Lecture 39

UNSUPERVISED LEARNING NETWORKS

INTRODUCTION

- Competition
- Winner take all
- Unsupervised – neural net seeks to find patterns or regularity in the input data
- Clustering nets –
    - no. of input units = no. of components in input vector
    - No. of output units = no. of clusters
    - Weight vector for an output serves as exemplar for input patterns placed in the cluster
    - During training, the net determines the output unit that is the best match for the current input vector and the weight vector for the winner is adjusted according the learning algorithm.
    - $W(new) = w(old) + \alpha[x - w(old)]$
- Determining the closest weight vector to a pattern vector

INTRODUCTION

- Determining the closest weight vector to a pattern vector
  - Euclidean distance – between weight vector and input vector
  - Dot product – of weight vector and input vector

UNSUPERVISED LEARNING

- No help from the outside.
- No training data, no information available on the desired output.
- Learning by doing.
- Used to pick out structure in the input:
    - Clustering,
    - Reduction of dimensionality → compression.
- Example: Kohonen’s Learning Law.

FEW UNSUPERVISED LEARNING NETWORKS

There exists several networks under this category, such as

- Max Net,
- Mexican Hat,
- Hamming Network,
- Kohonen Self-organizing Feature Maps,
- Adaptive Resonance Theory.
- Learning Vector Quantization,

COMPETITIVE LEARNING

- Output units compete, so that eventually only one neuron (the one with the most input) is active in response to each output pattern.

- The total weight from the input layer to each output neuron is limited. If some connections are strengthened, others must be weakened.

- A consequence is that the winner is the output neuron whose weights best match the activation pattern.

SELF-ORGANIZATION

➢ Network Organization is fundamental to the brain

- Functional structure.
- Layered structure.
- Both parallel processing and serial processing require organization of the brain.

SELF-ORGANIZING FEATURE MAP

Our brain is dominated by the cerebral cortex, a very complex structure of billions of neurons and hundreds of billions of synapses. The cortex includes areas that are responsible for different human activities (motor, visual, auditory, etc.) and associated with different sensory inputs. One can say that each sensory input is mapped into a corresponding area of the cerebral cortex. ***The cortex is a self-organizing computational map in the human brain.***

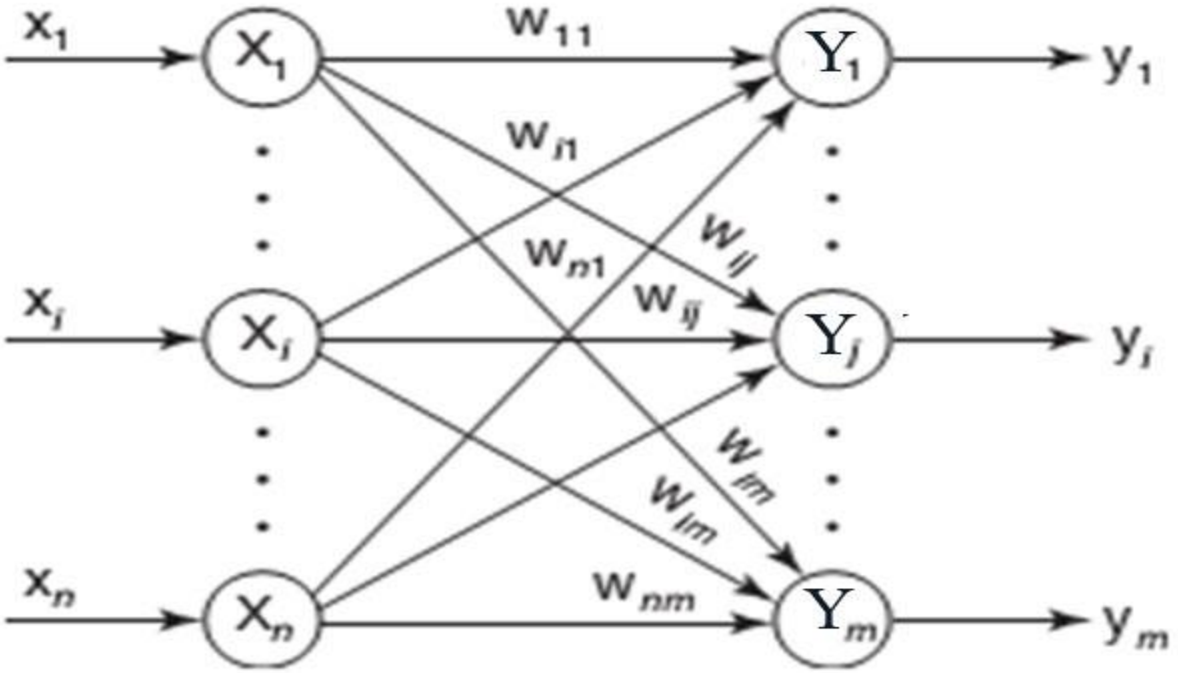
SELF-ORGANIZING NETWORKS

- Discover significant patterns or features in the input data.
- Discovery is done without a teacher.
- Synaptic weights are changed according to local rules.
- The changes affect a neuron’s immediate environment until a final configuration develops.

KOHONEN SELF-ORGANIZING FEATURE MAP (KSOFM)

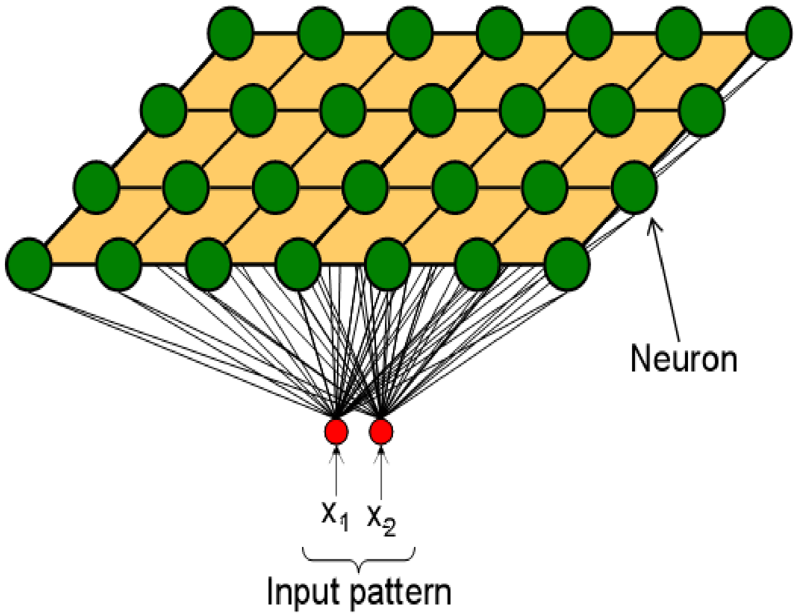
- Feature mapping : Converts the patterns of arbitrary dimensionality into a response of 1-D or 2-D arrays of neurons.
- Networks performing such mapping is called feature map.
- Capability to preserve the neighborhood relations of the input patterns i.e. topology preserving
    - M – output cluster units arranged in 1-D or 2-D array
    - Input signals are N – tuples
    - Weight vectors of cluster units serve as an exemplar vector
- Self-organization:
    - the weight vector of the cluster unit which matches the input pattern very closely is chosen the winner unit.
    - Weights are updated for winning unit and neighboring units.

ARCHITECTURE OF KSOFM



COMPETITION OF KSOFM

- Each neuron in an SOM is assigned a weight vector with the same dimensionality N as the input space.
- Any given input pattern is compared to the weight vector of each neuron and the closest neuron is declared the winner.
- The Euclidean norm is commonly used to measure distance.



CO-OPERATION OF KSOFM

- The activation of the winning neuron is spread to neurons in its immediate neighborhood.
  - This allows topologically close neurons to become sensitive to similar patterns.

- The winner's neighborhood is determined on the lattice topology.
  - Distance in the lattice is a function of the number of lateral connections to the winner.

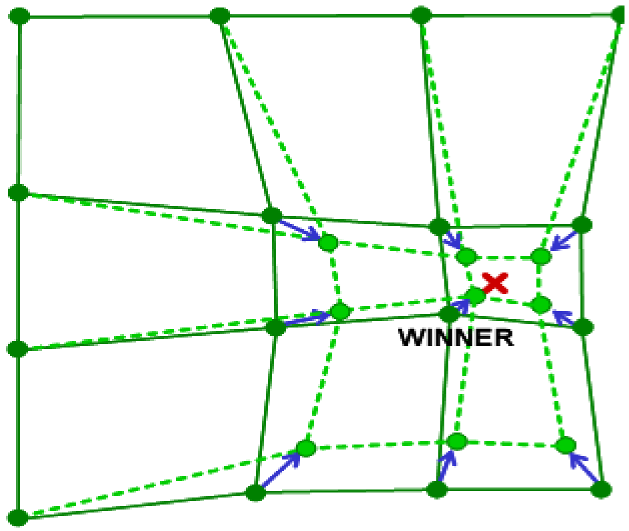
- The size of the neighborhood is initially large, but shrinks over time.
  - An initially large neighborhood promotes a topology-preserving mapping.
  - Smaller neighborhoods allow neurons to specialize in the

ADAPTATION OF KSOFM

During training, the winner neuron and its topological neighbors are adapted to make their weight vectors more similar to the input pattern that caused the activation.

Neurons that are closer to the winner will adapt more heavily than neurons that are further away.

The magnitude of the adaptation is controlled with a learning rate, which decays over time to ensure convergence of the SOM.



KSOFM ALGORITHM

Step 1: ***Initialization***

Set initial synaptic weights to small random values, say in an interval [0, 1], and assign a small positive value to the learning rate parameter α.

Step 2: ***Activation and Similarity Matching.***

Activate the Kohonen network by applying the input vector **X**, and find the winner-takes-all (best matching) neuron $j_{\mathbf{X}}$ at iteration p, using the minimum-distance Euclidean criterion

$$\boxed{j_{\mathbf{X}}(p) = \min_j \|\mathbf{X} - \mathbf{W}_j(p)\| = \left\{ \sum_{i=1}^{n} [x_i - w_{ij}(p)]^2 \right\}^{1/2},}$$

where n is the number of neurons in the input layer, and m is the number of neurons in the Kohonen layer.

Step 3: ***Learning.***

Update the synaptic weights

$$\boxed{w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)}$$

where $\Delta w_{ij}(p)$ is the weight correction at iteration p.

The weight correction is determined by the competitive learning rule:

$$\boxed{\Delta w_{ij}(p) = \begin{cases} \alpha\left[x_i - w_{ij}(p)\right], & j \in \Lambda_j(p) \\ 0, & j \notin \Lambda_j(p) \end{cases}}$$

where α is the *learning rate* parameter, and $\Lambda_j(p)$ is the neighbourhood function centred around the winner-takes-all neuron $j_{\mathbf{X}}$ at iteration p.

Step 4: ***Iteration.***

Increase iteration p by one, go back to Step 2 and continue until the minimum-distance Euclidean criterion is satisfied, or no noticeable changes occur in the feature map.

Construct a KSOFM net with two cluster units and four input units. [0 0 1 1], [1 0 0 0], [0 1 1 0], [0 0 0 1]. Assume initial learning rate of 0.5

The weight vectors for the cluster units are given by

- w1 = [0.2 0.4 0.6 0.8]
- w2 = [0.9 0.7 0.5 0.3]

Algorithm

- The radius of the neighborhood around a cluster unit also decreases as the clustering process progresses.
- The formation of a map occurs in two phases: the initial formation of the correct order and the final convergence.
- The second phase takes much longer than the first and requires a small value for the learning rate.
- Many iterations through the training set may be necessary, at least in some applications.

Example 4.4

- A Kohonen self-organizing map (SOM) to cluster four vectors.
- Let the vectors to be clustered be:

$$(1, 1, 0, 0); (0, 0, 0, 1); (1, 0, 0, 0); (0, 0, 1, 1).$$

- The maximum $m = 2.$ clusters to be formed is
- Suppose the l $\alpha(0) = .6,$ ıetric decrease) is:

$$\alpha(t + 1) = .5\,\alpha(t).$$

Example 4.4

- With only two clusters available, the neighborhood of node J (Step 4) is set so that only one cluster updates its weights at each step (i.e., R = 0).
- Step 0. Initial weight matrix: $$\begin{bmatrix} .2 & .8 \\ .6 & .4 \\ .5 & .7 \\ .9 & .3 \end{bmatrix}.$$
- Initial radius: R=0. $\alpha(0) = 0.6.$
- Initial learning rate:
- Step 1. Begin training.
- Step 2. For the first vector, (1, 1, 0, 0), do Steps 3-5.

Example 4.4

- Step 3.

$$D(1) = (.2 - 1)^2 + (.6 - 1)^2 + (.5 - 0)^2 + (.9 - 0)^2 = 1.86;$$

$$D(2) = (.8 - 1)^2 + (.4 - 1)^2 + (.7 - 0)^2 + (.3 - 0)^2 = 0.98.$$

- Step 4. The input vector is closest to output node 2, so J = 2.
- Step 5. Tl pdated:

$$w_{i2}(\text{new}) = w_{i2}(\text{old}) + .6\,[x_i - w_{i2}(\text{old})]$$
$$= .4\, w_{i2}(\text{old}) + .6\, x_i.$$

Example 4.4

- This gives the weight matrix $\begin{bmatrix} .2 & .92 \\ .6 & .76 \\ .5 & .28 \\ .9 & .12 \end{bmatrix}$

- Step 2. For the second vector, (0, 0 , 0 , 1) , do Steps 3-5

- Step 3.

$$D(1) = (.2 - 0)^2 + (.6 - 0)^2 + (.5 - 0)^2 + (.9 - 1)^2 = 0.66;$$

$$D(2) = (.92 - 0)^2 + (.76 - 0)^2 + (.28 - 0)^2 + (.12 - 1)^2 = 2.2768.$$

Example 4.4

- Step 4. The input vector is closest to output node 1 , so
- J=1.
- Step 5. Update the [illegible]mn of the weight matrix:

$$\begin{bmatrix} .08 & .92 \\ .24 & .76 \\ .20 & .28 \\ .96 & .12 \end{bmatrix}$$

Example 4.4

- Step 2. For the third vector, (1 , 0, 0, 0), do Steps 3-5.
- Step 3.

$$D(1) = (.08 - 1)^2 + (.24 - 0)^2 + (.2 - 0)^2 + (.96 - 0)^2 = 1.8656;$$

$$D(2) = (.92 - 1)^2 + (.76 - 0)^2 + (.28 - 0)^2 + (.12 - 0)^2 = 0.6768.$$

- Step 4. The input vec sest to output node 2, so

$$\begin{bmatrix} .08 & .968 \\ .24 & .304 \\ .20 & .112 \\ .96 & .048 \end{bmatrix}$$

- J = 2.

Example 4.4

- Step 2. For the fourth vector, (0 , 0, 1, 1), do Steps 3-5.
- Step 3.

$$D(1) = (.08 - 0)^2 + (.24 - 0)^2 + (.2 - 1)^2 + (.96 - 1)^2 = 0.7056;$$

$$D(2) = (.968 - 0)^2 + (.304 - 0)^2 + (.112 - 1)^2 + (.048 - 1)^2 = 2.724.$$

- Step 4. The input ve $\begin{bmatrix} .032 & .968 \\ .096 & .304 \\ .680 & .112 \\ .984 & .048 \end{bmatrix}$ est to output node 1, so
- J = 1.

Example 4.4

- Step 6. Reduce the learning rate:

$$\alpha = .5\,(0.6) = .3$$

- The weight update equations are now:

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + .3\,[x_i - w_{ij}(\text{old})]$$
$$= .7w_{ij}(\text{old}) + .3x_i.$$

- Modifying the adjustment procedure for the learning rate so that it decreases geometrically from .6 to .01 over 100 iterations (epochs) gives the following results:

Example 4.4

Iteration 2: Weight matrix: $\begin{bmatrix} .0053 & .9900 \\ -.1700 & .3000 \\ .7000 & .0200 \\ 1.0000 & .0086 \end{bmatrix}$

Iteration 10: Weight matrix: $\begin{bmatrix} 1.5e\text{-}7 & 1.0000 \\ 4.6e\text{-}7 & .3700 \\ .6300 & 5.4e\text{-}7 \\ 1.0000 & 2.3e\text{-}7 \end{bmatrix}$

Iteration 50: Weight matrix: $\begin{bmatrix} 1.9e\text{-}19 & 1.0000 \\ 5.7e\text{-}15 & .4700 \\ .5300 & 6.6e\text{-}15 \\ 1.0000 & 2.8e\text{-}15 \end{bmatrix}$

Iteration 100: Weight matrix: $\begin{bmatrix} 6.7e\text{-}17 & 1.0000 \\ 2.0e\text{-}16 & .4900 \\ .5100 & 2.3e\text{-}16 \\ 1.0000 & 1.0e\text{-}16 \end{bmatrix}$

Example 4.4

- These weight matrices appear to be converging to the matrix

$$\begin{bmatrix} 0.0 & 1.0 \\ 0.0 & 0.5 \\ 0.5 & 0.0 \\ 1.0 & 0.0 \end{bmatrix}$$

- the first column of which is the average of the two vectors placed in cluster 1 and the second column of which is the average of the two vectors placed in cluster 2.

ADAPTIVE RESONANCE THEORY (ART) NETWORK

- **Adaptive Resonance Theory** (ART) is a family of algorithms for unsupervised learning developed by Carpenter and Grossberg.

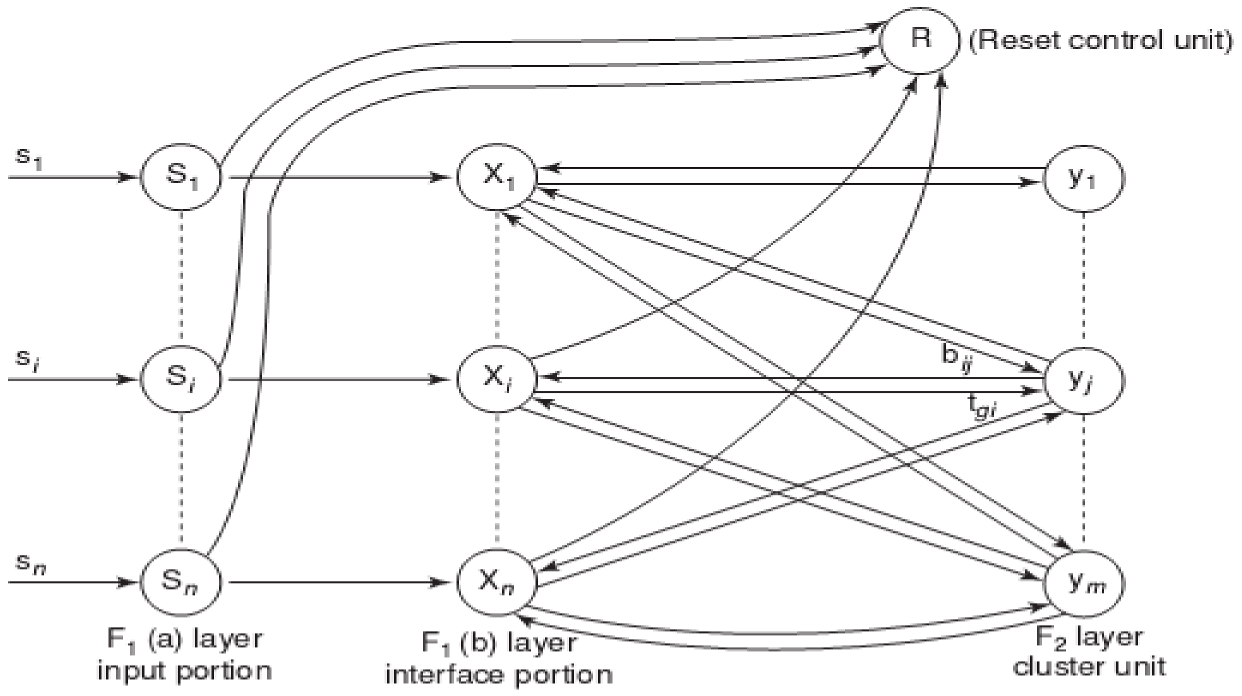
- **ART** is similar to many iterative clustering algorithms where each pattern is processed by

  - finding the "nearest" cluster (a.k.a. prototype or template) to that exemplar (desired).
  - updating that cluster to be "closer" to the exemplar.

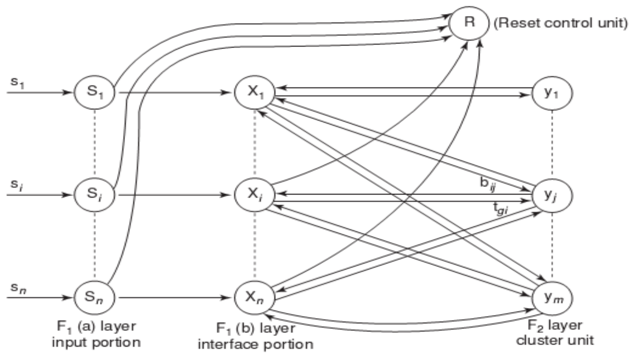
ADAPTIVE RESONANCE THEORY (ART) NETWORK

- Allow user to control the degree of similarity of pattern place on the same cluster.
- The *relative similarity* of input pattern with the weight vector is used rather than the *absolute difference*.
- Stability: a pattern does not oscillate among clusters.
- Plasticity: respond to a new pattern equally well at any stage of learning.
- ART nets are designed to be both stable and plastic.
- ART nets are also structured such that neural processes can control intricate operations of the net. This requires a number of neurons in addition to the input units, cluster units and units for the comparison of the input signal with the cluster unit's weight.

BASIC ARCHITECTURE OF ART1



Architecture

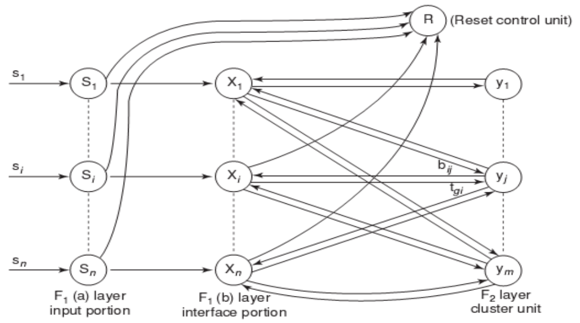


3 groups of neurons:

1. An input processing field (F1 layer).
2. Cluster unit (F2 layer).
3. A mechanism to control degree of similarity of patterns placed on the same cluster.

- ➢ F1 layer consists of
  1. Input portion F1(a)
  2. Interface portion F1(b) : combines the signals from input portion & F2 layer to compare the similarity of the input signal to the weight vector of the selected cluster unit
- ➢ F2 node is in one of the three states:
  - ➢ Active ("on", activation = d; d = 1 for ART1)
  - ➢ Inactive ("off", activation=0; but available to participate)
  - ➢ Inhibited ("off", activation = 0 & prevented for participate)
- ➢ Control of similarity: two sets of connections between each unit in the interface portion and cluster unit.
  - ➢ Bottom-up weights $\mathbf{b_{ij}}$ from ith F1 unit to jth F2 unit
  - ➢ Top-down weights $\mathbf{t_{ji}}$ from jth F2 unit to ith F1 unit

Operation

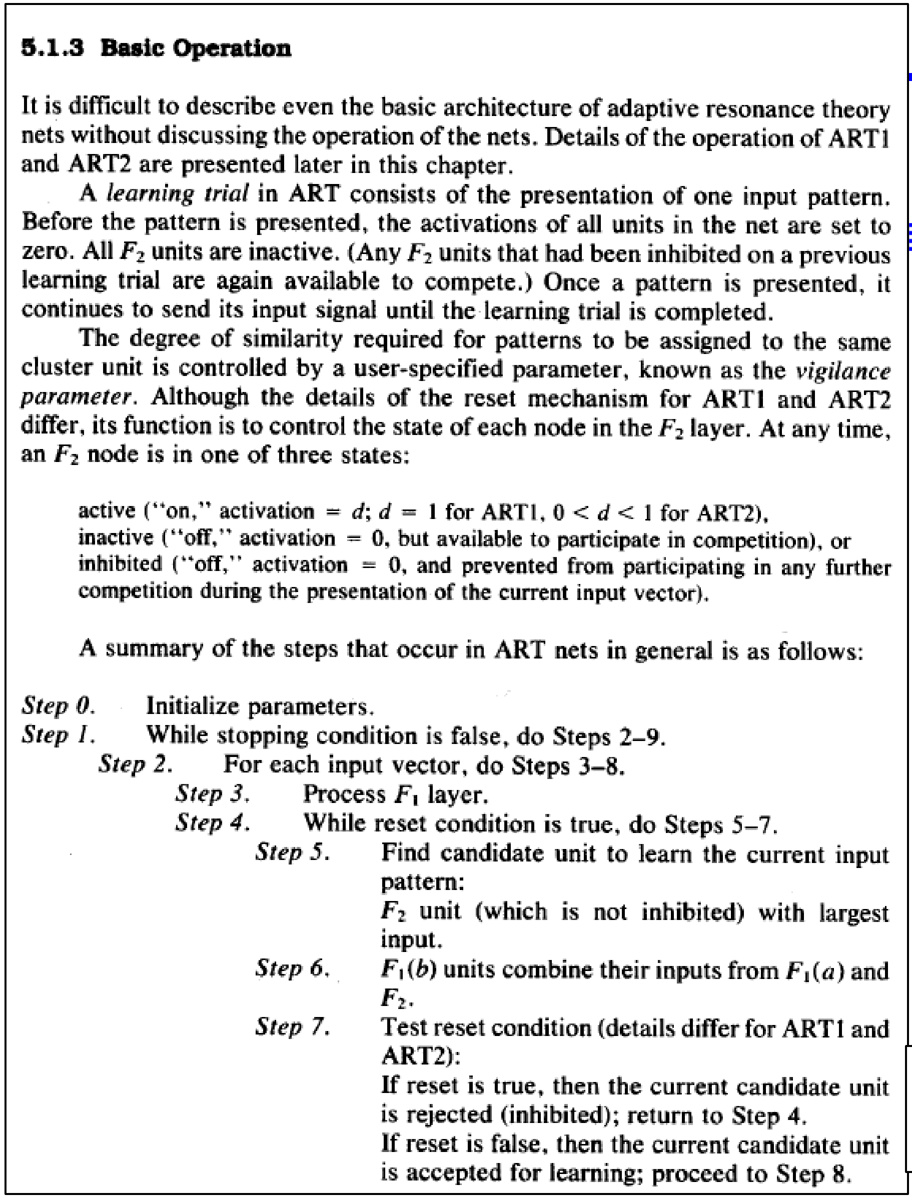


- the F2 layer is competitive layer.
  - Cluster unit with largest net input becomes the candidate to learn the input pattern.
  - activations of all other F2 units are set to zero.
- interface units combine the information from input & cluster units
- whether or not the cluster unit is allowed to learn the input patterns depends on how similar its weight vector is to the input vector and is decided by reset unit.
- if a cluster is not allowed to learn it is inhibited and a new cluster unit is selected as the candidate.
- degree of similarity is controlled by user defined vigilance parameter
- a pattern once presented continues to send its input signal until learning trial is completed.

ARCHITECTURES OF ART NETWORK

- **ART1**, designed for binary features.
- **ART2**, designed for continuous (analog) features.
- **ARTMAP**, a supervised version of ART.

Basic Architecture



5.1.3 Basic Operation

It is difficult to describe even the basic architecture of adaptive resonance theory nets without discussing the operation of the nets. Details of the operation of ART1 and ART2 are presented later in this chapter.

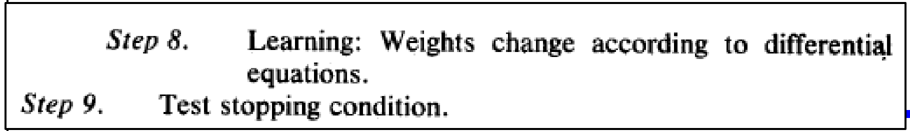
A *learning trial* in ART consists of the presentation of one input pattern. Before the pattern is presented, the activations of all units in the net are set to zero. All F_2 units are inactive. (Any F_2 units that had been inhibited on a previous learning trial are again available to compete.) Once a pattern is presented, it continues to send its input signal until the learning trial is completed.

The degree of similarity required for patterns to be assigned to the same cluster unit is controlled by a user-specified parameter, known as the *vigilance parameter*. Although the details of the reset mechanism for ART1 and ART2 differ, its function is to control the state of each node in the F_2 layer. At any time, an F_2 node is in one of three states:

active ("on," activation $= d$; $d = 1$ for ART1, $0 < d < 1$ for ART2),
inactive ("off," activation $= 0$, but available to participate in competition), or
inhibited ("off," activation $= 0$, and prevented from participating in any further competition during the presentation of the current input vector).

A summary of the steps that occur in ART nets in general is as follows:

```
Step 0.   Initialize parameters.
Step 1.   While stopping condition is false, do Steps 2–9.
    Step 2.   For each input vector, do Steps 3–8.
        Step 3.   Process $F_1$ layer.
        Step 4.   While reset condition is true, do Steps 5–7.
            Step 5.   Find candidate unit to learn the current input
                      pattern:
                      $F_2$ unit (which is not inhibited) with largest
                      input.
            Step 6.   $F_1(b)$ units combine their inputs from $F_1(a)$ and
                      $F_2$.
            Step 7.   Test reset condition (details differ for ART1 and
                      ART2):
                      If reset is true, then the current candidate unit
                      is rejected (inhibited); return to Step 4.
                      If reset is false, then the current candidate unit
                      is accepted for learning; proceed to Step 8.
```



```
        Step 8.   Learning: Weights change according to differential
                  equations.
Step 9.   Test stopping condition.
```

Learning trial: presentation of one pattern (Step 2)

Resonance: in order to learn an input vector by a cluster unit, the maintenance of the top-down and bottom-up signals for an extended period so that the weight changes occur.

Learning:
1. Fast Learning
2. Slow Learning

Basic Architecture

Fast Learning :

- weight updates occur rapidly during resonance as compared to the duration of time a pattern is presented in a trial
- Weights reach equilibrium faster
- Less presentations of patterns are required for learning as compared to slow learning
- Net is considered stabilized when each pattern chooses the correct cluster unit when it is presented
- ART1 – since the patterns are binary, weights associated with each cluster unit also stabilize fast. Equilibrium weights are easy to determine and differential equations control of weight updates is not required.
- ART2 – weights produced by fast learning continue to change each time pattern is presented. Equilibrium weights are not as easy to determine as in ART1. Net stabilizes after a few presentations of each pattern, but the differential equations for weight updates depend on the activation of units whose activations chan

Slow Learning:

- weight changes occur slowly relative to the duration of time a pattern is presented in a trial
- Weights reach equilibrium slowly
- More presentations of patterns are required for learning as compared to fast learning
- Weight changes do not reach equilibrium in any particular learning trial and more trials are required before the net stabilizes
- ART1 – Theoretically slow learning is possible, however generally fast learning is used
- ART2 – weights produced by slow learning are much more appropriate than those produced by fast learning

ART1 UNITS

ART1 is designed to cluster binary input vectors and has direct control of the degree of similarity among patterns placed on he same cluster unit.

ART1 Network is made up of two units

- Computational units
  - Input unit (F1 unit – input and interface).
  - Cluster unit (F2 unit – output).
- Supplemental units
  - One reset control unit. (controls degree of similarity).
  - Two gain control units.

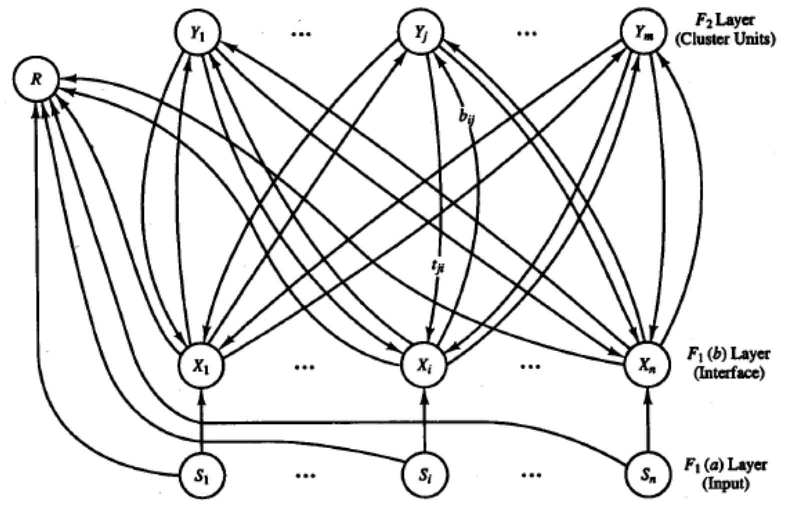
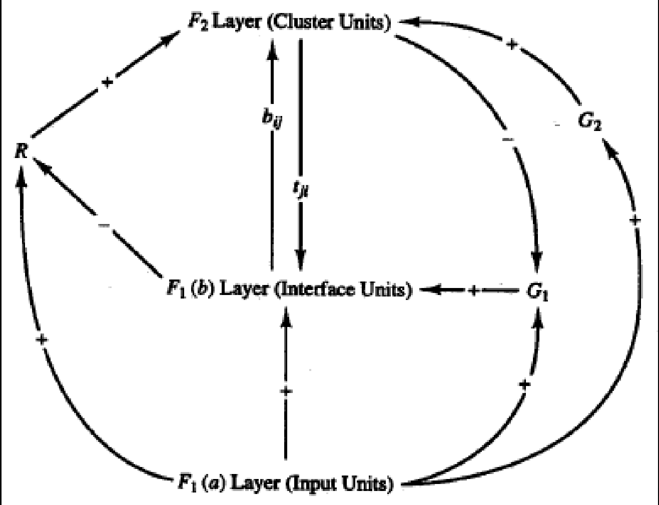
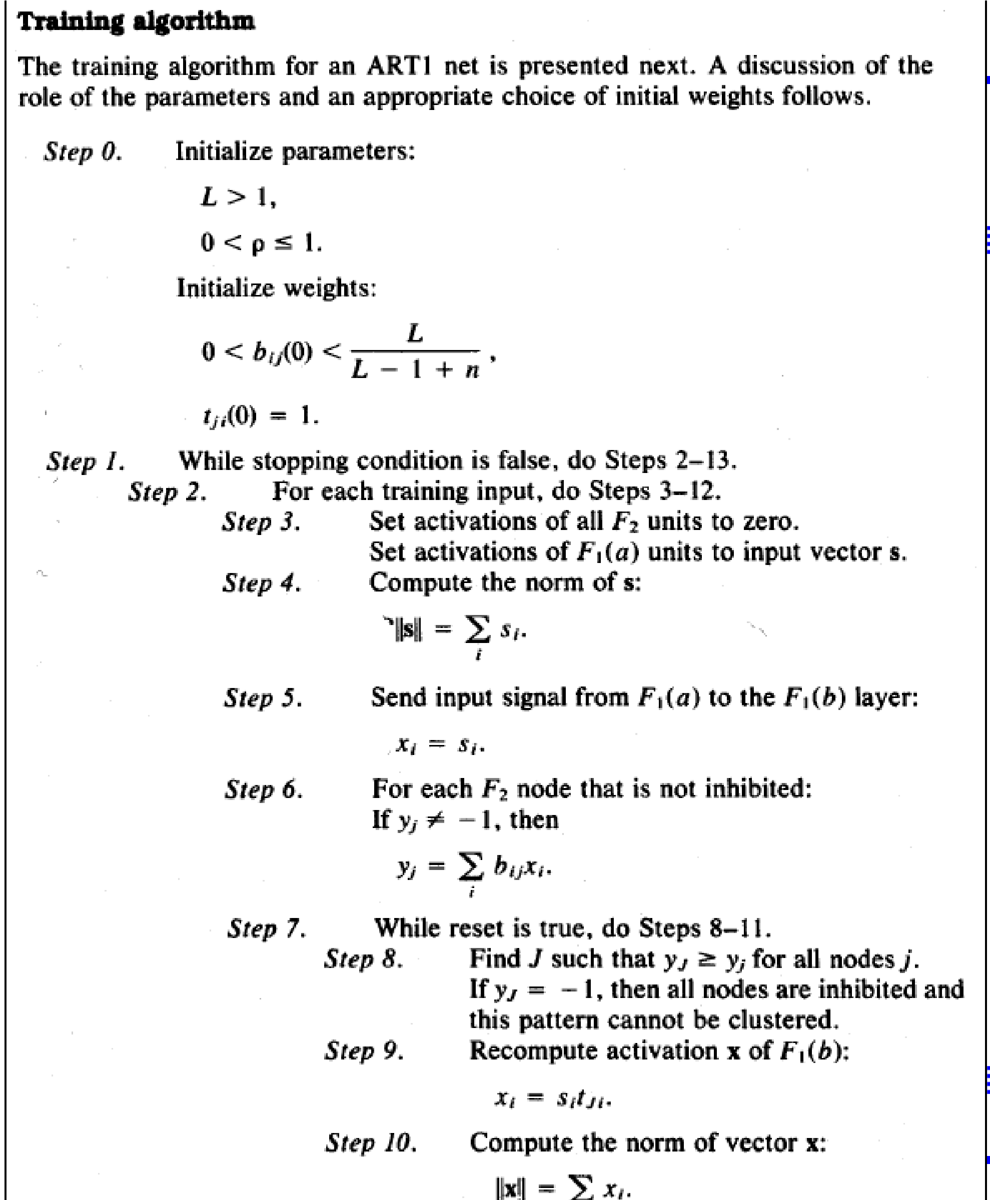


Figure 5.1 Basic structure of ART1.

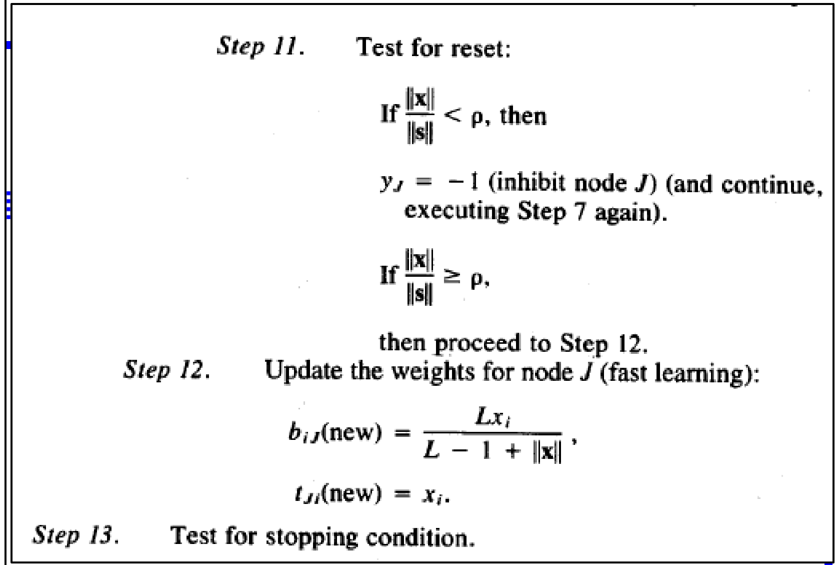

F_2 Layer (Cluster Units)
R
b_{ij}
t_{ji}
G_2
G_1
F_1 (b) Layer (Interface Units)
F_1 (a) Layer (Input Units)
+
−



Training algorithm

The training algorithm for an ART1 net is presented next. A discussion of the role of the parameters and an appropriate choice of initial weights follows.

```
Step 0.  Initialize parameters:
             $L > 1,$
             $0 < \rho \le 1.$
         Initialize weights:
             $0 < b_{ij}(0) < \frac{L}{L - 1 + n},$
             $t_{ji}(0) = 1.$
Step 1.  While stopping condition is false, do Steps 2–13.
    Step 2.  For each training input, do Steps 3–12.
        Step 3.  Set activations of all $F_2$ units to zero.
                 Set activations of $F_1(a)$ units to input vector s.
        Step 4.  Compute the norm of s:
                     $\|\mathbf{s}\| = \sum_i s_i.$
        Step 5.  Send input signal from $F_1(a)$ to the $F_1(b)$ layer:
                     $x_i = s_i.$
        Step 6.  For each $F_2$ node that is not inhibited:
                 If $y_j \neq -1$, then
                     $y_j = \sum_i b_{ij} x_i.$
        Step 7.  While reset is true, do Steps 8–11.
            Step 8.  Find $J$ such that $y_J \ge y_j$ for all nodes $j$.
                     If $y_J = -1$, then all nodes are inhibited and
                     this pattern cannot be clustered.
            Step 9.  Recompute activation x of $F_1(b)$:
                         $x_i = s_i t_{Ji}.$
            Step 10. Compute the norm of vector x:
                         $\|\mathbf{x}\| = \sum_i x_i.$
```



```
            Step 11. Test for reset:
                         If $\frac{\|\mathbf{x}\|}{\|\mathbf{s}\|} < \rho$, then
                         $y_J = -1$ (inhibit node $J$) (and continue,
                           executing Step 7 again).
                         If $\frac{\|\mathbf{x}\|}{\|\mathbf{s}\|} \ge \rho$,
                         then proceed to Step 12.
        Step 12. Update the weights for node $J$ (fast learning):
                     $b_{iJ}(\text{new}) = \frac{L x_i}{L - 1 + \|\mathbf{x}\|},$
                     $t_{Ji}(\text{new}) = x_i.$
    Step 13. Test for stopping condition.
```

Example 5.1 An ART1 net to cluster four vectors: low vigilance

The values and a description of the parameters in this example are:

n	$= 4$	number of components in an input vector;
m	$= 3$	maximum number of clusters to be formed;
ρ	$= 0.4$	vigilance parameter;
L	$= 2$	parameter used in update of bottom-up weights;
$b_{ij}(0)$	$= \dfrac{1}{1+n}$	initial bottom-up weights (one-half the maximum value al lowed);
$t_{ji}(0)$	$= 1$	initial top-down weights.

The example uses the ART1 algorithm to cluster the vectors (1, 1, 0, 0), (0, 0, 0, (1, 0, 0, 0), and (0, 0, 1, 1), in at most three clusters.

Application of the algorithm yields the following:

Step 0. Initialize parameters:

$L = 2,$

$\rho = 0.4;$

Initialize weights:

$b_{ij}(0) = 0.2,$

$t_{ji}(0) = 1.$

Step 1. Begin computation.

Step 2. For the first input vector, (1, 1, 0, 0), do Steps 3–12.

Step 3. Set activations of all F_2 units to zero.
Set activations of $F_1(a)$ units to input vector
$\mathbf{s} = (1, 1, 0, 0).$

Step 4. Compute norm of $\mathbf{s}$:

$\|\mathbf{s}\| = 2.$

Step 5. Compute activations for each node in the F_1 layer:

$\mathbf{x} = (1, 1, 0, 0).$

Step 6. Compute net input to each node in the F_2 layer:

$y_1 = .2(1) + .2(1) + .2(0) + .2(0) = 0.4,$

$y_2 = .2(1) + .2(1) + .2(0) + .2(0) = 0.4,$

$y_3 = .2(1) + .2(1) + .2(0) + .2(0) = 0.4.$

Step 7. While reset is true, do Steps 8–11.

Step 8. Since all units have the same net input,

$J = 1.$

Step 9. Recompute the F_1 activations:

$x_i = s_i t_{1i}$; currently, $t_1 = (1, 1, 1, 1)$;

therefore, $\mathbf{x} = (1, 1, 0, 0)$

Step 10. Compute the norm of $\mathbf{x}$:

$\|\mathbf{x}\| = 2.$

Step 11. Test for reset:

$\dfrac{\|\mathbf{x}\|}{\|\mathbf{s}\|} = 1.0 \geq 0.4;$

therefore, reset is false.
Proceed to Step 12.

Step 12. Update b_1; for $L = 2$, the equilibrium weights are

$$b_{i1}(\text{new}) = \frac{2x_i}{1 + \|\mathbf{x}\|}.$$

Therefore, the bottom-up weight matrix becomes

$$\begin{bmatrix} .67 & .2 & .2 \\ .67 & .2 & .2 \\ 0 & .2 & .2 \\ 0 & .2 & .2 \end{bmatrix}$$

Update $\mathbf{t}_1$; the fast learning weight values are

$t_{Ji}(\text{new}) = x_i,$

therefore, the top-down weight matrix becomes

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Step 2. For the second input vector, (0, 0, 0, 1), do Steps 3–12.

Step 3. Set activations of all F_2 units to zero.
Set activations of $F_1(a)$ units to input vector
$\mathbf{s} = (0, 0, 0, 1).$

Step 4. Compute norm of $\mathbf{s}$:

$\|\mathbf{s}\| = 1.$

Step 5. Compute activations for each node in the F_1 layer:

$\mathbf{x} = (0, 0, 0, 1).$

Step 6. Compute net input to each node in the F_2 laye

$y_1 = .67(0) + .67(0) + 0(0) + 0(1) = 0.0,$

$y_2 = .2(0) + .2(0) + .2(0) + .2(1) = 0.2,$

$y_3 = .2(0) + .2(0) + .2(0) + .2(1) = 0.2.$

Step 7. While reset is true, do Steps 8–11.

Step 8. Since units Y_2 and Y_3 have the same n

$J = 2.$

Step 9. Recompute the activation of the F_1 lay

$x_i = s_i t_{2i};$

currently $\mathbf{t}_2 = (1, 1, 1, 1)$; therefore,

$\mathbf{x} = (0, 0, 0, 1).$

Step 10. Compute the norm of $\mathbf{x}$:

$\|\mathbf{x}\| = 1.$

Step 11. Test for reset:

$\dfrac{\|\mathbf{x}\|}{\|\mathbf{s}\|} = 1.0 \geq 0.4;$

therefore, reset is false. Proceed to St

Step 12. Update $\mathbf{b}_2$; the bottom-up weight matrix becor

$$\begin{bmatrix} .67 & 0 & .2 \\ .67 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 1 & .2 \end{bmatrix}$$

Update $\mathbf{t}_2$; the top-down weight matrix becom

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Step 2. For the third input vector, (1, 0, 0, 0), do Steps 3–12.

Step 3. Set activations of all F_2 units to zero.
Set activations of $F_1(a)$ units to input vector
$\mathbf{s} = (1, 0, 0, 0).$

Step 4. Compute norm of $\mathbf{s}$:

$\|\mathbf{s}\| = 1.$

Step 5. Compute activations for each node in the F_1 la

$\mathbf{x} = (1, 0, 0, 0).$

Step 6. Compute net input to each node in the F_2 layer:

$$y_1 = .67(1) + .67(0) + 0(0) + 0(0) = 0.67,$$
$$y_2 = 0(1) + 0(0) + 0(0) + 1(0) = 0.0,$$
$$y_3 = .2(1) + .2(0) + .2(0) + .2(0) = 0.2.$$

Step 7. While reset is true, do Steps 8–11.

Step 8. Since unit Y_1 has the largest net input,

$$J = 1.$$

Step 9. Recompute the activation of the F_1 layer:

$$x_i = s_i t_{1i};$$

current, $t_1 = (1, 1, 0, 0)$; therefore,

$$\mathbf{x} = (1, 0, 0, 0).$$

Step 10. Compute the norm of $\mathbf{x}$:

$$\|\mathbf{x}\| = 1.$$

Step 11. $\|\mathbf{x}\| / \|\mathbf{s}\| = 1.0$ Proceed to Step 12.

Step 12. Update $\mathbf{b}_1$; the bottom-up weight matrix becomes

$$\begin{bmatrix} 1 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 1 & .2 \end{bmatrix}$$

Update $\mathbf{t}_1$; the top-down weight matrix becomes

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Step 2. For the fourth input vector, (0, 0, 1, 1), do Steps 3–12.

Step 3. Set activations of all F_2 units to zero.
Set activations of $F_1(a)$ units to input vector
$\mathbf{s} = (0, 0, 1, 1).$

Step 4. Compute norm of $\mathbf{s}$:

$$\|\mathbf{s}\| = 2.$$

Step 5. Compute activations for each node in the F_1 layer:

$$\mathbf{x} = (0, 0, 1, 1).$$

Step 6. Compute net input to each node in the F_2 layer:

$$y_1 = 1(0) + 0(0) + 0(1) + 0(1) = 0.0,$$
$$y_2 = 0(0) + 0(0) + 0(1) + 1(1) = 1.0,$$
$$y_3 = .2(0) + .2(0) + .2(1) + .2(1) = 0.4.$$

Step 7. While reset is true, do Steps 8–11.

Step 8. Since unit Y_2 has the largest net input,

$$J = 2.$$

Step 9. Recompute the activation of the F_1 layer:

$$x_i = s_i t_{2i};$$

currently, $t_2 = (0, 0, 0, 1)$; therefore,

$$\mathbf{x} = (0, 0, 0, 1).$$

Step 10. Compute the norm of $\mathbf{x}$:

$$\|\mathbf{x}\| = 1.$$

Step 11. Test for reset:

$$\frac{\|\mathbf{x}\|}{\|\mathbf{s}\|} = 0.5 \geq 0.4;$$

therefore, reset is false. Proceed to Step 12.

Step 12. Update $\mathbf{b}_2$; however, there is no change in the bottom-up weight matrix:

$$\begin{bmatrix} 1 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 0 & .2 \\ 0 & 1 & .2 \end{bmatrix}$$

Similarly, the top-down weight matrix remains

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

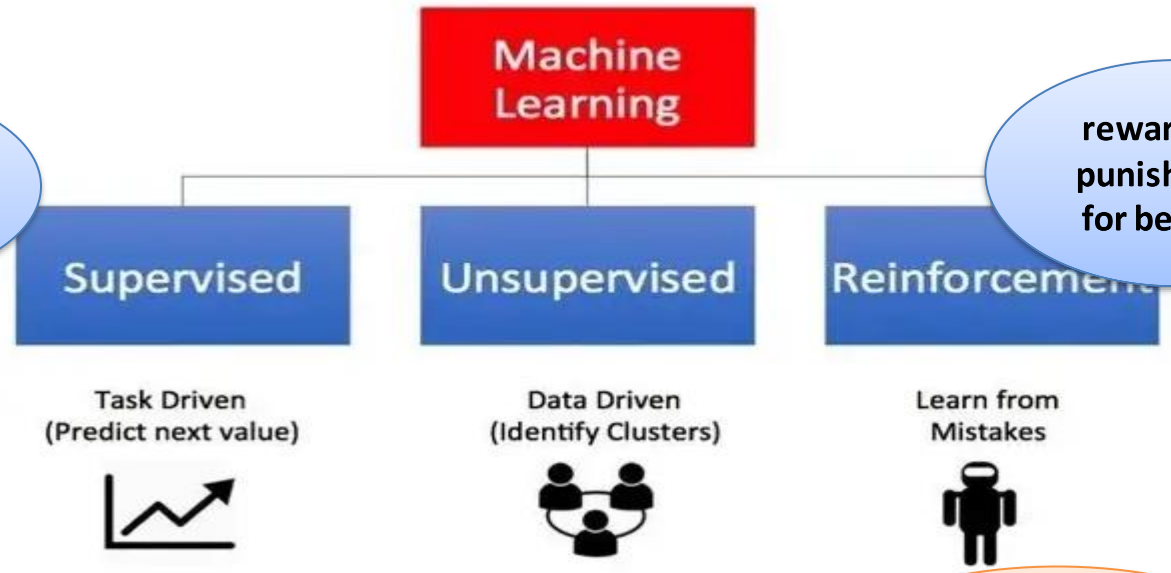
Step 13. Test stopping condition.
(This completes one epoch of training.)

The reader can check that no further learning takes place on subsequent presentations of these vectors, regardless of the order in which they are presented. Depending on the order of presentation of the patterns, more than one epoch may be required, but typically, stable weight matrices are obtained very quickly.

Reinforcement learning

Feedback is for Correct set of actions

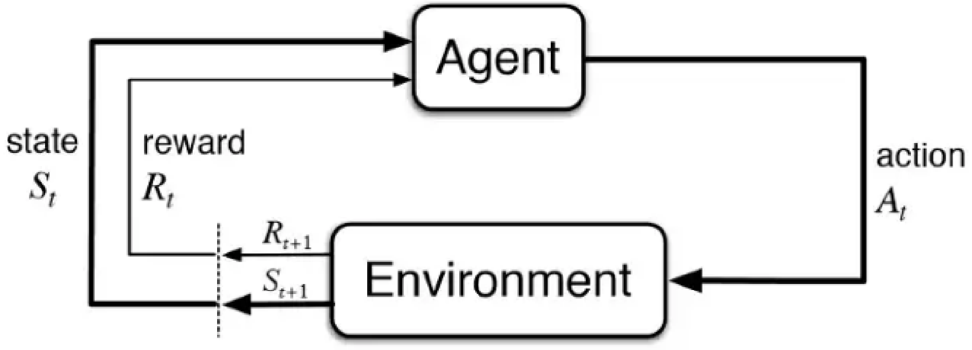
rewards and punishments for behavior



Find Similarities / differences

Find suitable action to maximize reward

RL problem



Agent(): An entity that can perceive/explore the environment and act upon it.

Environment(): A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.

Action(): Actions are the moves taken by an agent within the environment.

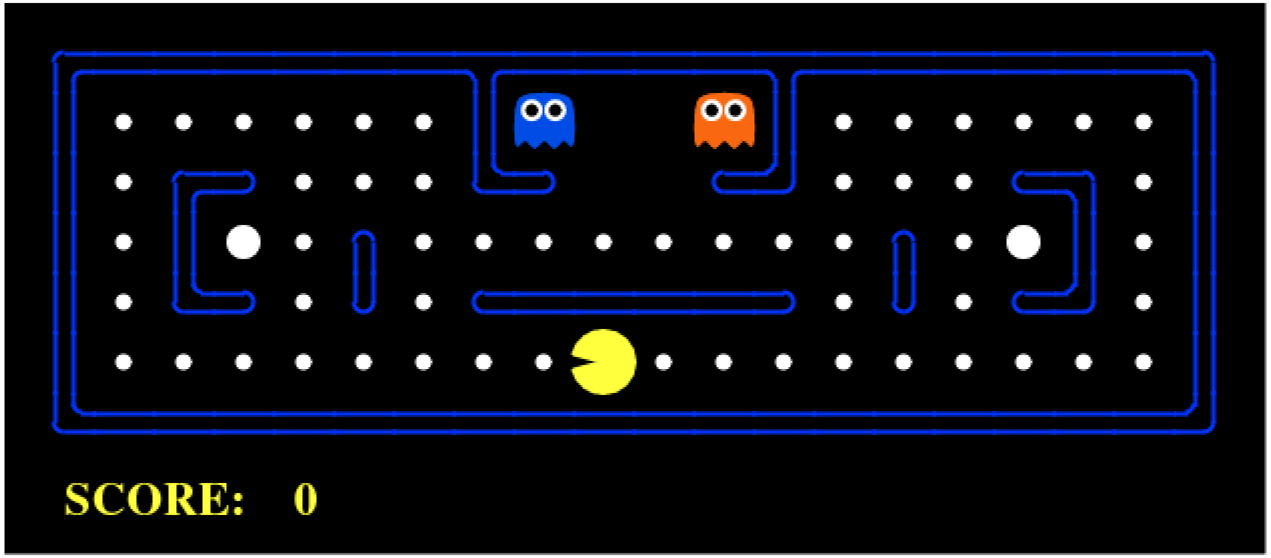
State(): State is a situation returned by the environment after each action taken by the agent.

Reward(): A feedback returned to the agent from the environment to evaluate the action of the agent.

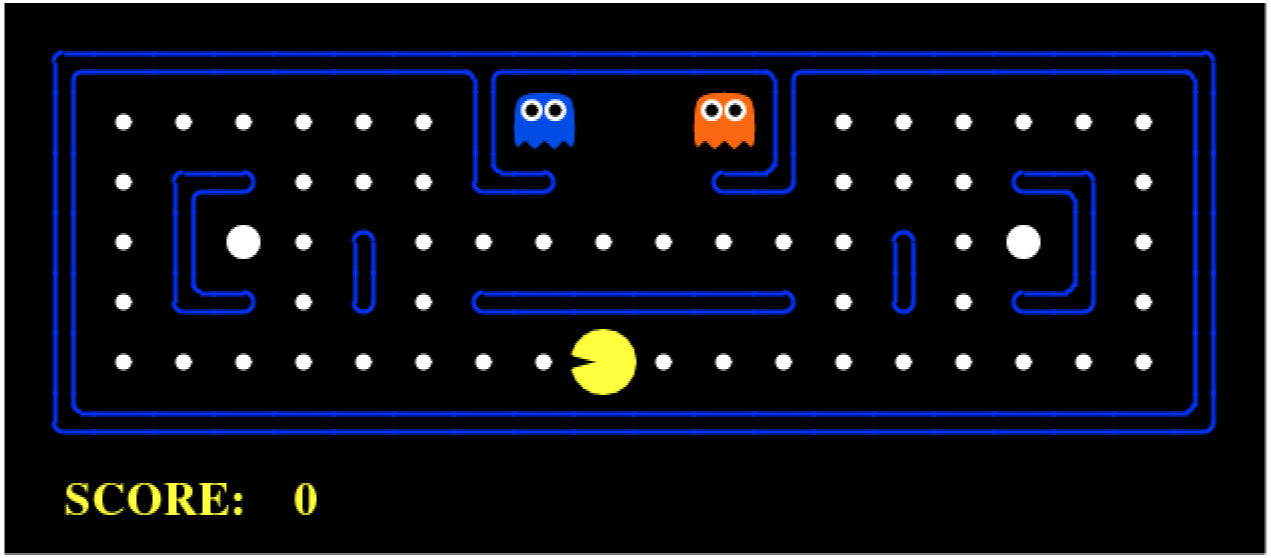
Policy(): Policy is a strategy applied by the agent for the next action based on the current state.

Value(): It is expected long-term retuned with the discount factor and opposite to the short-term reward.

Q-value(): It is mostly similar to the value, but it takes one additional parameter as a current action (a).



Goal - to eat the food in the grid while avoiding the ghosts on its way
Environment — Grid world
State — locations of the agent on the grid
Reward — earn points



PACMAN with deep learning

Key Features of Reinforcement Learning

- In RL, the agent is not instructed about the environment and what actions need to be taken.
- It is based on the hit and trial process.
- The agent takes the next action and changes states according to the feedback of the previous action.
- The agent may get a delayed reward.
- The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

Approaches to implement Reinforcement Learning

Value-based:

The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy п.

Policy-based:

Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the future reward.

Approaches to implement Reinforcement Learning

The policy-based approach has mainly two types of policy:

- **Deterministic:** The same action is produced by the policy (п) at any state.
- **Stochastic:** In this policy, probability determines the produced action.

Model-based: In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model representation is different for each environment.

Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

Positive Reinforcement:

The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

Negative Reinforcement:

The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.

It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

Markov property

"If the agent is present in the current state S1, performs an action a1 and move to the state s2, then the state transition from s1 to s2 only depends on the current state and future action and states do not depend on past actions, rewards, or states."

- the current state transition does not depend on any past action or state. E.g. Chess game: player focusses on the current state and not on the previous states

Representing an agent

The Markov state follows the **Markov property**, which says that the future is independent of the past and can only be defined with the present.

We can represent the agent state using the **Markov State** that contains all the required information from the history. The State St is Markov state if it follows the given condition:

$$P[S_t+1 \mid S_t] = P[S_t+1 \mid S_1,\ldots\ldots, S_t]$$

The RL works on fully observable environments, where the agent can observe the environment and act for the new state.

Markov Decision Process

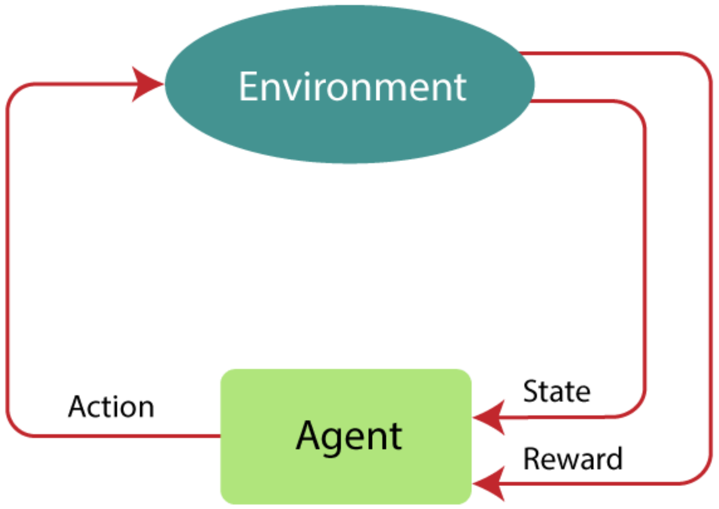
Markov Decision Process or MDP, is used to **formalize the reinforcement learning problems**.

If the environment is completely observable, then its dynamic can be modelled as a **Markov Process**.

In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.

MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP. Almost all the RL problem can be formalized using MDP.

Markov Decision Process



MDP contains a tuple of four elements (S, A, P_a, R_a):

- A set of finite States S
- A set of finite Actions A
- Rewards received after transitioning from state S to state S', due to action a.
- Probability P_a.

Finite MDP:
A finite MDP is when there are finite states, finite rewards, and finite actions. In RL, we consider only the finite MDP.

RL Algorithms

Q-Learning:

Q-learning is an **Off policy RL algorithm**, which is used for the temporal difference Learning. The temporal difference learning methods are the way of comparing temporally successive predictions.

It learns the value function Q (S, a), which means how good to take action "**a**" at a particular state "**s**."

State Action Reward State action (SARSA):

SARSA stands for **State Action Reward State action**, which is an **on-policy** temporal difference learning method. The on-policy control method selects the action for each state while learning using a specific policy.

Deep Q Neural Network (DQN):

DQN is a **Q-learning using Neural networks**.

For a big state space environment, it will be a challenging and complex task to define and update a Q-table.

To solve such an issue, we can use a DQN algorithm. Where, instead of defining a Q-table, neural network approximates the Q-values for each action and state.