Chapter 7

SEGMENTATION IN FEATURE SPACE

Segmentation by Classification in Feature Space

- Selection of an image acquisition technique is intentional in medical imaging.
- It can be assumed that pixel or voxel values in a medical image cover more semantics with respect to object class membership than intensity in a photograph.
- Hence, image segmentation can be done as classification in feature space where image intensities are the features.
- The dimensionality of feature space is usually low, and the number of samples characterizing object classes is high.
- Typical classifiers discussed in this chapter take this into account and estimate likelihood functions from samples. Classification is then done by computing a posteriori probabilities for each object class.

Clustering in Feature Space

- particularly useful if it is not known a priori into how many and which classes scene elements should be grouped -is called clustering.
- The underlying assumption is that scene elements from the same object have more similar features than those that belong to different objects.
- Similar to classification, clustering is a generic methodology that is able to group features of any kind. It makes sense to differentiate between clustering techniques for low-dimensional densely populated features spaces and those for high-dimensional sparsely populated features spaces.
- Clustering for segmentation works in low-dimensional feature space. In 2d feature space (and with some difficulty in 3d as well), the simplest way of clustering is to do it interactively. The 2d distribution is displayed, and the user points out or delineates clusters.

Partitional Clustering and K-means Clustering

K-means clustering is a variant of *partitional clustering*. The objective of a partitional clustering method is to find cluster centers $\mathrm{CC} = \{c_1, \ldots, c_K\}$ in feature space such that the distance of all samples to their center is minimal:

$$\mathrm{CC}_{min} = \underset{\mathrm{CC}}{\arg\min}\, d_{\mathrm{CC}}(\mathbf{f}) = \underset{\mathrm{CC}}{\arg\min} \sum_{i=1}^{M} \|\mathbf{f}_i - \mathbf{c}(\mathbf{f}_\mathrm{i})\|, \qquad (7.19)$$

where $\mathbf{c}(\mathbf{f}_i)$ delivers the cluster center $\mathbf{c}_k$ that is closest to $\mathbf{f}_i$.

A heuristic strategy is employed for finding optimal cluster centers. First, cluster centers are fixed and samples are assigned to clusters that minimize d_{CC} given the cluster centers. Then, a new center $\mathbf{c}_c$ is defined for each cluster c that minimizes $\sum_{f_i \in c_c} \|\mathbf{f}_i - \mathbf{c}_k\|$. The process is repeated until no further change of cluster assignment takes place (see Fig. 7.10).

Partitional Clustering

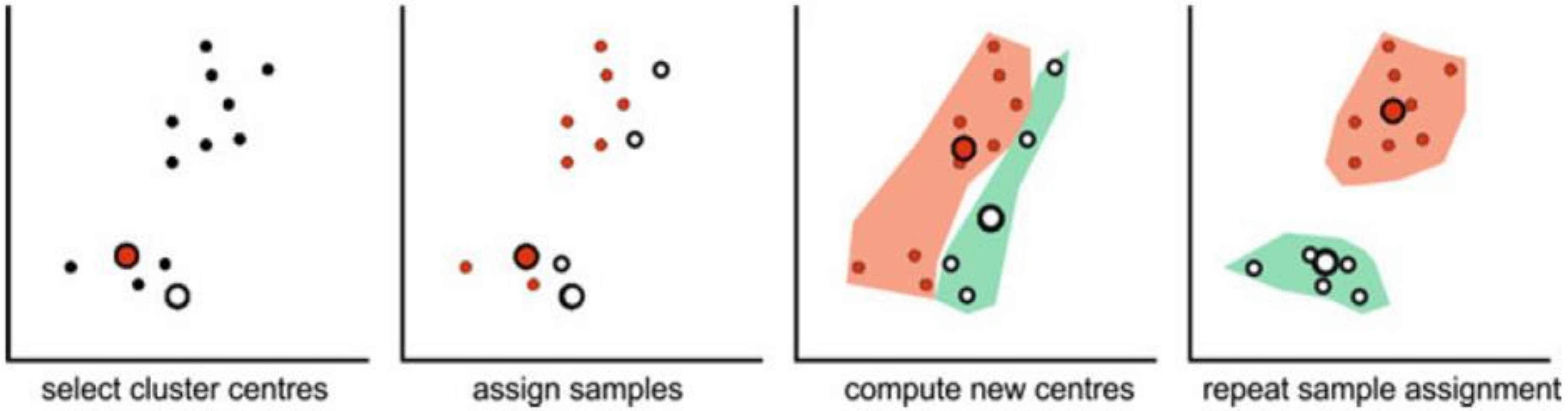


Fig. 7.10 Partitional clustering starts with initial cluster centers to which samples are assigned according to distance. New cluster centers are computed after the assignment phase. The process is repeated until there are no changes in cluster membership

K-means Clustering

Consider the following dataset with 10 data points:

Dataset:

(2, 3), (2, 4), (3, 4), (5, 6), (6, 5),

(6, 6), (7, 7), (8, 8), (8, 9), (9, 9)

We want to perform **K-means clustering with K=2** (i.e., we want to partition the data into two clusters).

Step 1: Initialization

1. Randomly initialize K cluster centroids. Let's say we randomly choose two initial centroids:
    1. Centroid 1: (3, 4)
    2. Centroid 2: (7, 7)

Step 2: Assignment

2. Assign each data point to the nearest centroid:
    - For each data point, calculate the distance to each centroid.
    - Assign the data point to the centroid with the minimum distance.

Step 3: Update

3. Update the centroids based on the mean of the data points assigned to each cluster:
   - Calculate the mean of the data points assigned to each cluster.
   - Update each centroid to the calculated mean.

Step 4: Repeat Assignment and Update

4. Repeat steps 2 and 3 until convergence:
   - Repeat the assignment step: reassign each data point to the nearest centroid.
   - Repeat the update step: update centroids based on the new assignments.

Step 5: Convergence

The algorithm converges when the centroids no longer change significantly between iterations or after a maximum number of iterations.

Iteration 1:

1. Centroids:
   1. Centroid 1: (3, 4)
   2. Centroid 2: (7, 7)
2. Assignment:
   1. Data points assigned to Centroid 1: (2, 3), (2, 4), (3, 4)
   2. Data points assigned to Centroid 2: (5, 6), (6, 5), (6, 6), (7, 7), (8, 8), (8, 9), (9, 9)
3. Update:
   1. Centroid 1: Mean of assigned points: ((2+2+3)/3, (3+4+4)/3) = (7/3, 11/3) ≈ (2.33, 3.67)
   2. Centroid 2: Mean of assigned points: ((5+6+6+7+8+8+9)/7, (6+5+6+7+8+9+9)/7) = (49/7, 50/7) ≈ (7, 7.14)

Iteration 2:

1. Centroids:
   1. Centroid 1: (2.33, 3.67)
   2. Centroid 2: (7, 7.14)
2. Assignment:
   1. Data points assigned to Centroid 1: (2, 3), (2, 4), (3, 4)
   2. Data points assigned to Centroid 2: (5, 6), (6, 5), (6, 6), (7, 7), (8, 8), (8, 9), (9, 9)
3. Update:
   1. Centroid 1: (2.33, 3.67) (unchanged)
   2. Centroid 2: (7, 7.14) (unchanged)

Mean Shift Clustering

- K-means clustering requires the number of clusters to be known. Misjudging the number of different clusters may lead to unwanted results.
- Mean shift clustering is an alternative that does not require this kind of information .Instead, mean shift clustering attempts to find all possible cluster centers in feature space.
- Samples are assumed to stem from a density function, which is a mixture of an unknown number of probability density functions. Each of the probability functions describes the probability of a sample to belong to one cluster.
- Ideal clustering would identify the probability functions of the mixture model and then assign cluster membership based on this probability.
- Determining parameters of an unknown number of probability functions of an unknown type will be difficult or even impossible. In mean shift clustering, heuristics are employed to arrive at a feasible solution.

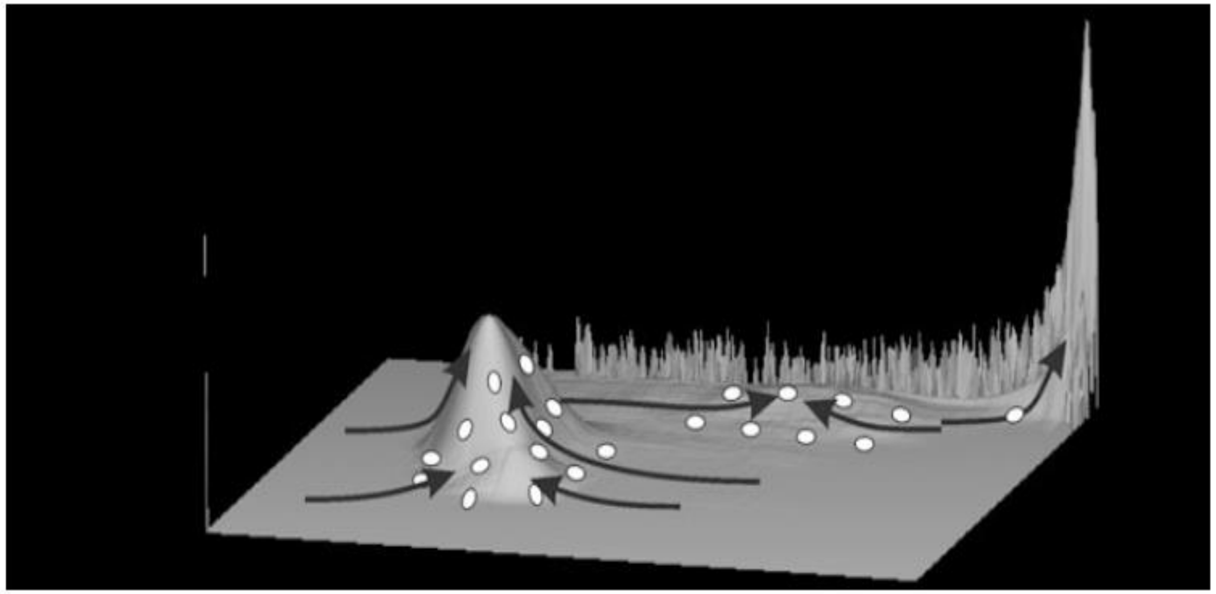


Fig. 7.15 The mean shift algorithm starts moving from every position in feature space toward the local maxima in the function

Mean Shift Clustering

The mean shift algorithm proceeds as follows:

• For each location in feature space, a marker is shifted toward the next local maximum by applying a gradient ascent algorithm.

• If a local maximum has been found and it has no cluster label, it is labeled.

• Irrespective of whether the local maximum has been labeled at this step or previously, the location from which it has been found receives the label as cluster label.

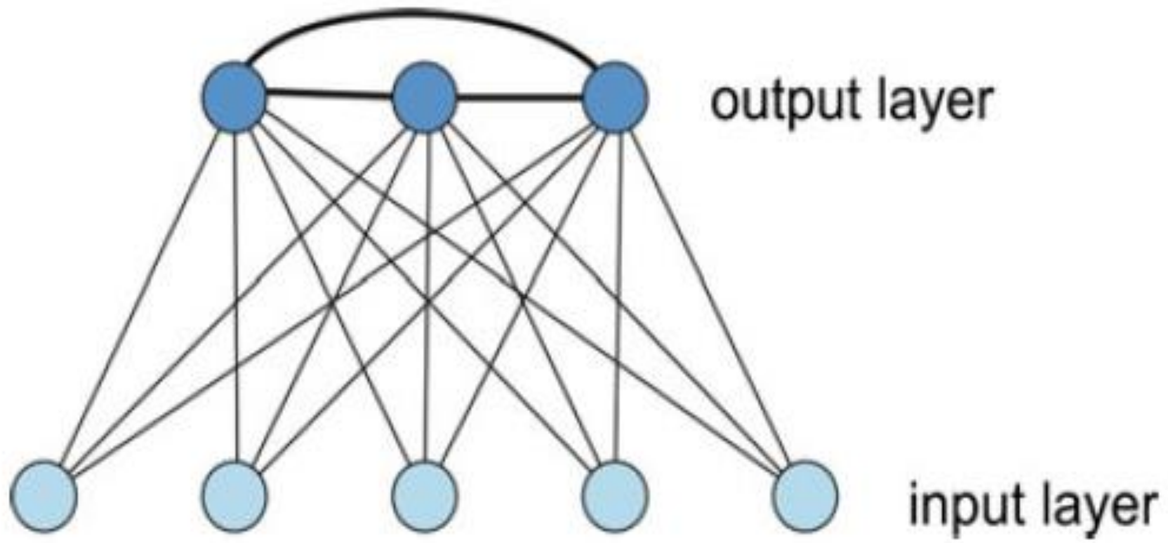
Kohonen’s Self-organizing Maps

- A Self-Organizing Feature Map (SOM) is a type of artificial neural network that is trained using **unsupervised learning to produce a two-dimensional discretized representation** of the input space of the training samples, called a map.
- These maps are useful for classification and visualizing lowdimensional views of high-dimensional data.
- Self-Organizing Maps (SOMs) is particularly similar to biological systems.
- In the human cortex, multi-dimensional sensory input spaces (e.g., visual input, tactile input) are represented by two-dimensional maps.
- The cortex is a self-organizing computational map in the human brain. Typically, SOMs have – like our brain – the task to map a high-dimensional input (N dimensions) onto areas in a low-dimensional grid of cells (G dimensions) to draw a map of the high-dimensional space.
- SOM is a visualization method to represent higher dimensional data in an usually 1- D, 2-D or 3-D manner

Kohonen’s Self-organizing Maps

Kohonen's SOM is called a topology-preserving map because there is a topological structure imposed on the nodes in the network. A topological map is simply a mapping that preserves neighborhood relations. The Kohonen map performs a mapping from a continuous input space to a discrete output space, preserving the topological properties of the input.

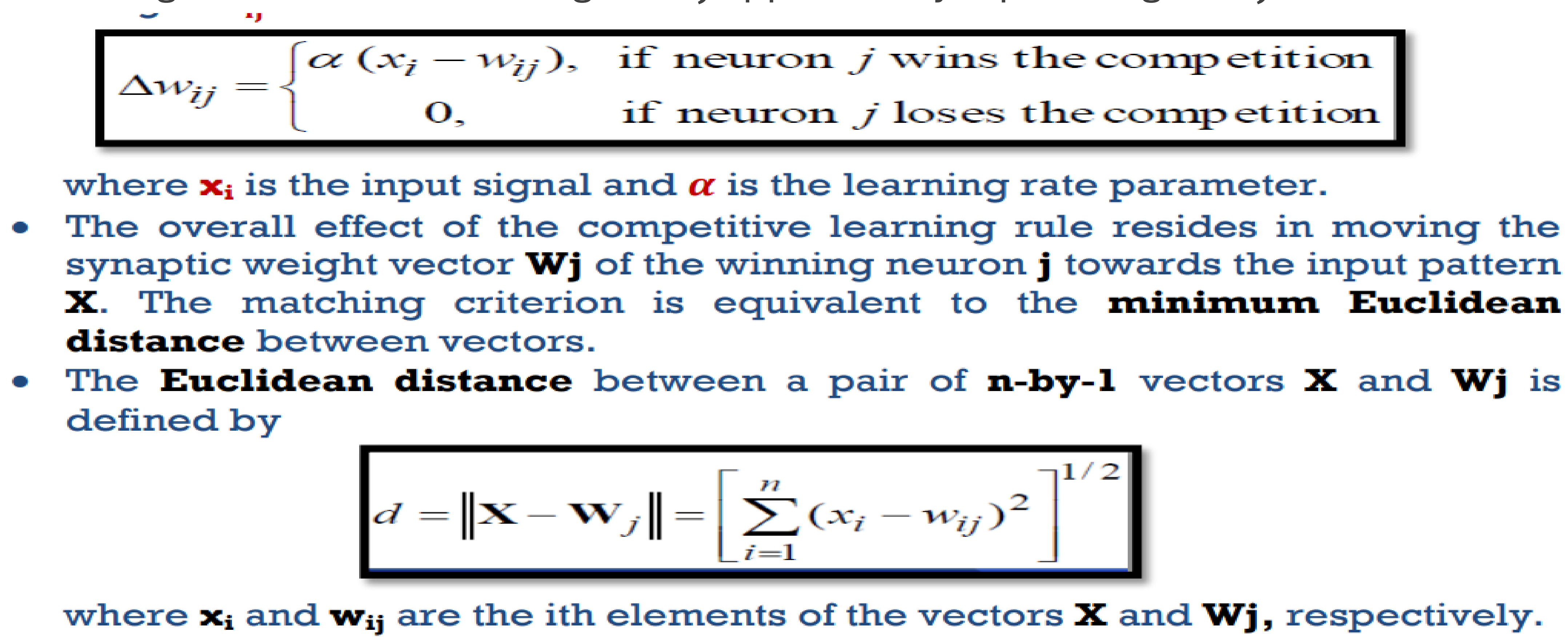
Fig. 7.16 Topology of a Kohonen network



In the Kohonen network, a neuron learns by shifting its weights from inactive connections to active ones. Only the winning neuron and its neighborhood 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

e learning rule defines the change **Δwij** applied to synaptic weight **wij** as:



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

where $\mathbf{x_i}$ 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 **Wj** of the winning neuron **j** towards the input pattern **X**. The matching criterion is equivalent to the **minimum Euclidean distance** between vectors.
- The **Euclidean distance** between a pair of **n-by-1** vectors **X** and **Wj** is defined by

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

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

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

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

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

The SOM Algorithm

The Self-Organizing Map algorithm can be broken up into 9 steps.

Step 0:

- Initialize synaptic weights $\mathbf{w_{ij}}$ to small random values, say in an interval [0, 1].
- Set topological neighborhood parameters.
- Set learning rate parameters (small positive value).

Step 1: While stopping condition is false, do **Steps 2-8**.

Step 2: For each input vector **x** chosen at random from the set of training data and **presented** to the network, do Steps 3-5.

Step 3: For each **j,** compute:

$$\boxed{D(j) = \sum_i (w_{ij} - x_i)^2.}$$

Euclidean distance is a measurement of **similarity** between two sets of data. (Every node in the network is examined to calculate which ones' weights are most like the input vector).

Step 4: Find index **J** such that **D(J)** is a minimum (The winning node is commonly known as the **Best Matching Unit (BMU)**).

input vector

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

Where: **n** is the number of neurons in the input layer,

m is the number of neurons in the Kohonen layer.

Step 5: Learning (Update the synaptic weights):

For all units **j** within a specified neighborhood of **J**, and for all **i**:

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

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

The weight correction is determined by the competitive learning rule:

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

Where α is the learning rate parameter, and $\Lambda_j(p)$ is the neighborhood function centered around the winner-takes-all neuron $\mathbf{j_X}$ at iteration **p.**

Any nodes found within the radius of the **BMU** are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its' weights are altered.

Step 6: Update learning rate (α is a slowly decreasing function of time).

Step 7: Reduce radius of topological neighborhood at specified times. The **radius** of the neighborhood of the BMU is calculated. This value starts large (typically it is set to be the radius of the network). The radius of the neighborhood around a cluster unit also decreases as the clustering process progresses.

Step 8: Test stopping condition.

Example: 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, **Wj**, are given by

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

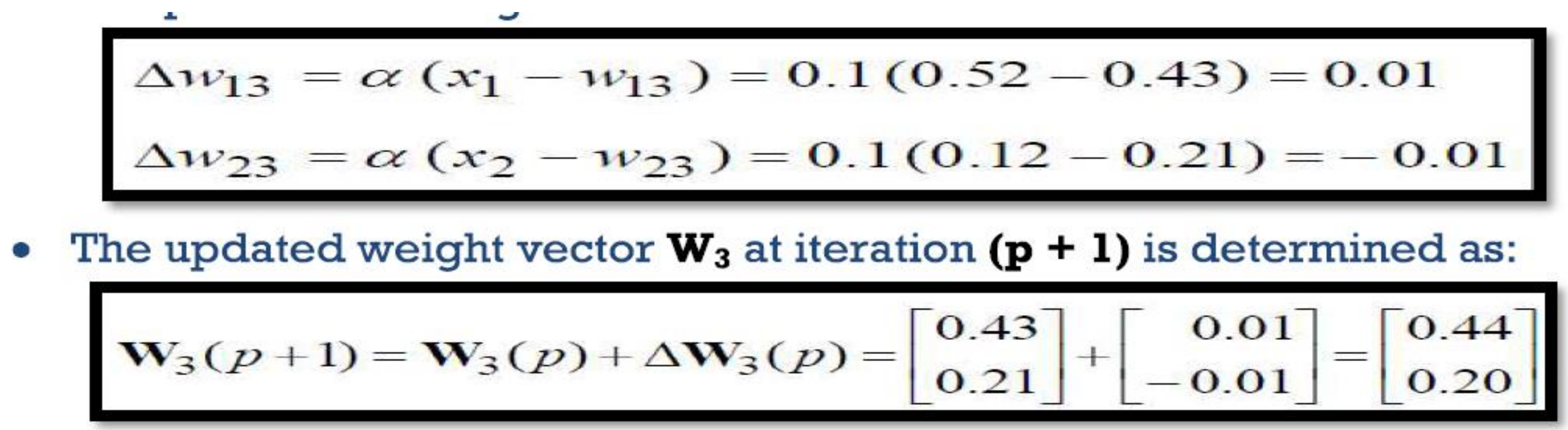
- We find the winning (**best-matching**) neuron $\mathbf{j_X}$ using the minimum-distance Euclidean criterion:

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

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

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

- 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 W3 of the wining neuron 3 becomes closer to the input vector X with each iteration.