Lecture 8

Artificial neural networks:

Unsupervised learning

- Introduction
- Hebbian learning
- Generalised Hebbian learning algorithm
- Competitive learning
- Self-organising computational map: Kohonen network
- Summary

Introduction

The main property of a neural network is an ability to learn from its environment, and to improve its performance through learning. So far we have considered **supervised** or **active learning** – learning with an external "teacher" or a supervisor who presents a training set to the network. But another type of learning also exists: **unsupervised learning**.

- In contrast to supervised learning, unsupervised or **self-organised learning** does not require an external teacher. During the training session, the neural network receives a number of different input patterns, discovers significant features in these patterns and learns how to classify input data into appropriate categories. Unsupervised learning tends to follow the neuro-biological organisation of the brain.
- Unsupervised learning algorithms aim to learn rapidly and can be used in real-time.

Hebbian learning

In 1949, Donald Hebb proposed one of the key ideas in biological learning, commonly known as **Hebb's Law**. Hebb's Law states that if neuron *i* is near enough to excite neuron *j* and repeatedly participates in its activation, the synaptic connection between these two neurons is strengthened and neuron *j* becomes more sensitive to stimuli from neuron *i*.

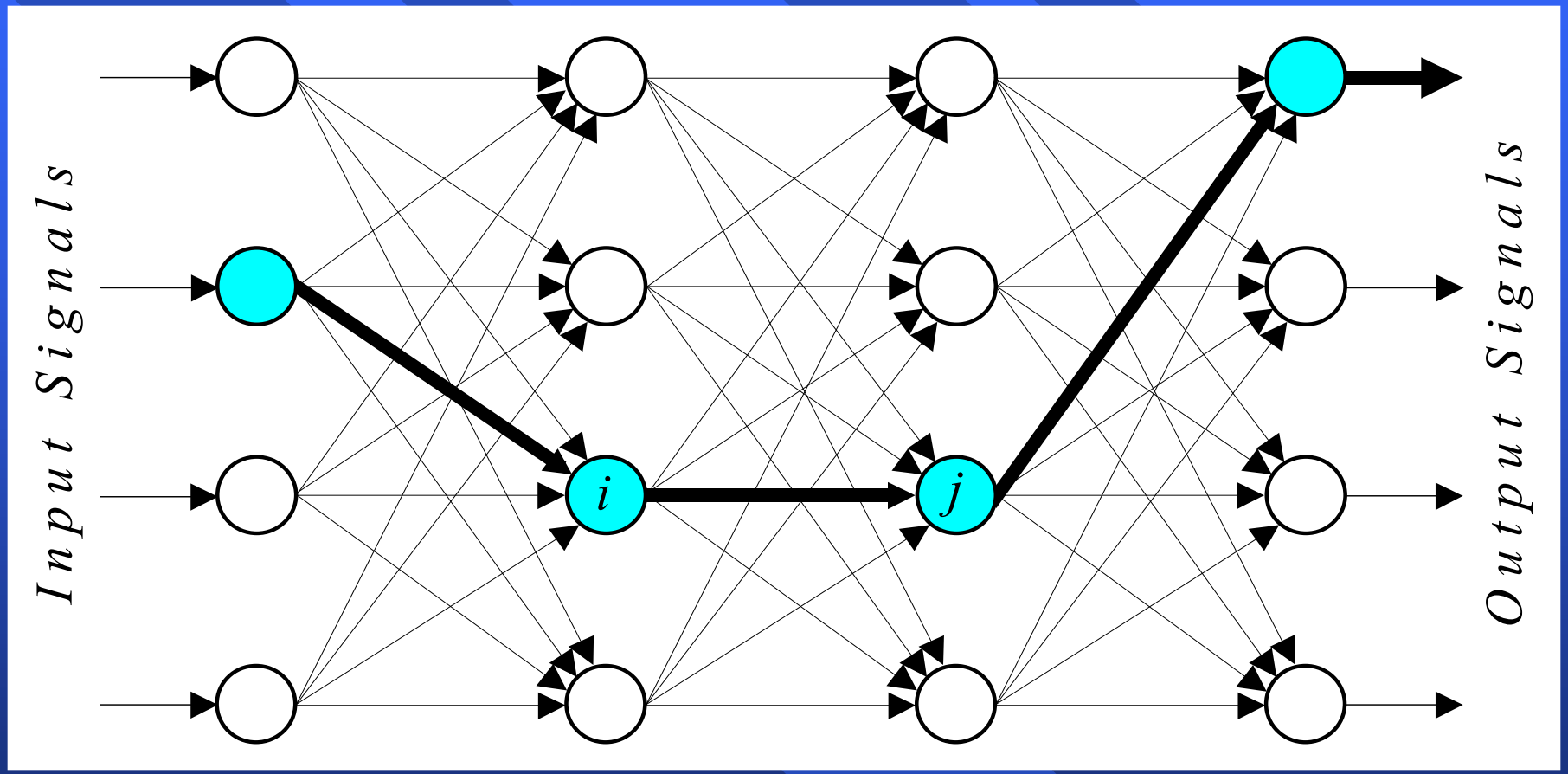
Hebb's Law can be represented in the form of two rules:

1. **If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.**

2. **If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.**

Hebb's Law provides the basis for learning without a teacher. Learning here is a **local phenomenon** occurring without feedback from the environment.

Hebbian learning in a neural network



- Using Hebb's Law we can express the adjustment applied to the weight w_{ij} at iteration p in the following form:

$$\Delta w_{ij}(p) = F[y_j(p), x_i(p)]$$

- As a special case, we can represent Hebb's Law as follows:

$$\Delta w_{ij}(p) = \alpha \; y_j(p) \; x_i(p)$$

where α is the *learning rate* parameter.

This equation is referred to as the **activity product rule**.

- Hebbian learning implies that weights can only increase. To resolve this problem, we might impose a limit on the growth of synaptic weights. It can be done by introducing a non-linear **forgetting factor** into Hebb's Law:

$$\Delta w_{ij}(p) = \alpha \; y_j(p) \; x_i(p) - \varphi \; y_j(p) \; w_{ij}(p)$$

where φ is the forgetting factor.

Forgetting factor usually falls in the interval between 0 and 1, typically between 0.01 and 0.1, to allow only a little "forgetting" while limiting the weight growth.

Hebbian learning algorithm

Step 1: Initialisation.

Set initial synaptic weights and thresholds to small random values, say in an interval [0, 1].

Step 2: Activation.

Compute the neuron output at iteration p

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

where n is the number of neuron inputs, and θ_j is the threshold value of neuron j.

Step 3: Learning.

Update the weights in the network:

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

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

The weight correction is determined by the generalised activity product rule:

$$\Delta w_{ij}(p) = \varphi\, y_j(p)[\lambda\ x_i(p) - w_{ij}(p)]$$

Step 4: Iteration.

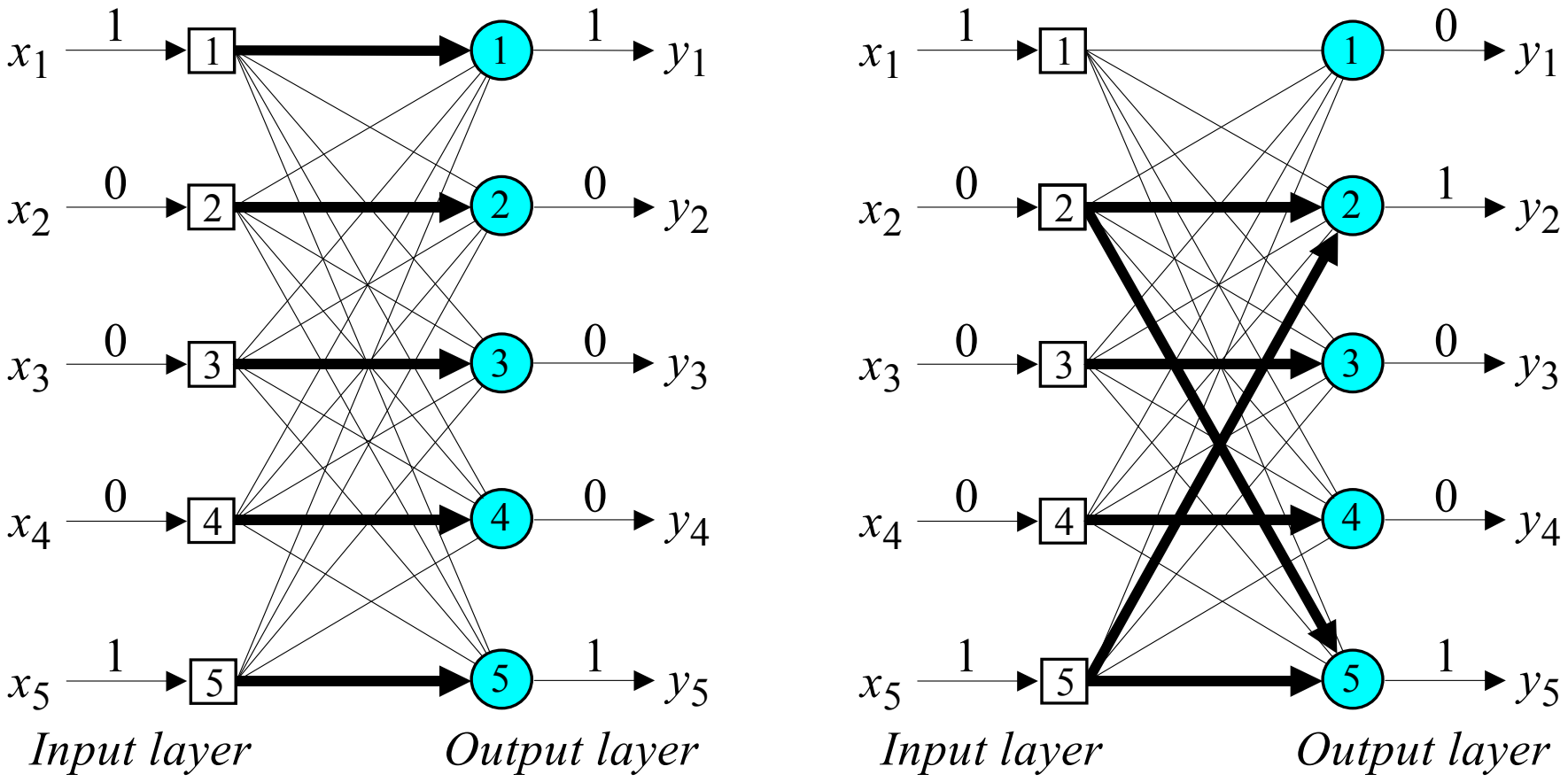
Increase iteration p by one, go back to Step 2.

Hebbian learning example

To illustrate Hebbian learning, consider a fully connected feedforward network with a single layer of five computation neurons. Each neuron is represented by a McCulloch and Pitts model with the *step* activation function. The network is trained on the following set of input vectors:

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

Initial and final states of the network



Initial and final weight matrices

Input layer \ Output 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

Input layer \ Output 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

- A test input vector, or probe, is defined as

$$\mathbf{X} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- When this probe is presented to the network, we obtain:

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

Competitive learning

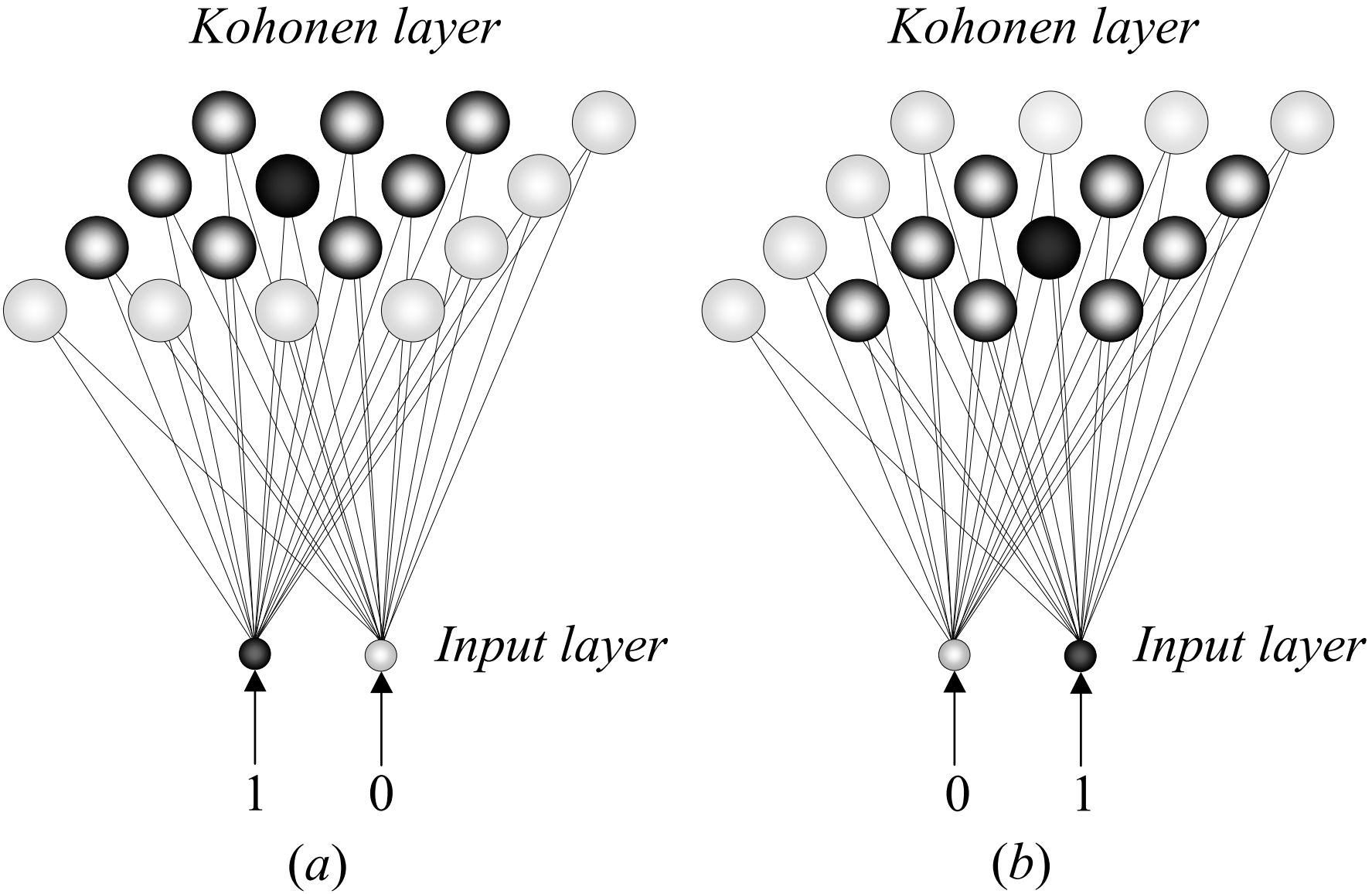
- In competitive learning, neurons compete among themselves to be activated.
- While in Hebbian learning, several output neurons can be activated simultaneously, in competitive learning, only a single output neuron is active at any time.
- The output neuron that wins the "competition" is called the ***winner-takes-all*** neuron.

- The basic idea of competitive learning was introduced in the early 1970s.
- In the late 1980s, Teuvo Kohonen introduced a special class of artificial neural networks called **self-organising feature maps**. These maps are based on competitive learning.

What is a self-organising 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, somatosensory, etc.), and associated with different sensory inputs. We can say that each sensory input is mapped into a corresponding area of the cerebral cortex. **The cortex is a self-organising computational map in the human brain.**

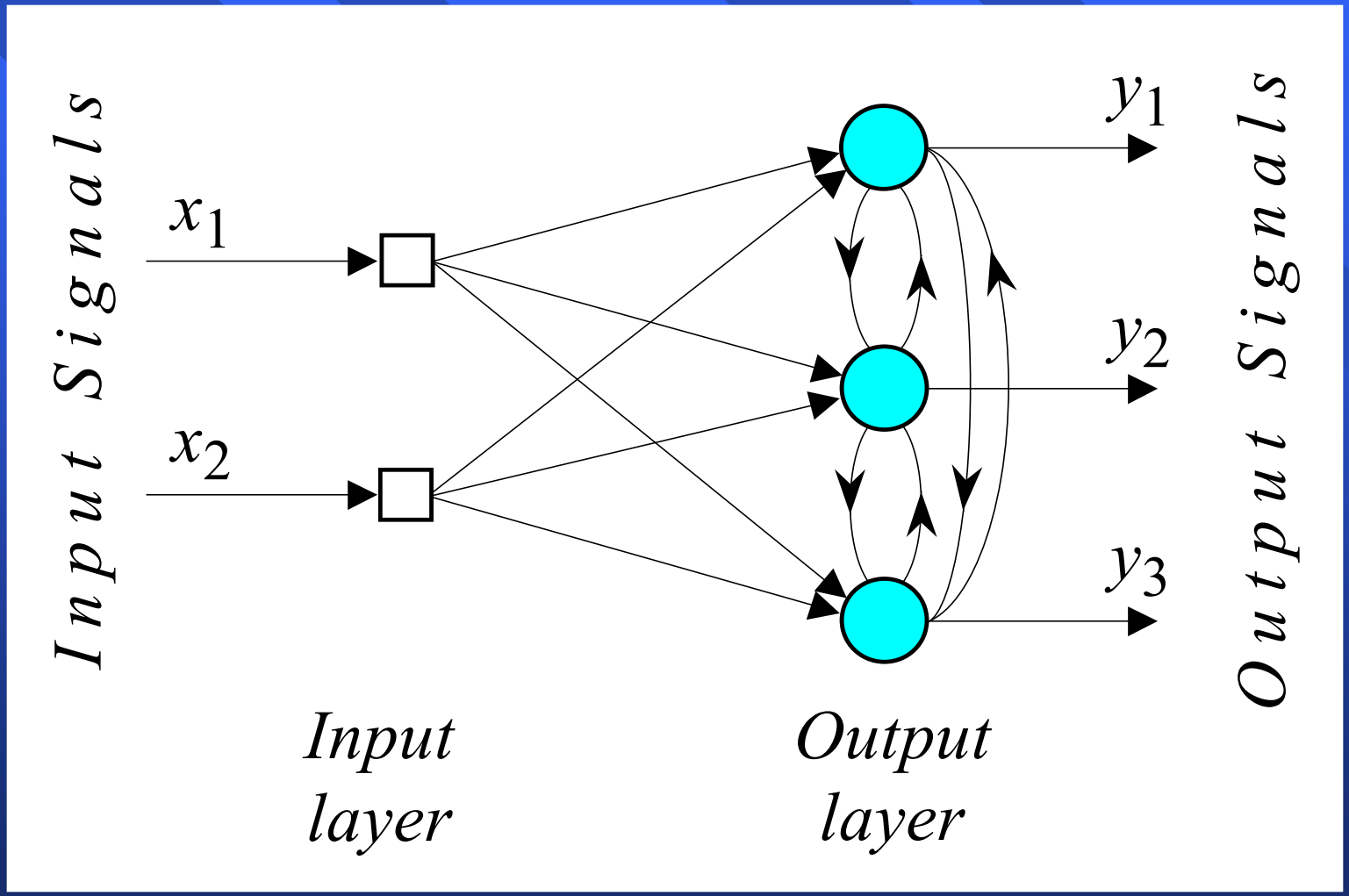
Feature-mapping Kohonen model



The Kohonen network

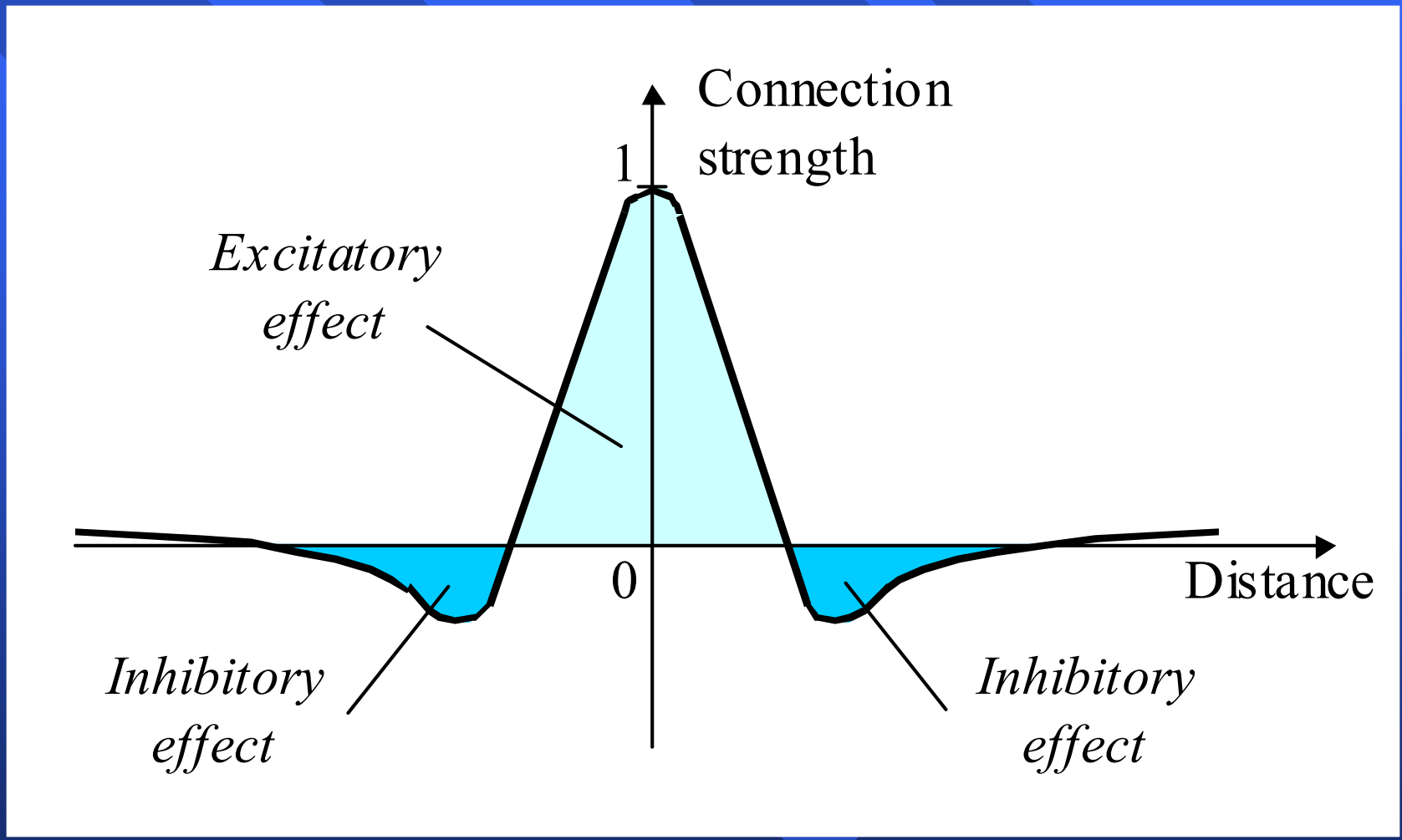
- The Kohonen model provides a topological mapping. It places a fixed number of input patterns from the input layer into a higher-dimensional output or Kohonen layer.
- Training in the Kohonen network begins with the winner's neighbourhood of a fairly large size. Then, as training proceeds, the neighbourhood size gradually decreases.

Architecture of the Kohonen Network



- The lateral connections are used to create a competition between neurons. The neuron with the largest activation level among all neurons in the output layer becomes the winner. This neuron is the only neuron that produces an output signal. The activity of all other neurons is suppressed in the competition.
- The lateral feedback connections produce excitatory or inhibitory effects, depending on the distance from the winning neuron. This is achieved by the use of a **Mexican hat function** which describes synaptic weights between neurons in the Kohonen layer.

The Mexican hat function of lateral connection



- In the Kohonen network, a neuron learns by shifting its weights from inactive connections to active ones. Only the winning neuron and its neighbourhood are allowed to learn. If a neuron does not respond to a given input pattern, then learning cannot occur in that particular neuron.
- The **competitive learning rule** defines the change Δw_{ij} applied to synaptic weight w_{ij} as

$$\Delta w_{ij} = \begin{cases} \alpha\,(x_i - w_{ij}), & \text{if neuron } j \text{ wins the competitio n} \\ 0, & \text{if neuron } j \text{ loses the competitio n} \end{cases}$$

where x_i is the input signal and α is the *learning rate* parameter.

- The overall effect of the competitive learning rule resides in moving the synaptic weight vector $\mathbf{W}_j$ of the winning neuron j towards the input pattern $\mathbf{X}$. The matching criterion is equivalent to the minimum **Euclidean distance** between vectors.
- The Euclidean distance between a pair of n-by-1 vectors $\mathbf{X}$ and $\mathbf{W}_j$ is defined by

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

where x_i and w_{ij} are the ith elements of the vectors $\mathbf{X}$ and $\mathbf{W}_j$, respectively.

- To identify the winning neuron, j_X, that best matches the input vector $\mathbf{X}$, we may apply the following condition:

$$j_X = \min_j \left\| \mathbf{X} - \mathbf{W}_j \right\|, \qquad j = 1, 2, \ldots, m$$

where m is the number of neurons in the Kohonen layer.

- Suppose, for instance, that the 2-dimensional input vector **X** is presented to the three-neuron Kohonen network,

$$\mathbf{X} = \begin{bmatrix} 0.52 \\ 0.12 \end{bmatrix}$$

- The initial weight vectors, $\mathbf{W}_j$, are given by

$$\mathbf{W}_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix} \qquad \mathbf{W}_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix} \qquad \mathbf{W}_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

- We find the winning (best-matching) neuron j_X using the minimum-distance Euclidean criterion:

$$d_1 = \sqrt{(x_1 - w_{11})^2 + (x_2 - w_{21})^2} = \sqrt{(0.52 - 0.27)^2 + (0.12 - 0.81)^2} = 0.73$$

$$d_2 = \sqrt{(x_1 - w_{12})^2 + (x_2 - w_{22})^2} = \sqrt{(0.52 - 0.42)^2 + (0.12 - 0.70)^2} = 0.59$$

$$d_3 = \sqrt{(x_1 - w_{13})^2 + (x_2 - w_{23})^2} = \sqrt{(0.52 - 0.43)^2 + (0.12 - 0.21)^2} = 0.13$$

- Neuron 3 is the winner and its weight vector $\mathbf{W}_3$ is updated according to the competitive learning rule.

$$\Delta w_{13} = \alpha (x_1 - w_{13}) = 0.1 (0.52 - 0.43) = 0.01$$

$$\Delta w_{23} = \alpha (x_2 - w_{23}) = 0.1 (0.12 - 0.21) = -0.01$$

- The updated weight vector $\mathbf{W}_3$ at iteration $(p + 1)$ is determined as:

$$\mathbf{W}_3(p+1) = \mathbf{W}_3(p) + \Delta\mathbf{W}_3(p) = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix} + \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.20 \end{bmatrix}$$

- The weight vector $\mathbf{W}_3$ of the wining neuron 3 becomes closer to the input vector $\mathbf{X}$ with each iteration.

Competitive Learning Algorithm

Step 1: ***Initialisation.***

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_X at iteration p, using the minimum-distance Euclidean criterion

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

$$j = 1, 2, \ldots, m$$

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

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

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

The weight correction is determined by the competitive learning rule:

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

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

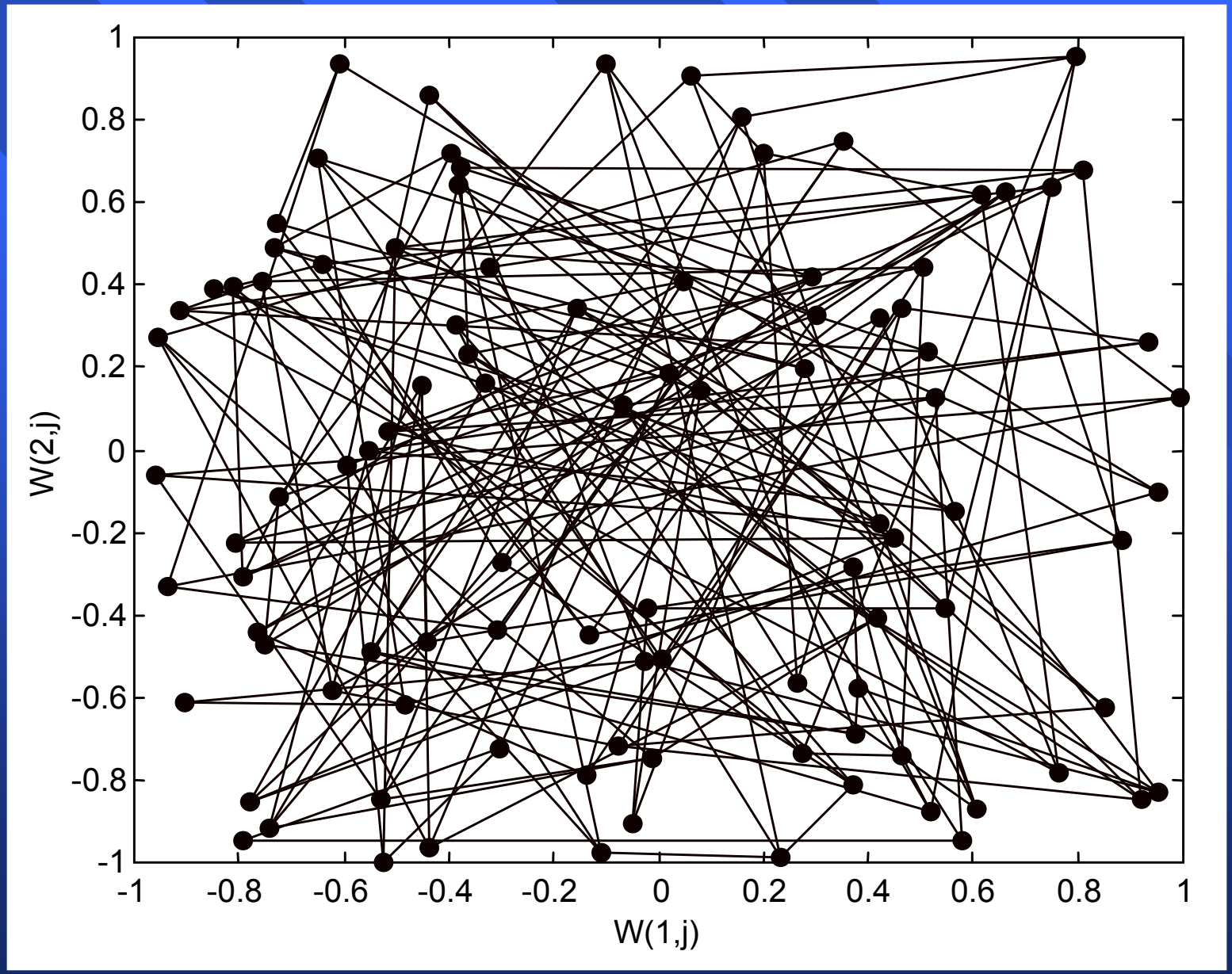
Step 4: ***Iteration.***

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.

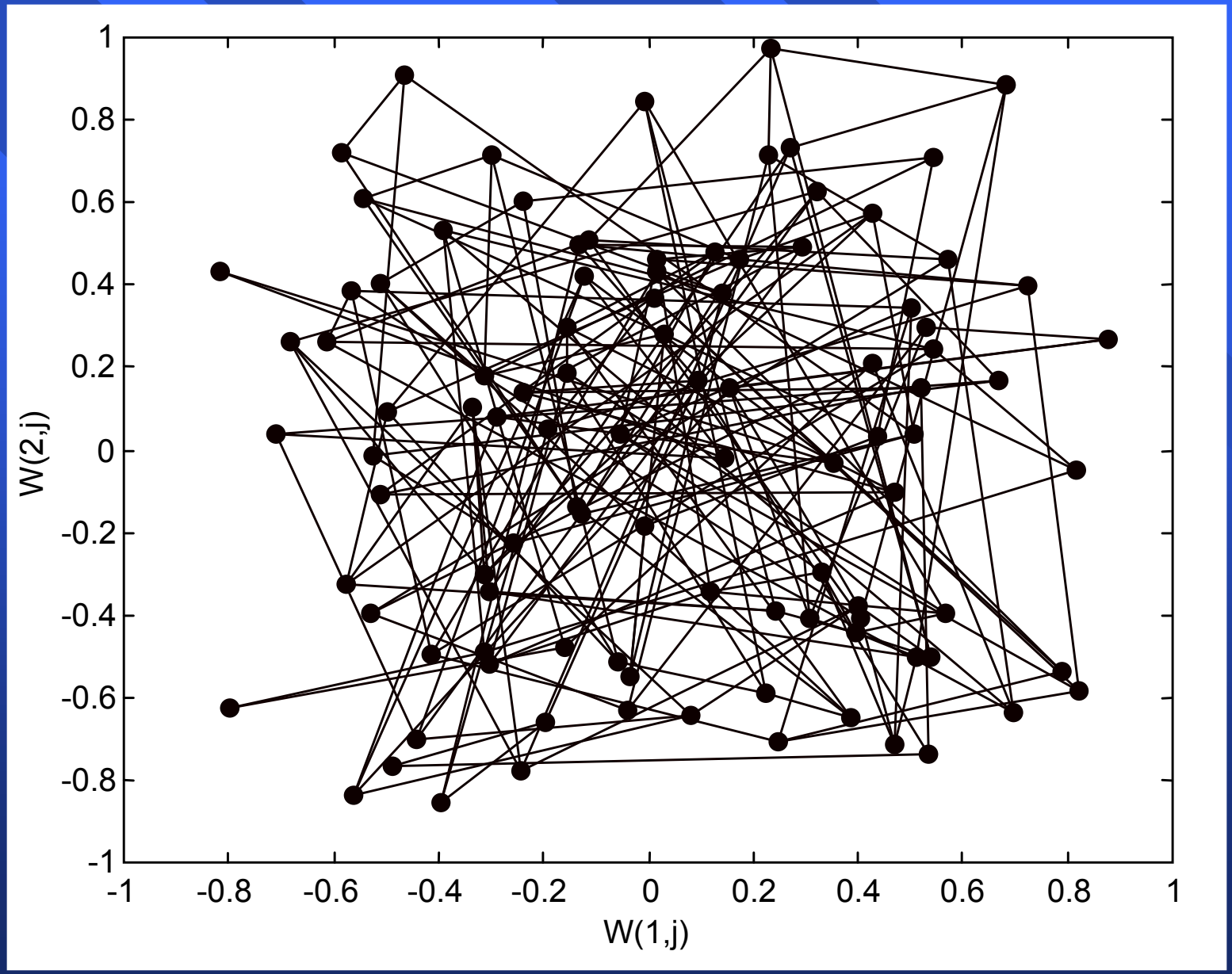
Competitive learning in the Kohonen network

- To illustrate competitive learning, consider the Kohonen network with 100 neurons arranged in the form of a two-dimensional lattice with 10 rows and 10 columns. The network is required to classify two-dimensional input vectors – each neuron in the network should respond only to the input vectors occurring in its region.
- The network is trained with 1000 two-dimensional input vectors generated randomly in a square region in the interval between –1 and +1. The learning rate parameter α is equal to 0.1.

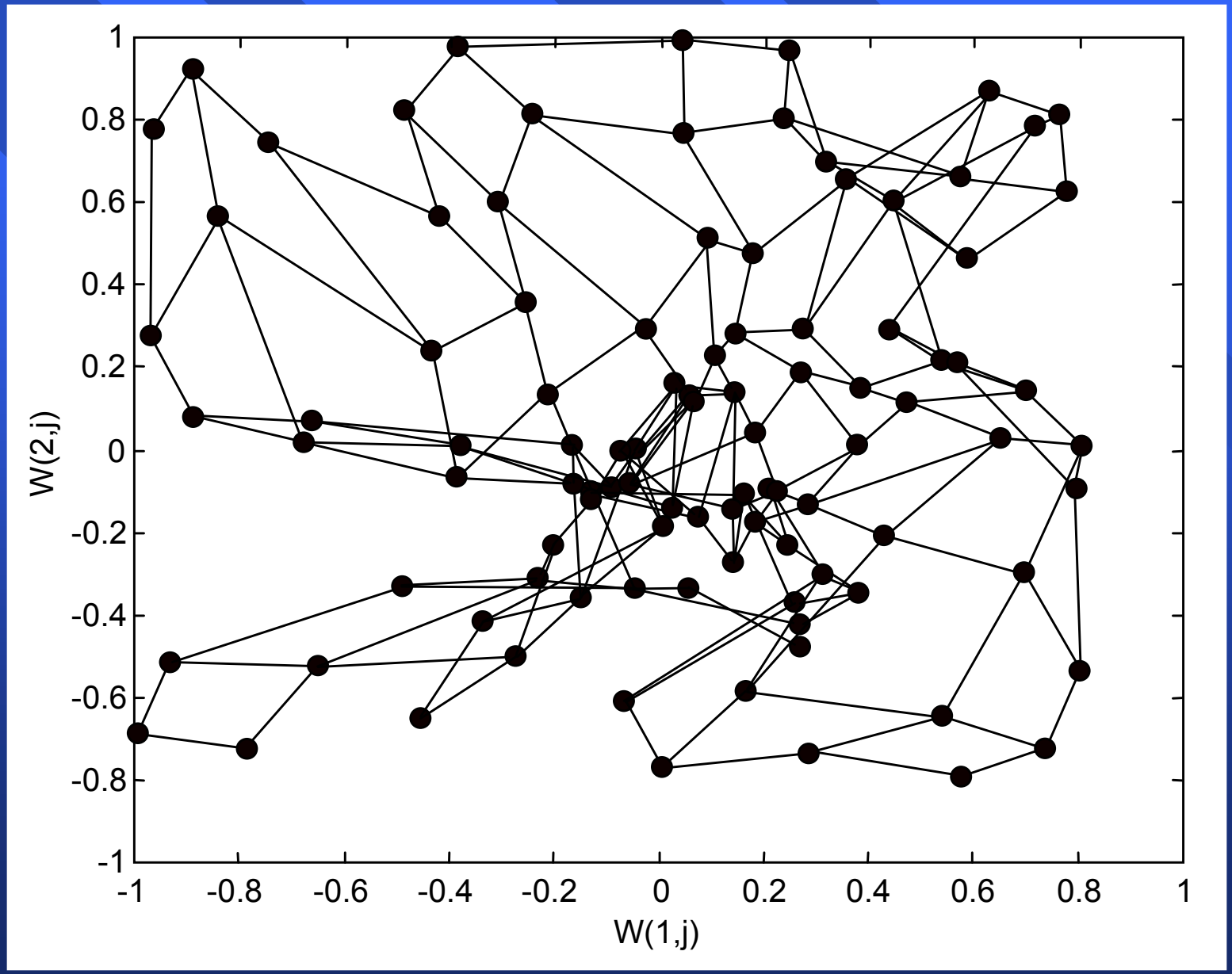
Initial random weights



Network after 100 iterations



Network after 1000 iterations



Network after 10,000 iterations