 0, 1)^T\) to the example network? Note that the pattern is a superset to one already encoded. Verify your hypothesis by direct calculation.

---

### 8.3 ART2

**On the surface,** ART2 differs from ART1 only in the nature of the input patterns: ART2 accepts analog (or gray-scale) vector components as well as binary components. This capability represents a significant enhancement to the system.

**Beyond the surface difference between** ART1 and ART2 lie architectural differences that give ART2 its ability to deal with analog patterns. These differences are sometimes more complex, and sometimes less complex, than the corresponding ART1 structures.

Aside from the obvious fact that binary and analog patterns differ in the nature of their respective components, ART2 must deal with additional complications. For example, ART2 must be able to recognize the underlying similarity of identical patterns superimposed on constant backgrounds having different levels. Compared in an absolute sense, two such patterns may appear entirely different when, in fact, they should be classified as the same pattern.

The price for this additional capability is primarily an increase in complexity on the \( P \) processing level. The ART2 \( P \) level consists of several sublevels and several gain-control systems. Processing on \( F_1 \) is the same for ART2 as it was for ART1. As partial compensation for the added complexity on the \( F_1 \) layer, the LTM equations are a bit simpler for ART2 than they were for ART1.

The developers of the architecture, Carpenter and Grossberg, have experimented with several variations of the architecture for ART2. At the time of this writing, that work is continuing. The architecture we shall describe here is one of several variations reported by Carpenter and Grossberg [2].

#### 8.3.1 ART2 Architecture

As we mentioned in the introduction to this section, ART2 bears a superficial resemblance to ART1. Both have an attentional subsystem and an orienting subsystem. The attentional subsystem of each architecture consists of two layers of processing elements, \( F_1 \) and \( F_2 \), and a gain-control system. The orienting subsystem of each network performs the identical function. Moreover, the basic differential equations that govern the activities of the individual processing elements are the same in both cases. To deal successfully with analog patterns in ART2, Carpenter and Grossberg have had to split the \( F_1 \) layer into a number of sublayers containing both feedforward and feedback connections. Figure 8.6 shows the resulting structure.

#### 8.3.2 Processing on \( F_1 \)

The activity of each unit on each sublayer of \( F_1 \) is governed by an equation of the form

\[
\dot{x}_k = -A x_k + (1 - B x_k) C z_k = (C + D x_k) z_k
\]

where \( A, B, C \) and \( D \) are constants. Equation (8.31) is almost identical to Eq. (8.1) from the ART1 discussion in Section 8.2.1. The only difference is...
Figure 8.6: The overall structure of the ART2 network is the same as that of ART1. The \( F_1 \) layer has been divided into six sublayers, \( w, x, u, v, p, \) and \( q \). Each node labeled \( G \) is a gain-control unit that sends a nonspecific inhibitory signal to each unit on the layer it feeds. All sublayers on \( F_1 \) as well as the \( r \) layer of the orienting subsystem, have the same number of units. Individual sublayers on \( F_1 \) are connected unit to unit; that is, the layers are not fully interconnected, with the exception of the bottom-up connections to \( F_2 \) and the top-down connections from \( F_2 \).

The appearance of the multiplicative factor in the first term on the right-hand side in Eq. (8.31). For the ART2 model presented here, we shall set \( B \) and \( C \) identically equal to zero. As with ART1, \( J^x_+ \) and \( J^q_+ \) represent net excitatory and inhibitory factors, respectively. Likewise, we shall be interested in only the asymptotic solution, so

\[
x_+ = \frac{J^x_+]_{+}}{A + D J^q_+}.
\]

The values of the individual quantities in Eq. (8.32) vary according to the sublayer being considered. For convenience, we have assembled Table 8.1, which shows all of the appropriate quantities for each \( F_1 \) sublayer, as well as the \( r \) layer of the orienting subsystem. Based on the table, the activities on each of the six sublayers on \( F_1 \) can be summarized by the following equations:

\[
w_i = I_i + a u_i.
\]

\[
x_i = \frac{w_i}{c + \|w_i\|}.
\]

\[
v_i = f(x_i) + b f(q_i).
\]

\[
u_i = \frac{v_i}{c + \|v_i\|}.
\]

\[p_i = u_i + \sum_j g(y_j) z_{ij}.
\]

\[
q_i = \frac{n}{r / \|p_i\|}.
\]

We shall discuss the orienting subsystem \( r \) layer shortly. The parameter \( e \) is typically set to a positive number considerably less than 1. It has the effect

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Layer</th>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( D )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>( x )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( u )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( v )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( p )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( q )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( r )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Table 8.1: Factors in Eq. (8.32) for each \( F_1 \) sublayer and the \( r \) layer. \( I_i \) is the \( i \)th component of the input vector. The parameters \( a, b, c, \) and \( e \) are constants whose values will be discussed in the text. \( y_i \) is the activity of the \( j \)th unit on the \( F_1 \) layer and \( g(y) \) is the output function on \( F_2 \). The function \( f(x) \) is described in the text.
of keeping the activations finite when no input is present in the system. We do not require the presence of \( e \) for this discussion so we shall set \( e = 0 \) for the remainder of the chapter.

The three gain control units in \( F_1 \) nonspecifically inhibit the \( x, u, \) and \( q \) sublayers. The inhibitory signal is equal to the magnitude of the input vector to those layers. The effect is that the activities of these three layers are normalized to unity by the gain control signals. This method is an alternative to the on-center off-surround interaction scheme presented in Chapter 6 for normalizing activities.

The form of the function, \( f(x) \), determines the nature of the contrast enhancement that takes place on \( F_1 \) (see Chapter 6). A sigmoid might be the logical choice for this function, but we shall stay with Carpenter's choice of

\[
\begin{align*}
   f(x) = \begin{cases} 
   0 & 0 \leq x \leq \theta \\
   \tanh(x) & x > \theta 
\end{cases} 
\end{align*}
\]

where \( \theta \) is a positive constant less than one. We shall use \( \theta = 0.2 \) in our subsequent examples.

It will be easier to see what happens on \( F_1 \) during the processing of an input vector if we actually carry through a couple of examples, as we did with \( \text{ART1} \). We shall set up a five-unit \( F_1 \) layer. The constants are chosen as follows:

\[
\begin{align*}
   \theta = 10, \alpha = 10, \gamma = 0.1. \text{ The first input vector is} \\
   x_1 = (0.2, 0.7, 0.1, 0.5, 0.4)^T
\end{align*}
\]

We propagate this vector through the sublayers in the order of the equations given.

As there is currently no feedback from \( u, \) \( w \) becomes a copy of the input vector:

\[
\begin{align*}
   w &= (0.2, 0.7, 0.1, 0.5, 0.4)^T \\
   x &= \text{a normalized version of the same vector:} \\
   x &= (0.205, 0.718, 0.103, 0.513, 0.410)^T
\end{align*}
\]

In the absence of feedback from \( q, \) \( v \) is equal to \( f(x) \):

\[
\begin{align*}
   v &= (0.205, 0.718, 0.0513, 0.410)^T
\end{align*}
\]

Note that the third component is now zero, since its value fell below the threshold. \( \theta \). Because \( F_3 \) is currently inactive, there is no top-down signal to \( F_3 \). In that case, all the remaining three sublayers on \( F_1 \) become copies of \( v \):

\[
\begin{align*}
   u &= (0.205, 0.718, 0.0513, 0.410)^T \\
   p &= (0.205, 0.718, 0.0513, 0.410)^T \\
   q &= (0.205, 0.718, 0.0513, 0.410)^T
\end{align*}
\]

We cannot stop here, however, as both \( u \) and \( q \) are now nonzero. Beginning again at \( w \), we find:

\[
\begin{align*}
   w &= (2.623, 7.920, 0.100, 3.567, 4.526)^T \\
   x &= (0.206, 0.722, 0.009, 0.516, 0.413)^T \\
   v &= (2.669, 7.942, 0.000, 3.573, 4.538)^T
\end{align*}
\]

where \( v \) now has contributions from the current \( x \) vector and the \( u \) vector from the previous time step. As before, the remaining three layers will be identical:

\[
\begin{align*}
   u &= (0.206, 0.723, 0.000, 0.516, 0.413)^T \\
   p &= (0.206, 0.723, 0.000, 0.516, 0.413)^T \\
   q &= (0.206, 0.723, 0.000, 0.516, 0.413)^T
\end{align*}
\]

Now we can stop because further iterations through the sublayers will not change the results. Two iterations are generally adequate to stabilize the outputs of the units on the sublayers.

During the first iteration through \( F_1 \), we assumed that there was no top-down signal from \( F_3 \) that would contribute to the activation on the \( p \) sublayer of \( F_2 \). This assumption may not hold for the second iteration. We shall see later from our study of the orienting subsystem that, by initializing the top-down weights to zero, \( z_i(0) = 0 \), we prevent reset during the initial encoding by a new \( F_2 \) unit. We shall assume that we are considering such a case in this example, so that the net input from any top-down connections sum to zero.

As a second example, we shall look at an input pattern that is a simple multiple of the first input pattern—namely,

\[
F_2 = (0.8, 2.8, 0.4, 2.0, 1.6)^T
\]

which is each element of \( I_1 \) times four. Calculating through the \( F_1 \) sublayers results in:

\[
\begin{align*}
   w &= (0.800, 2.800, 0.400, 2.000, 1.600)^T \\
   x &= (0.205, 0.718, 0.103, 0.513, 0.410)^T \\
   v &= (0.205, 0.718, 0.000, 0.513, 0.410)^T \\
   u &= (0.206, 0.722, 0.000, 0.516, 0.413)^T \\
   p &= (0.206, 0.722, 0.000, 0.516, 0.413)^T \\
   q &= (0.206, 0.722, 0.000, 0.516, 0.413)^T
\end{align*}
\]

The second time through gives:

\[
\begin{align*}
   w &= (2.623, 7.920, 0.100, 3.567, 4.526)^T \\
   x &= (0.206, 0.722, 0.038, 0.515, 0.412)^T \\
   v &= (2.669, 7.942, 0.000, 3.573, 4.538)^T \\
   u &= (0.206, 0.722, 0.000, 0.516, 0.413)^T \\
   p &= (0.206, 0.722, 0.000, 0.516, 0.413)^T \\
   q &= (0.206, 0.722, 0.000, 0.516, 0.413)^T
\end{align*}
\]
Notice that, after the v layer, the results are identical to the first example. Thus, it appears that ART2 treats patterns that are simple multiples of each other as belonging to the same class. For analog patterns, this would appear to be a useful feature. Patterns that differ only in amplitude probably should be classified together.

We can conclude from our analysis that ART2 performs a straightforward normalization and contrast-enhancement function before pattern matching is attempted. To see what happens during the matching process itself, we must consider the details of the remainder of the system.

### 8.3.3 Processing on $F_2$

Processing on $F_2$ of ART2 is identical to that performed on ART1. Bottom-up inputs are calculated as in ART1:

$$T_j = \sum_i x_{ji}$$  \hspace{1cm} (8.40)

Competition on $F_a$ results in contrast enhancement where a single winning node is chosen, again in keeping with ART1. The output function of $F_a$ is given by

$$g(x_j) = \begin{cases} d & \text{if } T_j = \max_k \{T_k\} \forall i \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (8.41)

This equation presumes that the set $\{T_k\}$ includes only those nodes that have not been reset recently by the orienting subsystem. We can now rewrite the equation for processing on the p sublayer of $F_1$ as (see Eq. 8.37)

$$p_i = \begin{cases} \mu_j & \text{if } F_j \text{ is inactive} \\ \frac{1}{d} \mu_j + d \lambda_j & \text{if the } i\text{th node on } F_p \text{ is active} \end{cases}$$

### 8.3.4 LTM Equations

The LTM equations on ART2 are significantly less complex than are those on ART1. Both bottom-up and top-down equations have the same form:

$$z_{ji} = g(x_j)/(P_i - z_{ji})$$  \hspace{1cm} (8.43)

for the bottom-up weights from $V_i$ on $F_j$ to $V_j$ on $F_a$, and

$$z_{ji} = g(x_j)/(P_i - z_{ji})$$  \hspace{1cm} (8.44)

for top-down weights from $V_j$ on $F_2$ to $q_j$ on $F_1$. If $q_j$ is the winning $F_2$ node, then we can use Eq. (8.42) in Eqs. (8.43) and (8.44) to show that

$$2z_{j1} = d(u_j + d_{z1}, - z_{j1})$$

and similarly

$$z_{j1} = d(u_j + d_{z1}, - z_{j1})$$

with all other $Z_{ij} = z_{ji} = 0$ for $j \neq J$. We shall be interested in the fast-learning case, so we can solve for the equilibrium values of the weights:

$$z_{j1} = \frac{d(u_j + d_{z1})}{1 + d}$$  \hspace{1cm} (8.45)

where we assume that $0 < d < 1$.

We shall postpone the discussion of initial values for the weights until after the discussion of the orienting subsystem.

### 8.3.5 ART2 Orienting Subsystem

From Table 8.1 and Eq. (8.32), we can construct the equation for the activities of the nodes on the r layer of the orienting subsystem:

$$r_i = \begin{cases} \frac{a}{||u|| + ||p||} & \text{if } \text{no output from } F_3 \text{ or } F_4 \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (8.46)

where we once again have assumed that $a > 0$. The condition for reset is

$$\frac{a}{||r||} > 1$$  \hspace{1cm} (8.47)

where $p$ is the vigilance parameter as in ART1.

Notice that two $F_1$ sublayers, p, and u, participate in the matching process. As top-down weights change on the p layer during learning, the activity of the units on the p layer also changes. The u layer remains stable during this process, so including it in the matching process prevents reset from occurring while learning of a new pattern is taking place.

We can rewrite Eq. (8.46) in vector form as

$$r = [u + cp]/[||u|| + ||p||]$$

Then, from $||r|| = (u \cdot p)^{1/2}$, we can write

$$r = \frac{1 + 2||p|| \cos(u, p) + ||p||^2/2}{1 + ||p||}$$  \hspace{1cm} (8.48)

where $\cos(u, p)$ is the cosine of the angle between $u$ and $p$. First, note that, if $u$ and $p$ are parallel, then Eq. (8.48) reduces to $||r|| = 1$, and there will be no reset. As long as there is no output from $F_3$, Eq. (8.37) shows that $u = p$, and there will be no reset in this case.

Suppose now that $F_3$ does have an output from some winning unit, and that the input pattern needs to be learned, or encoded, by the $F_3$ unit. We also do
not want a reset in this case. From Eq. (8.37), we see that \( p = u + \delta J \), where the \( J \)th unit on \( F_i \) is the winner and \( z_J = (z_{1J}, z_{2J}, \ldots, z_{J_iJ}) \). If we initialize all the top-down weights, \( z_{1J, 0} \) to zero, then the initial output from \( F_i \) will have no effect on the value of \( p \); that is, \( p \) will remain equal to \( u \).

During the learning process itself, \( z_J \) becomes parallel to \( u \) according to Eq. (8.45). Thus, \( p \) also becomes parallel to \( u \), and again \( ||r|| = 1 \) and there is no reset.

As with ART1, a sufficient mismatch between the bottom-up input vector and the top-down template results in a reset. In ART2, the bottom-up pattern is taken at the \( u \) sublevel of \( F_i \) and the top-down template is taken at \( p \).

Before returning to our numerical example, we must finish the discussion of weight initialization. We have already seen that top-down weights must be initialized to zero. Bottom-up weight initialization is the subject of the next section.

### 8.3.6 Bottom-Up LTM Initialization

We have been discussing the modification of LTM traces, or weights, in the case of fast-learning. Let’s examine the dynamic behavior of the bottom-up weights during a learning trial. Assume that a particular \( F_i \) node has previously encoded an input vector such that \( z_{ij} = u_{ij}/(1 - d) \), and, therefore, \( ||z_J|| = ||u_J||/(1 - d) - 100 - d \), where \( z_J \) is the vector of bottom-up weights on the \( J \)th, \( F_i \) node. Suppose the same node wins for a slightly different input pattern, one for which the degree of mismatch is not sufficient to cause a reset. Then, the bottom-up weights will be adjusted to match the new input vector. During this dynamic recording process, \( ||z_J|| \) can decrease before returning to the value \( 1.00 - d \). During this decreasing period, \( ||r|| \) will also be decreasing. If other nodes have had their weight values initialized such that \( ||z_{ij}(0)|| > 1/(1 - d) \), then the network might switch winners in the middle of the learning trial.

We must, therefore, initialize the bottom-up weight vectors such that

\[
||z_{ij}(0)|| < \frac{1}{1 - d}
\]

We can accomplish such an initialization by setting the weights to small random numbers. Alternatively, we could use the initialization

\[
z_{ij}(0) < \frac{1}{(1 - d)\sqrt{M}}
\]

This latter scheme has the appeal of a uniform initialization. Moreover, if we use the equality, then the initial values are as large as possible. Making the initial values as large as possible biases the network toward uncommitted nodes. Even if the vigilance parameter is too low to cause a reset otherwise, the network will choose an uncommitted node over a badly mismatched node. This mechanism helps stabilize the network against constant recording.

Similar arguments lead to a constraint on the parameters \( c \) and \( d_i \); namely,

\[
\frac{cd}{1/d} < 1
\]

As the ratio approaches 1, the network becomes more sensitive to mismatches because the value of \( ||r|| \) decreases to a smaller value, all other things being equal.

### 8.3.7 ART2 Processing Summary

In this section, we assemble a summary of the processing equations and constraints for the ART2 network. Following this brief list, we shall return to the numerical example that we began two sections ago.

As we did with ART1, we shall consider only the asymptotic solutions to the dynamic equations, and the fast-learning mode. Also, as with ART1, we let \( M \) be the number of units in each \( F_i \) sublayer, and \( N \) be the number of units on \( F_i \). Parameters are chosen according to the following constraints:

\[
a, b > 0 \quad 0 < d < 1 \\
\frac{c}{1 - d} < 1 \\
0 < 0 < 1 \\
0 < p < 1 \\
e < 1
\]

Top-down weights are all initialized to zero:

\[
z_{ij}(0) = 0
\]

Bottom-up weights are initialized according to

\[
z_{ij}(0) < \frac{1}{(1 - d)\sqrt{M}}
\]

Now we are ready to process data.

1. Initialize all layer and sublayer outputs to zero vectors, and establish a cycle counter initialized to a value of one.
2. Apply an input pattern, \( I \) to the \( I \) layer of \( F_i \). The output of this layer is

\[
w_i = I_i + au_j
\]

3. Propagate forward to the \( x \) sublayer.

\[
x_i = \frac{w_i}{c + ||w||}
\]
8.3 ART2

4. Propagate forward to the v sublayer.
   \[ V_i = f(z_{i1}) + b f(q_i) \]
   Note that the second term is zero on the first pass through, as q is zero at that time.

5. Propagate to the u sublayer.
   \[ U_i = \frac{V_i}{e + \|U\|} \]

6. Propagate forward to the p sublayer.
   \[ P_j = U_i + d z_{ji} \]
   where the jth node on \( F_2 \) is the winner of the competition on that layer.
   If \( F_2 \) is inactive, \( P_j = U_i \). Similarly, if the network is still in its initial configuration, \( P_j = U_i \) because \( z_{ji}(0) = 0 \).

7. Propagate to the q sublayer.
   \[ Q_i = \frac{P_i}{e + \|P\|} \]

8. Repeat steps 2 through 7 as necessary to stabilize the values on \( F_1 \).

9. Calculate the output of the \( r \) layer.
   \[ r_i = \frac{U_i + d P_i}{e + \|U\| + \|P\|} \]

10. Determine whether a reset condition is indicated. If \( p(e + \|U\|) > 1 \), then send a reset signal to \( F_2 \). Mark any active \( F_2 \) node as ineligible for competition, reset the cycle counter to one, and return to step 2. If there is no reset, and the cycle counter is one, increment the cycle counter and continue with step 11. If there is no reset, and the cycle counter is greater than one, then skip to step 14.

11. Propagate the output of the \( p \) sublayer to the \( F_1 \) layer. Calculate the net inputs to \( F_2 \).
   \[ T_j = \sum_{i=1}^{M} P_{ji} z_{ij} \]

12. Only the winning \( F_2 \) node has nonzero output.
   \[ g(T_j) = \begin{cases} 
   1 & T_j = \max \{ T_i \} \\
   0 & \text{otherwise}
   \end{cases} \]
   Any nodes marked as ineligible by previous reset signals do not participate in the competition.

13. Repeat steps 6 through 10.

14. Modify bottom-up weights on the winning \( F_1 \) unit.
   \[ z_{ji} = \frac{u_{ji}}{1 - d} \]

15. Modify top-down weights coming from the winning \( F_1 \) unit.
   \[ z_{ji} = \frac{x_{ji}}{1 - d} \]

16. Remove the input vector. Restore all inactive \( F_1 \) units. Return to step 1 with a new input pattern.

8.3.8 ART2 Processing Example

We shall be using the same parameters and input vector for this example that we used in Section 8.3.2. For that reason, we shall begin with the propagation of the \( p \) vector up to \( F_2 \). Before showing the results of that calculation, we shall summarize the network parameters and show the initialized weights.

We established the following parameters earlier: \( a = 10, b = 100, c = 0.1, d = 0.2 \). To that list we add the additional parameter, \( d = 0.9 \). We shall use \( N = 6 \) units on the \( F_1 \) layer.

The top-down weights are all initialized to zero, so \( z_{ji}(0) = 0 \) as discussed in Section 8.3.5. The bottom-up weights are initialized according to Eq. (8.49):

\[ z_{ji} = 0.5/(1 - d) = 2.236, \text{ since } M = 5. \]

Using \( I = (0.2, 0.7, 0.1, 0.5, 0.4) \) as the input vector, before propagation to \( F_2 \) we have \( P = (0.206, 0.722, 0.056, 0.043) \). Propagating this vector forward to \( F_2 \) yields a vector of activities across the \( F_2 \) units of

\[ T = (4.151, 4.51, 4.151, 4.151, 4.151, 4.151) \]

Because all of the activities are the same, the first unit becomes the winner and the activity vector becomes

\[ T = (4.151, 0, 0, 0, 0, 0) \]

and the output of the \( F_2 \) layer is the vector, \( 0.000, 0.000, 0.000, 0.000 \).

We now propagate this output vector back to \( F_1 \) and cycle through the layers again. Since the bottom-up weights are all initialized to zero, there is no change on the sublayers of \( F_2 \). We showed earlier that this condition will not result in a reset from the orienting subsystem; in other words, we have reached a resonant state. The weight vectors will now update according to the appropriate equations given previously. We find that the bottom-up weight matrix is

\[
\begin{bmatrix}
2.063 & 7.220 & 0.000 & 5.157 & 4.126 \\
2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\
2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\
2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\
2.236 & 2.236 & 2.236 & 2.236 & 2.236 \\
2.236 & 2.236 & 2.236 & 2.236 & 2.236
\end{bmatrix}
\]
8.4 The ART1 Simulator

and the top-down matrix is

\[
\begin{bmatrix}
2.06284 & 0 & 0 & 0 & 0 & 0 & 0 \\
7.21995 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.00000 & 0 & 0 & 0 & 0 & 0 & 0 \\
5.15711 & 0 & 0 & 0 & 0 & 0 & 0 \\
4.12568 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

Notice the expected similarity between the first row of the bottom-up matrix and the first column of the top-down matrix.

We shall not continue this example further. You are encouraged to build an ART2 simulator and experiment on your own.

8.4 THE ART1 SIMULATOR

In this section, we shall present the design for the ART network simulator. For clarity, we will focus on only the \textit{ART1} network in our discussion. The development of the ART2 simulator is left to you as an exercise. However, due to the similarities between the two networks, much of the material presented in this section will be applicable to the ART2 simulator. As in previous chapters, we begin this section with the development of the data structures needed to implement the simulator, and proceed to describe the pertinent algorithms. We conclude this section with a discussion of how the simulator might be adapted to implement the ART2 network.

8.4.1 ART1 Data Structures

The \textit{ART1} network is very much like the BAM network described in Chapter 4 of this text. Both networks process only binary input vectors. Both networks use connections that are initialized by performance of a calculation based on parameters unique to the network, rather than a random distribution of values. Also, both networks have two layers of processing elements that are completely interconnected between layers (the ART network augments the layers with the gain control and reset units).

However, unlike in the BAM, the connections between layers in the ART network are not bidirectional. Rather, the network units here are interconnected by means of two sets of unidirectional connections. As shown in Figure 8.7, one set ties all the outputs of the elements on layer $F_1$ to all the inputs on $F_2$, and the other set connects all $F_2$ unit outputs to inputs on layer $F_1$. Thus, for reasons completely different from those used to justify the BAM data structures, it turns out that the interconnection scheme needed to model the BAM is identical to the scheme needed to model the \textit{ART1} network.

As we saw in the case of the BAM, the data structures needed to implement this view of network processing fit nicely with the processing model provided by the generic simulator described in Chapter 4. To understand why this is so, recall the discussion in Section 4.5.2 where, in the case of the BAM, we claimed it was desirable to split the bidirectional connections between layers into two sets of unidirectional connections, and to process each individually. By organizing the network data structures in this manner, we were able to simplify the calculations performed at each network unit, in that the computer had only input values to process. In the case of the BAM, splitting the connections was done to improve performance at the expense of additional memory consumption. We can now see that there was another benefit to organizing the BAM simulator as we did: The data structures used to model the modified BAM network can be ported directly to the \textit{ART1} simulator.

By using the interconnection data structures developed for the BAM as the basis of the \textit{ART1} network, we eliminate the need to develop a new set of data structures, and now need only to define the top-level network structure used to tie all the \textit{ART1} specific parameters together. To do this, we simply construct a record containing the pointers to the appropriate layer structures and the learning parameters unique to the \textit{ART1} network. A good candidate structure is given by the following declaration:

\[
\text{record ART1 = \begin{align*}
&\text{F1 : \text{layer};} \\
&\text{F2 : \text{layer};} \\
&\text{A1, B1 : float;} \\
&\text{C1 : float;}
\end{align*}}
\]

Figure 8.7 The diagram shows the interconnection strategy needed to simulate the \textit{ART1} network. Notice that only the connections between units on the $F_1$ and $F_2$ layers are needed. The host computer can perform the function of the gain control and reset units directly, thus eliminating the need to model these structures in the simulator.
8.4 The ART1 Simulator

The ART1 Simulator  

where A, B, C, D, and L are network parameters as described in Section 8.2. You should also note that we have incorporated three items in the network structure that will be used to simplify the simulation process. These values—F2W, TNR, and magX—are used to provide immediate access to the winning unit on F2 to implement the inhibition mechanism from the attentional subsystem (A), and to store the computed magnitude of the template on layer F1, respectively. Furthermore, we have not specified the dimension of the TNR array directly, so you should be aware that we assume that this array contains as many values as there are units on layer F1. We will use the TNR array to selectively eliminate the input stimulation to each F1 layer unit, thus performing the reset function. We will elaborate on the use of this array in the following section.

As illustrated in Figure 8.8, this structure for the ART1 network provides us with access to all the network-specific data that we will require to complete our simulator, we shall now proceed to the development of the algorithms necessary to simulate the ART1 network.

8.4.2 ART1 Algorithms

As discussed in Section 8.2, it is desirable to simplify (as much as possible) the calculation of the unit activity within the network during digital simulation. For that reason, we will restrict our discussion of the ART1 algorithms to the asymptotic solution for the dynamic equations, and will implement the fast-learning case for the network weights.

Further, to clarify the implementation of the simulator, we will focus on the processing described in Section 8.2.3, and will use the data provided in that example as the basis for the algorithm design provided here. If you have not done so already, please review Section 8.2.3.

We begin by presuming that the network simulator has been constructed in memory and initialized according to the example data. We can define the algorithm necessary to perform the processing of the input vector on layer F1 as follows:

```plaintext
procedure prop to F1 (net:ART1, invvec:float[]):  
{compute outputs for layer F1 for a given input vector} 
var i : integer;  
unit : float[];  
begin 
unit = net.F1.OUTPUTS;  
for i = 1 to length(unit)  
{locate unit outputs}  
  for all F1 units} 
  if unit[i] = 0  
  {convert activation to output}  
  else unit[i] = 1;  
end if; 
end do; 
end procedure;
```

Figure 8.8 The complete data structure for the ART1 simulator is shown. Notice that we have added an additional array to contain the INHIBIT data that will be used to suppress invalid pattern matches on the F1 layer. Compare this diagram with the declaration in the text for the ART1 record, and be sure you understand how this model implements the interconnection scheme for the ART1 network.
Notice that the computation for the output of each unit on \( F_1 \) requires no modulating connection weights. This calculation is consistent with the processing model for the ART1 network, but it also is of benefit since we must use the input connection arrays to each unit on \( F_1 \) to hold the values associated with the connections from layer \( F_2 \). This makes the simulation process efficient, in that we can model two different kinds of connections (the inputs from the external world, and the top-down connections from \( F_2 \)) in the memory space required for one set of connections (the standard input connections for a unit on a layer).

The next step in the simulation process is to propagate the signals from the \( F_1 \) layer to the \( F_2 \) layer. This signal propagation is the familiar sum-of-products operation, and each unit in the \( F_2 \) layer will generate a nonzero output only if it had the highest activation level on the layer. For the ART1 simulation, however, we must also consider the effect of the inhibit signal to each unit on \( F_2 \) from the attentional subsystem. We assume this inhibition status is represented by the values in the \( ITH \) array, as initialized by a reader-provided routine to be discussed later, and further modified by network operation. We will use the values \( \{0,1\} \) to represent the inhibition status for the network, with a zero indicating the \( F_2 \) unit is inhibited, and a one indicating the unit is actively participating in the competition. Furthermore, as in the discussion of the counterpropagation network simulator, we will find it desirable to know, after the signal propagation to the competitive layer has completed, which unit won the competition so that it may be quickly accessed again during later processing.

To accomplish all of these operations, we can define the algorithm for the signal propagation to all units on layer \( F_2 \) as follows:

```plaintext
procedure prop_to_F2 (net:ART1) {
    propagate signals from layer \( F_1 \) to \( F_2 \)
    var i: integer;
    unit : 'float[]';  [pointer to \( F_2 \) unit outputs]
    inputs : 'float[]';  [pointer to \( F_1 \) unit outputs]
    connects : 'float[]';  [pointer to unit connections]
    largest : float;  [largest activation]
    winner : integer;  [index to winner]
    sum : float;  [accumulator]

    begin
        unit = net.F2.OUTS;  [locate \( F_2 \) output array]
        inputs = net.F1.OUTS;  [locate \( F_1 \) output array]
        largest = -100;  [initialize largest activation]
        for i = 1 to length(unit) do  [for all \( F_2 \) units]
            unit[i] = 0;  [deactivate unit output]
            end do;
        for i = 1 to length(unit) do  [for all \( F_2 \) units]
            sum = 0;
            for j = 1 to length(inputs) do  [for all inputs to unit]
                sum = sum + inputs[j] * connects[j];
            end do;
            sum = sum * net.ITH[i];  [inhibit if necessary]
            if (sum > largest) then
                winner = i;  [current winner]
                largest = sum;  [mark largest activation]
            end if;
        end do;
        unit[winner] = 1;  [mark winner]
        net.F2W = winner;  [remember winner]
    end procedure;
}
```

Now we have to propagate from the winning unit on \( F_2 \) back to all the units on \( F_1 \). In theory, we perform this step by computing the inner product between the connection weight vector and the vector formed by the outputs from all the units on \( F_1 \). For our digital simulation, however, we can reduce the amount of time needed to perform this propagation by limiting the calculation to only those connections between the units on \( F_1 \) and the single winning unit on \( F_2 \). Further, since the output of the winning unit on \( F_2 \) was set to one, we can again improve performance by eliminating the multiplication and using the connection weight directly. This new input from \( F_2 \) is then used to calculate a new output value for the \( F_1 \) units. The sequence of operations just described is captured in the following algorithm.

```plaintext
procedure prop_back_to_F1 (net:ART1, index:float[]);  [propagate signals from \( F_2 \) winner back to \( F_1 \) layer]
    var i: integer;  [iteration counter]
    winner : integer;  [index of winning \( F_2 \) unit]
    unit : 'float[]';  [\( F_1 \) units]
    connects : 'float[]';  [connections]
    X : float;  [new input activation]
    Vi : float;  [connection weight]
    begin
        unit = net.F1.OUTS;  [locate \( F_1 \) outputs]
        winner = net.F2W;  [set index of winning unit]
        for i = 1 to length(unit) do
            X = unit[i] + Vi * connects[i];
        end do;
    end procedure;
```
8.4 The ART1 Simulator

for i = 1 to length (unit)  
  do  
    connects = net.F1’.WEIGHTS;  
    V[i] = connects[i][winner];  
    if (X > 0)  
      then unit[i] = 1  
        (turn on unit output)  
      else unit[i] = 0  
        (if not, turn off)  
    end do;  
end procedure;

Now all that remains is to compare the output vector on F1 to the original input vector, and to update the network accordingly. Rather than trying to accomplish both of these operations in one function, we shall construct two functions (named match and update) that will determine whether a match has occurred between bottom-up and top-down patterns, and will update the network accordingly. These routines will both be constructed so that they can be called from a higher-level routine, which we call propagate. We first compute the degree to which the two vectors resemble each other. We shall accomplish this comparison as follows:

function match (net:ART1; invvec:float[]): return float;  
  input vector to activation values on F1
  var i : integer;  
  unit : output of F1
  magX : float;  
  begin
    unit = net.F1’.OUTS;  
    magX = 0;  
    for i = 1 to length (unit)  
      do
        magX = magX + unit[i];  
    end do;
    return (magX / net.L) * invvec[i];  
end function;

Once resonance has been established (as indicated by the degree of the match found between the template vector and the input vector), we must update the connection weights in order to reinforce the memory of this pattern. This update is accomplished in the following manner.

procedure update (net:ART1);  
  [update the connection weights to remember a pattern]
  var i: integer;  
  unit : output of F1
  winner : integer;  
  begins
    unit = net.F2’.OUTS;  
    winner = net.F2’.INDEX;  
    if (X > 0)  
      then unit[winner] = net.F2’.INPUTS[winner];  
        (turn on unit output)
      else unit[winner] = 0  
        (if not, turn off)  
    end do;
  end procedure;

You should note from inspection of the update algorithm that we have taken advantage of some characteristics of the ART1 network to enhance simulator performance in two ways:

- We update the connection weights to the winner on F1 by multiplying the computed value for each connection by the output of the F1 unit associated with the connection being updated. This operation makes use of the fact that the output from every F1 unit is always binary. Thus, connections are updated correctly regardless of whether they are connected to an active or inactive F1 unit.
8.4 The ART1 Simulator

- We update the top-down connections from the winning $F_1$ unit to the units on $F_2$ to contain the output value of the $F_1$ unit to which they are connected. Again, this takes advantage of the binary nature of the unit outputs on $F_2$ and allows us to eliminate a conditional test-and-branch operation in the algorithm.

With the addition of a top-level routine to tie them all together, the collection of algorithms just defined are sufficient to implement the ART1 network. We shall now complete the simulator design by presenting the implementation of the propagate routine. So that it remains consistent with our example, the top-level routine is designed to place an input vector on the network, and perform the signal propagation according to the algorithm described in Section 8.2.3.

Note that this routine uses a reader-provided routine (remove inhibit) to set all the values in the $F_1$ inhibit array to one. This routine is necessary in order to guarantee that all $F_1$ units participate in the signal-propagation activity for every new pattern presented to the network.

```plaintext
procedure propagate (net:ART1; invec:float[]);
{perform a signal propagation with learning in the network}
var done : boolean;  {true when template found}
begin
  done = false;  {start loop}
  remove_inhibit (net);  {enable all F2 units}
  while (not done)
    do
      prop_to_F1 (net, invec);  {update F1 layer}
      prop_to_F2 (net);  {determine F2 winner}
      prop_back_to_F1 (net, invec);  {send template back to F1}
      if (match (net, invec) = net.rho)
        if (pattern does not match)
          then net.INH[net.F2W] = 0  {inhibit winner}
          else done = true;  {else exit loop}
        else done = false;
      end do;
  update (net);  {reinforce template}
end procedure:
```

Note that the propagate algorithm does not take into account the case where all $F_2$ units have been encoded and none of them match the current input pattern. In that event, one of two things should occur: Either the algorithm should attempt to combine two already encoded patterns that exhibit some degree of similarity in order to free an $F_2$ unit (difficult to implement), or the simulator should allow for growth in the number of network units. This second option can be accomplished as follows:

1. When the condition exists that requires an additional $F_2$ unit, first allocate a new array of floats that contains enough room for all existing $F_2$ units, plus some number of extra units.
2. Copy the current contents of the output array to the newly created array so that the existing $n$ values occupy the first $n$ values in the new array.
3. Change the pointer in the $F_1$ record structure to locate the new array as the output array for the $F_2$ units.
4. Deallocate the old $F_2$ output array (optional).

The design and implementation of such an algorithm is left to you as an exercise.

8.5 ART2 SIMULATION

As we discussed earlier in this chapter, the ART2 model varies from the ART1 network primarily in the implementation of the $F_1$ layer. Rather than a single-layer structure of units, the $F_1$ layer contains a number of sublayers that serve to remove noise, to enhance contrast, and to normalize an analog input pattern. We shall not find this structure difficult to model, as the $F_1$ layer can be reduced to a superlayer containing many intermediate layer structures. In this case, we need only to be aware of the differences in the network structure as we implement the ART2 processing algorithms.

In addition, signals propagating through the ART2 network are primarily analog in nature, and hence must be modeled as floating-point numbers in our digital simulation. This condition creates a situation of which you must be aware when attempting to adapt the algorithms developed for the ART1 simulator to the ART2 model. Recall that, in several ART1 algorithms, we relied on the fact that network units were generating binary outputs in order to simplify processing. For example, consider the case where the input connection weights to layer $F_2$ are being modified during learning (algorithm update). In that algorithm, we multiplied the corrected connection weight by the output of the unit from the $F_1$ layer. We did this multiplication to ensure that the connections were updated to contain either the corrected connection value (if the $F_1$ unit was on) or to zero (if the $F_1$ unit was off). This approach will not work in the ART2 model, because $F_1$ layer units can now produce analog outputs.

Other than these two minor variations, the implementation of the ART2 simulator should be straightforward. Using the ART1 simulator and ART2 discussion as a guide, we leave it as an exercise for you to develop the algorithms and data structures needed to create an ART2 simulator.
Programming Exercises

8.1. Implement the ART1 simulator. Test it using the example data presented in Section 8.2.3. Does the simulator generate the same data values described in the example? Explain your answer.

8.2. Design and implement a function that can be incorporated in the propagate routine to account for the situation where all F2 units have been used and a new input pattern does not match any of the encoded patterns. Use the guidelines presented in the text for this algorithm. Show the new algorithm, and indicate where it should be called from inside the propagate routine.

8.3. Implement the ART2 simulator. Test it using the example data presented in Section 8.3.2. Does the simulator behave as expected? Describe the activity levels at each sublayer on F1 at different periods during the signal-propagation process.

8.4. Using the ART2 simulator constructed in Programming Exercise 8.3, describe what happens when all the inputs in a training pattern are scaled by a random noise function and are presented to the network after training. Does your ART2 network correctly classify the new input into the same category as it classifies the original pattern? How can you tell whether it does?

Suggested Readings

The most prolific writers of the neural-network community appear to be Stephen Grossberg, Gail Carpenter, and their colleagues. Starting with Grossberg's work in the 1970s, and continuing today, a steady stream of papers has evolved from Grossberg's early ideas. Many such papers have been collected into books. The two that we have found to be the most useful are Studies of Mind and Brain [10] and Neural Networks and Natural Intelligence [11]. Another collection is The Adaptive Brain, Volumes I and II [11, 12]. This two-volume compendium contains papers on the application of Grossberg's theories to models of vision, speech and language recognition and recall, cognitive self-organization, conditioning, reinforcement, motivation, attention, circadian rhythms, motor control, and even certain mental disorders such as amnesia. Many of the papers that deal directly with the adaptive resonance networks were coauthored by Gail Carpenter [1, 2, 3, 4, 5, 6].

A highly mathematical paper by Cohen and Grossberg proved a convergence theorem regarding networks and the latter's ability to learn patterns [8]. Although important from a theoretical standpoint, this paper is recommended for only the hardy mathematician.

Applications using ART networks often combine the basic ART structures with other, related structures also developed by Grossberg and colleagues. This fact is one reason why specific application examples are missing from this chapter. Examples of these applications can be found in the papers by Carpenter et al. [7], Kolodz and van Allen [14]. An alternate method for modeling the orienting subsystem can be found in the papers by Ryan and Winter [16] and by Ryan, Winter, and Turner [17].

Bibliography


