# CISC/CMPE452/COGS400/CISC874
# Unsupervised Learning I

Ch. 5 - Textbook

Farhana Zulkernine

# Unsupervised Learning

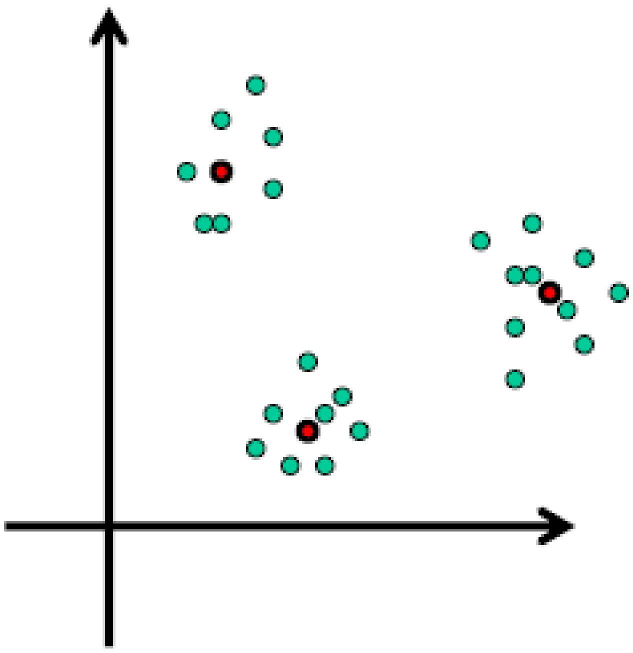
- Small kids are able to recognize patterns => a significant amount of learning is accomplished by biological processes as “unsupervised” without a teacher.
- “Unsupervised" learning proceeds to discover special features and pattern from available data without using external help.

# Unsupervised Learning Applications

- Tasks for which this is used are:
  - Clustering
  - Vector quantization
  - Approximation of data distribution
  - Feature extraction and
  - Dimensionality reduction
- Most unsupervised techniques bear resemblance to existing statistical methods.

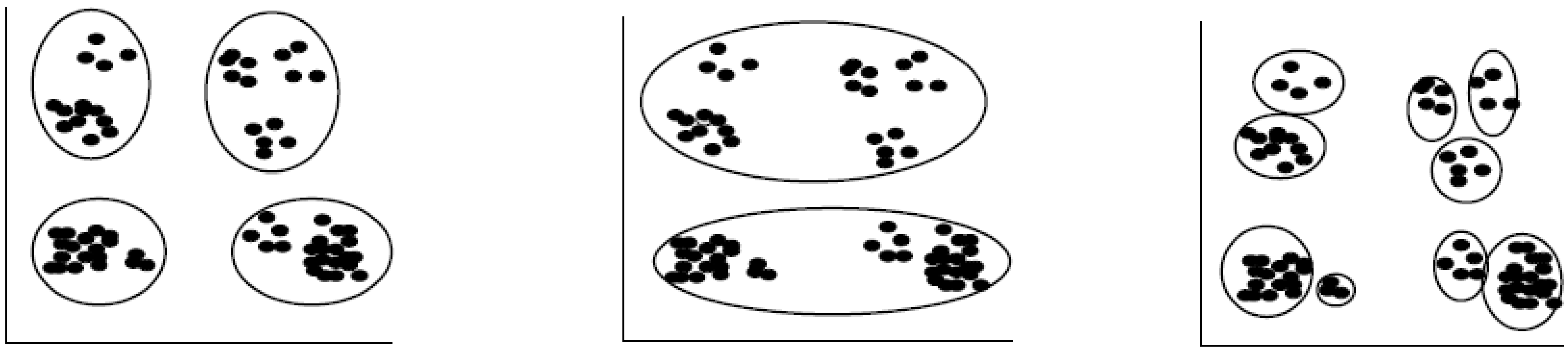
# Clustering

- Given a number of data points, determine a set of representative centroids, (or also called prototypes, cluster centers, or reference vectors)
- Find distances of a given pattern from the centroids.
- Typically it belongs to the cluster whose centroid is the closest.



*centroids indicated in red*

# Example: Clustering



Reasonable number of clusters | Small Number of Clusters | Too Many Clusters

Fig. Three different ways of clustering the same set of sample points.

# Clustering (cont…)

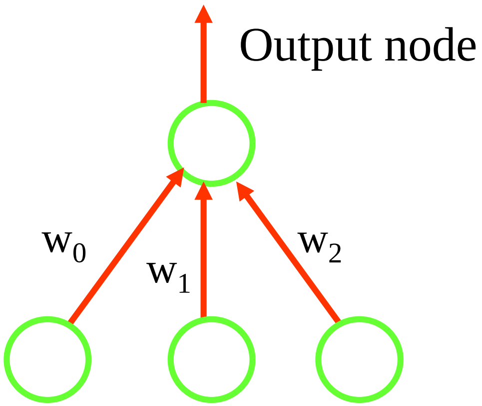
- Clusters are *evaluated* by measuring the average squared distance between each input pattern and the centroid of the cluster in which it is placed.

$$\mathbf{E_{cluster}} = \frac{\mathbf{1}}{\textbf{number of patterns}} \sum_{patterns} \| (\textbf{pattern} - \textbf{centroid}) \|^2$$

$$\mathbf{E_{total}} = \sum_{clusters} \boldsymbol{E_{cluster}}$$

# Clustering (cont…)

- In NN used for clustering, weights from the input layer to each output node constitute a ***weight vector w*** of that node, which represents the **centroid** of one cluster of input patterns.



# Vector Quantization

- This is a task that applies unsupervised learning to divide an input **space** into several connected regions called ***Voronoi Regions***, representing a quantization of the space.
- Each region is represented using a single vector called a ***Codebook Vector (CV)*** which are also called ***Voronoi centres***.

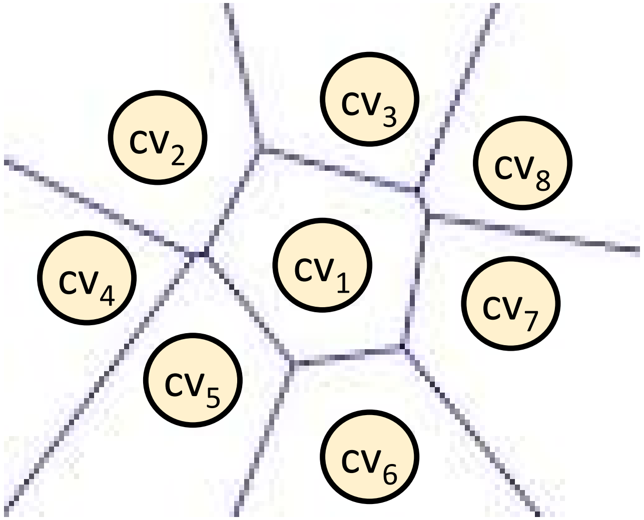


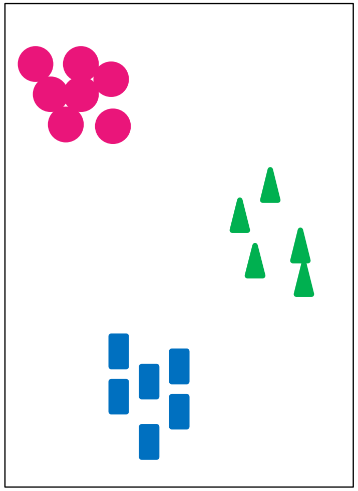

Figure: **Voronoi Diagram** with 8 voronoi regions and codebook vectors of $\{cv_1,\ldots,cv_8\}$

# Vector Quantization

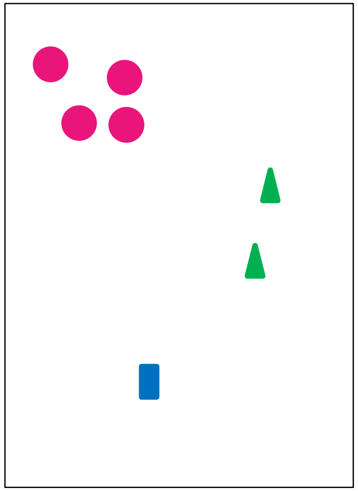
- Every point in the input space belongs to a region and is mapped to the corresponding CV.
  - Many input data vectors may be mapped to the same CV.
  - Two training approaches
    - Region may be given for each data point in a supervised learning and the CV has to be determined.
    - In unsupervised learning, a normal clustering approach can be applied.
- Therefore, the set of CVs is a **compressed form of information represented by all input data.**

# Approximation of Data Distribution

- The data distribution in *b* is a concise description of the larger amount of data in *a* drawn from a *probability distribution*.
  - Data distribution in *b* is considered to be an approximation of that in *a*.
  - Points in *b* **may not be** a subset of points in *a* ← can be done using **unsupervised learning**.
  - Clustering, if used, will extract only one point for each cluster.



a



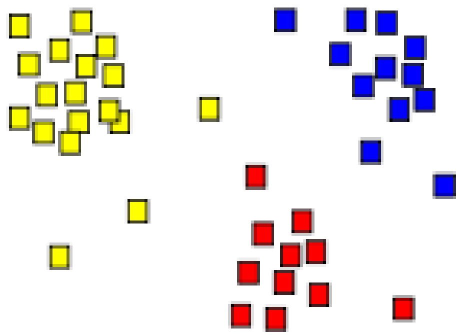
b

# Feature Extraction and Dimensionality Reduction

- Patterns in different clusters should ideally be distinguished by some *feature*. Ex. All red balls in one cluster and green in the other – 'color' is the identifying feature.
- The goal of **feature extraction** is to find the most important features, i.e., those with the highest variation in a given population.
- Important **side-effect** is **reduction of input dimensionality** and thereby, improve in processing time and cost.

# Winner-Take-All Networks

- Most unsupervised neural networks rely on **competitive learning** algorithms to compute and compare distances, determine the "winner" node with the highest level of activation, and use that to adapt weights.
- Patterns in the same cluster are as alike as possible.



# Hamming Networks

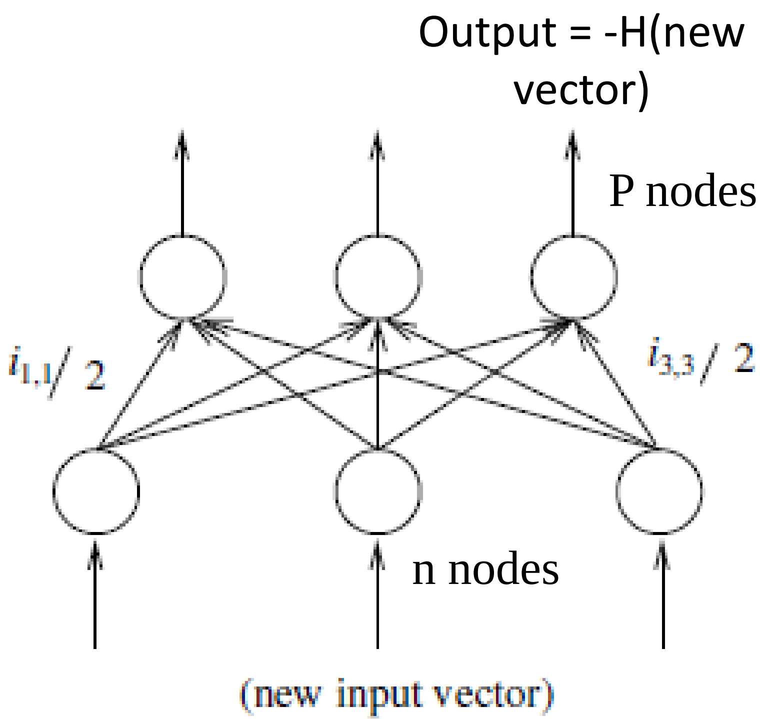



Figure: A network to calculate Hamming distance (H) between stored vectors and input vectors

- **Weights** on links from an input layer to an output layer **represent components of stored input patterns**.
- Hamming networks **compute the "hamming distance"**, the number of differing bits, of input and stored vectors.
- So, with P output nodes a NN can store P vectors each associated with a weight vector.

# Hamming Networks (cont…)

- Let $i_{p,j} \epsilon \{+1, -1\}$ be the $j$ th element of the $p$ th stored vector.
- Components of the weight matrix $W$ and threshold $\theta$ vector are given by $w_{p,j} = i_{p,j} /2$, j=1,…,n and p=1,…,P and $\theta = -(n/2)$, respectively.
  - Uses constant bias.
- Input layer has $n$ nodes (dimension of input vector) and output layer has $P$ nodes (total number of stored vectors)
- $p$th output node generates the ***negative of*** **Hamming distance** between $p$th stored pattern and the input pattern.

# Hamming Networks (cont…)

- When a new vector $i$ is presented to this network, its upper level nodes generate the output as given below where $\boldsymbol{i_p.i}$ represents the dot product $\sum_k i_{p,k} i_k$
- *One shot training* → Assign weights

$$W = \frac{1}{2}\begin{pmatrix} i_1^T \\ \vdots \\ i_P^T \end{pmatrix} \qquad \Theta = \begin{pmatrix} -\frac{n}{2} \\ \vdots \\ -\frac{n}{2} \end{pmatrix} \qquad o = Wi + \Theta = \frac{1}{2}\begin{pmatrix} i_1.i - n \\ \vdots \\ i_P.i - n \end{pmatrix}$$

# Example 5.1

- Given i1=(1, -1, -1, 1, 1), i2 = (-1, 1, -1, 1, -1), i3 = (1, -1, 1, -1, 1) and test data x=(1, 1, 1, -1, -1), design a Hamming network.
- Weight matrix represents the 3 stored vectors i1, i2 and i3.

$$W = \frac{1}{2}\begin{pmatrix} i_1^T \\ i_2^T \\ i_3^T \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{pmatrix} \qquad \Theta = \frac{1}{2}\begin{pmatrix} -5 \\ -5 \\ -5 \end{pmatrix}$$

# Example 5.1

**1 – 1 – 1 – 1 – 1 = -3**

- Given test data x=(1, 1, 1, -1, -1), output can be calculated as: (See book for details)

$$Wx = \frac{1}{2}\begin{pmatrix} 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{pmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -3 \\ -1 \\ 1 \end{bmatrix}$$

$$o = Wx + \Theta = \frac{1}{2}\begin{bmatrix} -3-5 \\ -1-5 \\ 1-5 \end{bmatrix} = \begin{pmatrix} -4 \\ -3 \\ -2 \end{pmatrix}$$

x and $i_1$ differs by 4 bits, $i_2$ by 3 bits, $i_3$ by 2 bits

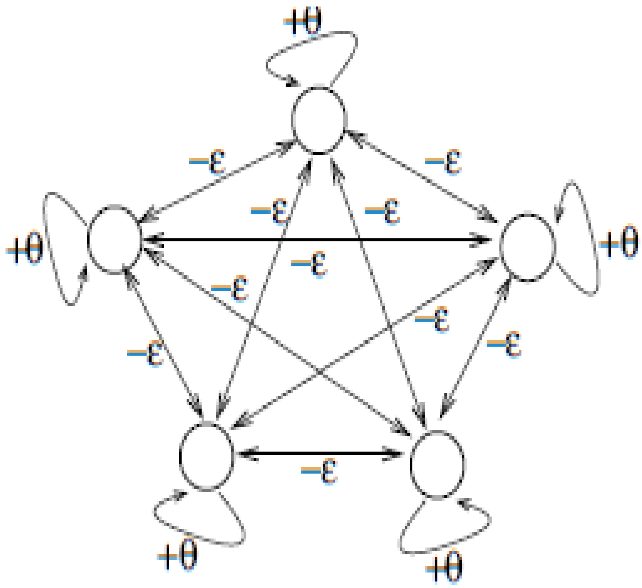
# Hamming Networks (cont…)

- Interestingly if all bits match ($wi = n/2$) then output Hamming distance = 0, otherwise < 0.
- So we can determine *which stored pattern is nearest to a new input pattern by taking the maximum of the outputs*.
- This can be accomplished by attaching a ***Maxnet*** on top of the second layer of the Hamming network.

# Maxnet

- A Maxnet is a **recurrent competitive one-layer** network used to determine which node has the highest initial activation.
- $\theta = 1$ and $\varepsilon \leq -1/(\#\text{ of nodes})$ ensures that the **node that has the initial highest value prevail** as "winner", while the others subside to zero.
- The node function is $\boldsymbol{f(net) = \max(0, net)}$ where $net = \sum_{i=1}^{n} w_i x_i$

Similar to inhibitory lateral connections in the IAM model.



# Maxnet (cont…)

- All nodes update their outputs simultaneously. Each node receives *inhibitory* inputs from all other nodes, via "lateral" (intra-layer) connections.
- The maxnet allows for greater parallelism in execution, since every computation is local to each node rather than centralized.

# Example 5.2

- Let initial activation values = (0.5,0.9, 1,0.9,0.9) and $\epsilon = -1/5$ and $\theta = +1$. Computing outputs $o_j$ after1st iteration

  $o_1 = \max(0, 0.5 - 1/5(0.9+1+0.9+0.9)) = \max(0, -0.24) = 0$

  $o_2 = \max(0, 0.9 - 1/5(0.5+1+0.9+0.9)) = \max(0, 0.24) = 0.24$

  $o_3 = \max(0, 1 - 1/5(0.5+0.9+0.9+0.9)) = \max(0, 0.36) = 0.36$

- In subsequent iterations,

  (0, 0.24, 0.36, 0.24, 0.24) → (0, 0.072, 0.216, 0.072, 0.072) → (0, 0, 0.1728, 0, 0)

- 3rd node becomes the winner although others had values very close to this node.

# Simple Competitive Learning

- The Hamming net and Maxnet assist more complex unsupervised learning networks, helping to determine the ***node whose weight vector is nearest to an input pattern***.
- The figure shows a generalized version where inputs are n-dimensional real value vectors, $\mathbb{R}^n \longrightarrow [0,1]$.
- Also known as **Kohonen Learning.**

$$\text{Output} = \begin{cases} 1 & \text{if node is a winner} \\ 0 & \text{otherwise} \end{cases}$$

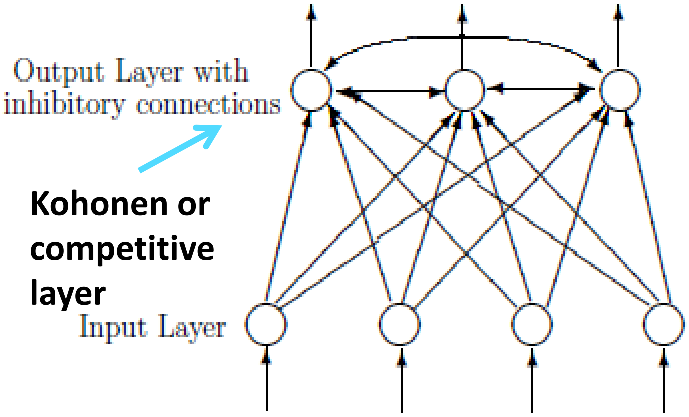
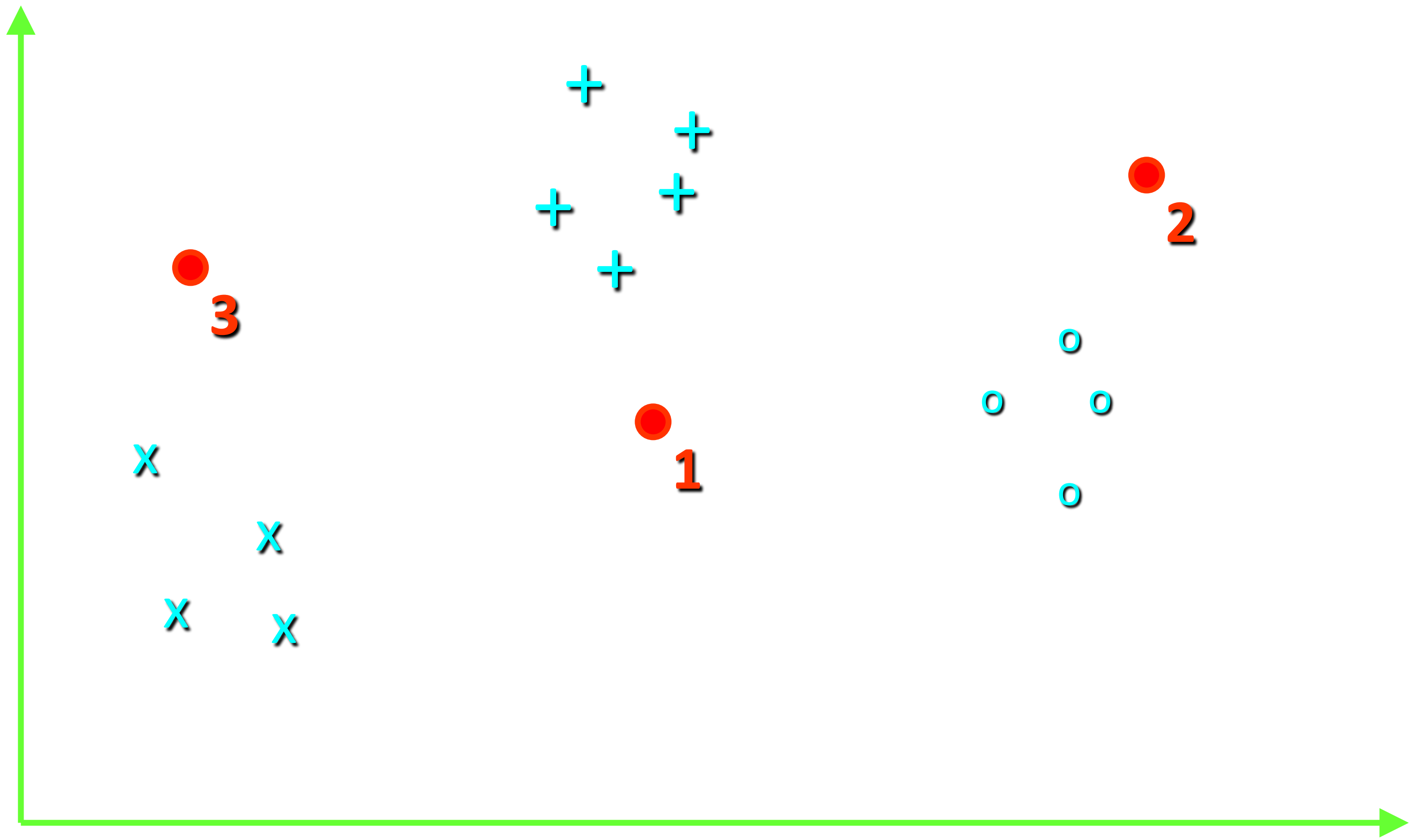



Figure: A simple competitive learning network

# Design Objectives

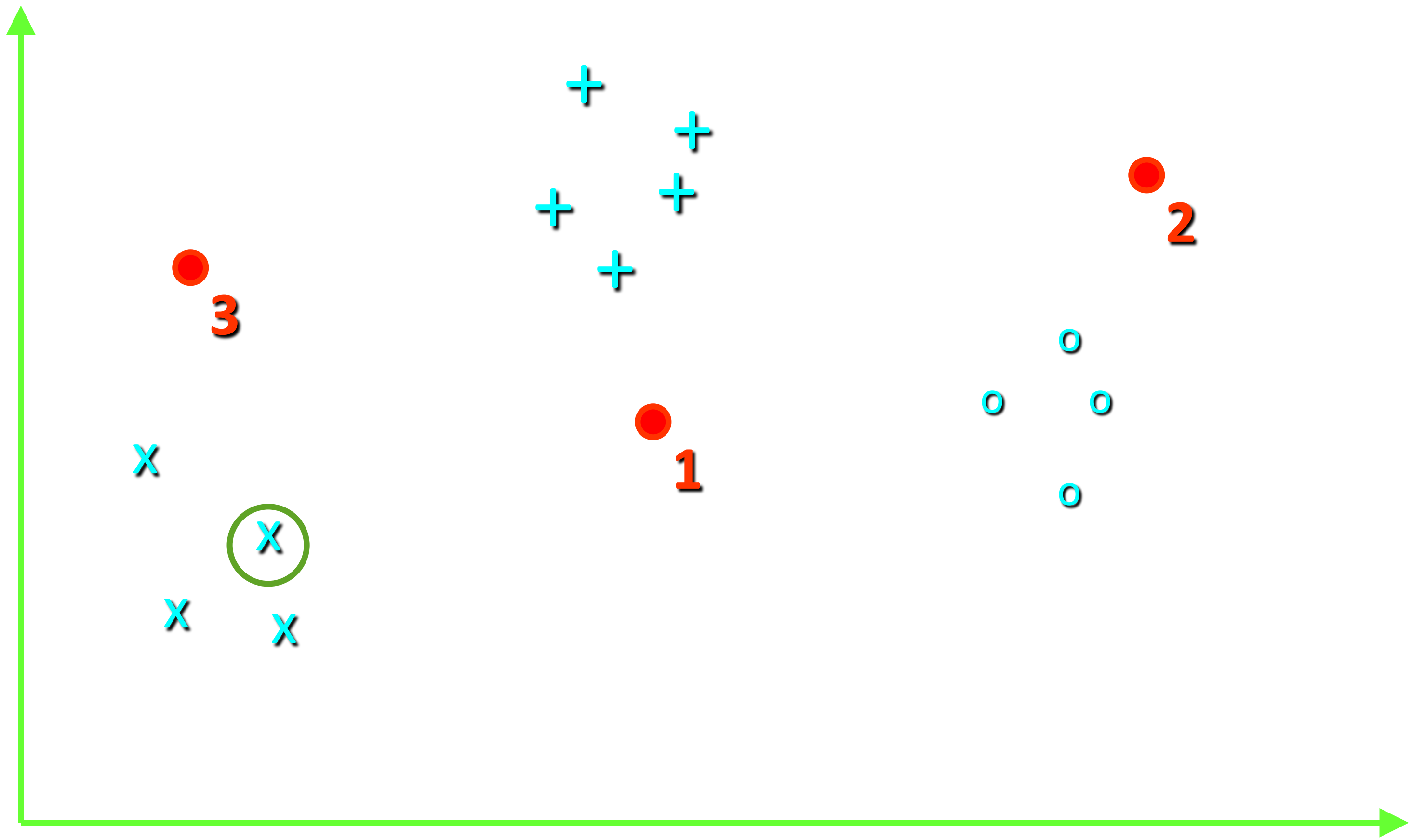
- The objective is to **attach each output node to a stored pattern as represented by its weight vector**.
- Any **distance measure** at the output layer estimate how similar an input vector is to a weight vector.
  - Winner node at Kohonen layer is closest to the input vector.
- **Iterative training → Weight update rule: Adjust the weights of the winner output node** such that $\mathbf{w}_i$ for Kohonen node $i$ is as near as possible to all input samples for which the node is winner of the competition.

# Clustering using Euclidean Distance

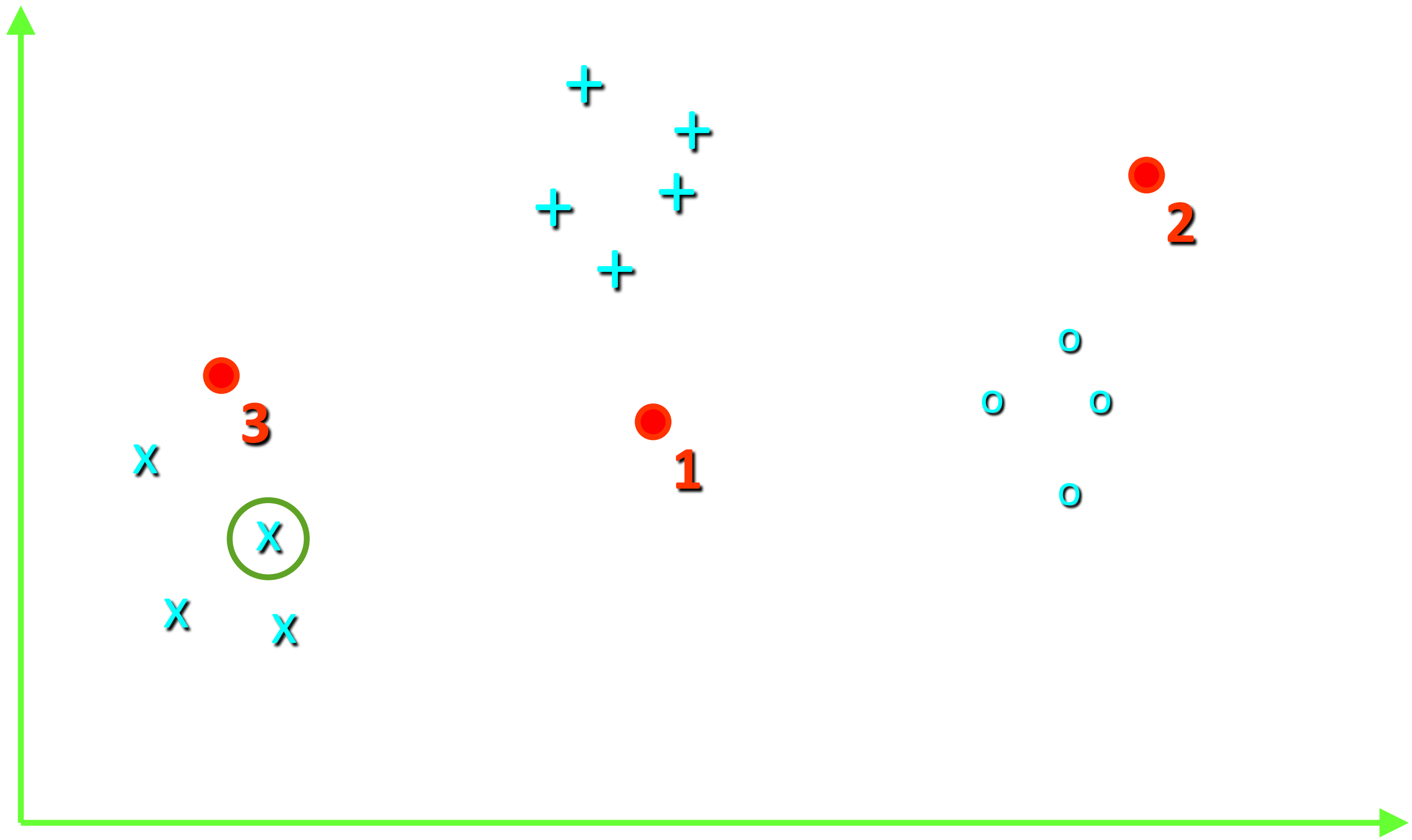



Example of competitive learning with three hidden nodes

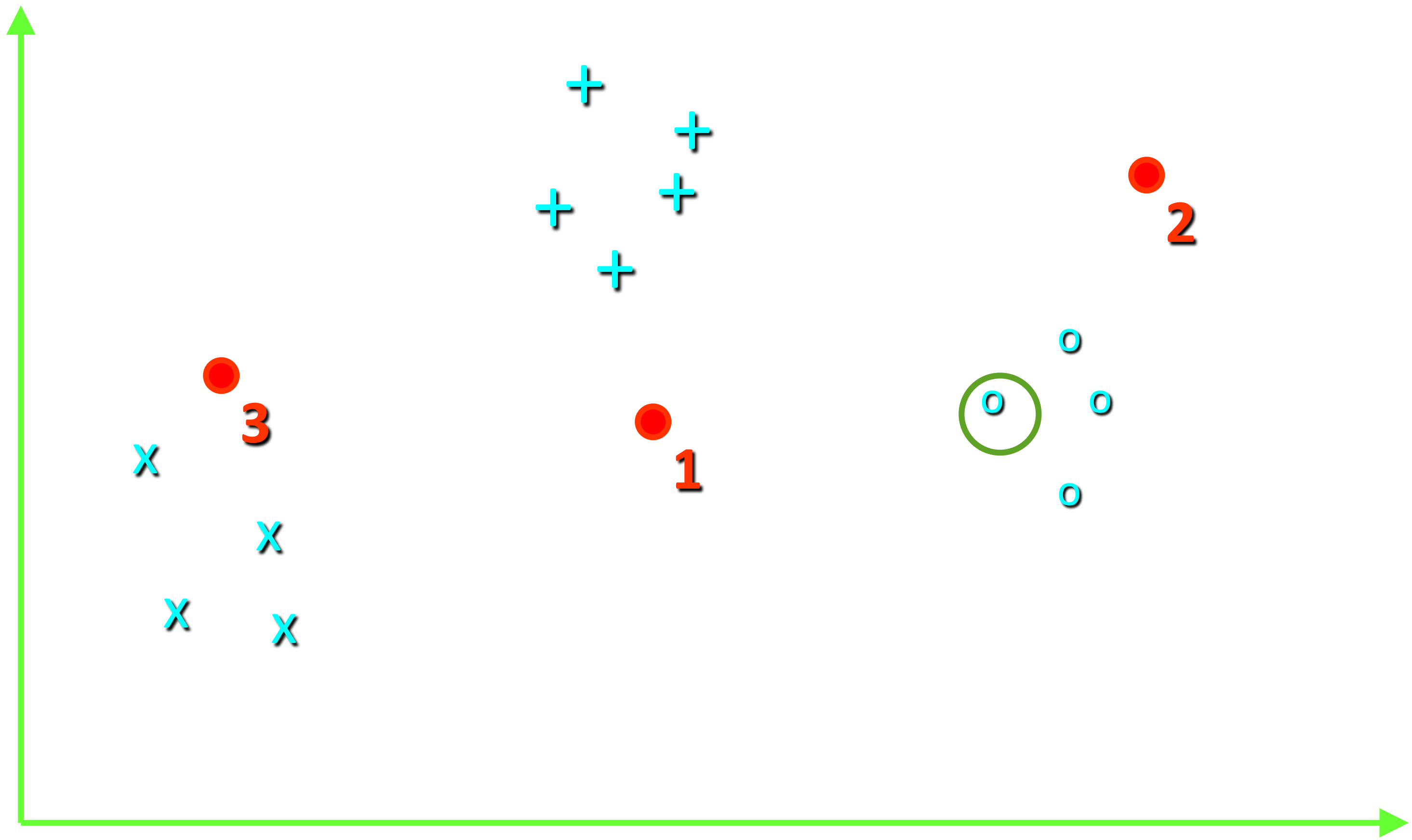
# Clustering (cont…)



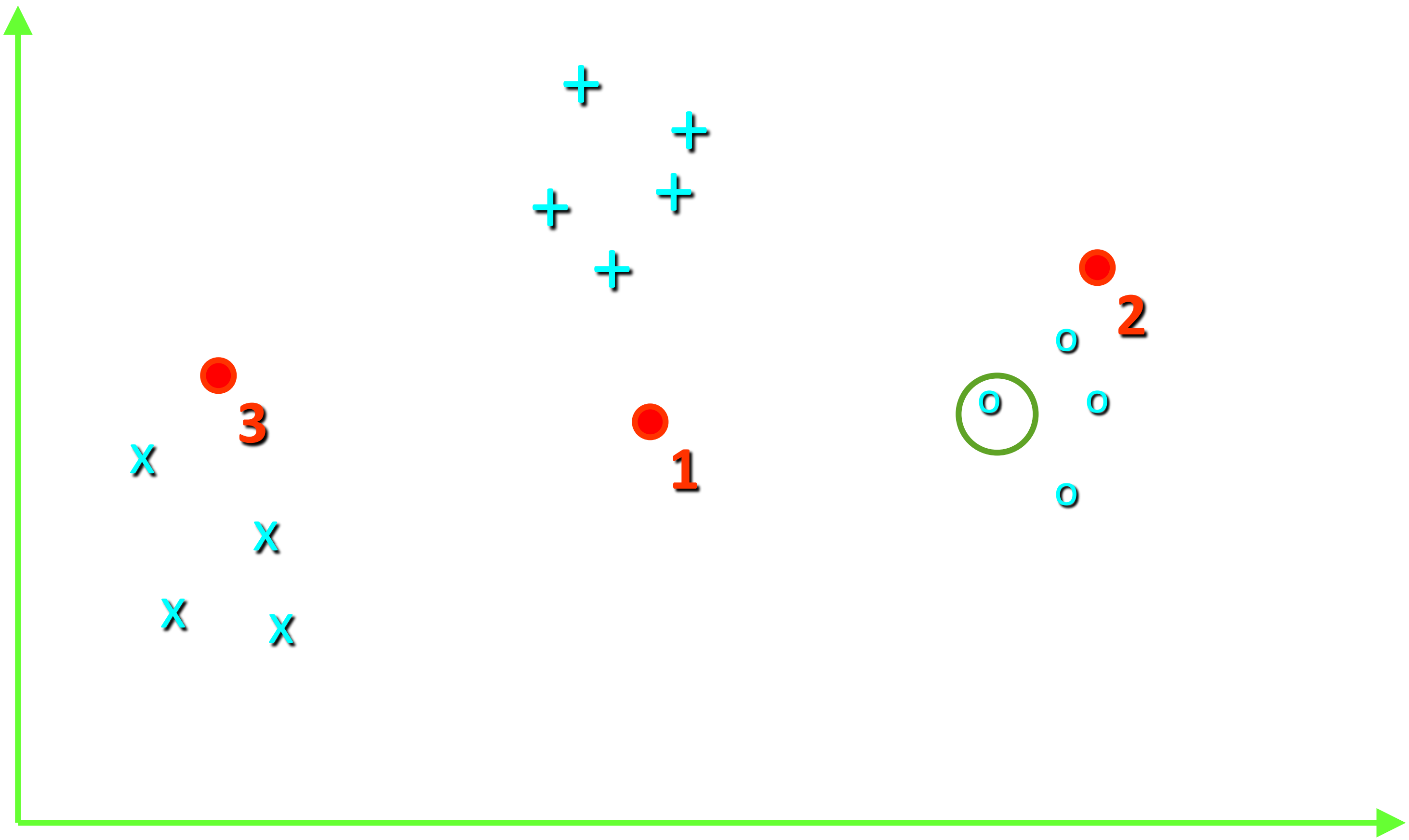
# Clustering (cont…)



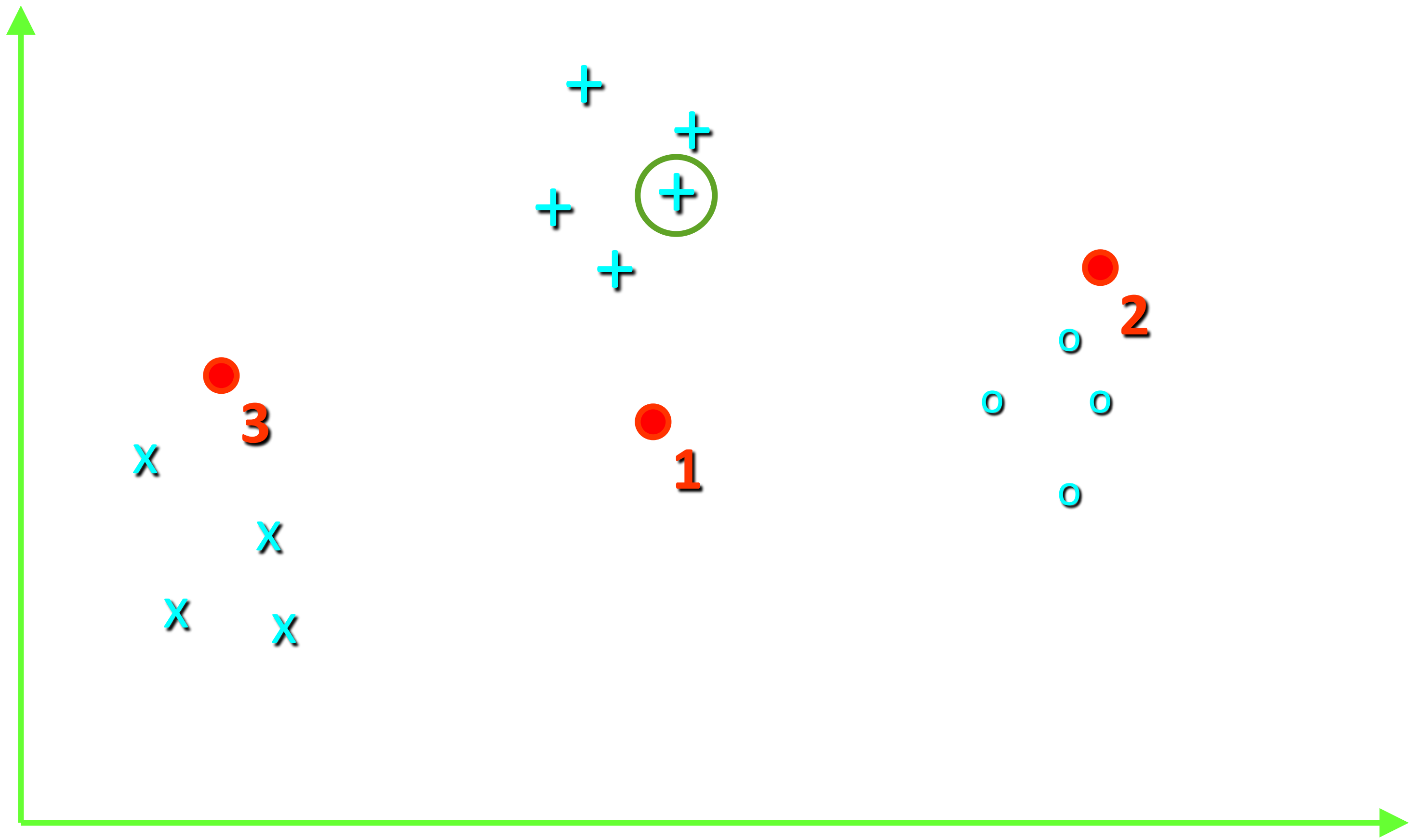
# Clustering (cont…)



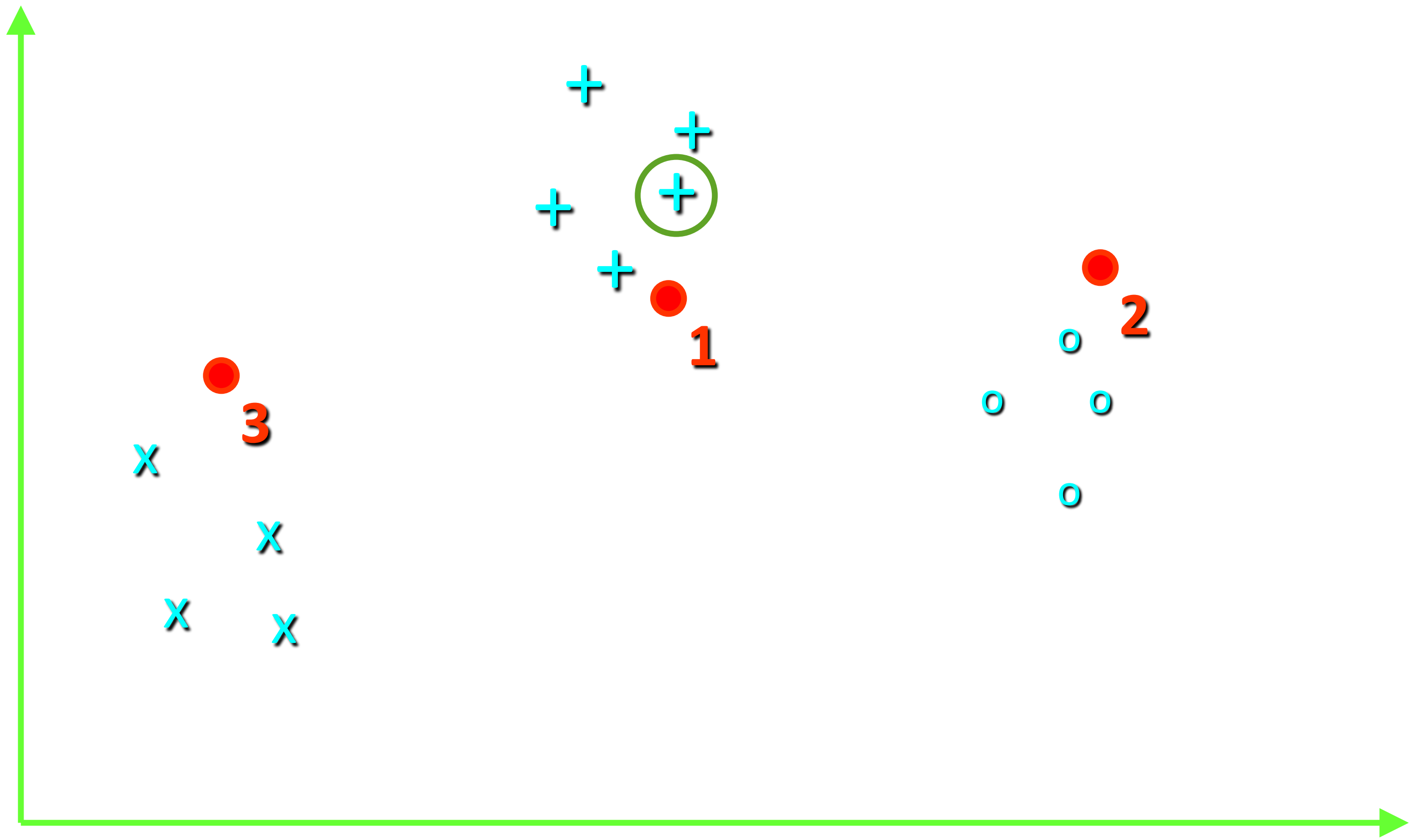
# Clustering (cont…)



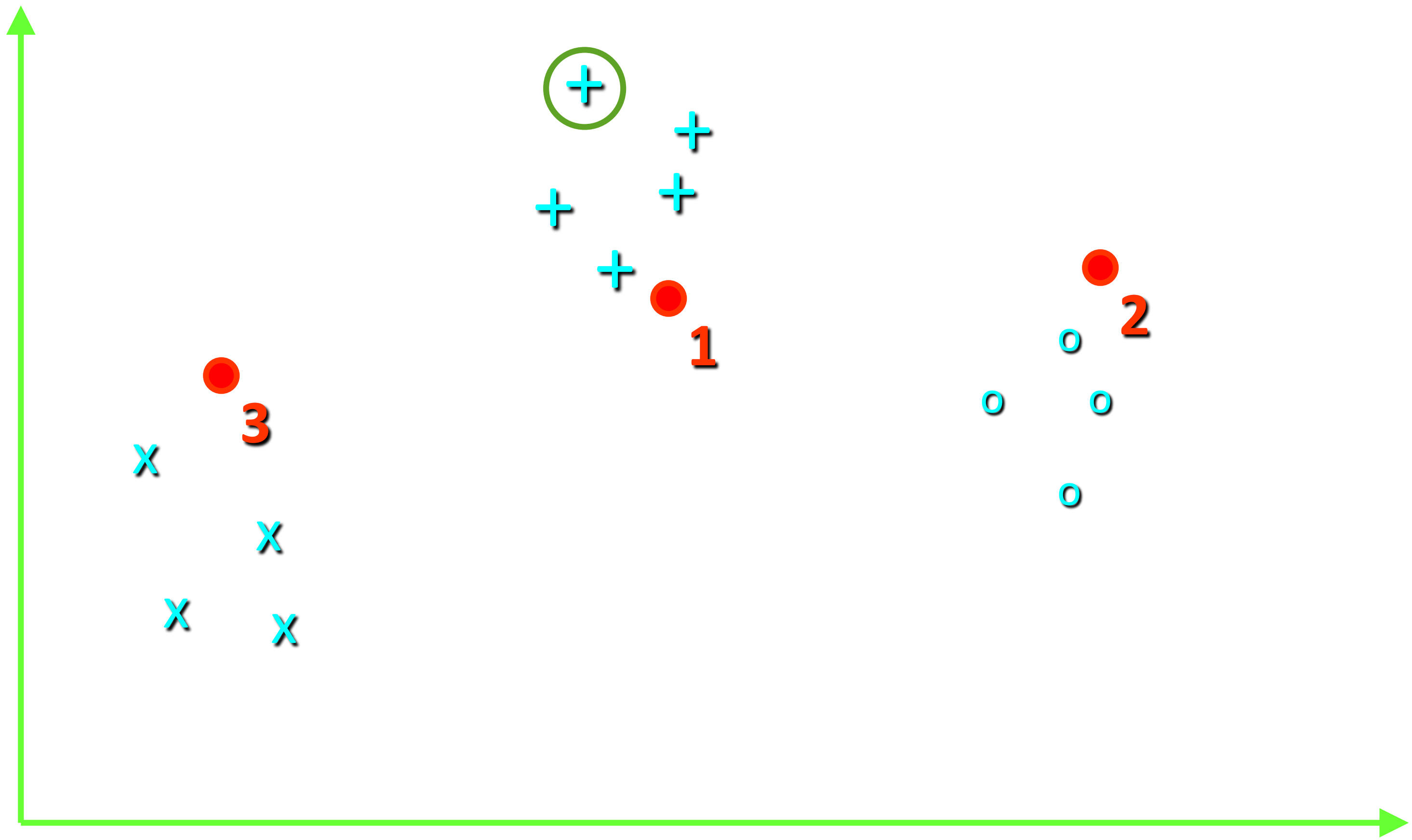
# Clustering (cont…)



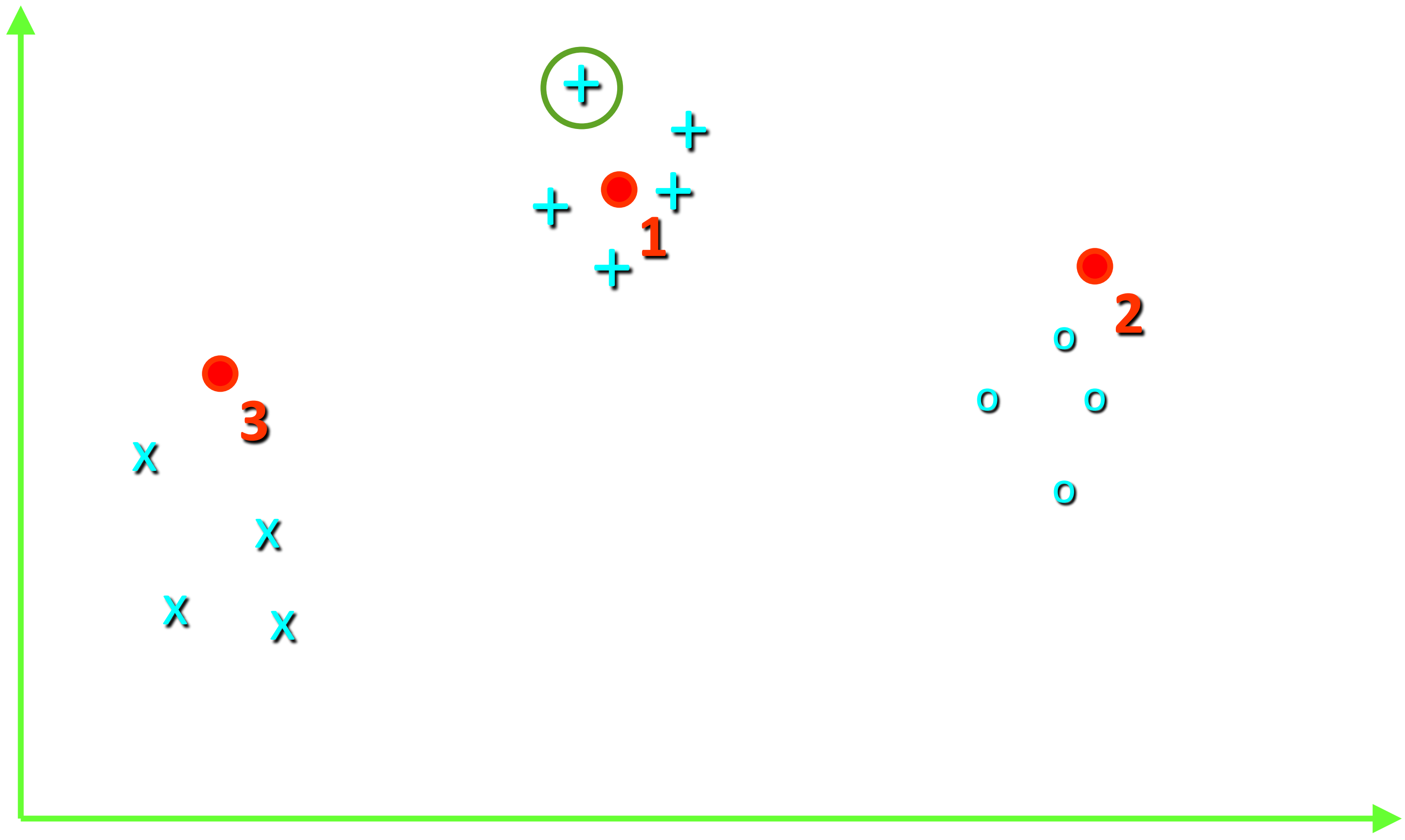
# Clustering (cont…)



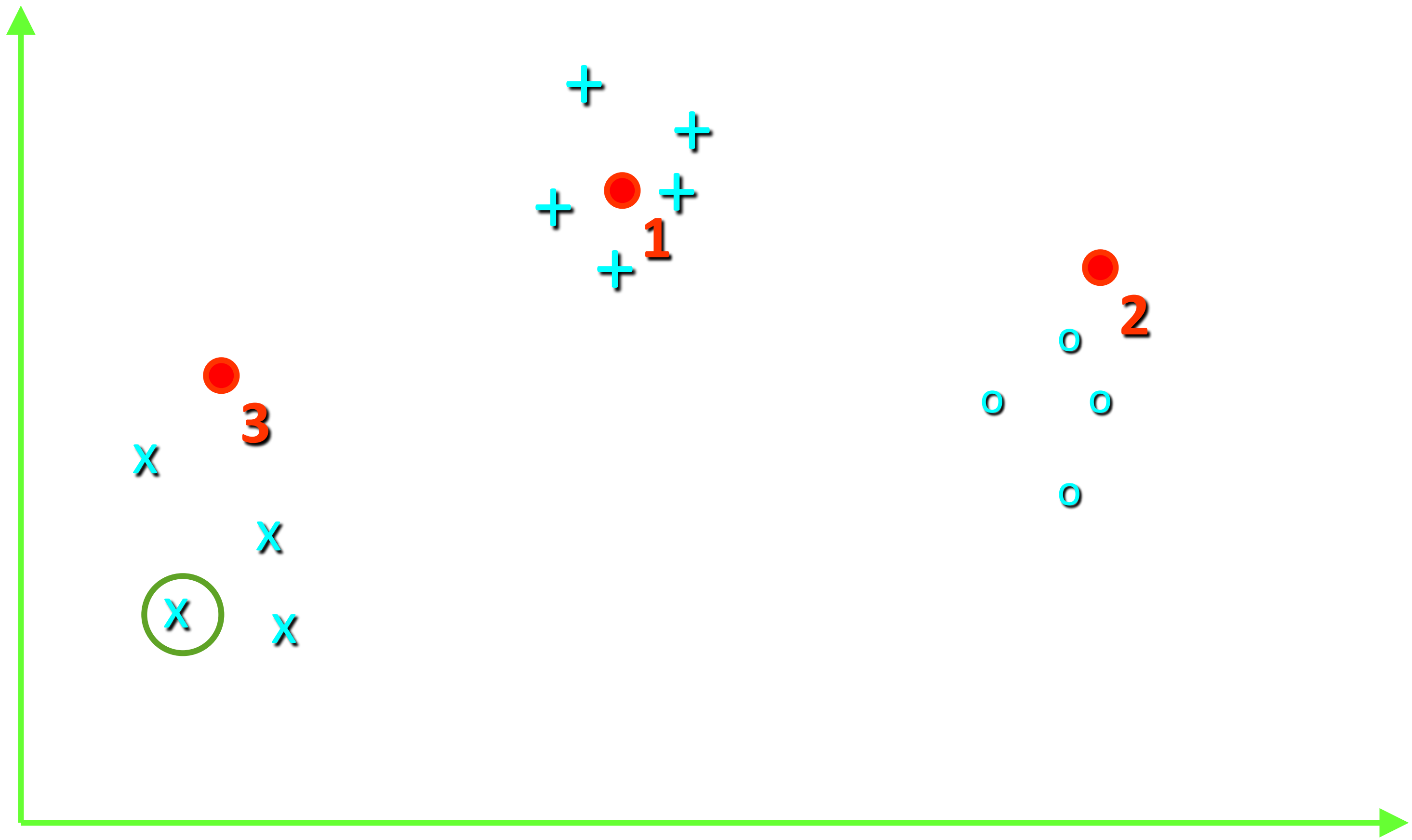
# Clustering (cont…)



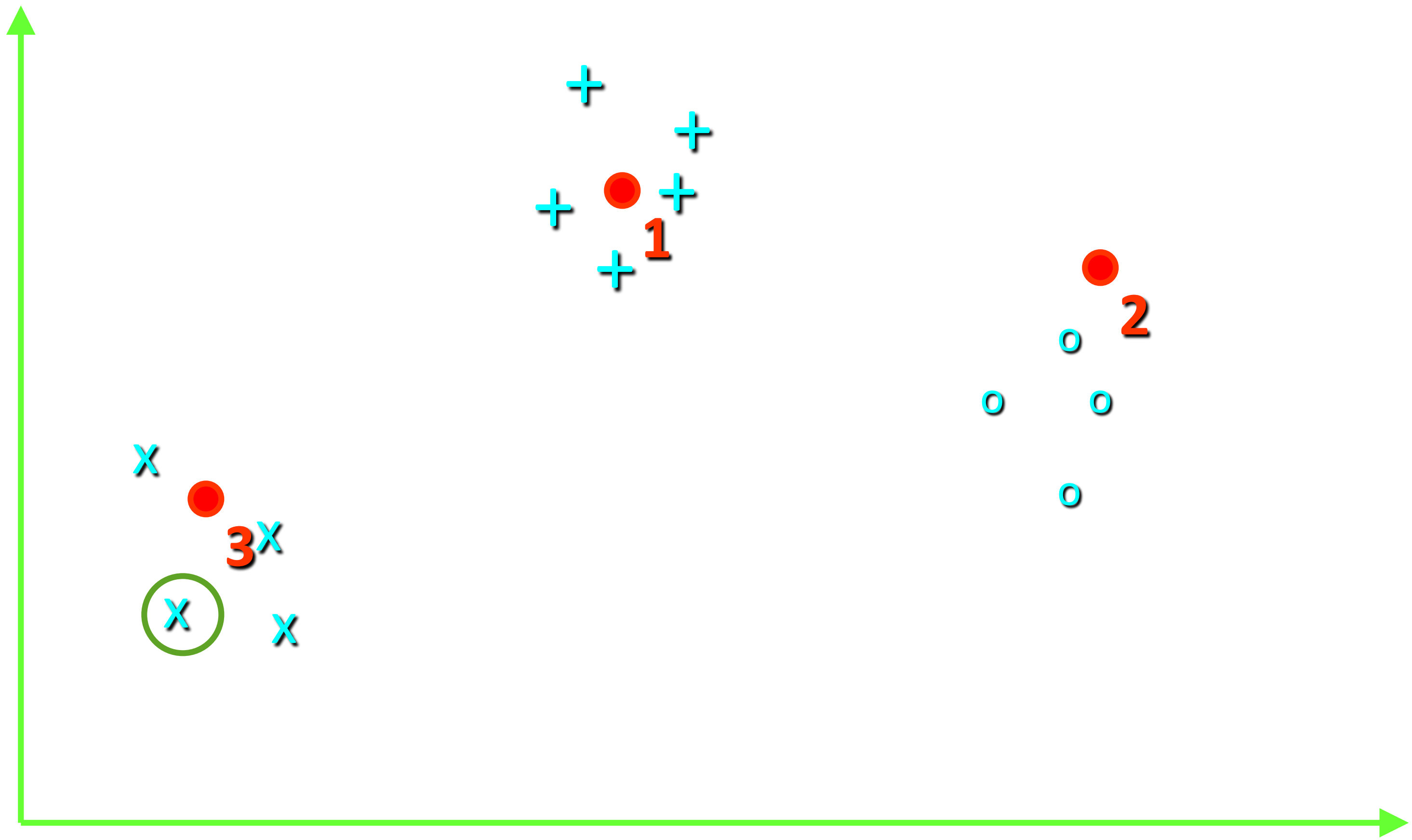
# Clustering (cont…)



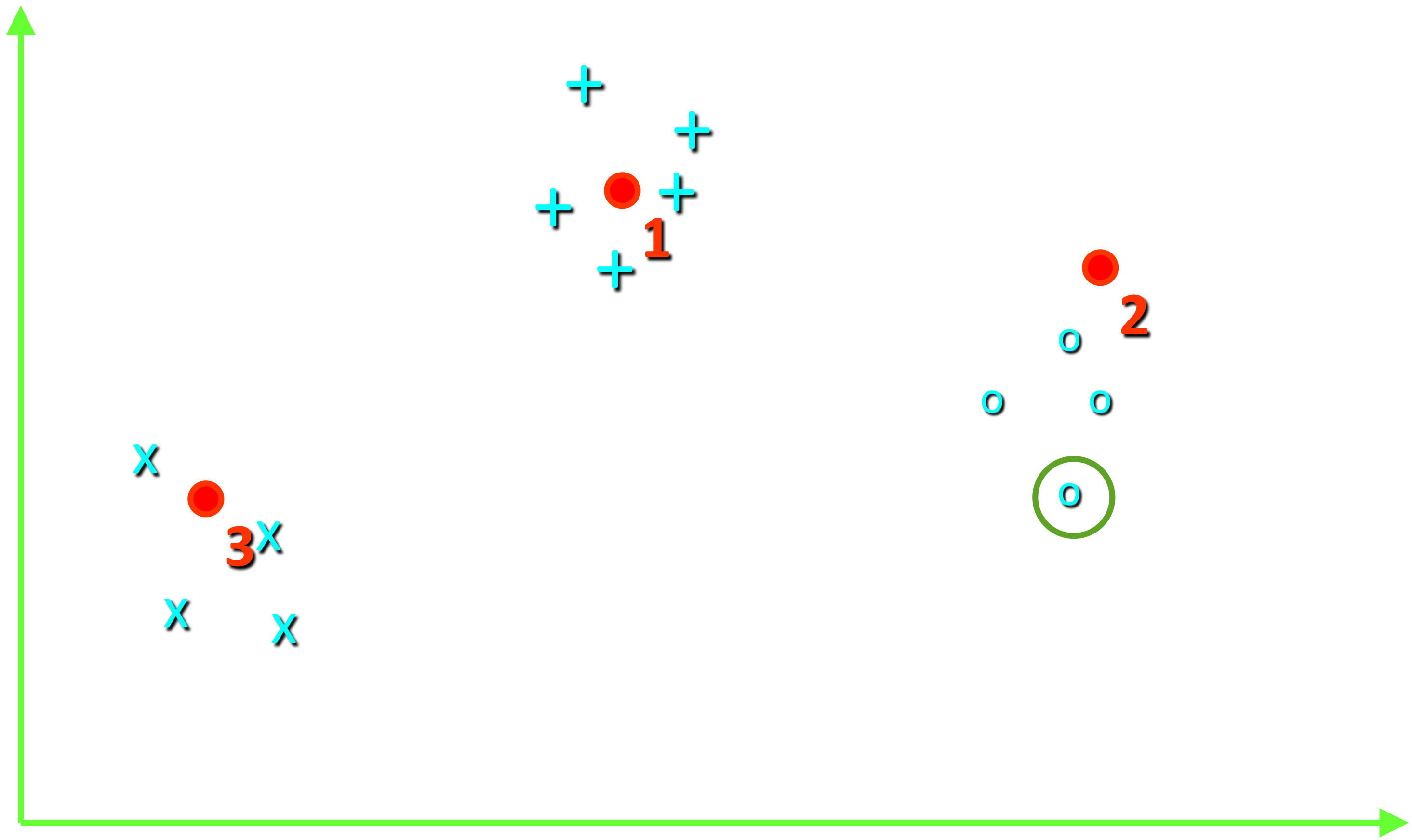
# Clustering (cont…)



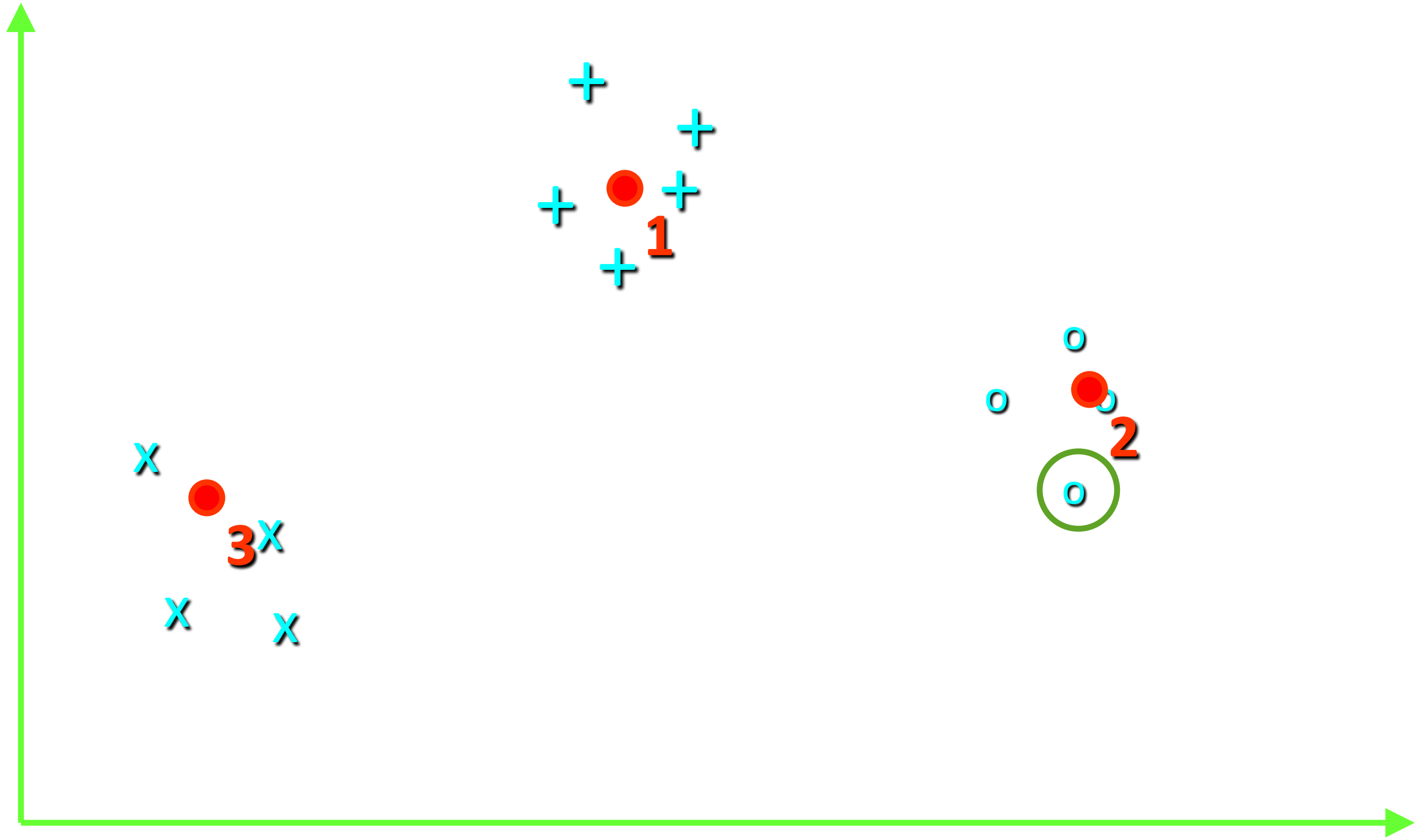
# Clustering (cont…)



# Clustering (cont…)



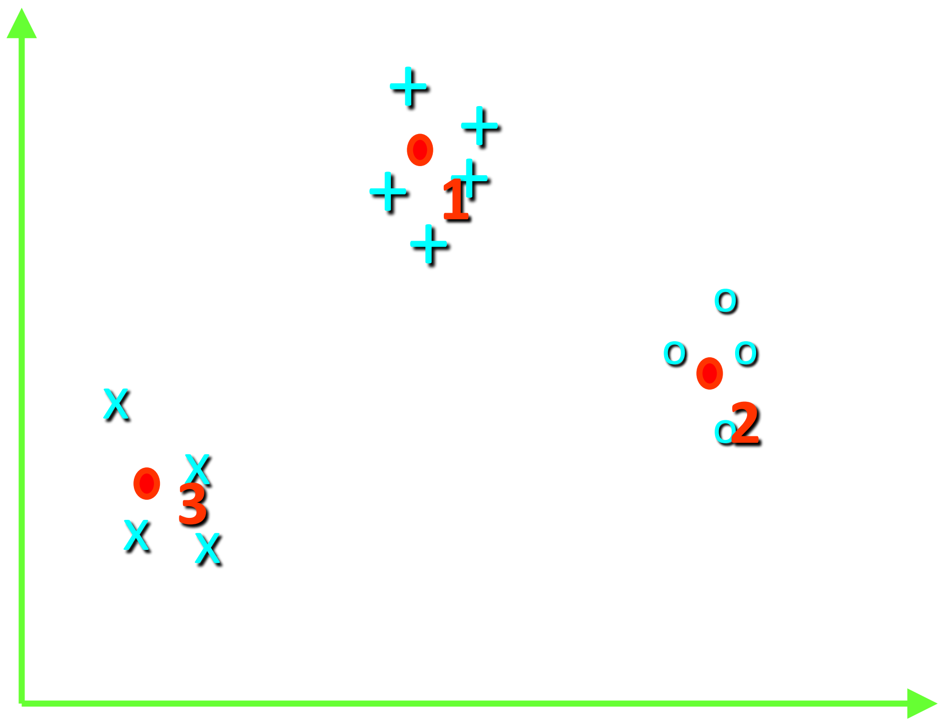
# Clustering (cont…)



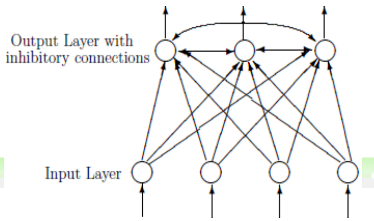

- Continues for the epoch and for multiple iterations

# Clustering (cont…)

- At the end the outputs of the network are at the center of the input data.
- Depending on their initial random values and the order of presentation of input data, the centroids of the clusters may be different.



# Kohonen Learning (cont...)




- Simple competitive learning can be accomplished using a Maxnet.
- The *j*th output node is described by its weight vector from the input nodes,

  $w_j = (w_{j,1}, \ldots, w_{j,n})$ $\quad\quad$ $l=\{1,\ldots,n\}$ input dimension

- A competition occurs to find the "winner" in the outer layer node j* whose weight vector (or "prototype" or "node position"), is nearest to the input vector $i_l$

  $d(w_{j*}, i_l) \leq d(w_j, i_l)$ $\quad$ where $j=\{1,\ldots,m\}$ # of output nodes

  $k=\{1,\ldots,P\}$ # of data points, $d$ is **Euclidean distance** =

$$d(w_j, i_k) = \sqrt{\sum_{l=1}^{n} (i_{l,k} - w_{j,l})^2}$$

# How Kohonen Learning Works

- $d(w_j, i_l) = \sqrt{\sum_{l=1}^{n} (i_{l,k} - w_{j,l})^2}$ When expanded with $\| i_l \| = \| w_j \| = 1$ (unit vectors $\|\mathbf{p}\| = \sqrt{p_1^2 + p_2^2 + \cdots + p_n^2} = \sqrt{\mathbf{p} \cdot \mathbf{p}}$ )

  $d^2(w, i) = \boldsymbol{i_k}.\boldsymbol{i_k} + \boldsymbol{w_j}.\boldsymbol{w_j} - 2 \sum i_{lk} w_{jl} = 2 - 2 \sum i_{lk} w_{jl} = 2 - 2\, \boldsymbol{i} \,.\, \boldsymbol{w}$
- So, the distance will be minimum when ***i.w*** (dot product) is maximum (||i|| ||w|| cos $\theta$ is maximum at $\theta = 0$). To generate output ***i.w***, use linear output neuron.
- *Hamming network* helps detect the distance as ***i.w*** in the first phase.
- Highest ***i.w*** indicates closest centroid. In the *winner-takes-all* phase, the *maxnet* recurrent network in the output layer finds the maximum distance ***i.w*** as the winning node.

# Kohonen Learning (cont…)

- Weights $w_j$* of the winning node $j$* are adjusted keeping other weights unchanged so that $w_j$* moves closer to the input $i_l$.

$$\Delta \boldsymbol{w}_{j*} = \boldsymbol{\eta}\, (\boldsymbol{i}_l - \boldsymbol{w}_{j*})$$

- It can be shown that in the limiting case, i.e., if and when **there is no significant change in the weight vector** when an input pattern is presented, $\Delta \boldsymbol{w}_j < \boldsymbol{\varepsilon}$, a predefined minimum threshold, $w_j$ converges to

$$w_j = \frac{1}{\sum_\ell \delta_{j,\ell}} \sum_\ell i_\ell\, \delta_{j,\ell},$$

$$\delta_{j,\ell} = \begin{cases} 1 & \text{if the } j\text{th node is "winner" for input } i_\ell, \\ 0 & \text{otherwise,} \end{cases}$$

averaging input vectors for which $w_j$ is the winner.

- The learning rate may vary such that $\eta(t+1) \le \eta(t)$, resulting in faster convergence (gradually reduce the rate).

**See example 5.3 in the book**

# Kohonen Learning (cont…)

**Figure 5.4** Simple competitive learning algorithm

Initialize weights randomly;

**repeat**

- (Optional:) Adjust learning rate $\eta(t)$;
- Select an input pattern $i_k$;
- Find node $j*$ whose weight vector $w_{j*}$ is closest to $i_k$;
- Update each weight $w_{j*,1}, \ldots, w_{j*,n}$ using the rule:

$$\Delta w_{j*,l} = \eta(t)(i_{k,l} - w_{j*,l}) \qquad \text{for } \ell \in \{1, \ldots, n\}$$

**until** network converges or computational bounds are exceeded

# Example 5.3

Input vectors $T = \{i_1 = (1.1, 1.7, 1.8), i_2 = (0, 0, 0), i_3 = (0, 0.5, 1.5), i_4 = (1, 0, 0), i_5 = (0.5, 0.5, 0.5), i_6 = (1, 1, 1)\}$

The network contains three input nodes; assume that there are also three output nodes, $A, B, C$ with initial weights randomly chosen:

$$W(0) = \begin{pmatrix} w_1 : & 0.2 & 0.7 & 0.3 \\ w_2 : & 0.1 & 0.1 & 0.9 \\ w_3 : & 1 & 1 & 1 \end{pmatrix}.$$

Assuming $\eta = 0.5$.

# Example 5.3 (cont…)

The first sample presented is $i_1 = (1.1, 1.7, 1.8)$. Squared Euclidean distance between $A$ and $i_1$:
$d_{1,1}^2 = (1.1 - 0.2)^2 + (1.7 - 0.7)^2 + (1.8 - 0.3)^2 = 4.1$. Similarly, $d_{2,1}^2 = 4.4$ and $d_{3,1}^2 = 1.1$.

$C$ is the "winner" since $d_{3,1}^2 < d_{1,1}^2$ and $d_{3,1}^2 < d_{2,1}^2$. $A$ and $B$ are therefore not perturbed by this sample whereas $C$ moves halfway towards the sample (since $\eta = 0.5$).

The resulting weight matrix is:

$$W(1) = \begin{pmatrix} w_1: & 0.2 & 0.7 & 0.3 \\ w_2: & 0.1 & 0.1 & 0.9 \\ w_3: & 1.05 & 1.35 & 1.4 \end{pmatrix}$$

E.g. w31 = w31 + Δw31 = 1 + 0.5 * (1.1 - 1.0) = 1.05

# Example 5.3 (cont…)

Assuming input vectors are presented repeatedly in the sequence, the weight matrix after 12 steps is :

$$W(12) = \begin{pmatrix} w_1: & 0.55 & 0.3 & 0.3 \\ w_2: & 0 & 0.4 & 1.35 \\ w_3: & 1 & 1.2 & 1.25 \end{pmatrix}.$$

- Each node tends to be the winner for the same input vectors, in later iterations, and moves towards their centroid.
- Convergence is not smooth when $\eta$ is high.
- Different results are observed when other distance measures are used. E.g. use **Manhattan distance $d(x, y) = \sum|x_l - y_l|$, and you will have different centroids**.
- **Using three nodes does not guarantee that three "clusters" will be obtained** by the network. Close initial values may result in splitting of a cluster into two.
- Results depend on the **initial weight vectors and the sequence in which samples are presented** especially when $\eta$ is high.

# k-means Clustering

- A statistical procedure closely related to simple competitive learning, which **computes cluster centroids directly instead of making small updates** to node positions.
- The k-means algorithm is fast and converges to a state in which each prototype changes little, assuming that successive vectors presented to the algorithm are drawn by independent random trials from the input data distribution.
- Both k-means and simple competitive learning can lead to local minima of E.

# Algorithm k-means Clustering

**Figure 5.5** k-means clustering algorithm

Initialize $k$ prototypes

$$w_j = i_\ell, j \in \{1, \ldots, k\}, \; \ell \in \{1, \ldots, P\}$$

Each cluster $C_j$ is associated with prototype $w_j$.

**repeat**

  **for** each input vector $i_\ell$ **do**

    Place $i_\ell$ in the cluster with nearest prototype $w_{j*}$

  **end for**

  **for** each cluster $C_j$ **do**

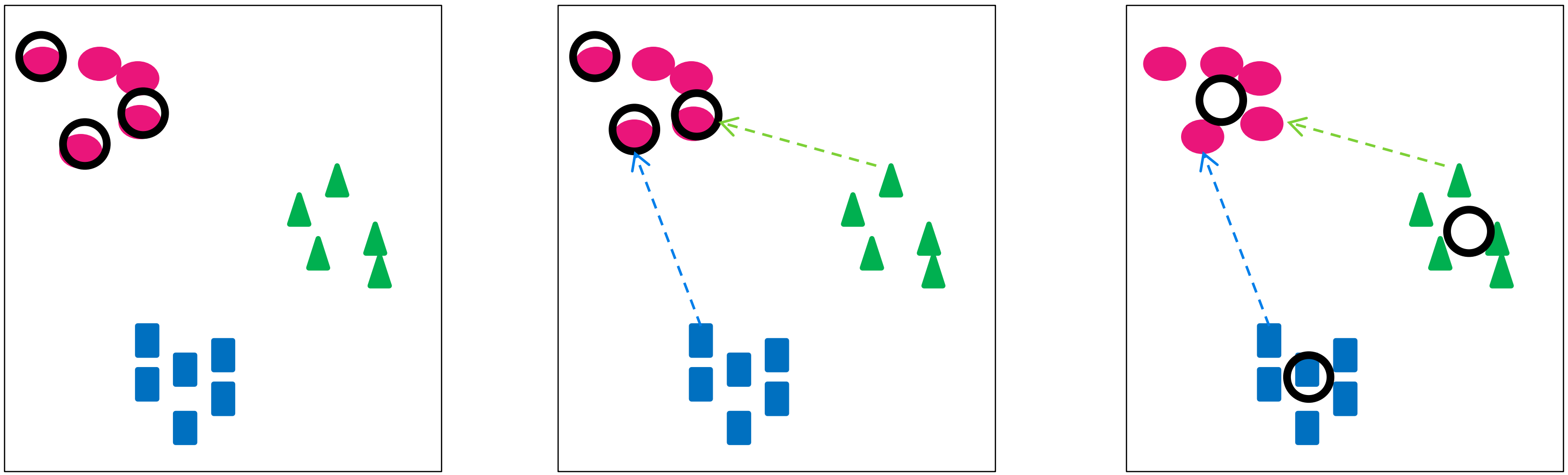
$$w_j = \frac{1}{|c_j|} \sum_{i_l \in c_j} i_l \quad \text{where} \; |c_j| \; \text{is the cluster size}$$

  **end for**

$$\text{Compute } E = \sum_{j=1}^{k} \sum_{i_l \in c_j} | i_l - w_j |^2$$

**until** $E$ no longer decreases, or cluster memberships stabilize

# k-means Clustering



- k= 3. Initially 3 centroids are initialized by 3 training samples
- Takes only two iterations to stabilize

# Kohonen as k-means

The simple competitive learning algorithm conducts stochastic gradient descent on the *quantization error*

$$E = \sum_{p} | i_p - \mathbb{W}(i_p) |^2$$

where $\mathbb{W}(i_p)$ is the weight vector nearest to $i_p$

The number of nodes is assumed to be fixed, but the right choice may not be obvious. We may attempt to minimize $E + c(\text{number of nodes})$ instead of $E$, for $c > 0$.

The above adjustment is called **Regularization** (one approach).

# Learning Vector Quantizers (LVQ)

- Sometimes clustering is used as a useful preprocessing step for solving classification problem.
- A LVQ is an application of the above, uses *winner-take-all network* and illustrates how unsupervised learning can be adapted to solve supervised learning.
- **Class membership is known for each training pattern**.
- Learns the codebook vectors.

# Learning Vector Quantizers

- Each output node is associated with an arbitrary class label in the beginning.
  - Each node should be finally associated with approximately the number of training data belonging to that class.
- Initial weights are chosen randomly.
- The learning rate decreases with time - helps the network converge to a state in which weight vectors are stable.
  - e.g., $\eta(t) = 1/t$ or $\eta(t) = a[1 – (t/A)]$ where $a > 0$ and $A > 1$ i.e., variable rate → decrease more with time.

# LVQ (cont…)

- When pattern $i$ from class C($i$) is presented to the network, let the winner node $j^*$ ϵ C($j^*$).
- **If this is the correct class** i.e., C($i$)=C($j^*$), $j^*$ moves closer to $i$

$$\Delta w_{j^*,l} = \eta(t)(i_{k,l} - w_{j^*l})$$

- Otherwise $j^*$ moves away from $i$.

$$\Delta w_{j^*,l} = -\eta(t)(i_{k,l} - w_{j^*,l})$$

# LVQ1 Algorithm

**Figure 5.6** LVQ1 algorithm

Initialize all weights $\in [0, 1]$
**repeat**
  Adjust $\eta(t)$;
  **for** each $i_k$ **do**
    find node $j*$ whose weight vector $w_{j*}$ is closest to $i_k$;
  **end for**
  **for** $\ell = 1, \ldots, n$ **do**
    **if** the class label of node $j*$ equals the desired class of $i_k$ **then**
      $\Delta w_{j*,l} = \eta(t)(i_{k,l} - w_{j*,l})$
    **else**
      $\Delta w_{j*,l} = -\eta(t)(i_{k,l} - w_{j*,l})$
    **end if**
  **end for**
**until** network converges or computational bounds are exceeded

# Example 5.5

$\{i_1 = (1.1, 1.7, 1.8),\ i_2 = (0, 0, 0),\ i_3 = (0, 0.5, 1.5),\ i_4 = (1, 0, 0),$ $i_5 = (0.5, 0.5, 0.5)$, and $i_6 = (1, 1, 1)\}$. Assume only the first and last samples come from Class 1, and

$$W(0) = \begin{pmatrix} w_1: & 0.2 & 0.7 & 0.3 \\ w_2: & 0.1 & 0.1 & 0.9 \\ w_3: & 1 & 1 & 1 \end{pmatrix}.$$

$w_1$ is associated with Class 1, and the other two nodes with Class 2.

Simply because there are twice as many training data associated with class 2 than class 1.

# Example 5.5 (cont…)

Let $\eta(t) = 0.5$ until $t = 6$, then $\eta(t) = 0.25$ until $t = 12$ and $\eta(t) = 0.1$ thereafter.

1. Sample $i_1$, winner $w_3$ (distance 1.07), $w_3$ changed to (0.95, 0.65, 0.60). → But $w_3$ is class 2 and $i_1$ is class 1. So $-\Delta w$ applied.
2. $\vdots \; \vdots \; \vdots \; \vdots$

Associations between input samples and weight vectors stabilize by the second cycle of pattern presentations. Weight vectors continue to change, converging to centroids of associated input vectors in 150 iterations.