# Neural Networks
# Lecture 7: Self Organizing Maps

**H.A Talebi**
**Farzaneh Abdollahi**

Department of Electrical Engineering

Amirkabir University of Technology

Winter 2011

Hamming Net and Maxnet
- Hamming Network

Unsupervised Learning of Clusters
- Kohonen network (Kohonen 1988)
  - Winner Take-All Learning

Self Organizing Feature Map

LVQ

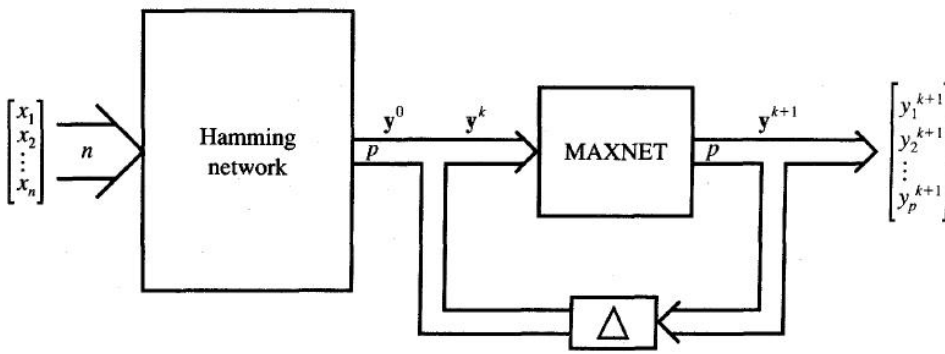
CPN

ART
- ART1

- In this lecture we want to explore the unsupervised training algorithms of neural networks where their weights can adapt by self-organization algs.
- Both inputs and their probability density will affect the training.
  - Rare inputs will have less impact for learning than those occurring frequently.
- In the absence of target responses or guidelines provided directly by the teacher, the network can build classification / recognition /mapping performance based on competitive learning.
  - The winning output node will receive the only reward during learning
  - Sometimes, the extension of rewards for the node's neighborhood will also be allowed.

# Hamming Net and Maxnet (Lippmann 1987)

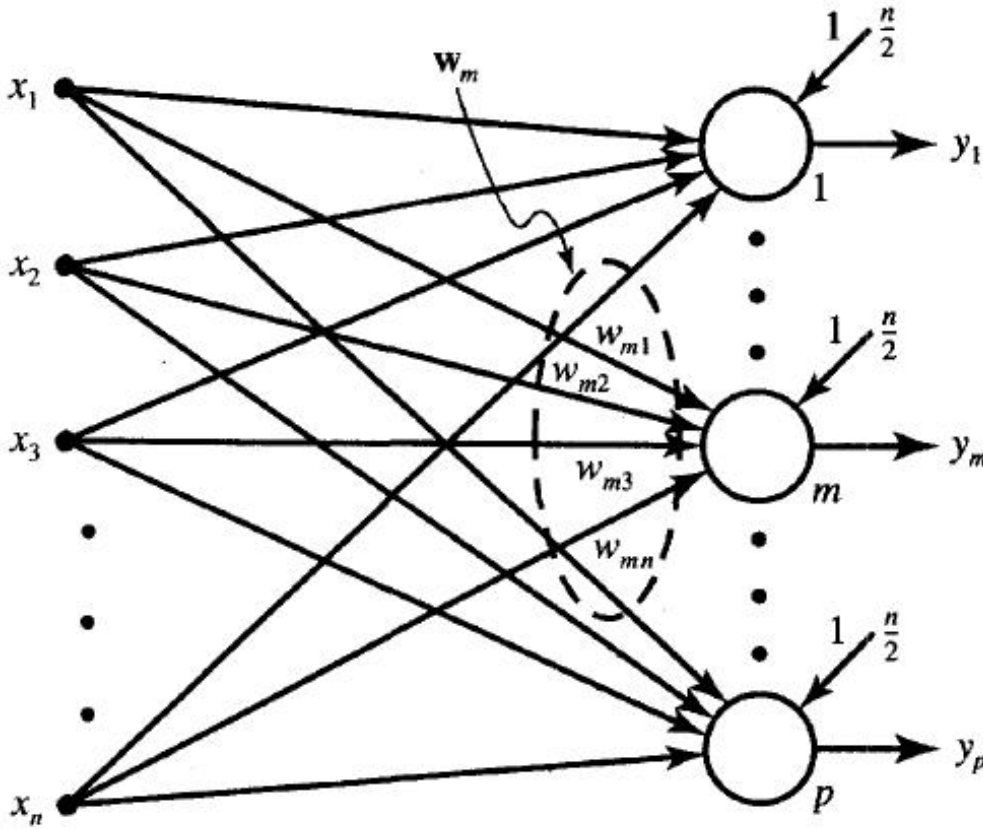
- It is a two-layer bipolar classifier
  1. **First Layer:Hamming Network**
     - It is feedforward type the classifier.
     - It classifies based on minimum Hamming distance
     - The strongest response of a neuron is indicative of the min HD value between the input and the class this neuron represents.
     - The p-class Hamming network has p output neurons
  2. **Second Layer: MAXNET**
     - it operates as a recurrent recall network.
     - It suppresses all the outputs except the initially max output of the first layer




Block diagram of the minimum HD classifier.

# Hamming Network

- Consider the $n$-tuple prototype vector of the $m$'th class is $s^{(m)}$, for $m = 1, 2, ..., p$
- the $n$-tuple input vector is $x$.
- The entries of the weight vector $w_m = [w_{m1}\ w_{m2}\ ...\ w_{mn}]'$ connect inputs to the $m$'th neuron, which performs as the class indicator
- The network should be designed s.t. the $m$'th output is 1 iff $x = s^{(m)}$.
- $\therefore w_m = s^{(m)}$ and the outputs: $x^T s^{(1)}, ..., x^T s^{(m)}, ..., x^T s^{(p)}$



- $x = s^{(m)} \Rightarrow s^{(m)T} s^{(1)},\ s^{(m)T} s^{(2)}, \ldots, s^{(m)T} s^{(m)}, \ldots, s^{(m)T} s^{(p)}$
- considering $\pm 1$ as input $s^{(m)}$, only the $m$th output is $n$ and others are less then $n$
- Total we can say:
  - The scalar product $x^T s^{(m)}$ of two bipolar vectors can be written as the total # positions in which they agree - # positions they differ.

$$x^T s^{(m)} = (n - HD(x^T, s^{(m)})) - HD(x^T s^{(m)})$$

$$\rightsquigarrow \frac{1}{2} x^T s^{(m)} = (\frac{n}{2} - HD(x^T, s^{(m)}))$$

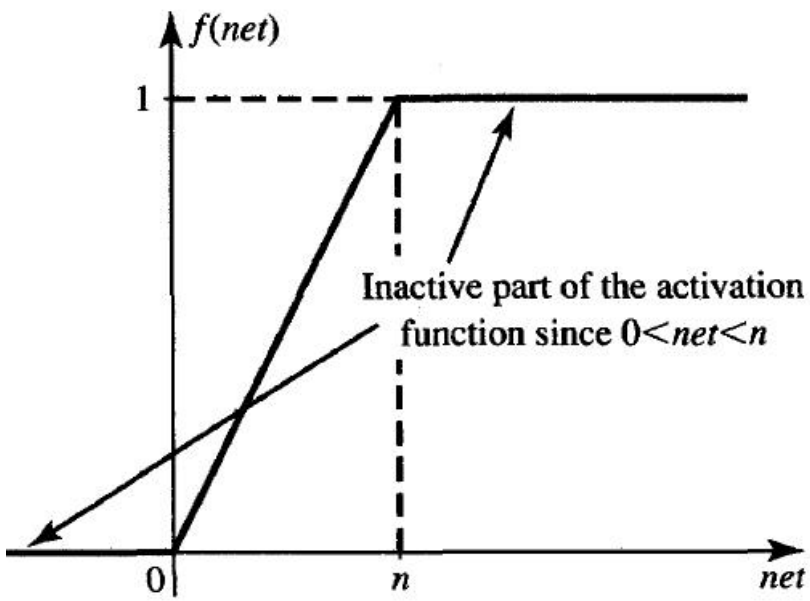
  - $\frac{1}{2}$ is for convenient
- The weight matrix $W_H$ of the Hamming network s defined s.t its rows are the prototypes

$$W_H = \frac{1}{2} \begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \ldots & s_n^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \ldots & s_n^{(2)} \\ \vdots & \vdots & \ldots & \vdots \\ s_1^{(p)} & s_2^{(p)} & \ldots & s_n^{(p)} \end{bmatrix}$$

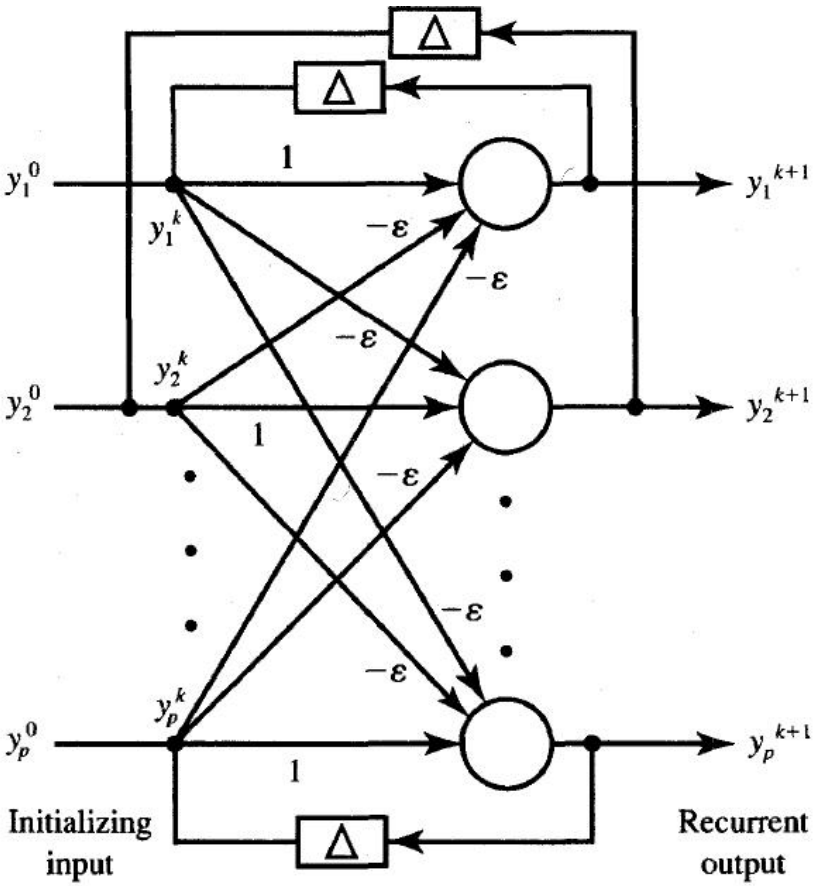
- Adding the fixed bias value of $\frac{1}{2}$ to the input of each neuron results:

$$\begin{aligned} net_m &= \frac{1}{2}x^T s^{(m)} + \frac{n}{2} \\ &= n - HD(x^T s^{(m)}) \\ &\quad \text{for } m = 1, 2, ..., p \end{aligned}$$

- Activation function is defined as $f(net_m) = \frac{1}{n} net_m$ for $m = 1, ..., p$
- $0 \leq net_m \leq n \Rightarrow 0 \leq f(net_m) \leq 1$.
- $\therefore$ The neurons need to perform only the linear scaling of activation function.



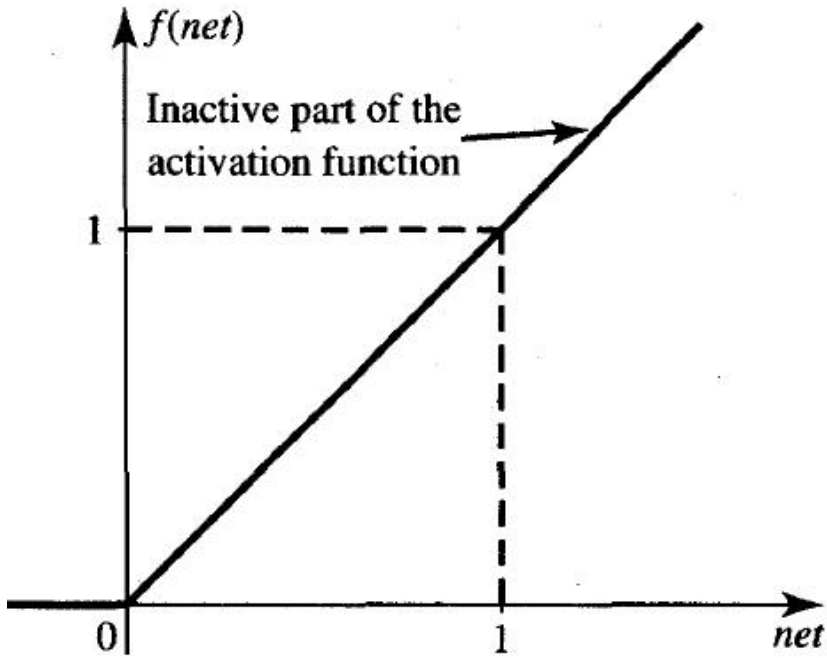
- The more similar input vector to a prototype yields the smaller $HD$.
- A perfect match of input vector to class m: $HD = 0 \rightsquigarrow f(net_m) = 1$.
- The complement of the prototype of class $m$: $f(net_m) = 0$.
- The response of the network essentially terminates the classification in only the first layer.
- MAXNET is a recurrent processing to keep the max output neuron high and decay the remaining nodes to zero.
- It has pos. self-fb with weighting 1.
- and $M - 1$ cross-fb with weighting $-\epsilon$



- The weight matrix

$$\begin{bmatrix} 1 & -\epsilon & \dots & -\epsilon \\ -\epsilon & 1 & \dots & -\epsilon \\ \vdots & \vdots & \ddots & \vdots \\ -\epsilon & -\epsilon & \dots & 1 \end{bmatrix}$$

- $0 < \epsilon < \frac{1}{p}$
- Considering the activation function shown in Fig. and the initializing vector condition $0 \leq y_i^0 \leq 1$ for $i = 1, ..., p$:
  - After a number of recurrences, the only unsuppressed node will be the one with the largest initializing entry $y_m^0$
  - $\therefore$ the only nonzero output response node is the node closest to the input vector argument in sense of HD.

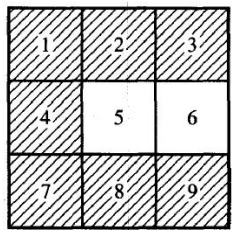


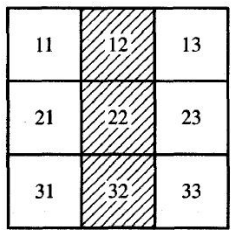
- Therefore, recurrent processing by MAXNET is expressed as $y^{k+1} = \Gamma[W_M y^k]$.
- $\Gamma$ is a nonlinear diagonal matrix operator with entries

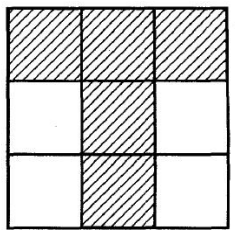
$$f(net) = \begin{cases} 0 & net < 0 \\ n & net \geq 0 \end{cases}$$

- **Example:** Let us design an HD classifier for three characters C, I, and T
- The prototypes are

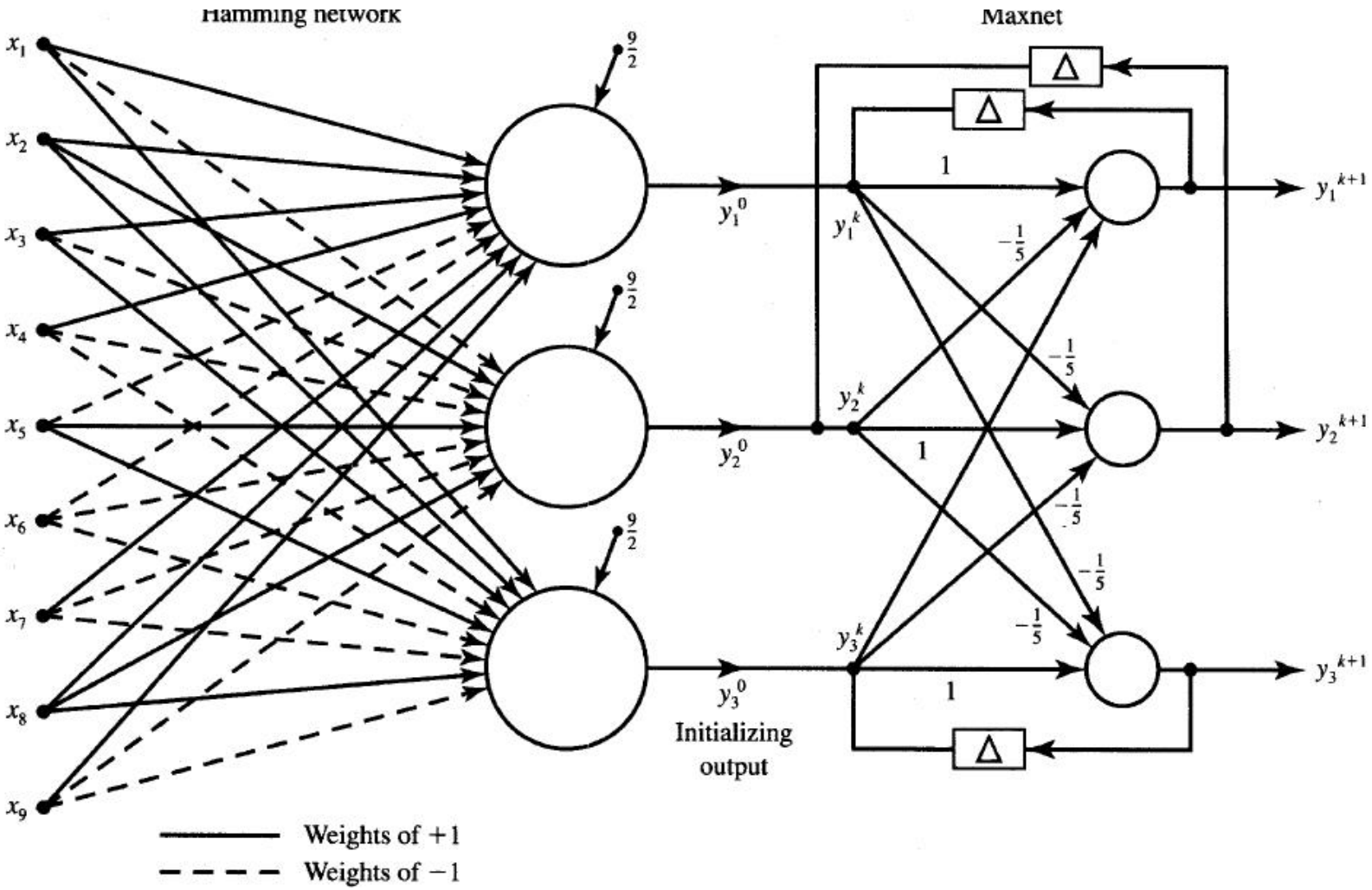
$$\begin{aligned} s^{(1)} &= [1\ 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1\ 1] \\ s^{(2)} &= [-1\ 1\ -1\ -1\ 1\ -1\ -1\ 1\ -1] \\ s^{(3)} &= [1\ 1\ 1\ -1\ 1\ -1\ -1\ 1\ -1] \end{aligned}$$



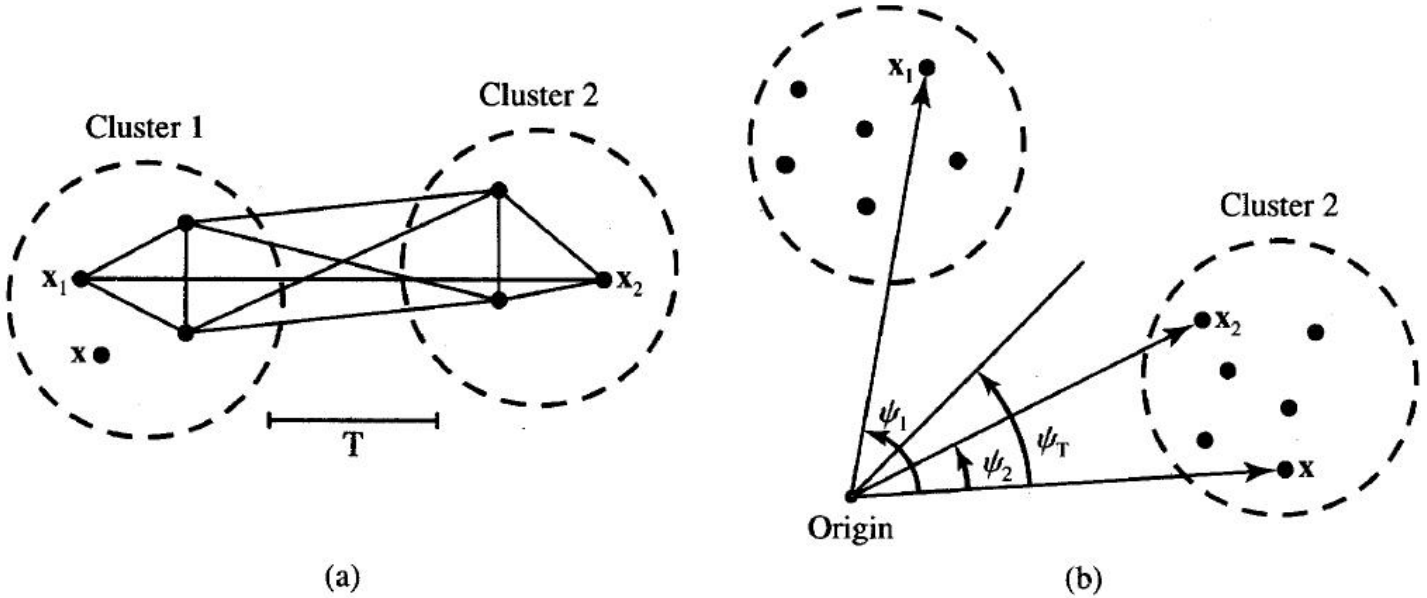



- So $W_H = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \end{bmatrix}$
- $net = W^T x + \begin{bmatrix} 9/2 \\ 9/2 \\ 9/2 \end{bmatrix}$
- $\epsilon = 0.2 < 1/3$


Hamming network
Maxnet
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
$x_6$
$x_7$
$x_8$
$x_9$
$\frac{9}{2}$
$\frac{9}{2}$
$\frac{9}{2}$
$y_1^0$
$y_2^0$
$y_3^0$
Initializing output
$y_1^k$
$y_2^k$
$y_3^k$
$y_1^{k+1}$
$y_2^{k+1}$
$y_3^{k+1}$
1
1
1
$-\frac{1}{5}$
$-\frac{1}{5}$
$-\frac{1}{5}$
$-\frac{1}{5}$
$-\frac{1}{5}$
Δ
Δ
Δ
Weights of +1
Weights of −1

# Kohonen network (Kohonen 1988)

- Unsupervised classification is called clustering
- Clustering is considered as the grouping of similar objects and separating of dissimilar ones.
- Sometimes even number of clusters are not known a priori.
  - The clustering technique should
    1. identify # of classes according to a certain criterion
    2. assign the membership of the patterns in these classes.
- The clustering technique presented here, knows # of clusters a priori
- The input is required to be identifies as member of one of the possible clusters.

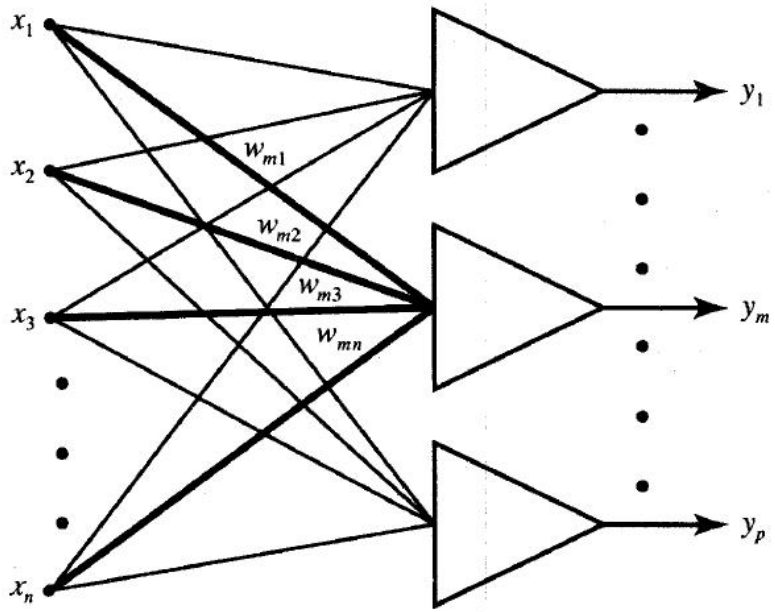



Measures of similarity for clustering data: (a) distance and (b) a normalized scalar product.

- Since it is unsupervised, criterion for similarity should be defined:
  1. Euclidean distance between two patterns $x$, for $x_i$: $\|x - x_i\| = \sqrt{(x - x_i)^T (x - x_i)}$
  2. scalar product (cos of the angle between $x$ and $x_i$): $\cos\psi = \frac{x^T x_i}{\|x\|\|x_i\|}$ .
     For $cos\psi_2 > \cos\psi_1$ : $x$ is more similar to $x_2$ than $x_1$.
- scalar product is simpler for implementation
- For normalaized vectors both method yield similar results

# Winner Take-All Learning

- Objective is learning the weights s.t. classifies input vectors into one of the specified number of $p$ categories
- Training set $\{x_1, x_2, ..., x_N\}$.
- $y = \Gamma[Wx]$
- $\Gamma$ is diagonal continuous activation function
- $W = [w_1^T \ w_2^T \ ... \ w_p^T]^T$, $w_i = [w_{i1} \ ... \ w_{in}] \ i = 1, ...p$
- Each neuron represent a cluster
- Winner take-all learning: Only the weight corresponding to the winner neuron is updated




(Adapted weights highlighted)

- Before start learning all weights should be normalized
  $\hat{w}_i = \frac{w_i}{\|w_i\|} \quad i = 1, ..., p$
- For $x = s^m$, by selecting $w = s^m$ $m$th neuron will be max.
- But $s^m$. center of cluster is not known a priori
- Proper learning rule is required to find the centers
- The weight vectors should be modified accordingly so that they become more similar to the current input vector
- The winning neuron (the closest approximation of the current input $x$), to be updated is selected s.t.
  $\|x - \hat{w}_m\| = (x^T x - 2\hat{w}_m^T x + 1)^{1/2} = \min_{i=1,...p}\{\|x - \hat{w}_i\|\}$
- Searching for the min of $p$ distances corresponds to finding the max among the p scalar products $\hat{w}_m^T x = \max_{i=1,..,p}(\hat{w}_i^T x)$
- If the weights are not normalized the above statement is not always true

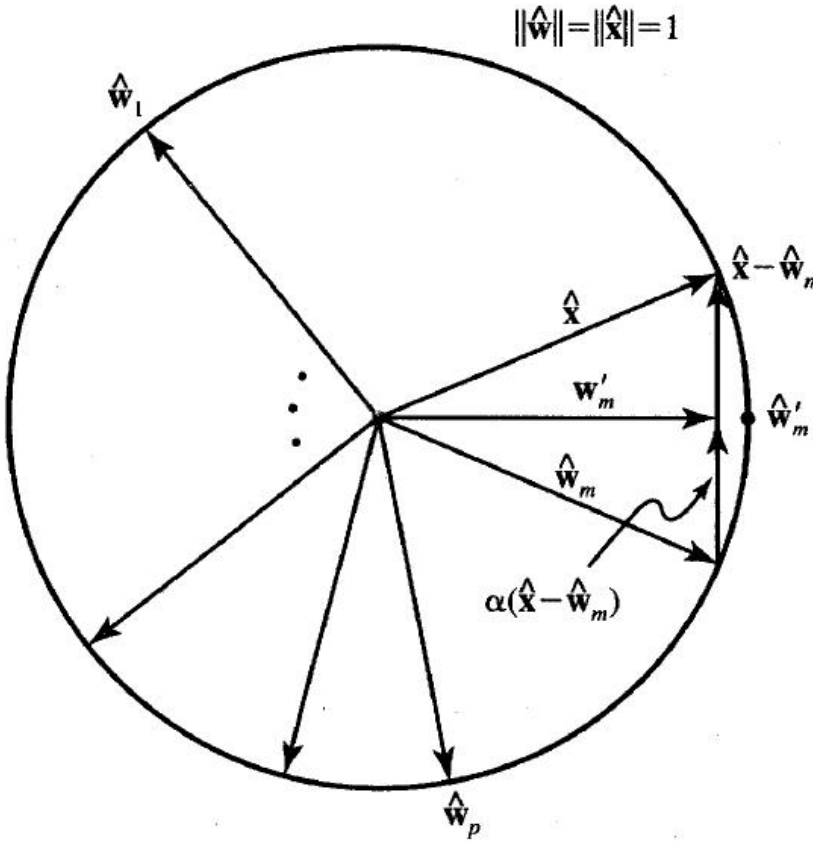
- After identifying the winning neuron, its weights must be adjusted s. t. the distance $\|x - w_m\|$ is reduced.
- The wining weight is updated along the gradient direction in the weight space $\nabla_{w_m}\|x - w_m\|^2 = -2(x - w_m)$
- $\therefore$ Learning rule in $k$th step is

$$\begin{aligned} \hat{w}_m^{k+1} &= \hat{w}_m^k + \alpha^k(x - \hat{w}_m^k) \\ \hat{w}_i^{k+1} &= \hat{w}_i^k \quad i \neq m \end{aligned}$$

- $\alpha^k$: a suitable learning constant
- $m$ is the winning neuron selected based on the scalar product comparison (largest $net_i$)
- During the learning the clusters are developed,
- The network weights acquire similarity to input data within clusters.
- To avoid unconstrained growth of weights, $\alpha$ is usually reduced monotonically and the learning slows down.

- Let us check the impact of the learning rule:
  - The rule should increase the chances of winning by the $m$'th neuron, if the same input pattern using the updated weights is repeated
- If so the following inequality should be valid
  $\hat{w}_m^T x < (\hat{w}_m^T + \triangle\ \hat{w}_m^T)x \rightsquigarrow \triangle\ \hat{w}_m^T x > 0$
- Based on the learning rule :
  $x^T x - \hat{w}_m^T x > 0 \rightsquigarrow \|x^T\|\|x\| cos0 - \|\hat{w}_m^T\|\|x\| cos\phi > 0$
- Assuming all vectors are normalized: $1 - cos\phi > 0$ which is always true, $(\phi = \angle(x, \hat{w}_m^T))$
- $\therefore$ **the winner-take-all learning rule produces an update of the weight vector in the proper direction.**

- Geometrical interpretation of the rule is shown in Fig.
- Assume that in this step $\hat{x}$ is the normalized input vector of $x$ and $\hat{w}_m$ yield the maximum scalar product $\hat{w}_m^T \hat{x}$, for $i = 1, 2, ..., p$.
- To implement the rule for $x = \hat{x}$, an increment of the weight vector is computed as a fraction of $\hat{x} - \hat{w}_m^T$
- $\therefore$ weight adjustment is the rotation of the weight vector $\hat{w}_m$ toward the input vector without a significant length change.
- The adjusted weight vector $w'_m$'s length is below unity $\rightsquigarrow$ for next step it should be renormalized



- (Simpson 1990) proposed a supervised Kohonen network:
  - $\alpha > 0$ for proper node responses
  - $\alpha < 0$ otherwise
- Another modification of the winner-take-all learning rule for the cases which clusters may be hard to distinguish named Leaky competitive learning
  - Both the winners' and losers' weights are adjusted in proportion to their level of responses.
- **Recall Mode** The network trained in the winner-take-all mode responds instantaneously during feedforward recall
- The response is $y = \Gamma[Wx]$
- The layer now performs as a filter of the input vectors such that the largest output neuron is found as $y_m = max(y_1, ..., y_p)$

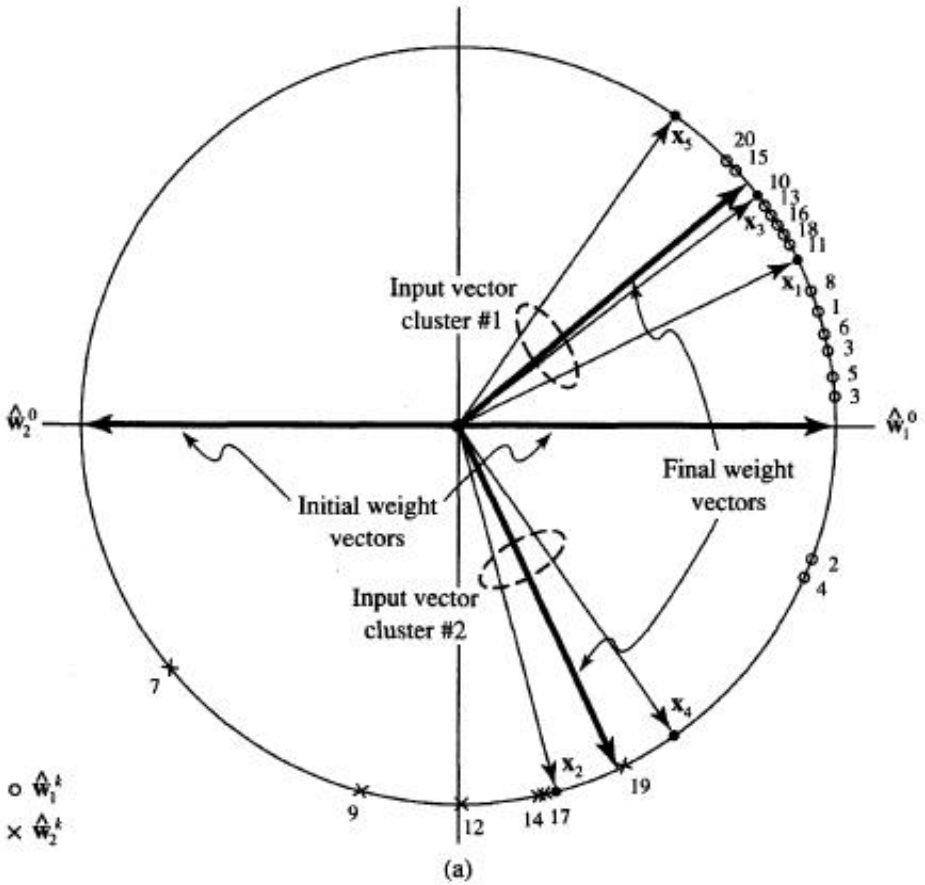
- **Weight Initializing**
- Random initial weight vectors should be selected s.t. uniformly distributed on the unity hypersphere in *n*-dimensional pattern space.
- Self-organization of the network suffers from some limitations:
  - Because of its single-layer architecture, linearly nonseparable patterns cannot be efficiently handled by this network.
  - The network training may not always be successful even for linearly separable patterns.
- The weights may get stuck in isolated regions without forming adequate clusters.
- In such cases the training must be reinitialized with new initial weights,
- After the weights have been trained to provide coarse clustering, the learning constant $\alpha$ should be reduced

- The simple winner-take-all training alg can be modified, for K winners in the competition
  - Such learning is called multiple-winner unsupervised learning.(Hecht-Nielsen 1987).
  - The winning K neurons would be the ones best matching the input vector, instead of a single one as in the winnertake- all mode.
  - The outputs of winning neurons can be set so that they sum to unity.

# Example

- Consider the normalized patterns for training: $\{x_1, x_2, x_3, x_4, x_5\} = \{\begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.1736 \\ -0.9848 \end{bmatrix}, \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}, \begin{bmatrix} 0.342 \\ -0.9397 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.8 \end{bmatrix}\}$
- $\#$ of clusters: 2, and $\alpha = 1/2$
- The normalized initial weights: $w_1^0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, w_2^0 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$
- The inputs are submitted in ascending sequence and recycled
- The first neuron won all of the first six competitions in a row $\rightsquigarrow$ the first weight is only updated.
- In the second training cycle, $x_2$ and $x_4$ has been detected by the second neuron.
- After 20 steps the weights are adjusted properly and index the center of clusters.

# Example Cont'd




(a)

| Step k | $\hat{w}_1^k$ ∠ deg | $\hat{w}_2^k$ ∠ deg |
|---|---|---|
| 1 | 18.46 | −180.00° |
| 2 | −30.77 | — |
| 3 | 7.11 | — |
| 4 | −31.45 | — |
| 5 | 10.85 | — |
| 6 | 23.86 | — |
| 7 | — | −130.22 |
| 8 | 34.43 | — |
| 9 | — | −100.01 |
| 10 | 43.78 | — |
| 11 | 40.33 | — |
| 12 | — | −90.00 |
| 13 | 42.67 | — |
| 14 | — | −80.02 |
| 15 | 47.90 | — |
| 16 | 42.39 | — |
| 17 | — | −80.01 |
| 18 | 43.69 | — |
| 19 | — | −75.01 |
| 20 | 48.42 | — |

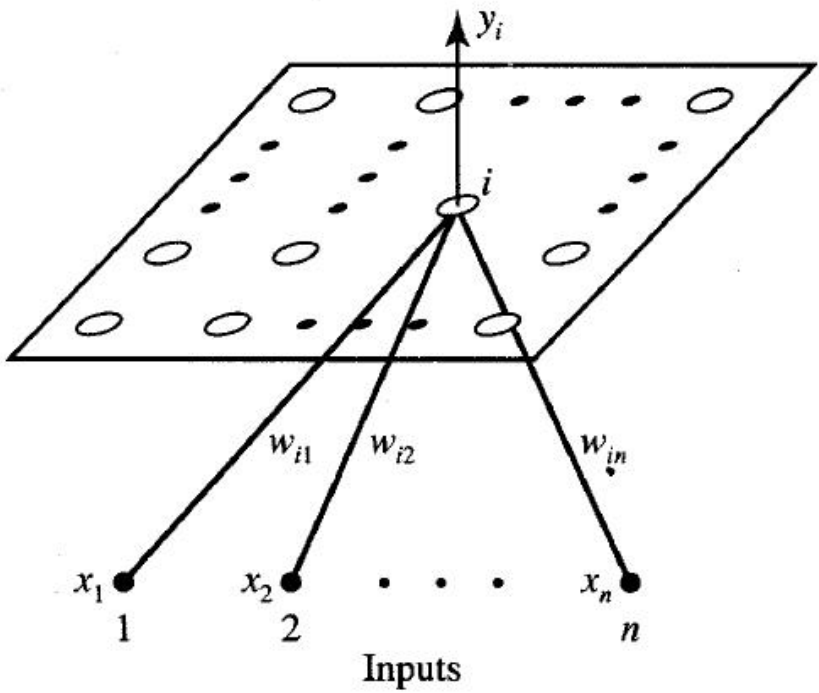
(weight vectors of unity length)
(— means no change)

(b)

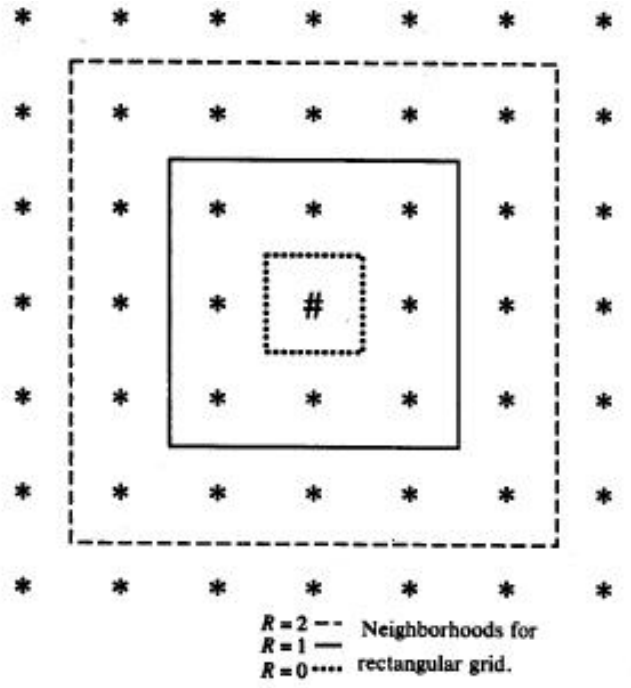
(a) training patterns and weight assignments and (b) weight learning, Steps 1 through 20.

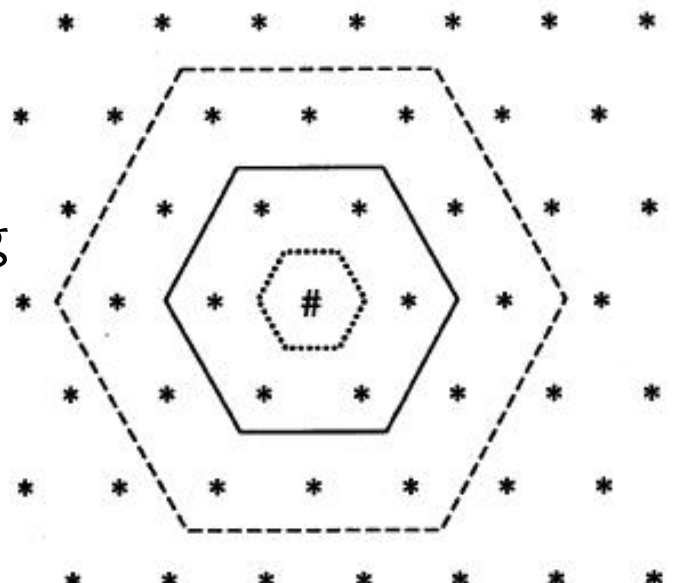
# Self Organizing Feature Map

- The feature mapping alg converts patterns of arbitrary dimensionality into the responses of one-or two-dimensional arrays of neurons
- It is assumed there are $P$ clusters in a one-or- two dimensional array
- Like the previous network the weights of each neuron should point to the center of the cluster
- Learning within self-organizing feature maps results in finding the best matching neuron cells which also activate their spatial neighbors to react to the same input.



- After learning, each input causes a localized response having a position on the neurons' array that reflects the dominant feature characteristics of the input.
- Input $x$ is applied simultaneously to all nodes.
- The best matching neuron to the input is announced as winner
- The weights affecting the currently winning node and its neighborhood undergo adaptation at the current learning step, other weights remain unaffected.
- Consider the winning neuron by: #
- Rectangular and hexagonal neighboring is shown in Fig.



- Neighboring in one dimension neuron is shown in Fig.
- The best matching neuron $m$ is defined as $\|x - w_m\| = \min_i\{\|x - w_i\|\}$
- The radius of neighboring $N_m$ should be decreasing as the training progresses, $N_m(t_1) > N_m(t_2) > N_m(t_3)...$, where $t_l < t_2 < t_3, ...$
- The weight updating rule for self-organizing feature maps is defined: $\triangle w_i(t) = \alpha(N_i, t)[x(t) - w_i(t)]$ for $i \in N_m(t)$
- $\alpha$ is positive valued learning function
- $\alpha$ can be expressed as a function of the neighborhood radius
- The initial weights are defined randomly

# Defining Neighboring Sets

1. In defined neighboring set, all weights are updated with similar rate
2. All weights of the network is updated properly with different rate
   - The rate for the winning neuron weight is max
   - The closer neuron to the winner updates with higher rate

- In the first type the radius of the neighboring set in decreasing in time
- In the second type, the learning rate is decreasing in time
- In the first type there is significant difference on the edge of neighboring set
- The first type is faster since # of weights of be updated is less

# Example

- Let us investigate the effect of topological structure in updating the weights.
- **Objective:** Clustering input patterns representing letters in three different fonts
- Initial number of neurons(Maximum of possible clusters): 25
- Learning rate is reduced linearly from 0.6 to 0.01.
- Three topology is considered:
  1. No structure ( only the winner is updated)

| NEURON | PATTERNS |
|---|---|
| 3 | C1,C2,C3 |
| 13 | B1,B3,D1,D3,E1,K1,K3,E3 |
| 16 | A1,A2,A3 |
| 18 | J1,J2,J3 |
| 24 | B2,D2,E2,K2 |

# Example

## 2. Linear structure (R=1)

- If Neuron $J$ is the winner, the neurons $J \pm 1$ are also updated

| NEURON | PATTERNS |
|---|---|
| 6 | K2 |
| 10 | J1,J2,J3 |
| 14 | E1,E3 |
| 16 | K1,K3 |
| 18 | B1,B3,D1,D3 |
| 20 | C1,C2,C3 |
| 22 | D2 |
| 23 | B2,E2 |
| 25 | A1,A2,A3 |

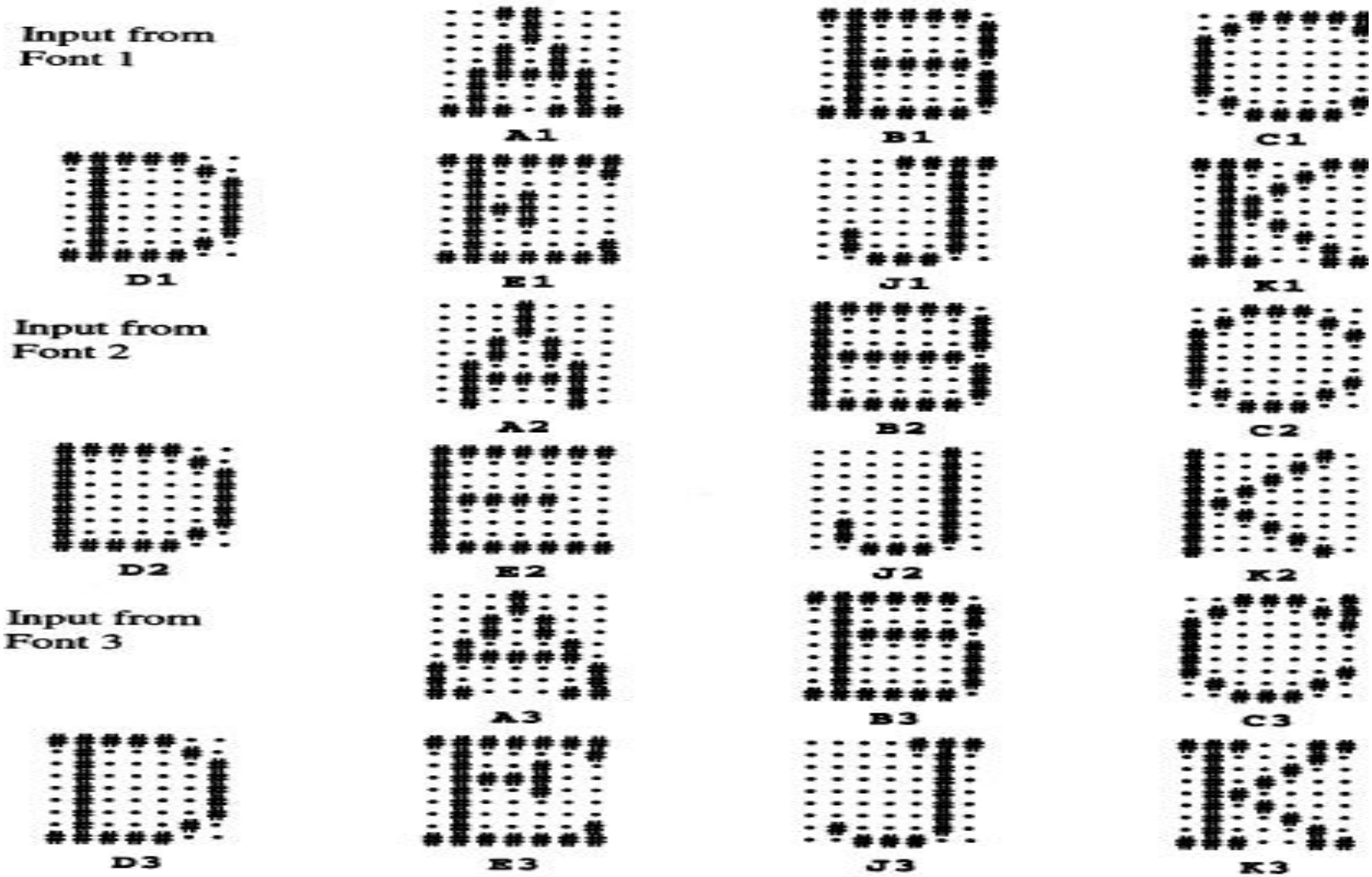
# Example

## 3. Diamond Structure

- If Neuron $X_{IJ}$ wins then neurons $X_{I\pm1J}$, $X_{IJ\pm1}$ also update

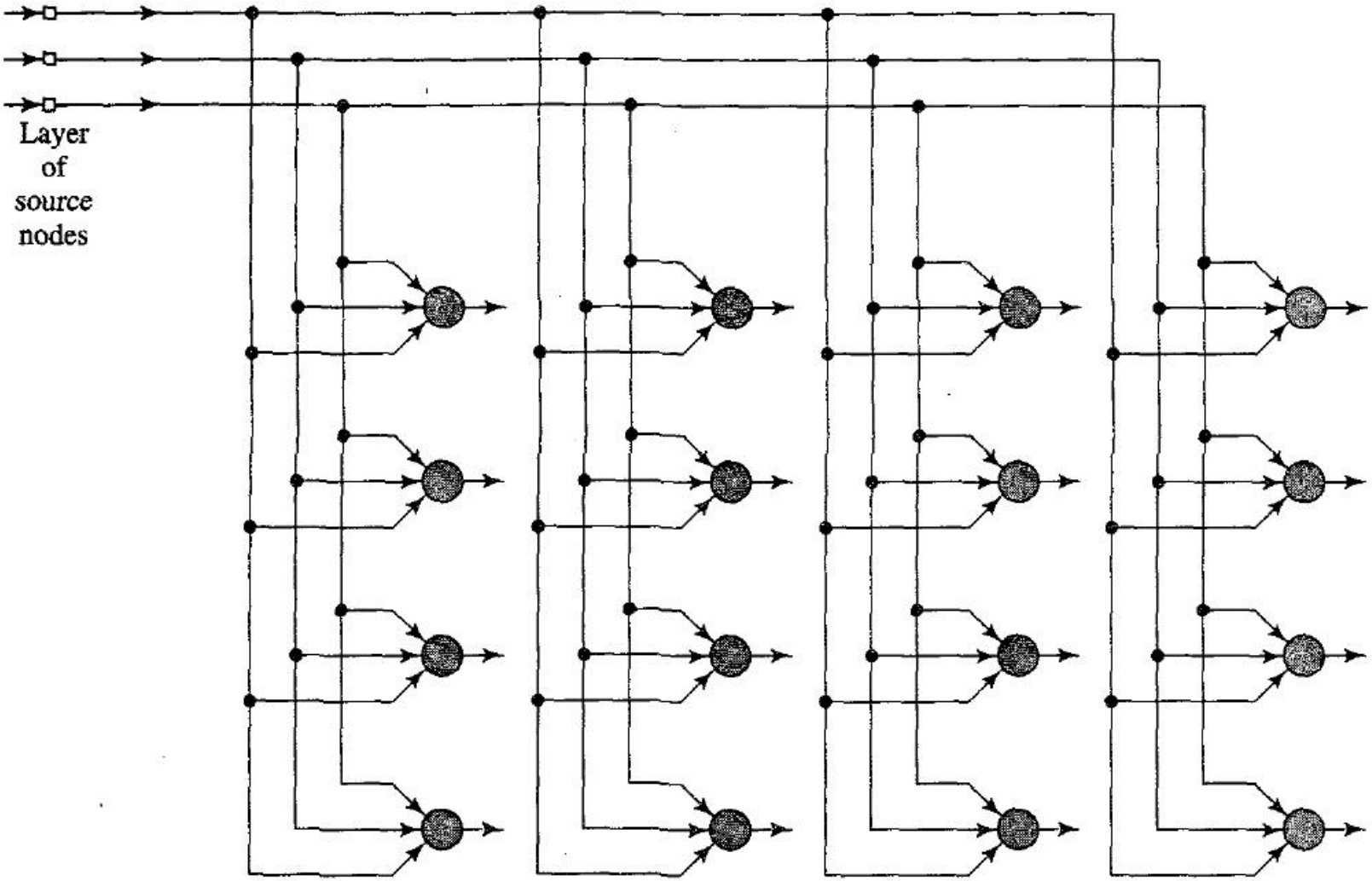
| $i \setminus j$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | | J1,J2,J3 | D2 | | |
| 2 | C1,C2,C3 | | D1.D3 | | B2 E2 |
| 3 | | B1 | K2 | | |
| 4 | | | E1,E3,B3 | | A3 |
| 5 | | K1,K3 | | A1,A2 | |

# Example: Character Recognition




Training input patterns for character recognition examples.

# SOM




Two-dimensional lattice of neurons.

# Summary of SOM Algorithm

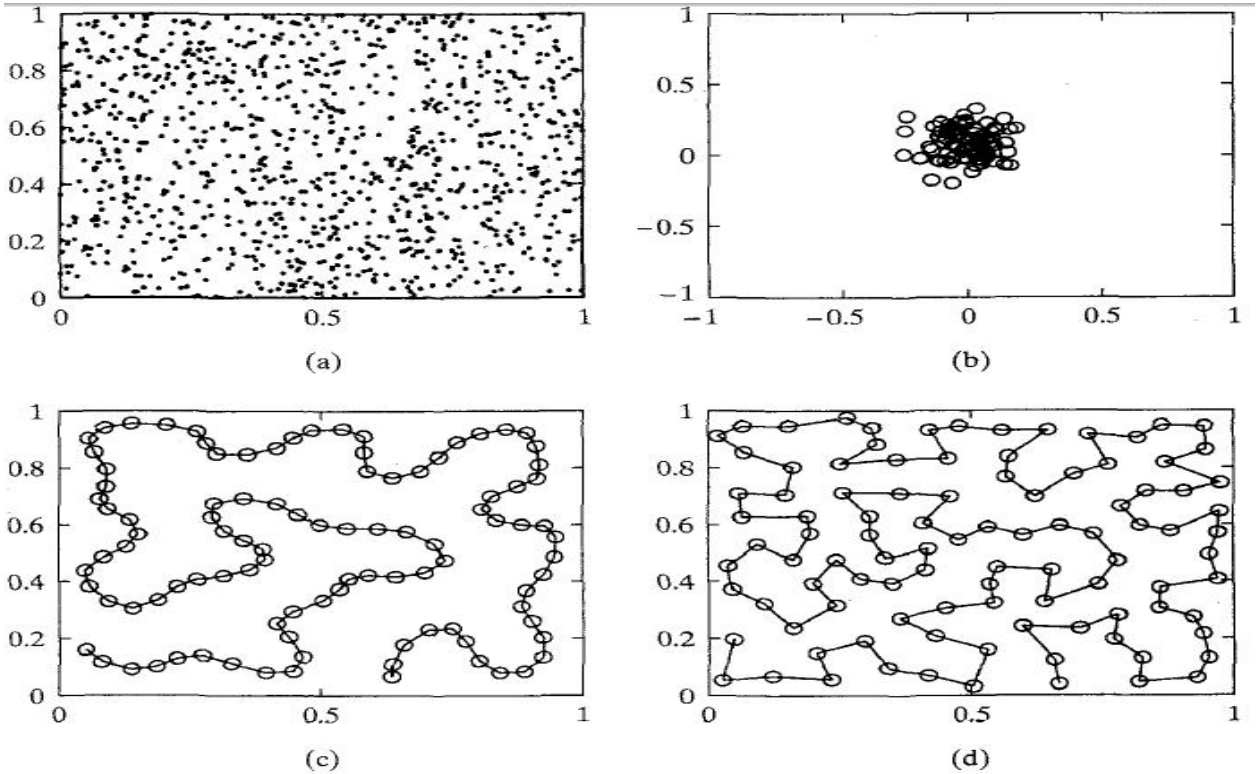
- Includes one layer of a lattice neurons which defines a discrete output space
- A time varying neighborhood function $N_m$ is defined around a winning neuron $m$
- A learning rate parameter $\alpha$ is defined that also can be considered decreasing in time gradually
- The algorithm includes five main steps:
  1. Initialization Choose random values for the initial weight vectors $W_j(0)$. Keep the magnitude of the weights small and different from each other
  2. Sampling Draw a sample $x$ from input space. It is applied to the lattice.
  3. Similarity Matching Find the winner neuron $m$th, based on Euclidean distance criterion
  4. Updating Adjust the weights of the winner and its neighbors based on $\triangle w_i(t) = \alpha(N_i, t)[x(t) - w_i(t)]$ for $i \in N_m(t)$
  5. Continuation Continue with step 2 until no noticeable changes in the feature map is observed

# Properties of Feature Map

- Let $\mathcal{X}$: Input space; $\mathcal{A}$: Discrete output space; $\Phi : \mathcal{X} \rightarrow \mathcal{A}$ a nonlinear transformation called feature map represented by weight vectors $\{w_j\}$
- **P1:Approximation of Input Space**: SOM alg. stores a large set of input vectors $x \in \mathcal{X}$ by finding a smaller set of prototypes $w_j \in \mathcal{A}$
  - It is useful for dimension reduction or data compression
- **P2: Topological Ordering**: By approximating the input space $\mathcal{X}$ in the form of weights $w_j$, $\Phi$ provides a faithful representation of the important features that characterize the input vector $x \in \mathcal{X}$ in terms of a certain criterion.
- **P3: Density Matching**: The input vectors $x$ with higher probability of occurrence are mapped into larger domain of the output space with better resolution
- **P4: Feature Selection**: SOM selects the best features for approximating the underlying distribution

# Example

- Consider SOM with 100 neurons in 1 dimension; $x$ in two dimetions: $0 \leq x_1 \leq 1; 0 \leq x_2 \leq 1$
- The initial weights are chosen randomly




(a) Two-dimensional input data distribution. (b) Initial condition of the one-dimensional lattice. (c) Condition of the lattice at the end of the ordering phase. (d) Condition of the lattice at the end of the convergence phase.

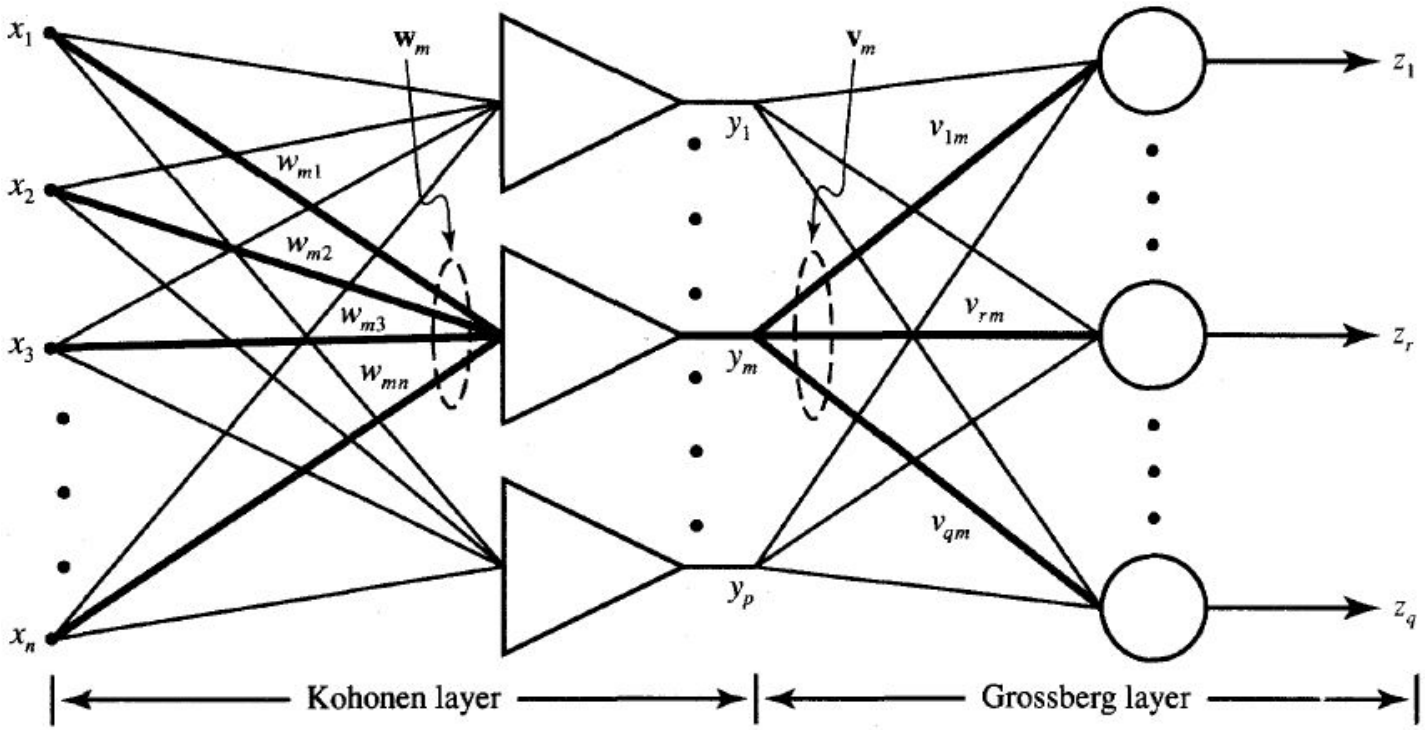
# Learning Vector Quantization (LVQ) (Kohonen 1990)

- It is a supervised pattern classification
- Each output represents a particular class
- The weights of each neuron represents the center of the corresponding class
- It has similar structure of Kohonen
- the only difference is in training which is supervised
- First the wining neuron is found
- Since it is supervised, it can be verified whether or not the winner is chosen correctly
  - The correct chosen winner is updated toward input vector
  - The wrong chosen winner is updated in away form the input vector

- $x$: training vector
- $T$: correct category or class for training vector
- $C_j$: the winner neuron
- $\alpha > 0$: learning rate
- After finding the winner neuron based on Euclidean distance
  - If $T = C_j$ $W_j(k+1) = W_j(k) + \alpha(x - W_j(k))$
  - If $T \neq C_j$ $W_j(k+1) = W_j(k) - \alpha(x - W_j(k))$

# Counterpropagation Network (CPN)

- CPN is a two-layer network consisting of two feedforward layers.
- It was introduced by Hecht-Nielsen (1987, 1988).
- Considering the vector pairs $(x_1, z_1), ..., (x_r, z_r)$, CPN can map $z = \Phi(x)$
  - relates $x_i$ to $z_i$ based in $\Phi(.)$
- If the inverse of $\phi$ exist, CPN can also learn inverse mapping $x = \Phi^{-1}(z)$
- Similar to hetro-associative memories, it relates input vector to output vector
- Compared to bidirectional associative memory , there is no feedback and delay activated during the recall operation mode.
  - $\therefore$ it can be trained to perform associative mappings much faster than a typical two-layer network.
- CPN is useful in pattern mapping and associations, data compression, and classification.




- It has two layers similar to ff networks
- BUT it has different learning strategy

- Consider the $r$ pairs of training data $(x_1, z_1^d), ..., (x_r, z_r^d)$
- The first layer is Kohonen Layer
  - The training is unsupervised; winner take-all mode
  - For example, if by applying $x_i$ the $m$th neuron wins:
    - The weights of the $m$th neuron is updated as $w_m^{k+1} = w_m^k + \alpha^k(x_i - w_m^k)$
    - $\alpha^k$ may be reduced at each iteration
  - The training is continued by applying the next training data and updating the weights of the winner neuron
  - When training is terminated, the output of this layer, $y$ represents an input cluster
  - Therefore in recall mode, if the input $x_i$ belongs to $m$th cluster, the output vector $y_i$ of the Kohonen layer becomes

$$y_i = [0\ 0\ ...\ 1\ ...\ 0] \quad (1)$$

- The second layer is Grossberg Layer
  - It has outstar learning mode which is a supervised learning
  - Activation fcn is $sgn(.)$
  - By applying $x_i$, the weights are updated as : $\Delta v_m = \beta(z_i^d - v_m)$
    - $\beta$: learning rate can be reduced gradually
    - $z_i^d$: desired output of $x_i$
    - $v_m$: Weight vector Output layer
  - The training is continued by applying the next training data and updating the weights of the corresponding neuron
  - If $E = \sum_{i=1}^{r}(z_i^d - z_i) > E_{max}$, the training data is resubmitted to the network and weight matrix V is updated
  - After the training is ceased, in recall mode $z = \Gamma[Vy]$, where $V = [v_1\ v_2\ ...v_p]$ is the weight matrix
  - Considering (1) as input of Gorssberg layer $\rightsquigarrow z = \Gamma[v_m]$

- After training first layer, $p$ clusters are formed among sample input x; each $w_k; k = 1, ..., p($ $p : \#$ of first layer neurons) is a representative of a cluster (average).
- After training the second layer, cluster $k$ is mapped to an output vector z, which is the average of $\phi(x); x \in$ cluster $k$
- It can be shown that after training $w_k \rightarrow < x >, v_k \rightarrow < \phi(x) >$
  - $< x >$ is the mean of all training samples that make the $k$th neuron win
- CPN works like a look-up of math table.
- For any input $x$, finds a region where $x$ falls (represented by the wining y node);
- Then uses the region as the index to look-up the table for the function value.
- The more cluster nodes $y$, the more accurate mapping is achieved.

# Full CPN

- If both $z = \Phi(x)$ and $x = \Phi^{-1}(z)$ exist, we can establish bi-directional approximation
- Two pairs of weights matrices should be trained:
  - $W(x\ to\ y)$ and $V(y\ to\ z)$ for approx. map $x$ to $z = \Phi(x)$
  - $U(z\ to\ y)$ and $T(y\ to\ x)$ for approx. map $z$ to $x = \Phi^{-1}(z)$
- When training sample $(x_i, z_i)$ is applied, they may separately determine the winner $y_{m(x)}$ or $y_{m(z)}$

# Adaptive Resonance Theory (ART)

- It is developed by Carpenter and Grossberg (1987-1988)
- **Motivations**: In Previous networks:
- Number of class nodes is pre-determined and fixed.
  - Under- and over- classification may result from training
  - Some classes may have empty members.
  - There is no control on similarity degree of inputs grouped in one class.
- Training is non-incremental:
  - Adding new samples often requires re-train the network with the enlarged training set until a new stable state is reached.

# ART

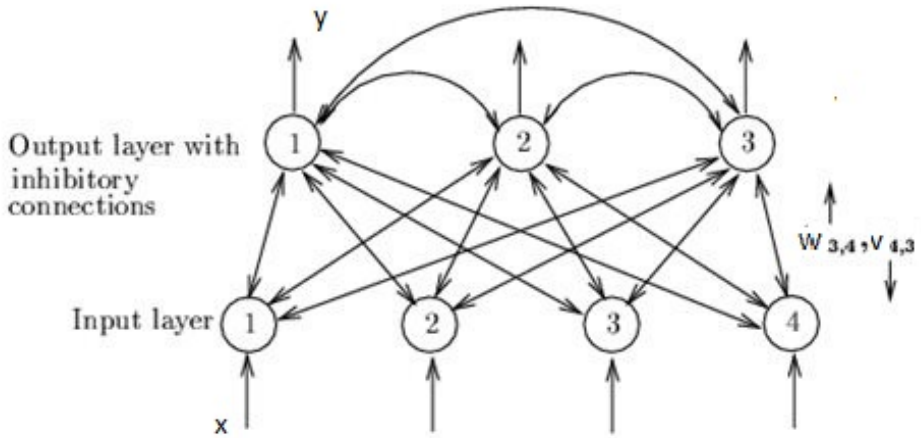
- It is applied for cluster discovery
- The network learns the clusters in an unsupervised mode.
- The novelty of ART: It can accommodate new clusters without affecting the stored clusters
- The network produces the cluster by itself without a priori knowledge info about the possible number of and type of clusters
- There are three types of ART:
  1. ART1: for binary data [**?**],[**?**]
  2. ART2: for continuous data [**?**],[**?**]
  3. ART3: for hybrid data

- Art gets its name from the way it learns or recall
- In physics, resonance occurs when a small-amplitude vibration of a proper freq. causes large-amplitude vibration
- If the proper patterns develop, a stable oscillation ensues, which is the NN equivalent of resonance.
- When an input vector is presented:
  - If the network has learned it previously, a resonant state is achieved quickly.
  - Otherwise, it searches to the stored patterns for a match.
  - If no match is found, a new pattern is stored.
- The previously learned patterns remains without change

# How ART Performs

- Suppose the input samples have been appropriately classified into k clusters (by a competitive learning).
- The weight vector of $j$th neuron, $w_j$, is a representative (average) of all samples in that cluster.
- When a new input vector $x$ arrives
- Find the winner $j$ among all $k$ cluster nodes
- Compare, $w_j$ with x
  - If they are sufficiently similar then update $w_j$ properly
  - Otherwise, find/create a free class node and make $x$ as its first member.
- To Design ART, we need:
  1. A mechanism to determine (dis)similarity between $x$ and $w_j$
  2. A control for finding/creating new class nodes.
  3. All operations to be implemented by units of local computation.

# ART Architecture

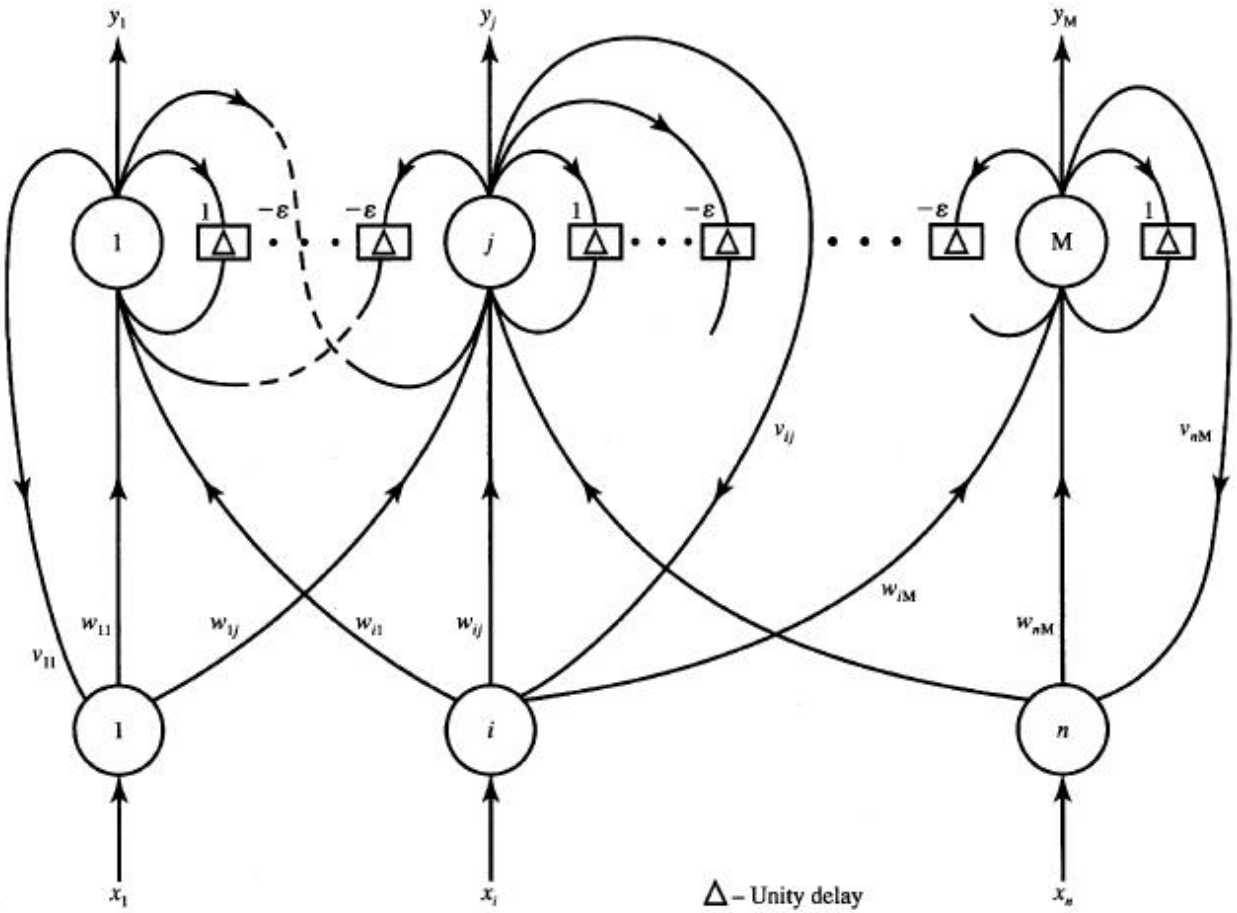



- $x$: Input vector
- $y$: Output vector (classes)
- $w_{ji}$: Bottom-Top weights from $x_i$ to $y_j$
- $v_{ji}$: Top-Bottom weights from $y_j$ to $x_i$
- $\rho$: vigilance parameter for similarity comparison $(0 < \rho < 1)$

# ART1

## 1. Bottom-top processing

- It functions like Hamming net MAXNET
- Finds the best match and winner neuron of output (e.x. $m$th cluster)



# ART1

2. Top-Bottom Processing
   - Checks how much similar $x$ and cluster $m$ is
   - $0 < \rho < 1$ is the threshold for vigilance test
   - If it is similar both $W$ and $V$ are updated properly
   - Otherwise a new cluster is added then the weights are updated

# ART1 Algorithm

1. Set vigilance threshold $0 < \rho < 1$; for $n$-tuple input vectors, and $(m = 1)$-tuple output vectors. Input $\mathbf{x} = [x_i]$ is assumed binary unipolar.
2. Initialize the Bottom-top weight matrix, $W_{n\times m}$, and Top-bottom weight matrix, $V_{n\times m}$: $w_{ij} = \frac{1}{1+n}, v_{ji} = 1$
3. Hamming net-MAXNET
   - Find all matching score: $y_r^0 = \sum_{i=1}^{n} w_{ir} x_i$ for $r = 1, ..., m$
   - Then find the best matching cluster, $j$: $y_j^0 = max_{r=1,...,m}(y_r^0)$
4. Check the similarity test for the winning neuron $j$: $\frac{1}{\|x\|} \sum_{i=1}^{n} v_{ij} x_i > \rho$; where $\|x\| = \sum_{i=1}^{n} |x_i|$
   - If the test is passed go to the next step;
   - Otherwise if the only top layer has more than a single active node left goes to step 6;
   - Otherwise add new cluster in output $m = m + 1$ and go to the next step

# ART1 Algorithm

5. The weights of the winner cluster or newly created one, $j$, is updated : $w_{ij} = \frac{v_{ij}(t)x_i}{0.5+\sum_{i=1}^{n} v_{ij}(t)x_i}$, $v_{ij}(t+1) = x_i v_{ij}(t)$
6. The node $j$ is deactivated by setting $y_j$ to 0. So this mode will not participate in current cluster search. Now go back to step 3

- The vigilance threshold $\rho$ indicates how close an input must be to a stored cluster prototype for a desirable match.
- If *rho* is closed to 1 $\rightsquigarrow$ a close match is required;
- Smaller $\rho$ indicates a poorer match is acceptable. $\rightsquigarrow$ a smaller number of clusters being discovered by the ARTI network.
- In recall Mode:
  - Similar to learning mode
  - But the vigilance test is always passed and no weight updates

# Example

- Consider four patterns with 25-dimensional vector with 0 and 1 entries
- Learn with two different vigilance factors: $\rho_1 = 0.7, \rho_2 = 0.3$
- Initialized weights: $w_{ij} = \frac{1}{26}, v_{ij} = 1, i = 1, ..., 25, j = 1$
- Present pattern $A$
- The vigilance test is unconditionally passed
- If $x_i = 1$ then $w_{i1} = \frac{2}{11}; v_{i1} = 1,$
  - $w_{1,1} = w_{7,1} = w_{13,1} = w_{19,1} = w_{25,1} = \frac{2}{11}$, other $w_{i1} = 0$
  - $v_{1,1} = v_{7,1} = v_{13,1} = v_{19,1} = v_{25,1} = 1$, other $v_{i1} = 0$

# Example Cont'd

- Present pattern $B$
  - Only one neuron exist $\rightsquigarrow$ no competition
  - The vigilance test: $\frac{1}{\|x\|}\sum_{i=1}^{25} v_{i1}x_i = \frac{5}{9} < 0.7$
  - The test is failed $\rightsquigarrow$ new neuron is added to to top layer
  - $w_{1,2} = w_{5,2} = w_{7,2} = w_{9,2} = w_{13,2} = w_{17,2} = w_{19,2} = w_{21,2} = w_{25,2} = \frac{2}{19}$, other $w_{i2} = 0$
  - $v_{1,2} = v_{5,2} = v_{7,2} = v_{9,2} = v_{13,2} = v_{17,2} = v_{19,2} = v_{21,2} = {}_{25,2} = 1$, other $v_{i2} = 0$
- Present Pattern $C$
  - $y_1^0 = 5(\frac{2}{11}) + 8(\frac{1}{26}) = 1.217$, $y_2^0 = 9(\frac{2}{19}) + 4(\frac{1}{26}) = 1.101$
  - Neuron 1 is winner
  - The vigilance test: $\frac{5}{13} < 0.7 \rightsquigarrow$ it fails
  - Node 1 is disabled, only 1 neuron is remained
  - The vigilance test for neuron 2: $\frac{9}{13} < 0.7 \rightsquigarrow$ it fails
  - Add new neuron and update the wights accordingly

# Example Cont'd

- Present pattern $D$
  - Neuron 3 is the winner and passes the vigilance test.
- By $\rho_1 = 0.7$ the patterns are classified in three classes
- repeat this example for $\rho_2 = 0.3$ classifies the patterns in two classes only

# Example Cont'd