# Self Organizing Map

Nguyen Ngoc Thao

Department of Computer Science, FIT
University of Science, VNU-HCM

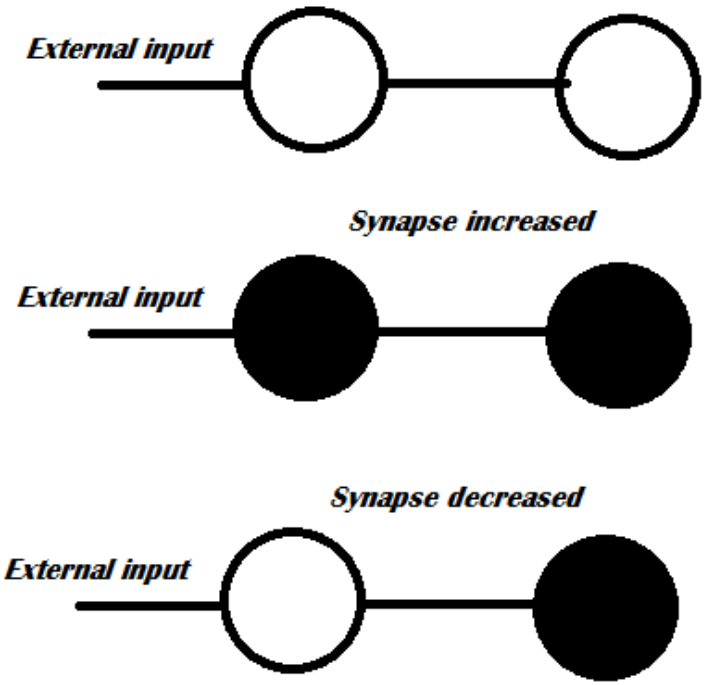
# Content outline

- Hebbian learning
- Competitive learning

# Hebbian Learning

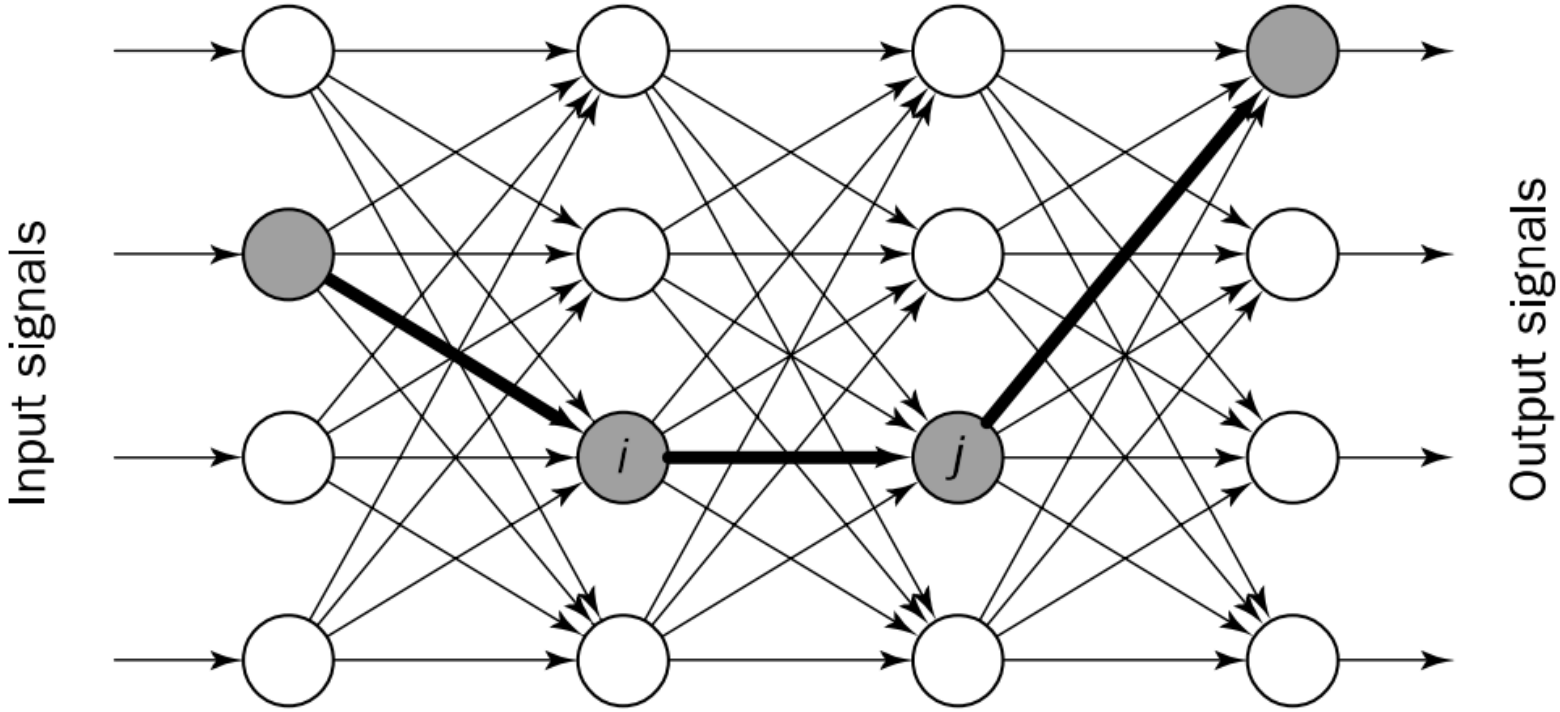
# Hebbian learning (Hebb, 1949)

- If neuron $i$ is near enough to excite neuron $j$ and repeatedly joins in its activation, their synaptic connection is enhanced.
- Neuron $j$ becomes more sensitive to stimuli from neuron $i$.



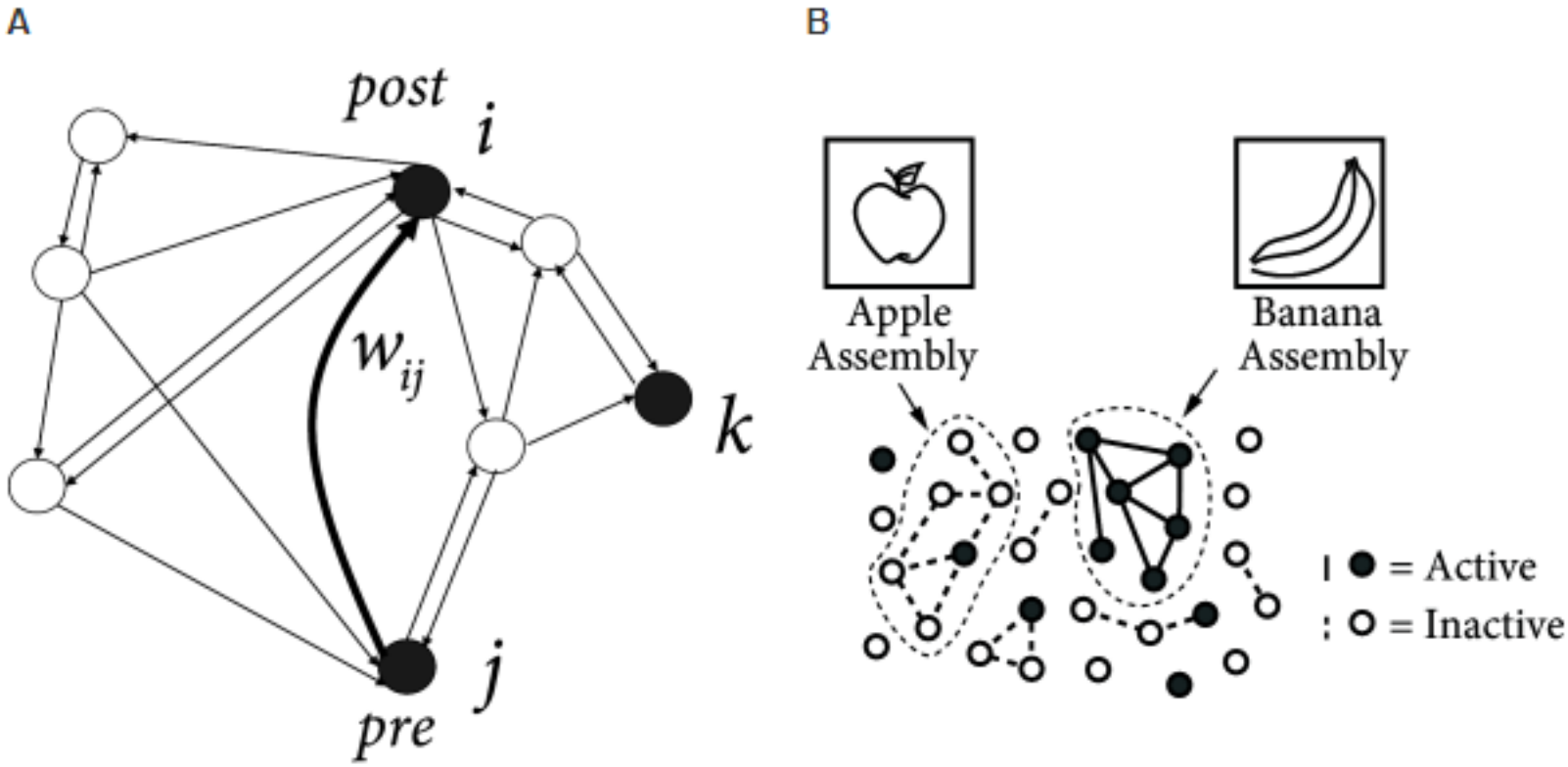
# Hebbian learning in ANN

- A neural network really can learn to associate stimuli commonly presented together.
- Learning here occurs without feedback from the environment.




Hebbian learning in a neural network

# Hebbian learning: An example




(A) The change of a synaptic weight $w_{ij}$ depends on the state of the presynaptic neuron $j$ and the postsynaptic neuron $i$ and the present efficacy $w_{ij}$, but not on the state of other neurons $k$. (B) Hebbian learning strengthens the connectivity within assemblies of neurons that fire together, e.g., during the perception of banana.

# Hebbian learning rule

- Step 1: Initialization
  - Initial weights $\boldsymbol{w_1, w_2, \ldots, w_n}$ and threshold $\boldsymbol{\theta}$ are randomly assigned to numbers $\in [0,1]$.
  - The learning rate $\boldsymbol{\alpha}$ and forgetting factor $\boldsymbol{\phi}$ are small positive values.
- Step 2: Activation
  - Compute the output at iteration $\boldsymbol{p}$:

$$\boldsymbol{y_j(p) = \sum_{i=1}^{n} x_i(p) w_{ij}(p) - \theta_j}$$

where $\boldsymbol{n}$ is the number of inputs, and $\boldsymbol{\theta_j}$ is the threshold of neuron $\boldsymbol{j}$.

# Hebbian learning rule

- Step 3: Learning
  - Update the weights in the network: $w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$

    where $\boldsymbol{\Delta w_{ij}(p)}$ is the weight correction at iteration $\boldsymbol{p}$
  - The generalized activity product rule from Hebb's Law

    $$\boldsymbol{\Delta w_{ij}(p) = \phi\, y_j(p)\left[\lambda x_i(p) - w_{ij}(p)\right]} \text{ where } \boldsymbol{\lambda = \alpha/\phi}$$
- Step 4: Iteration
  - Increase iteration $\boldsymbol{p}$ by one, go back to Step 2 and continue until the synaptic weights reach their steady-state values.

# The forgetting factor $\phi$

- The **forgetting factor** $\phi$ (usually $\in [0, 1]$) specifies the weight decay in a single learning cycle.
- Recalling the Hebb's Law:

$$\Delta w_{ij}(p) = \alpha\, y_j(p) x_i(p) - \phi y_j(p) w_{ij}(p)$$

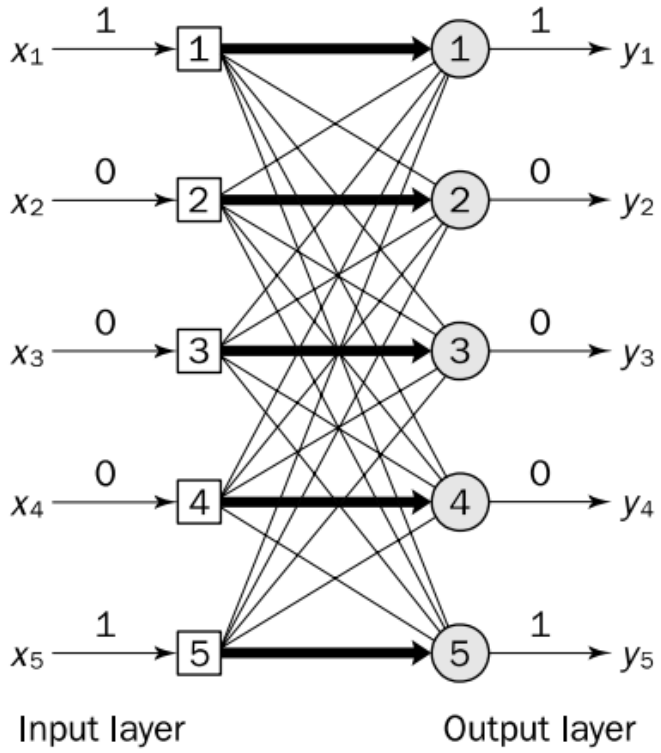
- $\phi = 0$: the network only strengthens its synaptic weights $\rightarrow$ weights may grow towards infinity.
- $\phi = 1$: the network remembers very little of what it learns
- $\phi \in [0.01, 0.1]$: a little "forgetting" while limiting the weight growth

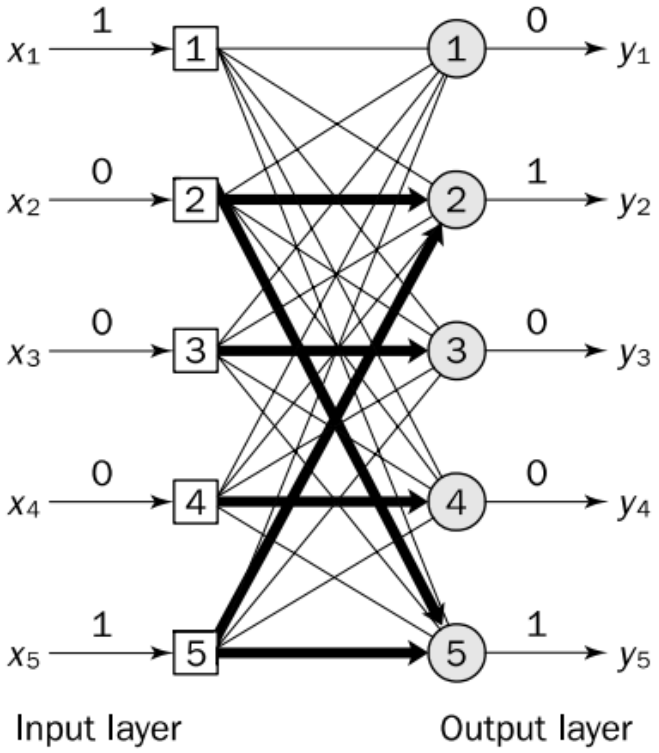
# Hebbian learning: An example

- Consider a fully connected feedforward network with a single layer of five computation neurons.
  - McCulloch-Pitt neurons with the sign activation function
  - Trained with generalized activity product rule
- The set of input vectors are

$$\mathbf{X}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{X}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{X}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{X}_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{X}_5 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- The thresholds are random numbers $\in [0,1]$.
- $\alpha = 0.1$ and $\phi = 0.02$




Output layer

| Input layer | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 |

Output layer

| Input layer | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 2.0204 | 0 | 0 | 2.0204 |
| 3 | 0 | 0 | 1.0200 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0.9996 | 0 |
| 5 | 0 | 2.0204 | 0 | 0 | 2.0204 |

Unsupervised Hebbian learning in a single-layer network: (top) initial and final states of the network; (bottom) initial and final weight matrices

# Hebbian learning: An example

- The test input vector $X = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ will have $Y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$.

$$\mathbf{Y} = sign\left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \\ 0 & 0 & 1.0200 & 0 & 0 \\ 0 & 0 & 0 & 0.9996 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.4940 \\ 0.2661 \\ 0.0907 \\ 0.9478 \\ 0.0737 \end{bmatrix} \right\} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- The input $x_5$ is associated with outputs $y_2$ and $y_5$ because inputs $x_2$ and $x_5$ were coupled during training.
- The input $x_1$ is not associated with the output $y_1$ because it did not appear during training.

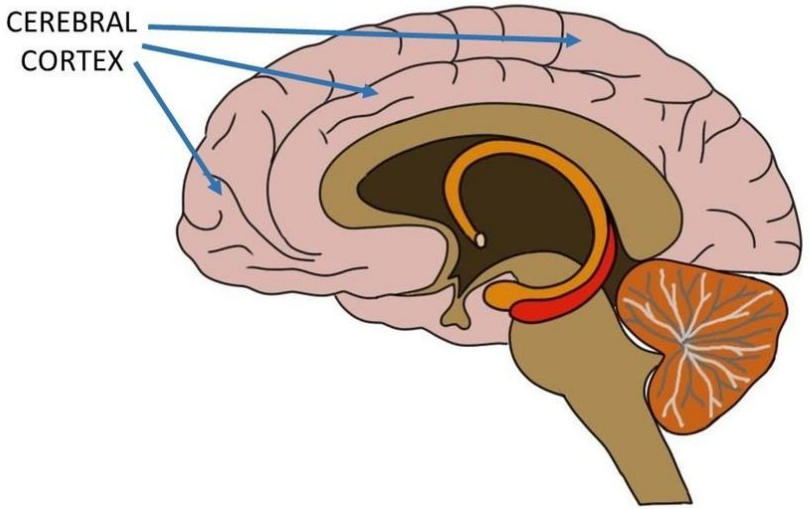
# Competitive Learning for Self Organizing Map

# Competitive learning (early 1970s)

- Neurons compete among themselves to be activated → only a single output neuron is active at any time
  - Meanwhile, in Hebbian learning, several output neurons can be activated simultaneously
- The output neuron that wins the "competition" is called the **winner-takes-all** neuron.
- This learning paradigm is the basis for the Self-organizing feature maps (SOM), (Kohonen, 1989)

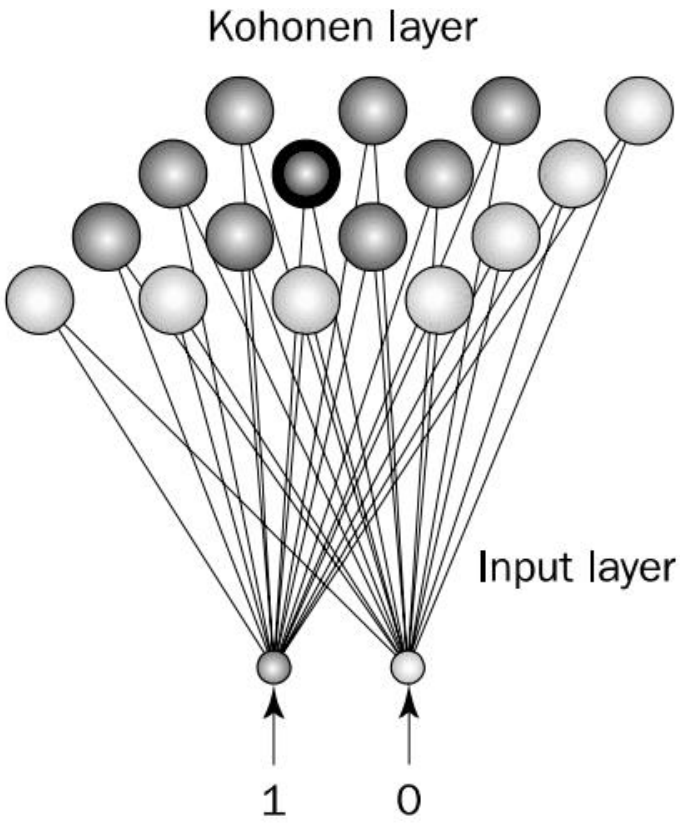
# What is a self-organising feature map?

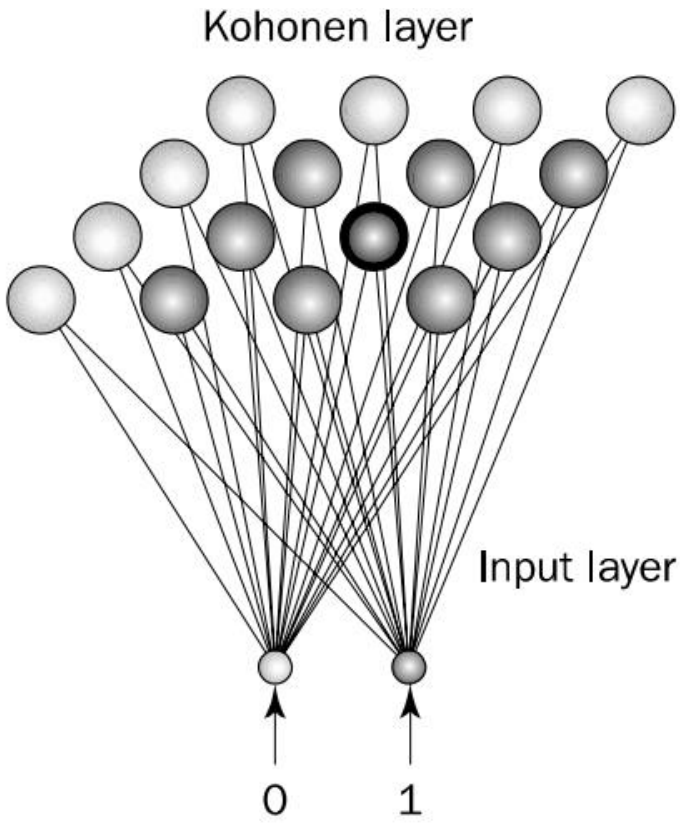
- Areas in a cerebral cortex are responsible for different human activities (motor, visual, auditory, etc.)
  - Different sensory inputs are mapped into corresponding areas.
- The cortex is a self-organizing computational map in the human brain.



# Kohonen model (Kohonen, 1990)

- Principle of topographic map formation: The spatial location of an output neuron in the topographic map corresponds to a particular feature of the input pattern.



# How do Kohonen networks work?

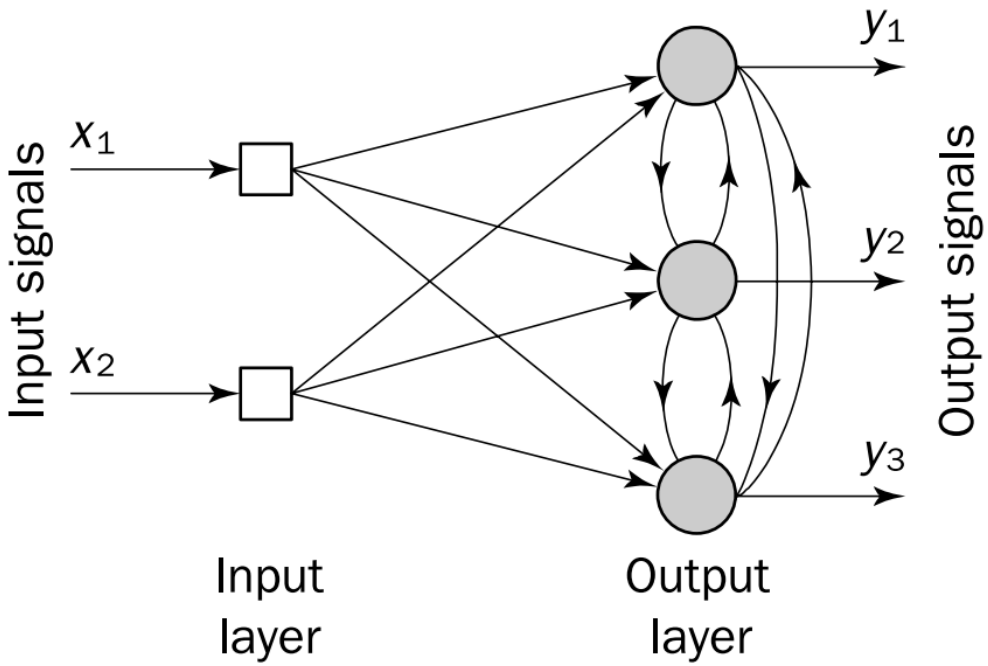
- Kohonen network consists of a single layer of computation neurons and two different types of connections.

- Forward connections

from the neurons in the input layer to those in the output layer, and

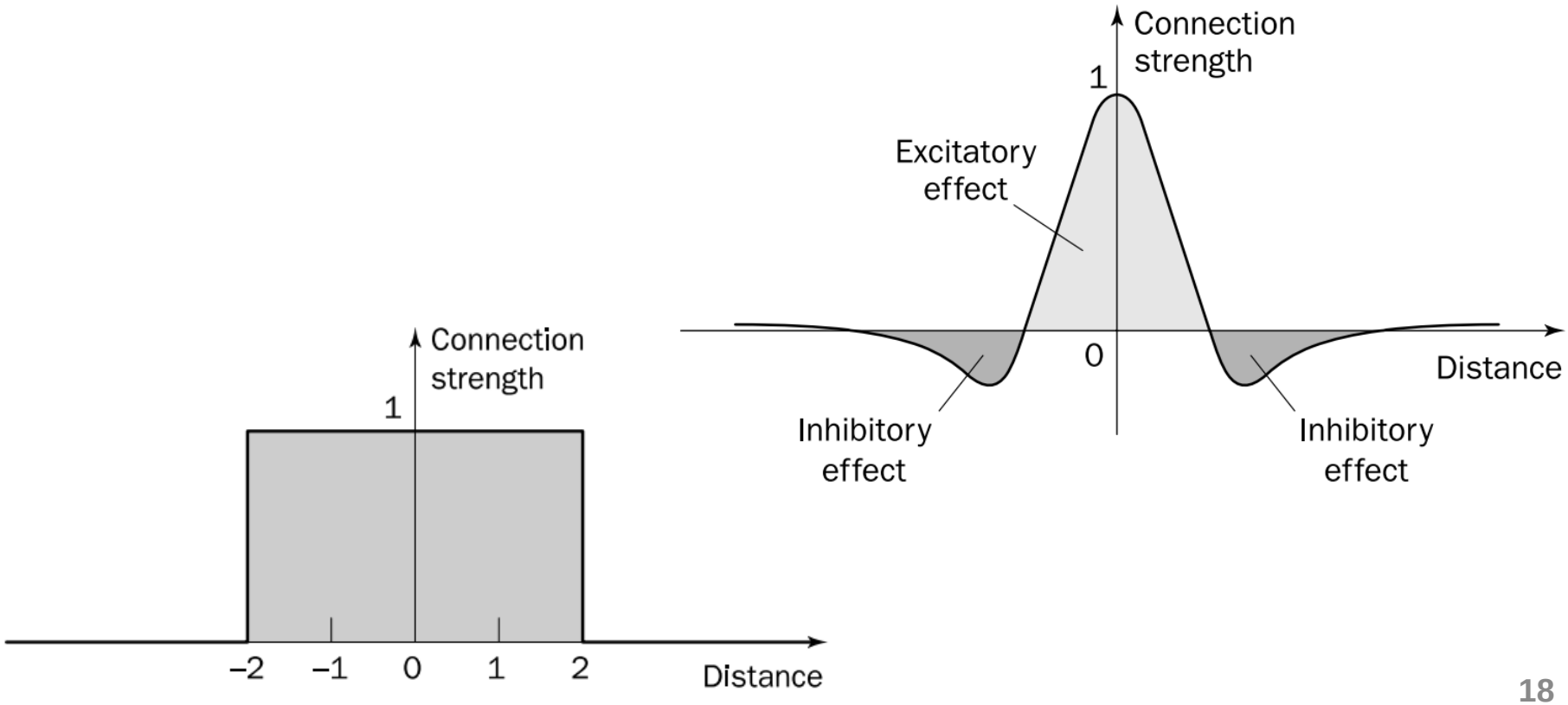
- Lateral connections

between neurons in the output layer



# Lateral connections

- Neurons competes with others through lateral connections.
- The winner-takes-all neuron has the largest activation level among all neurons, while the activity of others is suppressed.



# Learning in Kohonen network

- A fixed number of input patterns are placed into a higher-dimensional output (also called Kohonen layer)
- A neuron learns by shifting its weights from inactive connections to active ones.
- Only the winner-takes-all neuron and its neighbors can learn.
  - The output signal, $y_j$, of the winning neuron $j$ is set equal to 1 and those of all the other neurons (that lose the competition) are set to 0.

# Kohonen learning algorithm

- Step 1: Initialization
  - Initial synaptic weights are set to small random values $\in [0,1]$ and the learning rate $\alpha$ is a small positive value.
- Step 2: Activation and similarity matching
  - Apply the input vector $\mathbf{X}$ and find the winner-takes-all neuron $j_{\mathbf{X}}$ at iteration $\boldsymbol{p}$, using the minimum-distance Euclidean criterion

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

where $\boldsymbol{n}$ is the number of neurons in the input layer, and $\boldsymbol{m}$ is the number of neurons in the output or Kohonen layer.

# Kohonen learning algorithm

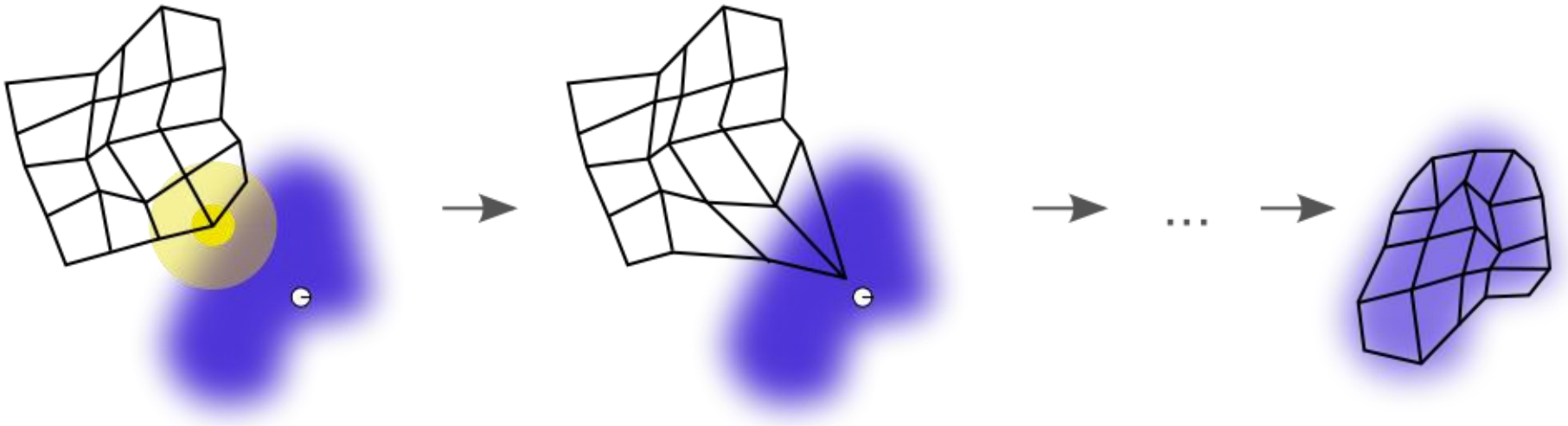
- Step 3: Learning
  - Update the synaptic weights: $w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$

    where $\boldsymbol{\Delta w_{ij}(p)}$ is the weight correction at iteration $\boldsymbol{p}$
  - The **competitive learning rule** (Haykin, 1999) for weight correction

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

    where $\boldsymbol{\alpha}$ is the learning rate $(0 < \alpha < 1)$, and $\boldsymbol{\Lambda_j(p)}$ is the neighborhood function centered around the winner-takes-all neuron $\boldsymbol{j_X}$ at iteration $\boldsymbol{p}$
- Step 4: Iteration
  - Increase iteration $\boldsymbol{p}$ by one, go back to Step 2 and continue until the stopping criterion is satisfied.

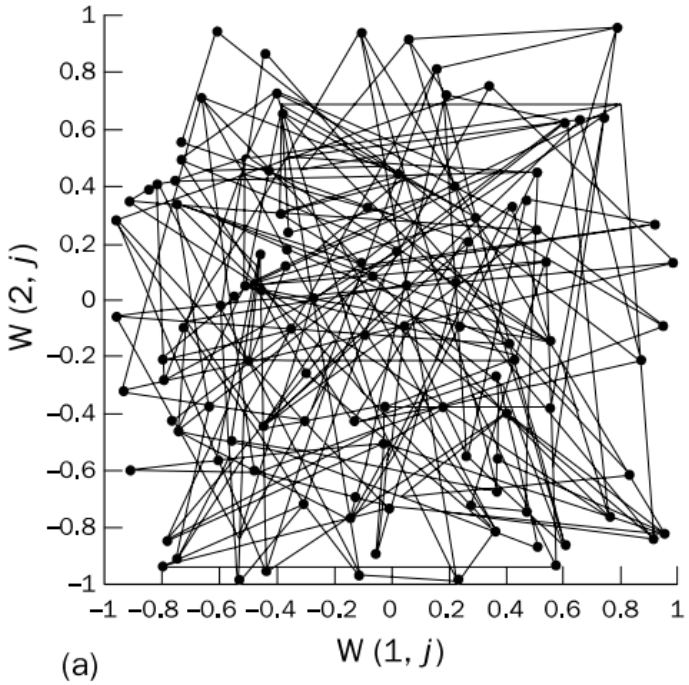
# Kohonen learning: An illustration

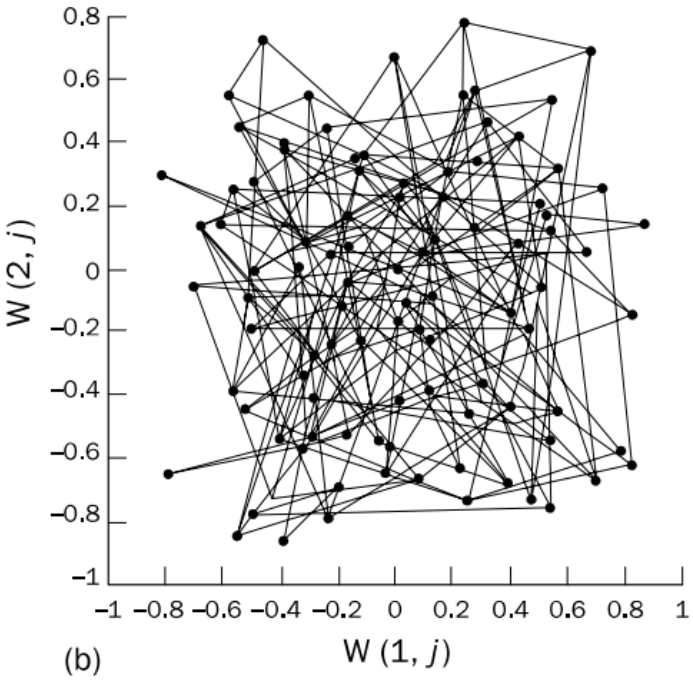


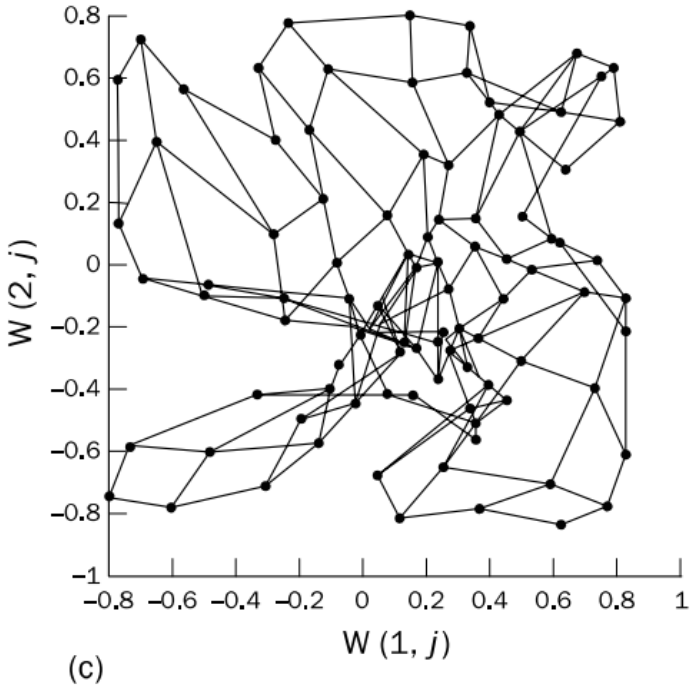
The blue blob is the distribution of the training data, and the small white disc is the current training sample drawn from that distribution. At first (left) the SOM nodes are arbitrarily positioned in the data space. The node nearest to the training node (highlighted in yellow) is selected, and is moved towards the training datum, as (to a lesser extent) are its neighbors on the grid. After many iterations the grid tends to approximate the data distribution (right).

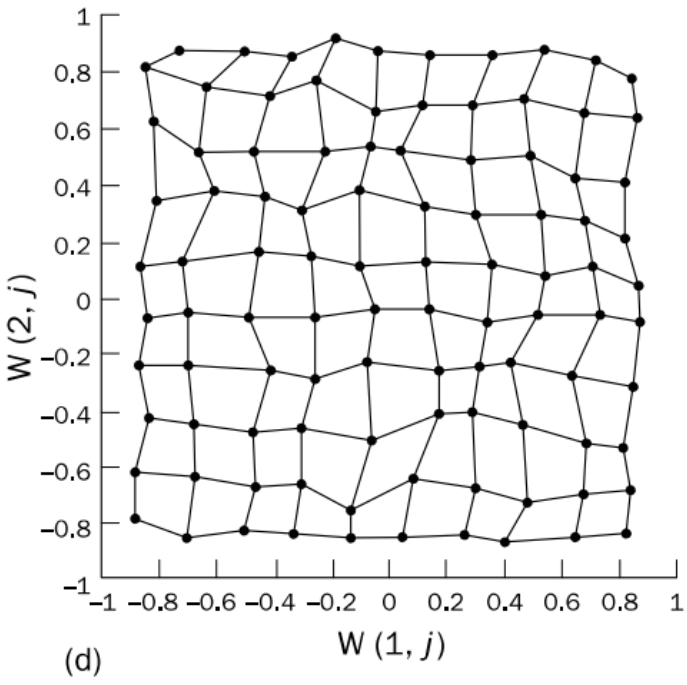
# Kohonen network: An example

- Consider the Kohonen network with 100 neurons arranged in a two-dimensional lattice of size $10 \times 10$.
- **Target:** classify two-dimensional input vectors
- Each Kohonen neuron should respond only to the input vectors occurring in its region.
- 1000 input vectors place randomly in the region $[-1, +1]$
- Initial synaptic weights are random values $\in [-1, +1]$, and learning rate $\alpha = 0.1$.



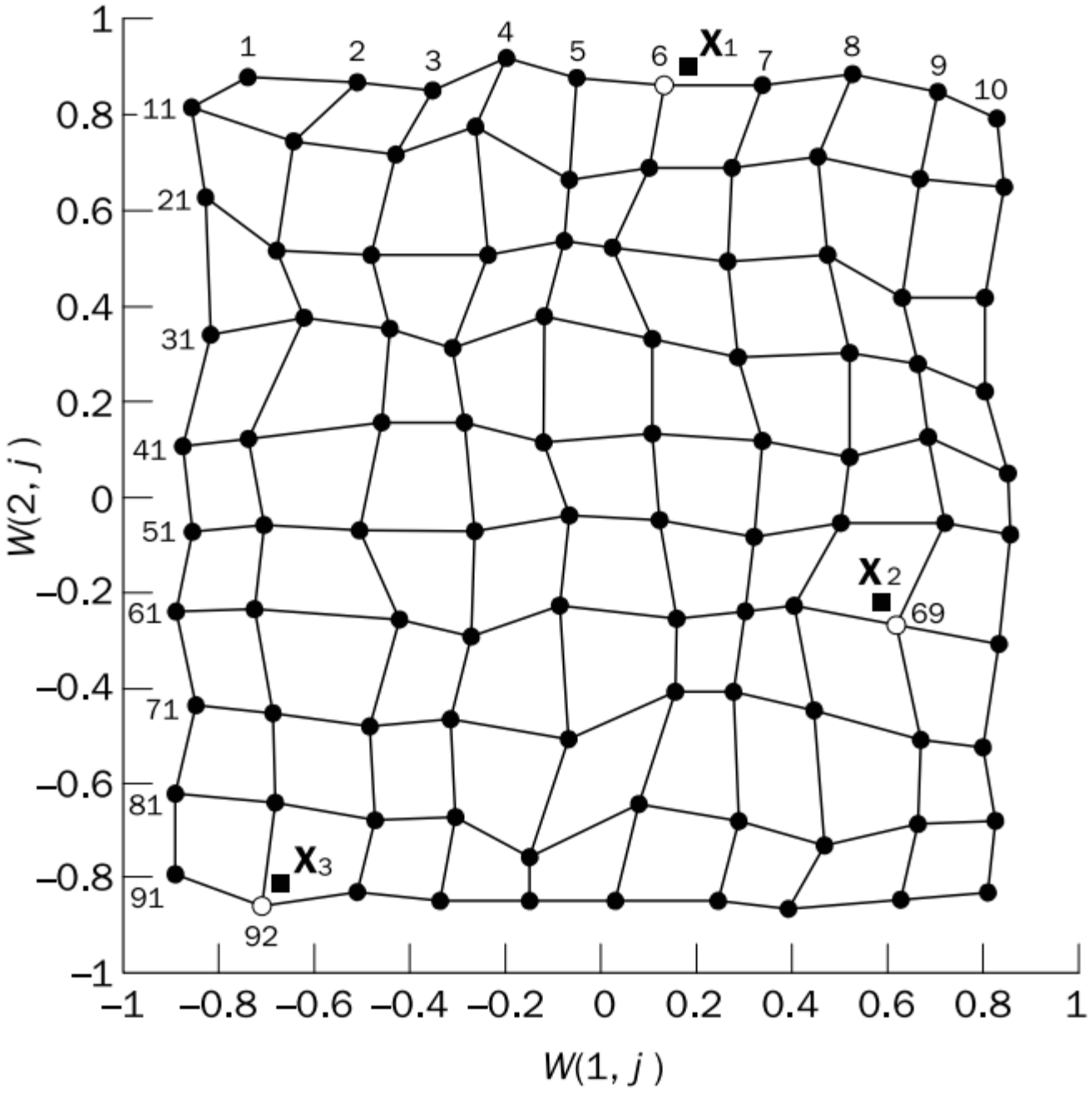

Competitive learning in the Kohonen network:

(a) initial random weights;

(b) network after 100 iterations;

(c) network after 1000 iterations;

(d) network after 10,000 iterations




$$\mathbf{X}_1 = \begin{bmatrix} 0.2 \\ 0.9 \end{bmatrix}$$

$$\mathbf{X}_2 = \begin{bmatrix} 0.6 \\ -0.2 \end{bmatrix}$$

$$\mathbf{X}_3 = \begin{bmatrix} -0.7 \\ -0.8 \end{bmatrix}$$

Topologically ordered feature map displayed in the input space


THE
END