8 Adaptive Resonance Theory

T. Tanaka and A. Weitzenfeld[1]

8.1 Introduction

The adaptive resonance theory (ART) has been developed to avoid the stability-plasticity dilemma in competitive networks learning. The stability-plasticity dilemma addresses how a learning system can preserve its previously learned knowledge while keeping its ability to learn new patterns. ART architecture models can self-organize in real time producing stable recognition while getting input patterns beyond those originally stored.

ART is a family of different neural architectures. The first and most basic architecture is ART1 (Carpenter and Grossberg, 1987). ART1 can learn and recognize binary patterns. ART2 (Carpenter and Grossberg, 1987) is a class of architectures categorizing arbitrary sequences of analog input patterns. ART is used in modeling such as invariant visual pattern recognition (Carpenter et al 1989) where biological equivalence is discussed (Carpenter and Grossberg 1990).

An ART system consists of two subsystems, an attentional subsystem and an orienting subsystem. The stabilization of learning and activation occurs in the attentional subsystem by matching bottom-up input activation and top-down expectation. The orienting subsystem controls the attentional subsystem when a mismatch occurs in the attentional subsystem. In other words, the orienting subsystem works like a novelty detector.

An ART system has four basic properties. The first is the self-scaling computational units. The attentional subsystem is based on competitive learning enhancing pattern features but suppressing noise. The second is self-adjusting memory search. The system can search memory in parallel and adaptively change its search order. Third, already learned patterns directly access their corresponding category. Finally, the system can adaptively modulate attentional vigilance using the environment as a teacher. If the environment disapproves the current recognition of the system, it changes this parameter to be more vigilant.

There are two models of ART1, a slow-learning and a fast-learning one. The slow learning model is described by in terms of differential equations while the fast learning model uses the results of convergence in the slow learning model. In this chapter we will not show a full implementation on ART1, instead an implementation of the fast learning model will be more efficient and sufficient to show the ART1 architecture behavior.

8.2 Model Description

ART1 is the simplest ART learning model specifically designed for recognizing binary patterns. The ART1 system consists of an attentional subsystem and an orienting subsystem as shown in figure 8.1.

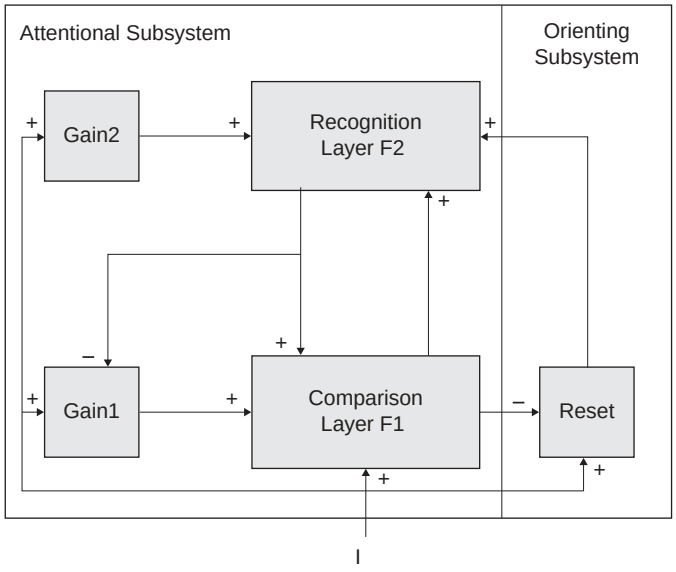

Figure 8.1
ART1 consists of an attentional subsystem and an orienting subsystem. The attentional subsystem has two short term memory (STM) stages, *F1* and *F2*. Long term memory (LTM) traces between *F1* and *F2* multiply the signal in these pathways. Gain control signals enable *F1* and *F2* to distinguish current stages of a running cycle. STM reset wave inhibits active *F2* cells when mismatches between bottom-up and top-down signals occur at *F1*.

The attentional subsystem consists of two competitive networks, the comparison layer *F1* and the recognition layer *F2*, and two control gains, Gain 1 and Gain 2. The orienting subsystem contains the reset layer for controlling the attentional subsystem overall dynamics.

The comparison layer receives the binary external input passing it to the recognition layer responsible for matching it to a classification category. This result is passed back to the comparison layer to find out if the category matches that of the input vector. If there is a match a new input vector is read and the cycle starts again. If there is a mismatch the orienting system is in charge of inhibiting the previous category in order to get a new category match in the recognition layer. The two gains control the activity of the recognition and comparison layer, respectively.

A processing element x_{1i} in layer *F1* is shown in figure 8.2.

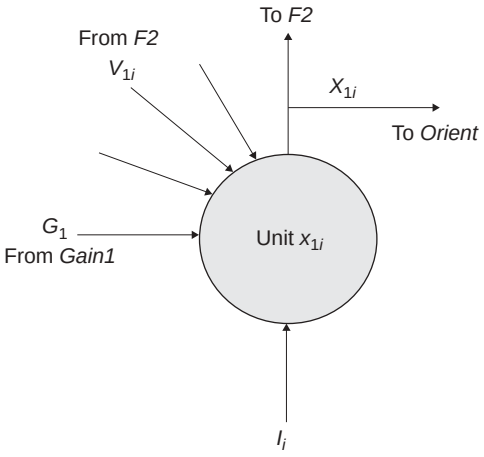

Figure 8.2
A processing unit x_{1i} in *F1* receives input from: pattern I_i, gain control signal G_1 and V_{1i} equivalent to output X_{2j} from *F2* multiplied by interconnection weight E_{21ij}. The local activity serving also as unit output is X_{1i}.

The excitatory input to x_{1i} in layer *F1* comes from three sources: (1) the external input vector I_i, (2) the control gain $G_1$and (3) the internal network input V_{1i} made of the output from *F2* multiplied appropriate connections weights. There is no inhibitory input to the neuron. The output of the neuron is fed to the *F2* layer as well as the orient subsystem.

A processing element x_{2j} in layer *F2* is shown in figure 8.3.

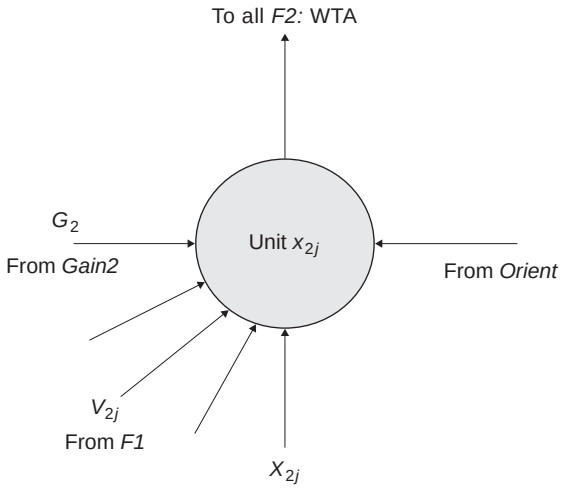

Figure 8.3
A processing element x_{2j} in *F2* receives input from: gain control signal G_2 and V_{2j} equivalent to output X_{1i} from *F1* multiplied by interconnection weight W_{12ji}. The local activity is also the unit output X_{2j}.

The excitatory input to x_{2j} in *F2* comes from three sources: (1) the orient subsystem, (2) the control gain G_2 and (3) the internal network input V_{2j} made of the output from *F1* multiplied appropriate connections weights. There is no inhibitory input to the neuron. The output of the neuron is fed to the *F1* layer as well as the Gain 1 control.

The original dynamic equations (Carpenter and Grossberg 1987) handle both binary and analog computations. We shall concentrate here on the binary model. Processing in ART1 can be divided into four phases, (1) recognition, (2) comparison, (3) search, and (4) learning.

Recognition

Initially, in the recognition or bottom-up activation, no input vector I is applied disabling all recognition in *F2* and making the two control gains, G_1 and G_2, equal to zero. This causes all *F2* elements to be set to zero, giving them an equal chance to win the subsequent recognition competition. When an input vector is applied one or more of its components must be set to one thereby making both G_1 and G_2 equal to one.

Thus, the control gain G_1 depends on both the input vector I and the output X_2 from *F2*,

$$G_1 = \begin{cases} 1 & \text{if } I \neq 0 \text{ and } X_2 = 0 \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

In other words, if there is an input vector I and *F2* is not actively producing output, then $G_1 = 1$. Any other combination of activity on I and *F2* would inhibit the gain control from exciting units on *F1*.

On the other hand, the output G_2 of the gain control module depends only on the input vector I,

$$G_2 = \begin{cases} 1 & \text{if } I \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{8.2}$$

In other words, if there exists an input vector then $G_2 = 1$ and recognition in *F2* is allowed.

Each node in *F1* receiving a nonzero input value generates an STM pattern activity greater than zero and the node's output is an exact duplicate of input vector. Since both X_{1i} and I_i are binary, their values would be either 1 or 0,

$$X_1 = I, \text{ if } G_1 = 1 \tag{8.3}$$

Each node in *F1* whose activity is beyond the threshold sends excitatory outputs to the *F2* nodes. The *F1* output pattern X_1 is multiplied by the LTM traces W_{12} connecting from *F1* to *F2*. Each node in *F2* sums up all its LTM gated signals

$$V_{2j} = \sum_i X_{1i} W_{12ji} \tag{8.4}$$

These connections represent the input pattern classification categories, where each weight stores one category. The output X_{2j} is defined so that the element that receives the largest input should be clearly enhanced. As such, the competitive network *F2* works as a winner-take-all network described by.

$$X_{2j} = \begin{cases} 1 & \text{if } G_2 = 1 \cap V_{2j} = \max_k \{V_{2k}\} \forall k \\ 0 & \text{otherwise} \end{cases} \tag{8.5}$$

The *F2* unit receiving the largest *F1* output is the one that best matches the input vector category, thus winning the competition. The *F2* winner node fires, having its value set to one, inhibiting all other nodes in the layer resulting in all other nodes being set to zero.

Comparison

In the comparison or top-down template matching, the STM activation pattern X_2 on *F2* generates a top-down template on *F1*. This pattern is multiplied by the LTM traces W_{12} connecting from *F2* to *F1*. Each node in *F1* sums up all its LTM gated signals

$$V_{1i} = \sum_j X_{2j} W_{21ij} \tag{8.6}$$

The most active recognition unit from *F2* passes a one back to the comparison layer *F1*. Since the recognition layer is now active, G_1 is inhibited and its output is set to zero.

In accordance with the "2/3" rule, stating that from three different input sources at least two are required to be active in order to generate an excitatory output, the only comparison units that will fire are those that receive simultaneous ones from the input vector and the recognition layer. Units not receiving a top down signal from *F2* must be inactive even if they receive input from below. This is summarized as follows

$$X_{1i} = \begin{cases} 1 & I_i \cap V_{1i} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{8.7}$$

If there is a good match between the top-down template and the input vector, the system becomes stable and learning may occur.

If there is a mismatch between the input vector and the activity coming from the recognition layer, this indicates that the pattern being returned is not the one desired and the recognition layer should be inhibited.

Search

The reset layer in the orienting subsystem measures the similarity between the input vector and the recognition layer output pattern. If a mismatch between them, the reset layer inhibits the *F2* layer activity. The orienting systems compares the input vector to the *F1* layer output and causes a reset signal if their degree of similarity is less than the vigilance level, where ρ is the vigilance parameter set as $0 < \rho \leq 1$.

The input pattern mismatch occurs if the following inequality is true,

$$\rho < \frac{|X_1|}{|I|} \tag{8.8}$$

If the two patterns differ by more than the vigilance parameter, a reset signal is sent to disable the firing unit in the recognition layer *F2*. The effect of the reset is to force the output of the recognition layer back to zero, disabling it for the duration of the current classification in order to search for a better match.

The parameter ρ determines how large a mismatch is tolerated. A large vigilance parameter makes the system to search for new categories in response to small difference between I and X_2 learning to classify input patterns into a large number of finer categories. Having a small vigilance parameter allows for larger differences and more input patterns are classified into the same category.

When a mismatch occurs, the total inhibitory signal from *F1* to the orienting subsystem is increased. If the inhibition is sufficient, the orienting subsystem fires and sends a reset signal. The activated signal affects the *F2* nodes in a state-dependent fashion. If an *F2* node is active, the signal through a mechanism known as gated dipole field causes a long-lasting inhibition.

When the active *F2* node is suppressed, the top-down output pattern X_2 and the top-down template V_1 are removed and the former *F1* activation pattern X_1 is generated again. The newly generated pattern X_1 causes the orienting subsystem to cancel the reset signal and bottom-up activation starts again. Since *F2* nodes having fired receive the long-lasting inhibition, a different *F2* unit will win in the recognition layer and a different stored pattern is fed back to the comparison layer. If the pattern once again does not match the input, the whole process gets repeated. .

If no reset signal is generated this time, the match is adequate and the classification is finished.

The above three stages, that is, recognition, comparison, and search, are repeated until the input pattern matches a top-down template X_1. Otherwise a *F2* node that has not learned any patterns yet is activated. In the latter case, the chosen *F2* node becomes a learned new input pattern recognition category.

Learning

The above three stages take place very quickly relative to the time constants of the learning equations of the LTM traces between *F1* and *F2*. Thus, we can assume that the learning occurs only when the STM reset and search process end and all STM patterns on *F1* and *F2* are stable.

The LTM traces from *F1* to *F2* follow the equation

$$\tau_1 \frac{dW_{12ij}}{dt} = \begin{cases} \left(1 - W_{12ij}\right)L - W_{12ij}\left(|X_1| - 1\right) & \text{if } V_{1i} \text{ and } V_{1j} \text{ are active} \\ -|X_1|W_{12ij} & \text{if only } V_{1j} \text{ is active} \\ 0 & \text{if only } V_{1j} \text{ is inactive} \end{cases} \tag{8.9}$$

where τ_1 is the time constant and L is a parameter with a value greater than one. Because time constant τ is sufficiently larger than the STM activation and smaller than the input pattern presentation, the above is a slow learning equation that converges in the fast learning equation

$$W_{12ij} = \begin{cases} \dfrac{L}{L - 1 + |X_1|} & \text{if } V_{1i} \text{ and } V_{1j} \text{ are active} \\ 0 & \text{if only } V_{1j} \text{ is active} \\ \text{no change} & \text{if only } V_{1j} \text{ is inactive} \end{cases} \tag{8.10}$$

The initial values for W_{12ij} must be randomly chosen while satisfying the inequality

$$0 < W_{12ij} < \frac{L}{L-1+|M|} \tag{8.11}$$

where M is the input pattern dimension equal to the number of nodes in *F1*.

The LTM traces from *F2* to *F1* follows the equation,

$$\tau_2 \frac{dW_{21ji}}{dt} = X_{2j}\left(-W_{21ji} + X_{1i}\right) \tag{8.12}$$

where τ_2 is the time constant and the equation is defined to converge during a presentation of an input pattern. Thus, the fast learning equation of the for W_{21ji} is

$$W_{21ji} = \begin{cases} 1 & \text{if } V_{1i} \text{ and } V_{1j} \text{ are active} \\ 0 & \text{if only } V_{1i} \text{ is inactive} \end{cases} \tag{8.13}$$

The initial value for W_{21ji} must be randomly chosen to satisfy the inequality

$$1 \geq W_{21ji}(0) > C \tag{8.14}$$

where C is decided by the slow learning equation parameters. However, all $W_{21ji}(0)$ may be set 1 in the fast learning case.

Theorems

The theorems describing ART1 behavior are described next with proofs given in Carpenter and Grossberg (1987). These theorems hold in the fast learning case with initial LTM traces satisfying constraints (10) and (14). If parameters are properly set, however, the following results also hold in the slow learning case.

(Theorem 1) Direct Access of Learned Patterns

If an *F2* node has already learned input pattern *I* as its template, then input pattern *I* activates the *F2* node at once.

The theorem states that a pattern that has been perfectly memorized by an *F2* node activates the node immediately.

(Theorem 2) Stable Category Learning

This theorem guarantees that the LTM traces W_{12ij} and W_{21ji} become stable after a finite number of learning trials in response to an arbitrary list of binary input patterns. The V_{1j} template corresponding to the *j*th *F2* node remains constant after at most *M*-1 times.

In stable states, the LTM traces W_{12ij} become $L/(L\text{-}1+M)$ if the *i*th element of the top-down template corresponding to the *j*th *F2* node is one. Otherwise, it is zero. The LTM traces W_{21ji} become one if the *i*th element of the template of corresponding to the *j*th *F2* node is one. Otherwise, it is zero.

However, theorem 2 doesn't guarantee that a perfectly coded input pattern by an *F2* node will be coded by the same *F2* node after presentation. The *F2* node may forget the input pattern in successive learning, though the template of the *F2* node continues to be a subset of the input pattern.

(Theorem 3) Direct Access after Learning Stabilizes

After learning has stabilized in response to an arbitrary list of binary input patterns, each input pattern *I* either directly activates the *F2* node which possesses the largest subset template with respect to *I*, or *I* cannot activate any *F2* node. In the latter case, *F2* contains no uncommitted nodes.

This theorem guarantees that a memorized pattern activates an *F2* node at once after learning and that all *F2* nodes have been already committed if any input patterns cannot be coded. If an input pattern list contains many different input patterns and *F2* contains fewer nodes, all input patterns cannot be coded with ρ close to 1.

However, the theorem doesn't guarantee that an input pattern having activated an *F2* node during learning should have been coded. If there are many input patterns with respect to the number of *F2* nodes, input patterns which have smaller $|X_1|$ tend to be coded while input patterns with larger $|X_1|$ tend to be coded by their subsets or not coded at all after learning.

8.3 Model Implementation

The complete model incorporates the Attentional and Orient Subsystem into a single **Art** module, as shown in figure 8.4, together with the **ArtModel** instantiating the **Art** module with the appropriate sizes for its layers.

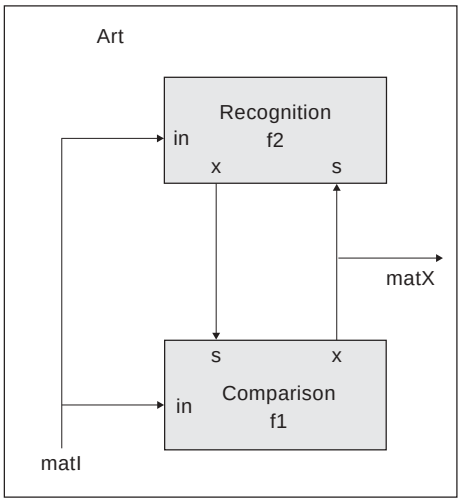

Figure 8.4
ART module containing the F2 and F2 submodules incorporating the functionality of both the Attentional and Orienting subsystems.

Art Module

Due to the limited process complexity of some of the components of the model only two submodules **F1** and **F2** are defined within the Art module. These two submodules correspond to layers *F1* and *F2* in the Attentional subsystem and include their respective gains. Also considering the simplicity of the orienting subsystem structures, it is incorporated directly into module F1.

Every simulation run initialization, corresponding to the beginning of a new epoch, a new input pattern is sent to the F1 and F2 input vector ports `in`. Since the input ports in is a vector and `matI` is a matrix we do a corresponding conversion between the two.

```
public void initRun() {
   matrixToVector(matI,in);
}
```

After completing a simulation run the **endRun** method is called, in this case we want to update the `matX` array in order to display to the user the letter output in a visually appropriate form.

```
public void endRun() {
    vectorToMatrix(x,matX);
}
```

Comparison Module

The Comparison module contains the corresponding data structure for the F1 layer including gain 1. Input layer *s* and activity layer *x* are both initialized to 0 while weights are initialized 1.0. This is all done in the **initModule** method. The **initTrain** method resets the active elements. Simulation processing is specified in the **simTrain** method as follows

```
public void simTrain() {
   if (resetActive == 1) { // input vector G1 condition, eq
(8.1)
            resetActive = 0;
         active = -1;
         x = in;
    }
    else { // eq (8.7)
       if (s.nslMax() > 0)
          s.nslMax(active);
          v = w*s; // eq (8.6)
   // this is a step function: x=nslStep(in+s,1.99)
       for (int i = 0;i < in.getSize();i++) {
             if (in[i] + v[i] >= 1.99)
                x[i] = 1.0;
             else
                x[i] = 0.0;
       }
    }
}
```

This module executes the bottom-up activation, the top-down template matching, and the STM reset and search. The activation cycle is repeated until matching is complete.

After running a complete simulation for a single pattern the **endTrain** method gets called. The module changes the LTM traces F12 and F21 after the system reaches stable responding to an input pattern. This modifies bottom-up and top-down traces F12 and F21 by the fast learning equations. The LTM learning module may be turned off when learning is unnecessary.

```
public void endTrain() {
   s.nslMax(active); // eq. (8.9)

   for (int i = 0;i < w.getRows();i++) {
      if (x[i] == 1.0)
            w[i][active] = 1.0;
      else
            w[i][active] = 0.0;
   }
}
```

Recognition Module

The Recognition module contains the corresponding data structures for the F2 layer. Simulation variables are initialized in the **initModule** method as follows:

```
public void initModule() {
// initialization of all LTM weights // eq (8.10)
    float max_value = l.getData()/(l.getData() - 1.0 +
    in.getSize());
    for (int xi = 0; xi < w.getRows(); xi++) {
        for (int yi = 0; yi < w.getCols(); yi++) {
              w[xi][yi] = uniformRandom(float(0.0),max_value);
        }
     }
}
```

The **initTrain** method resets the active elements. Simulation processing is specified in the **simTrain** method where LTM traces are multiplied to the input from F1 and F2 activation *x* is computed. The F2 unit that receives the biggest input from F1 that has not been reset is activated while the other units are deactivated.

```
public void simTrain() {
    if (s.nslSum() / in.nslSum() < rho.getData()) { // eq (8.8)
        resetY[active] = -1.0;
        active = -1;
    }
    if (active >= 0) {
        nslPrintln("Matching is passed");
        system.breakCycle();
        return;
    }
    v = w*s; // eq (8.6)
    num_type  maxvalue;
    int    i;
    active = -1;
    x = 0.0;
    float  BIG_MINUS = -1.0; // the smallest value in this
    program
    // To exclude units which have been already reset
    for (i = 0;i < resetY.getSize();i++) {
        if (resetY[i] == -1.0) {
              v[i] = BIG_MINUS;
        }
    }
    // search for the unit which receives maximum input
    maxvalue = v.nslMax();
    // In the case that there is no available unit
    if (maxvalue == BIG_MINUS) {
        active = -1;
        nslPrint("An error has occured");
        system.breakCycle();
        return;
    }
```

```
    // To find the maximum input // eq (8.5)
    for (i = 0;i < v.getSize();i++) {
       if (v[i] == maxvalue) {
             x[i] = 1.0;
             active = i;
             break;
       }
    }
    // For the error
    if (i >= v.getSize()) {
          nslPrintln("An error has occured");
          system.breakCycle();
          return;
    }
    if (active < 0) {
          nslPrintln("There are no available units");
          system.breakCycle();
          return;
    }
 }
```

After running a complete simulation for a single pattern the **endTrain** method gets called.

```
 public void endTrain() {
    nslPrintln("Top-Down Template Unit:" ,active);
    if (active < 0) {
       nslPrintln("There are no units for this input");
       system.breakCycle();
       return;
    }
    float val = l.getData() / (l.getData() - 1.0 + s.sum()); //
    eq (8.11)
    for (int i = 0; i < w.getCols(); i++) {
       if (s[i] == 1.0)
          w[active][i] = val;
       else
          w[active][i] = 0.0;
    }
 }
```

8.4 Simulation and Results[2]

The ART1 model simulation will be illustrated with character recognition example (Carpenter and Grossberg, 1987). The NSLS command file ART1.nsls contains NSL command to set parameters and prepare graphics. The parameters to be set are only the vigilance parameter and the weight initialization parameter besides the usual simulation steps specification.

```
nsl set art.f2.rho 0.7
nsl set art.f2.l 2.0
```

The system may run without learning by setting the epoch steps to 0.

A window frame with two windows inside corresponding to the input vector and *F1* activation pattern *X*, both shown as a square pattern, are opened in the simulation. A second frame with a single window shows the F2 activation pattern *X*. The latter layer is shown as a vector representing a group of classified categories.

Execution

A typical ART1 simulation session is as follows;

1. **Loading ArtModel.nsl:** "nsl source artModel.nsl."
2. **Initialization:** Execute the NSL command "nsl init." This initializes LTM traces and variables.
3. **Setting character:** Characters may be interactively fed by the user or read from a script file. For example read the "nsl source patI1.nsl" file for a single letter.
4. **Activation and Learning:** Type "nsl train" to train a single cycle of the Art model. After either the maximum number of simulation steps are executed or X_2 stabilizes, **endTrain** is executed. Learning may be disabled, only by setting the epoch step number to 1.

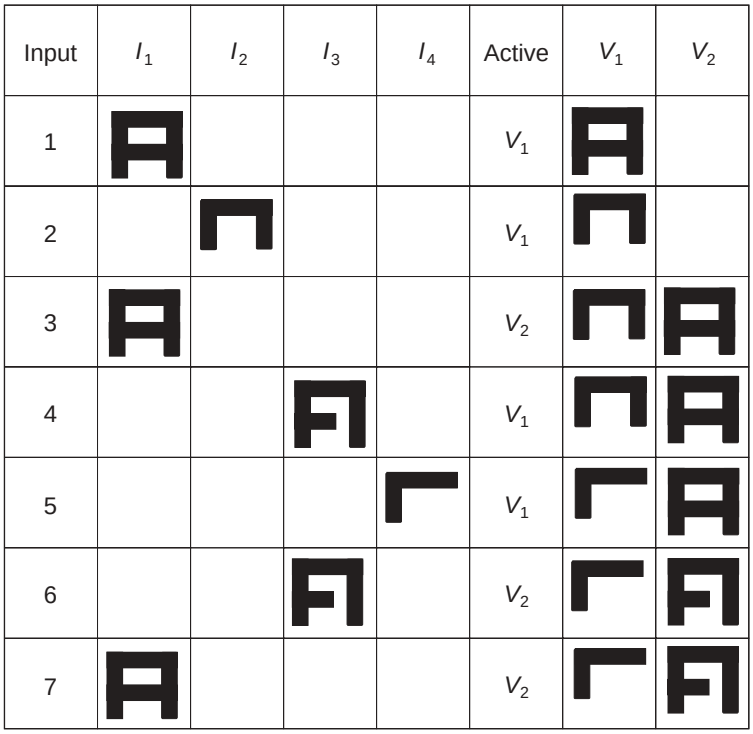

Figure 8.5
Four two-dimensional 5 by 5 (I_1, I_2, I_3 and I_4) patterns are presented to the ART1 system. The correct output is specified by the active *V* element.

Output

We give a simple simulation example in this section. Four input patterns are presented to the model for a total of seven times. The input patterns, the *F2* nodes activated by them, and top-down template of the activated *F2* nodes are shown in figure 8.5.

5. An input pattern I_1 is given in the first presentation. Because no patterns have been memorized yet, the input pattern is completely learned by an *F2* node n_1 and the top-down template of n_1 is I_1 after learning.
6. An input pattern I_2 is then given. Because I_2 is a subset of I_1, I_2 directly activates the same *F2* node n_1, and I_2 becomes a new template of n_1.

7. The input pattern I_1 is presented again in the third trial. The *F2* node n_1 is activated at first, but it is reset because its template pattern I_2 and the input pattern I_1 are very different. Thus, another *F2* node n_2 is activated and I_1 becomes its template.
8. An input pattern I_3 is given in the fourth presentation. Though I_3 looks closer to I_1 than I_2, I_3 directly accesses n_1 and the activated pattern on *F1* is I_2. The top-down template of n_1 doesn't change and it is still I_2.
9. The next input pattern I_4 activates n_1 because I_4 is a subset of the current template I_2 of n_1. Then, the template of n_1 becomes I_4 instead of I_2.
10. Next, the input pattern I_3 is given again. It activates n_1 at first, but it is reset because its current template I_4 and I_3 are very different. Thus, I_3 activates the *F2* node n_2 at the second search, and it becomes the template of the node.
11. Finally, the input pattern I_1 is given again. It directly activates the *F2* node n_2 and the activated pattern on *F1* is I_3.

The NSL simulation displays for the *V* elements are shown in figure 8.6.

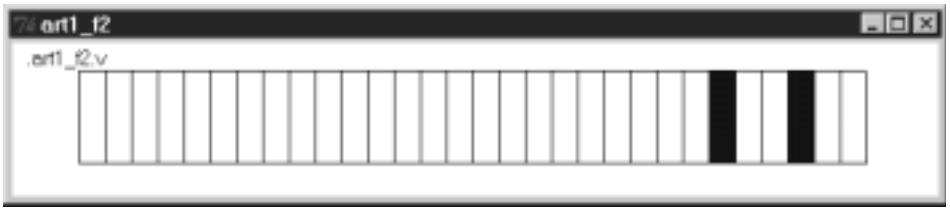

Figure 8.6
V elements in the recognition module of the ART1 system.

The NSL simulation displays comparing the letter input to the corresponding output is shown in figure 8.7. The above example illustrates some of the features of the model:

- An F2 node that memorizes an input pattern will not necessarily keep memorizing it. Though the F2 node n1 first memorizes the input pattern I1 in the above simulation, for example, the node doesn't respond to I1 in the final presentation. This means that the final stable state of the model may be largely different from early stages.
- Simpler patterns which have smaller |I|'s tend to be learned. Thus, when the number of the F2 nodes are limited, complex patterns may not be learned. Skilled adjustment of a vigilance parameter is indispensable for balanced learning.
- The criterion to classify input patterns is not intuitive. For example, the input pattern I3 is judged closer to I2 than I1.
- The previous top-down template n presented as an input pattern is not necessarily the final activation pattern on F1. This means that the model cannot restore pixels erased by noise though it can remove pixels added by noise.
- These features may be flaws of the model, but they can be taken also as good points.

8.5 Summary

Though we chose a simplified way to simulate ART1 on NSL, some interesting features of ART1 have been made clear. Different extensions can be made to the NSL implementation of ART1:

- The first extension would be a full implementation of ART1 original dynamic equations, in particular the inclusion of membrane potential equations of *F1* and *F2* nodes and the slow learning equations.
- The second extension would be to improve ART1. Some features present in our simulation are not desirable for many applications. We believe some improvements of the learning equations and matching rules would extend to further applications while keeping the basic structure of ART.

The third extension would be the implementation of other ART models. ART is a theory applying to many models, such as ART2, FUZZY-ART (Carpenter et al 1991), besides various practical applications.

A good exercise here would be to use the *Maximum Selector* model instead of the simple WTA used in ART.

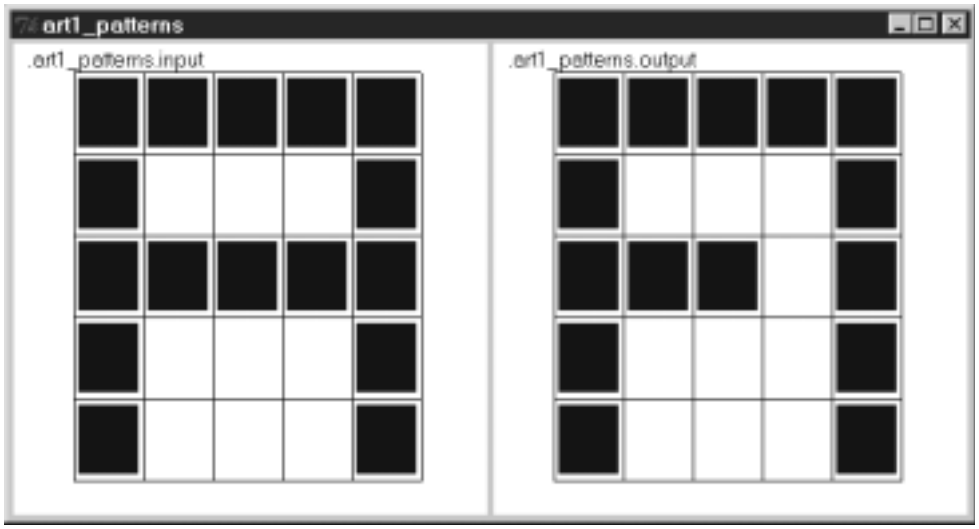

Figure 8.7
Sample letter input and output in the ART1 system.

Notes

1. A. Weitzenfeld developed the NSL3.0 version from the original NSL2.1 model implementation written by T. Tanaka as well as contributed Section 8.3 and part of Section 8.4 to this chapter.
2. The Art model was implemented and tested under NSLC.