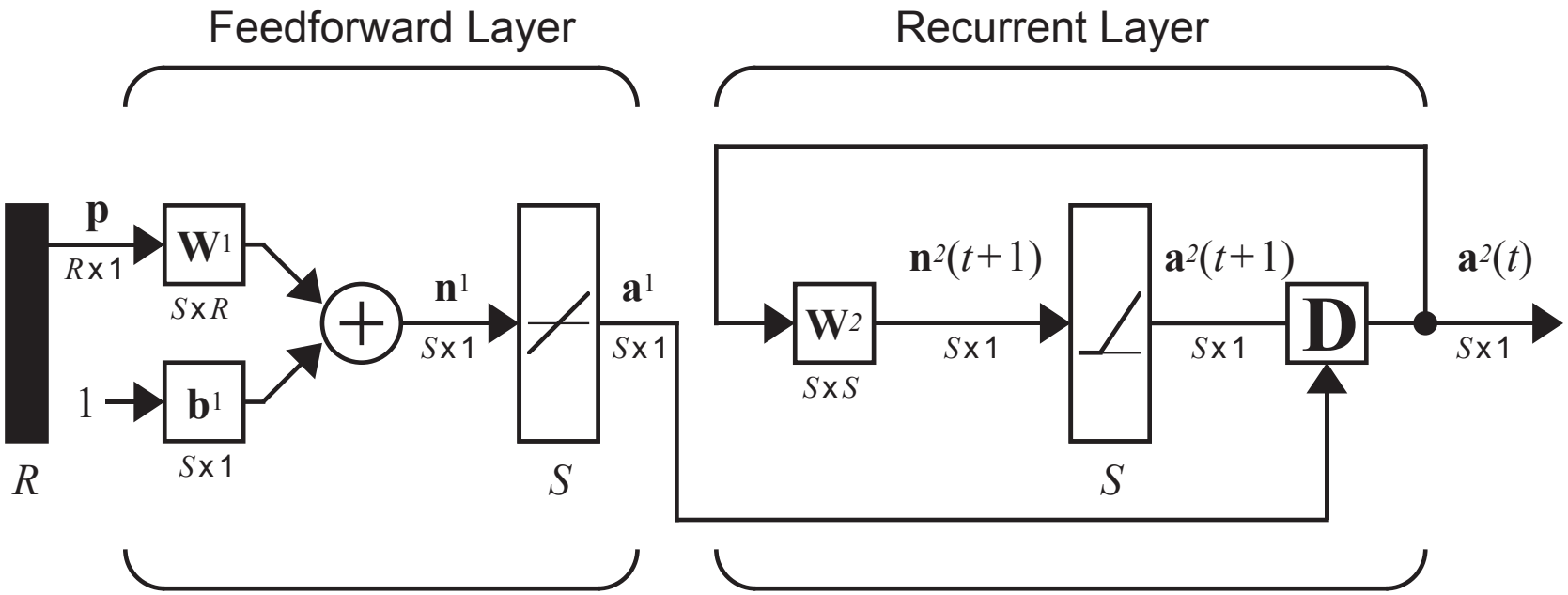
16

# Competitive Networks

# Hamming Network




$$\mathbf{a}^1 = \mathbf{purelin}(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) \qquad \mathbf{a}^2(0) = \mathbf{a}^1 \qquad \mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$$

# Layer 1 (Correlation)

We want the network to recognize the following prototype vectors:

$$\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_Q\}$$

The first layer weight matrix and bias vector are given by:

$$\mathbf{W}^1 = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} \qquad \mathbf{b}^1 = \begin{bmatrix} R \\ R \\ \vdots \\ R \end{bmatrix}$$

The response of the first layer is:

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^T\mathbf{p} + R \\ \mathbf{p}_2^T\mathbf{p} + R \\ \vdots \\ \mathbf{p}_Q^T\mathbf{p} + R \end{bmatrix}$$

The prototype closest to the input vector produces the largest response.

# Layer 2 (Competition)

$$\mathbf{a}^2(0) = \mathbf{a}^1$$

*The second layer is initialized with the output of the first layer.*

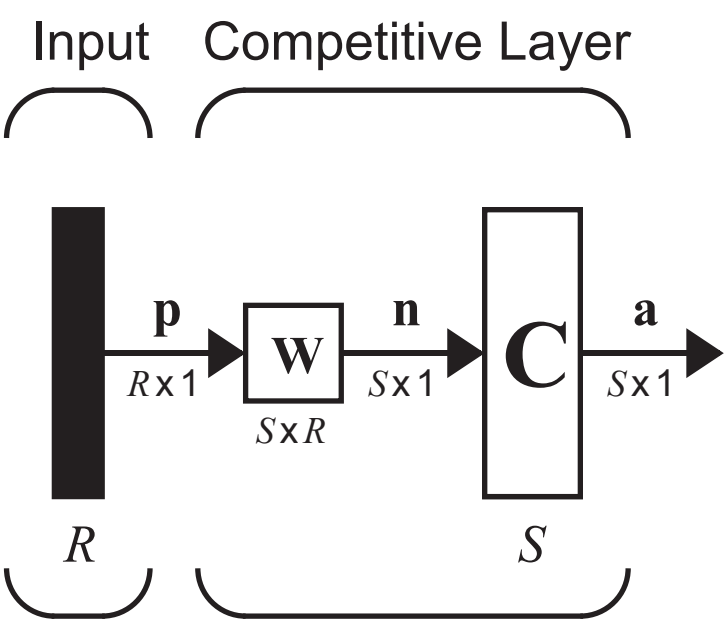
$$\mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$$

$$w_{ij}^2 = \begin{cases} 1, & \text{if } i = j \\ -\varepsilon, & \text{otherwise} \end{cases} \qquad 0 < \varepsilon < \frac{1}{S-1}$$

$$a_i^2(t+1) = poslin\left(a_i^2(t) - \varepsilon \sum_{j \neq i} a_j^2(t)\right)$$

*The neuron with the largest initial condition will win the competiton.*

# Competitive Layer




$$\mathbf{a} = \mathbf{compet}(\mathbf{Wp})$$

$$\mathbf{n} = \mathbf{W}\mathbf{p} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix} \mathbf{p} = \begin{bmatrix} {}_1\mathbf{w}^T\mathbf{p} \\ {}_2\mathbf{w}^T\mathbf{p} \\ \vdots \\ {}_S\mathbf{w}^T\mathbf{p} \end{bmatrix} = \begin{bmatrix} L^2\cos\theta_1 \\ L^2\cos\theta_2 \\ \vdots \\ L^2\cos\theta_S \end{bmatrix}$$

$$\mathbf{a} = \mathbf{compet}(\mathbf{n})$$

$$a_i = \begin{cases} 1, & i = i^* \\ 0, & i \neq i^* \end{cases} \qquad n_{i^*} \geq n_i, \forall i \qquad i^* \leq i, \forall n_i = n_{i^*}$$

# Competitive Learning

## Instar Rule

$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) $$

For the competitive network, the winning neuron has an ouput of 1, and the other neurons have an output of 0.

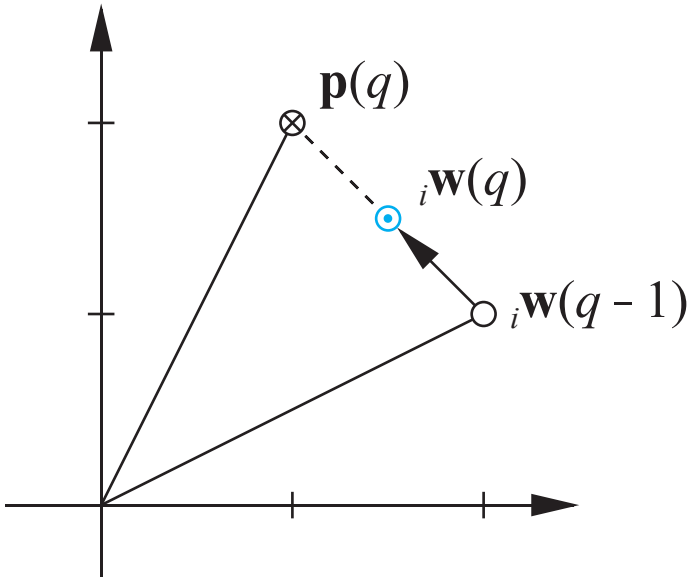
## Kohonen Rule

$$ {}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1)) $$

$$ {}_{i^*}\mathbf{w}(q) = (1-\alpha){}_{i^*}\mathbf{w}(q-1) + \alpha\mathbf{p}(q) $$

$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) \qquad i \neq i^* $$

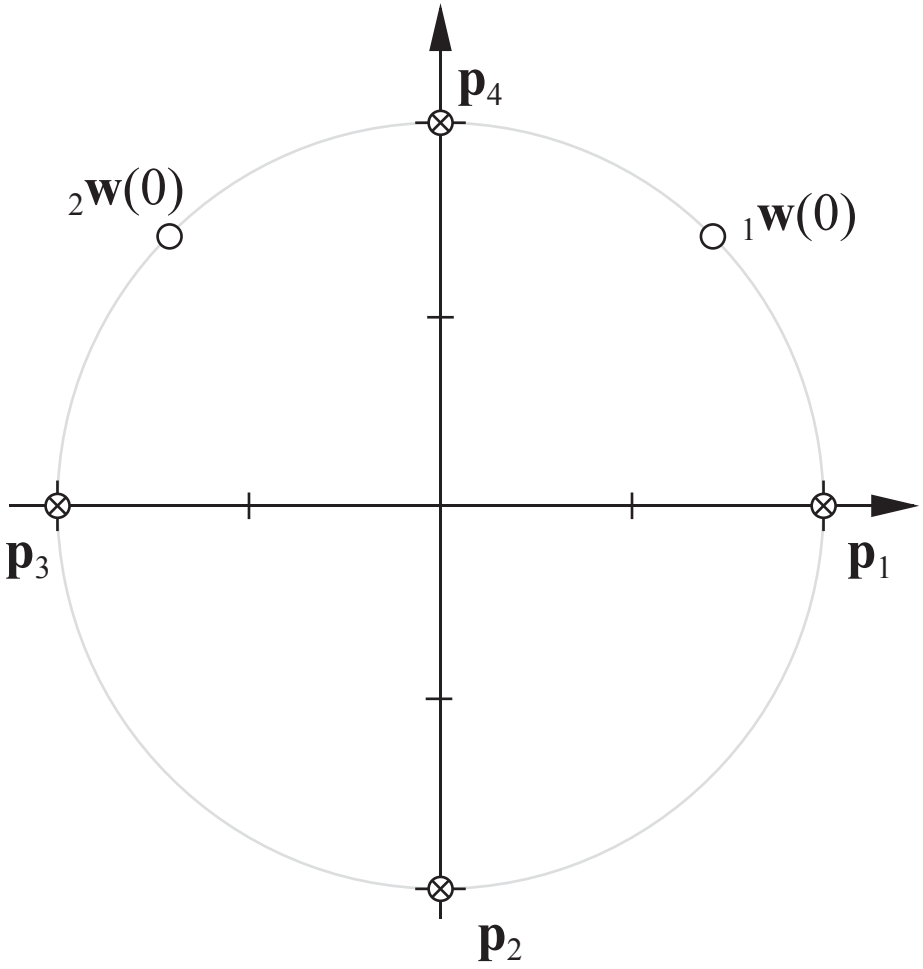
# Graphical Representation



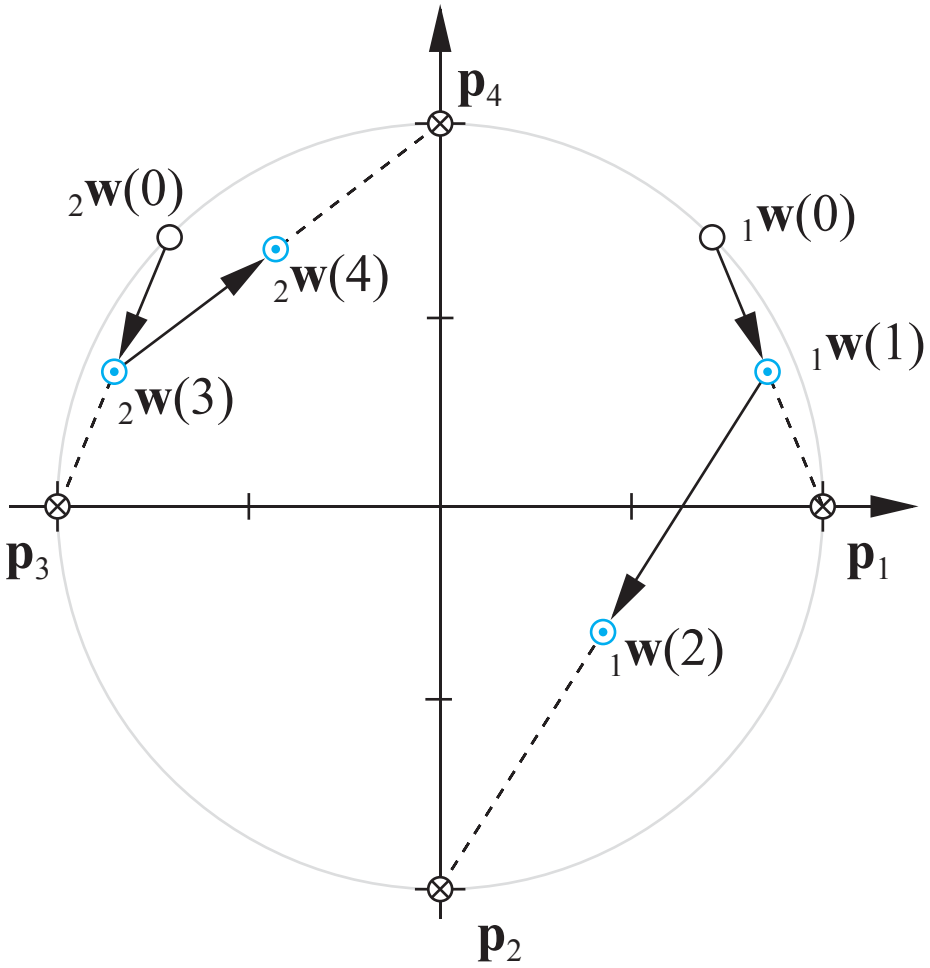

$$ {}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1)) $$

$$ {}_{i^*}\mathbf{w}(q) = (1-\alpha){}_{i^*}\mathbf{w}(q-1) + \alpha\mathbf{p}(q) $$

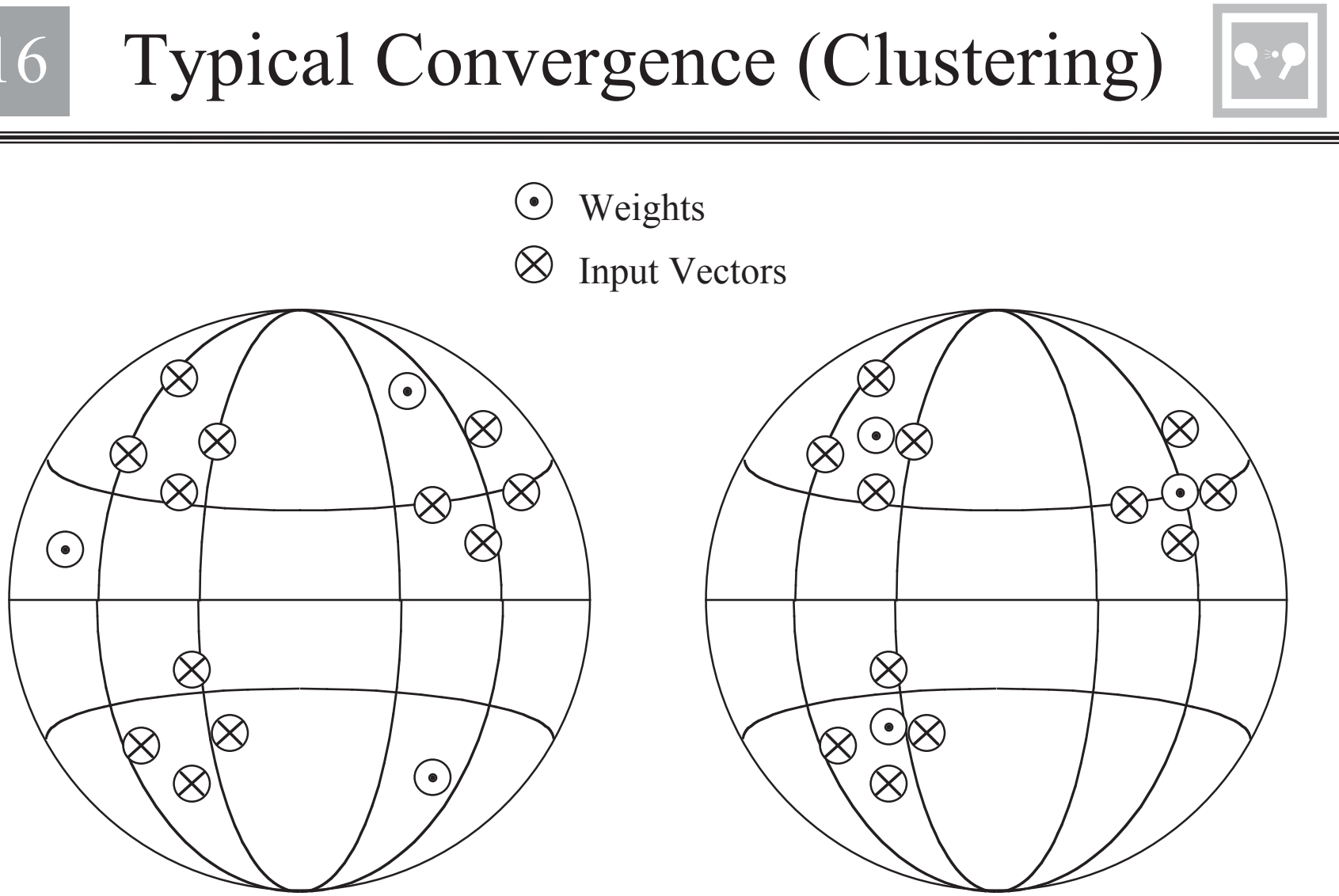
# Example



# Four Iterations



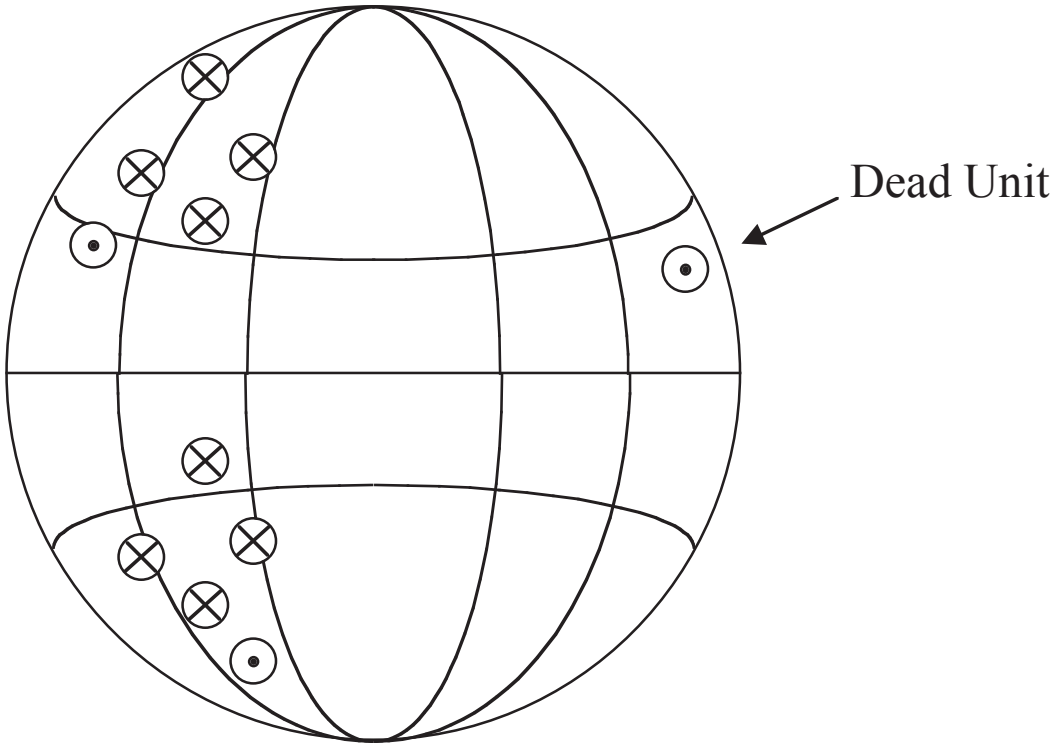
# 16 Typical Convergence (Clustering)



# Dead Units



One problem with competitive learning is that neurons with initial weights far from any input vector may never win.

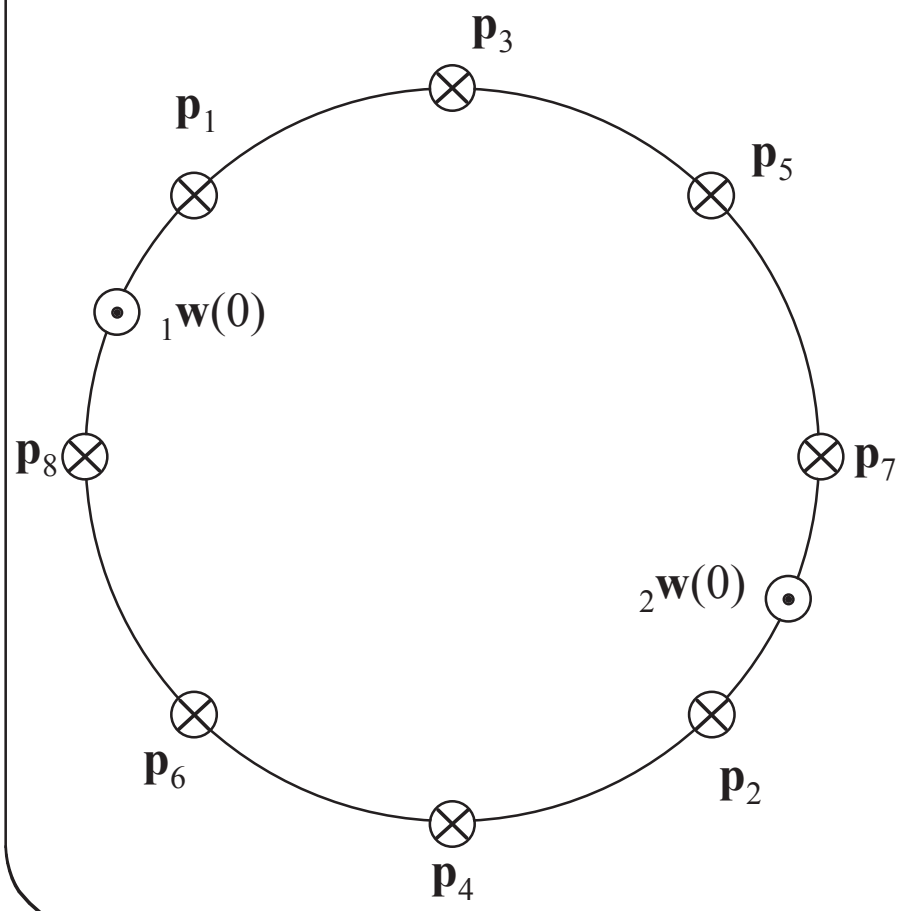


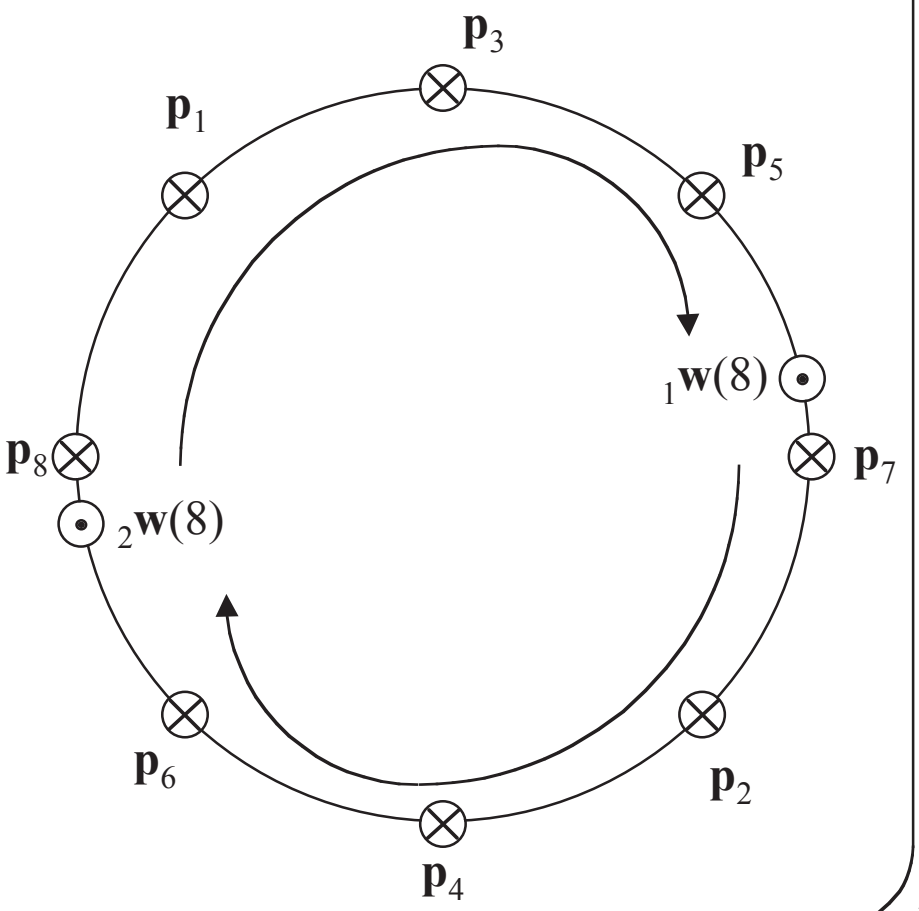

Solution: Add a negative bias to each neuron, and increase the magnitude of the bias as the neuron wins. This will make it harder to win if a neuron has won often. This is called a "conscience."

# Stability



If the input vectors don't fall into nice clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will undergo continual evolution.



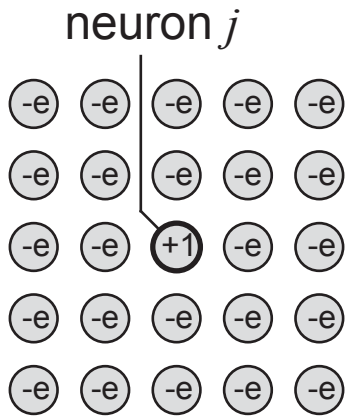
# Competitive Layers in Biology

## On-Center/Off-Surround Connections for Competition

Weights in the competitive layer of the Hamming network:

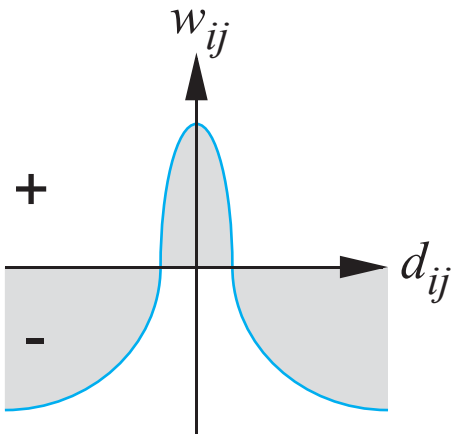
$$w_{i,j} = \begin{cases} 1, & \text{if } i = j \\ -\varepsilon, & \text{if } i \neq j \end{cases}$$

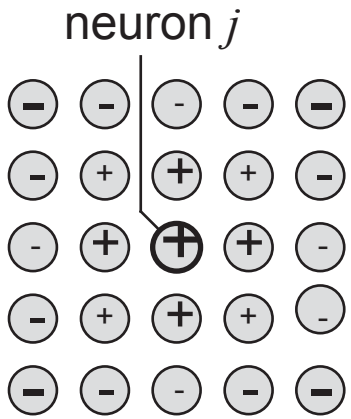
Weights assigned based on distance:




$$w_{i,j} = \begin{cases} 1, & \text{if } d_{i,j} = 0 \\ -\varepsilon, & \text{if } d_{i,j} > 0 \end{cases}$$

# Mexican-Hat Function



# Feature Maps

Update weight vectors in a neighborhood of the winning neuron.

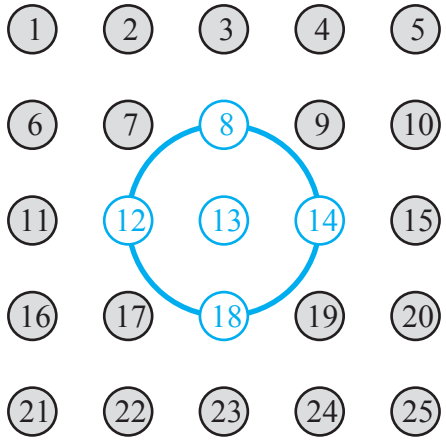
$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) $$

$$ {}_i\mathbf{w}(q) = (1-\alpha){}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q) \qquad i \in N_{i^*}(d) $$

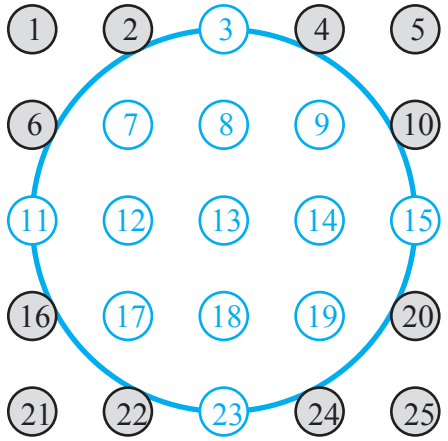
$$ N_i(d) = \{j, \mathrm{d}_{i,j} \le d\} $$

$$ N_{13}(1) = \{8, 12, 13, 14, 18\} $$

$$ N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\} $$

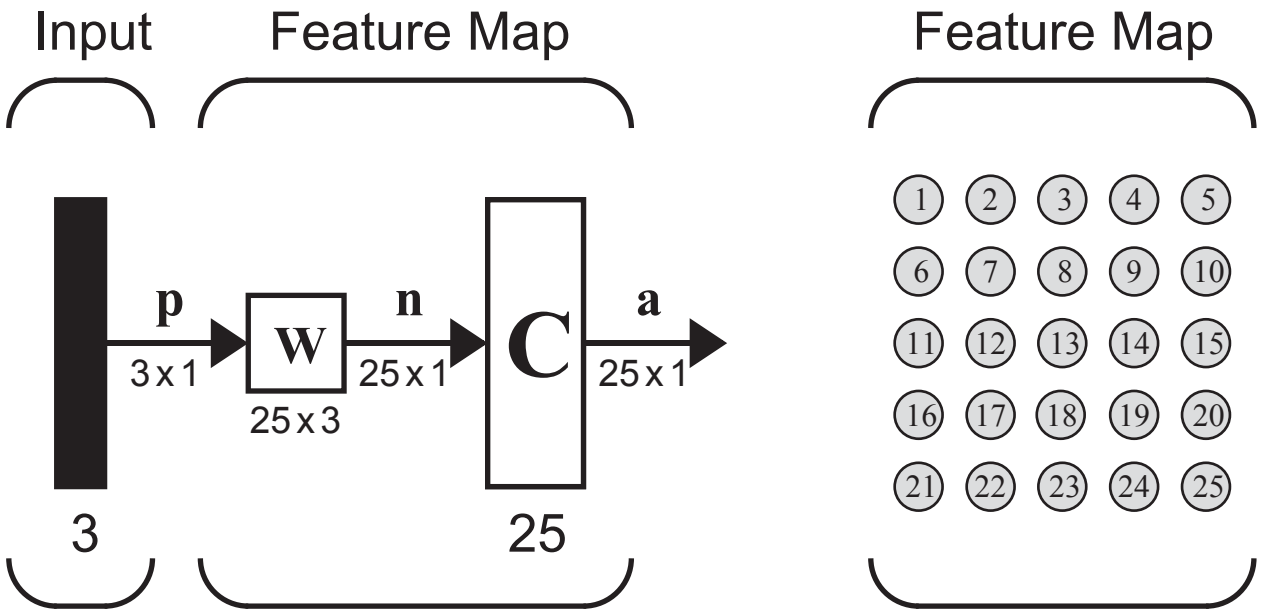



$N_{13}(1)$



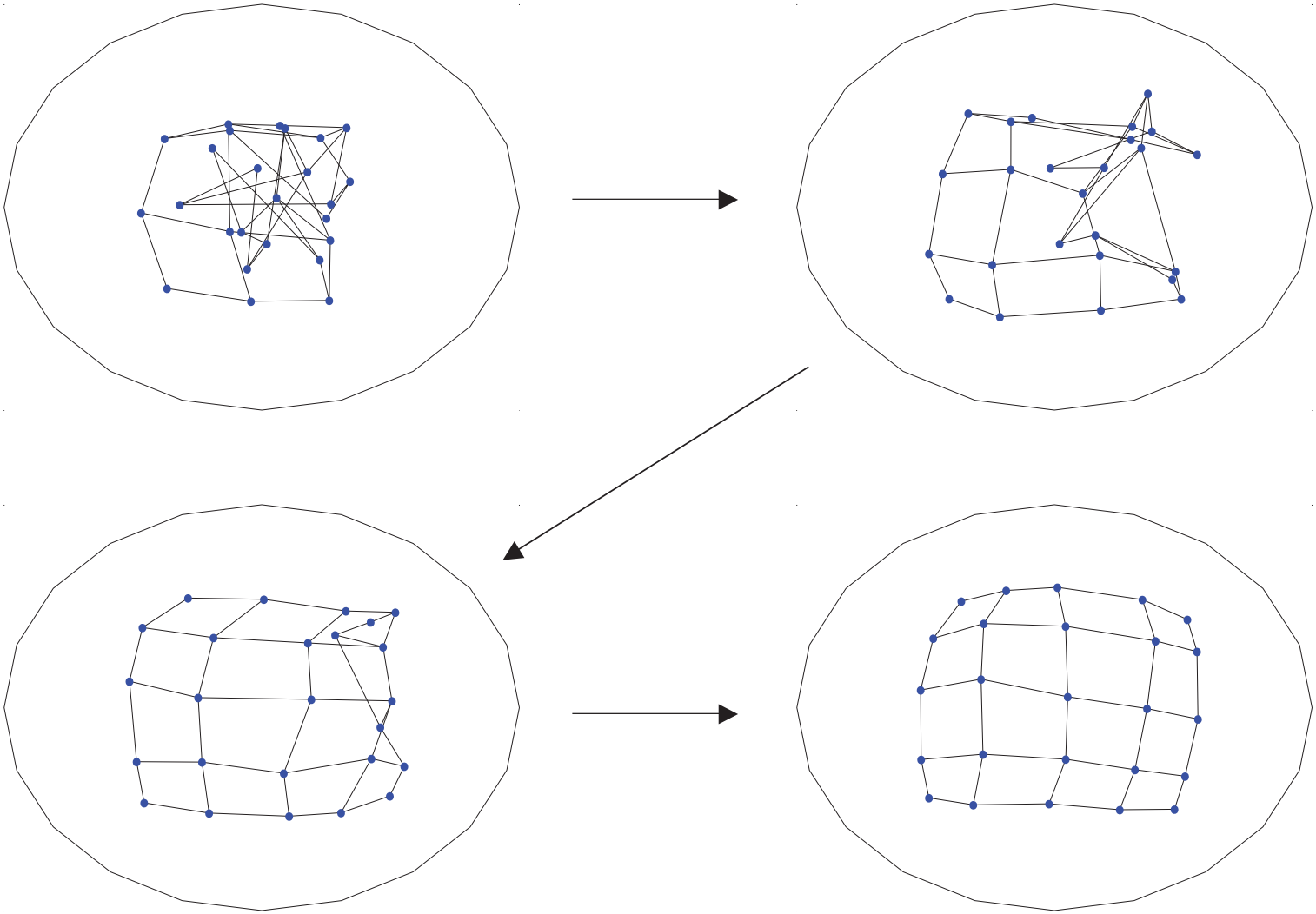

$N_{13}(2)$

# Example

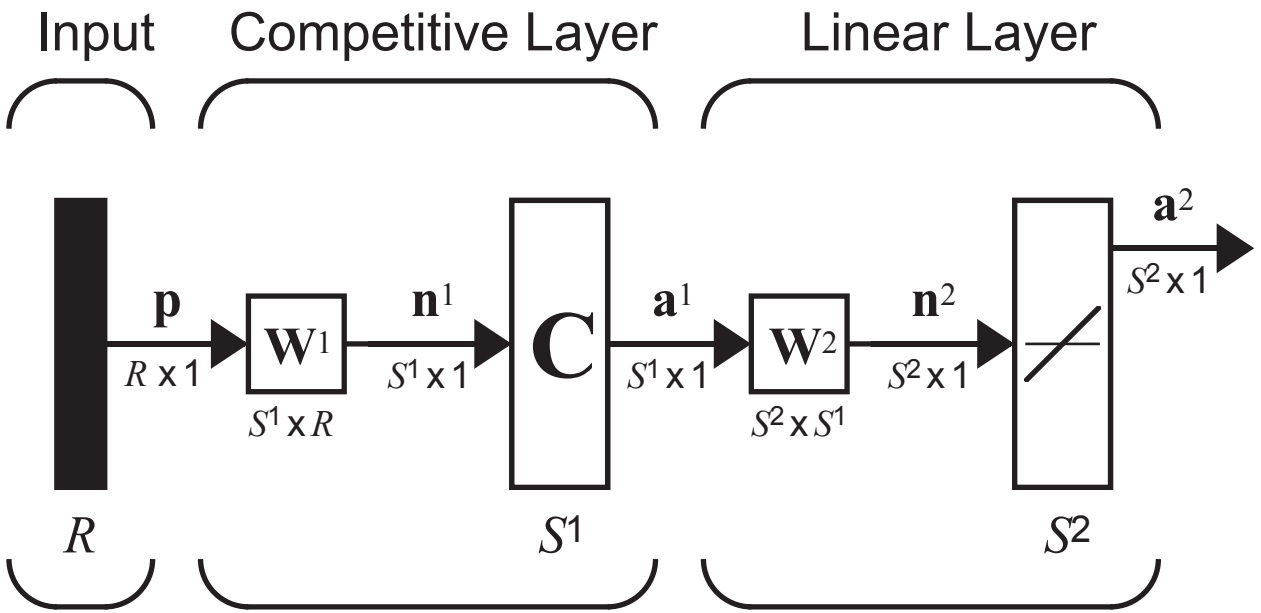



$$\mathbf{a} = \mathbf{compet}(\mathbf{Wp})$$

# Convergence



# Learning Vector Quantization




$$n_i^1 = -\|_i\mathbf{w}^1 - \mathbf{p}\| \qquad \mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1$$

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1)$$

The net input is not computed by taking an inner product of the prototype vectors with the input. Instead, the net input is the negative of the distance between the prototype vectors and the input.

# Subclass

For the LVQ network, the winning neuron in the first layer indicates the **subclass** which the input vector belongs to. There may be several different neurons (subclasses) which make up each class.

The second layer of the LVQ network combines subclasses into a single class. The columns of $\mathbf{W}^2$ represent subclasses, and the rows represent classes. $\mathbf{W}^2$ has a single 1 in each column, with the other elements set to zero. The row in which the 1 occurs indicates which class the appropriate subclass belongs to.

$$(w^2_{k,\,i} = 1) \Rightarrow \text{subclass } i \text{ is a part of class } k$$

# Example

$$\mathbf{w}^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

- Subclasses 1, 3 and 4 belong to class 1.
- Subclass 2 belongs to class 2.
- Subclasses 5 and 6 belong to class 3.

A single-layer competitive network can create convex classification regions. The second layer of the LVQ network can combine the convex regions to create more complex categories.

# LVQ Learning

LVQ learning combines competive learning with supervision. It requires a training set of examples of proper network behavior.

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

If the input pattern is classified correctly, then move the winning weight toward the input vector according to the Kohonen rule.

$$_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)) \qquad a_{k^*}^2 = t_{k^*} = 1$$

If the input pattern is classified incorrectly, then move the winning weight away from the input vector.

$$_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)) \qquad a_{k^*}^2 = 1 \neq t_{k^*} = 0$$

# Example

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\0\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix}0\\1\end{bmatrix}\right\} \quad \left\{\mathbf{p}_4 = \begin{bmatrix}0\\0\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}0\\1\end{bmatrix}\right\}$$

$$\mathbf{W}^1(0) = \begin{bmatrix}({}_1\mathbf{w}^1)^T\\({}_2\mathbf{w}^1)^T\\({}_3\mathbf{w}^1)^T\\({}_4\mathbf{w}^1)^T\end{bmatrix} = \begin{bmatrix}0.25 & 0.75\\0.75 & 0.75\\1 & 0.25\\0.5 & 0.25\end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix}1 & 1 & 0 & 0\\0 & 0 & 1 & 1\end{bmatrix}$$

# First Iteration

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1) = \mathbf{compet}\left(\begin{bmatrix} -\|{}_1\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|{}_2\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|{}_3\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|{}_4\mathbf{w}^1 - \mathbf{p}_1\| \end{bmatrix}\right)$$

$$\mathbf{a}^1 = \mathbf{compet}\left(\begin{bmatrix} -\|[0.25\ 0.75]^T - [0\ 1]^T\| \\ -\|[0.75\ 0.75]^T - [0\ 1]^T\| \\ -\|[1.00\ 0.25]^T - [0\ 1]^T\| \\ -\|[0.50\ 0.25]^T - [0\ 1]^T\| \end{bmatrix}\right) = \mathbf{compet}\left(\begin{bmatrix} -0.354 \\ -0.791 \\ -1.25 \\ -0.901 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

# Second Layer

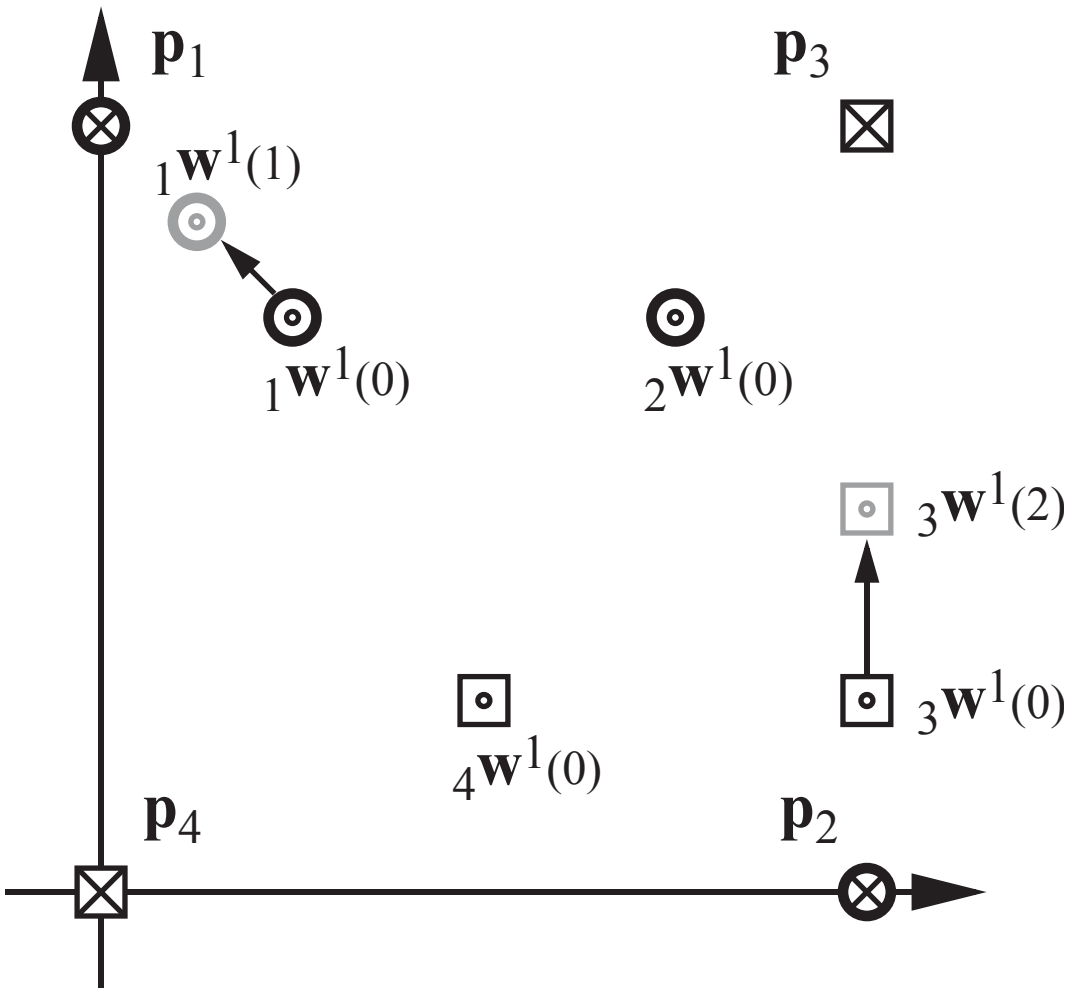
$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

This is the correct class, therefore the weight vector is moved toward the input vector.

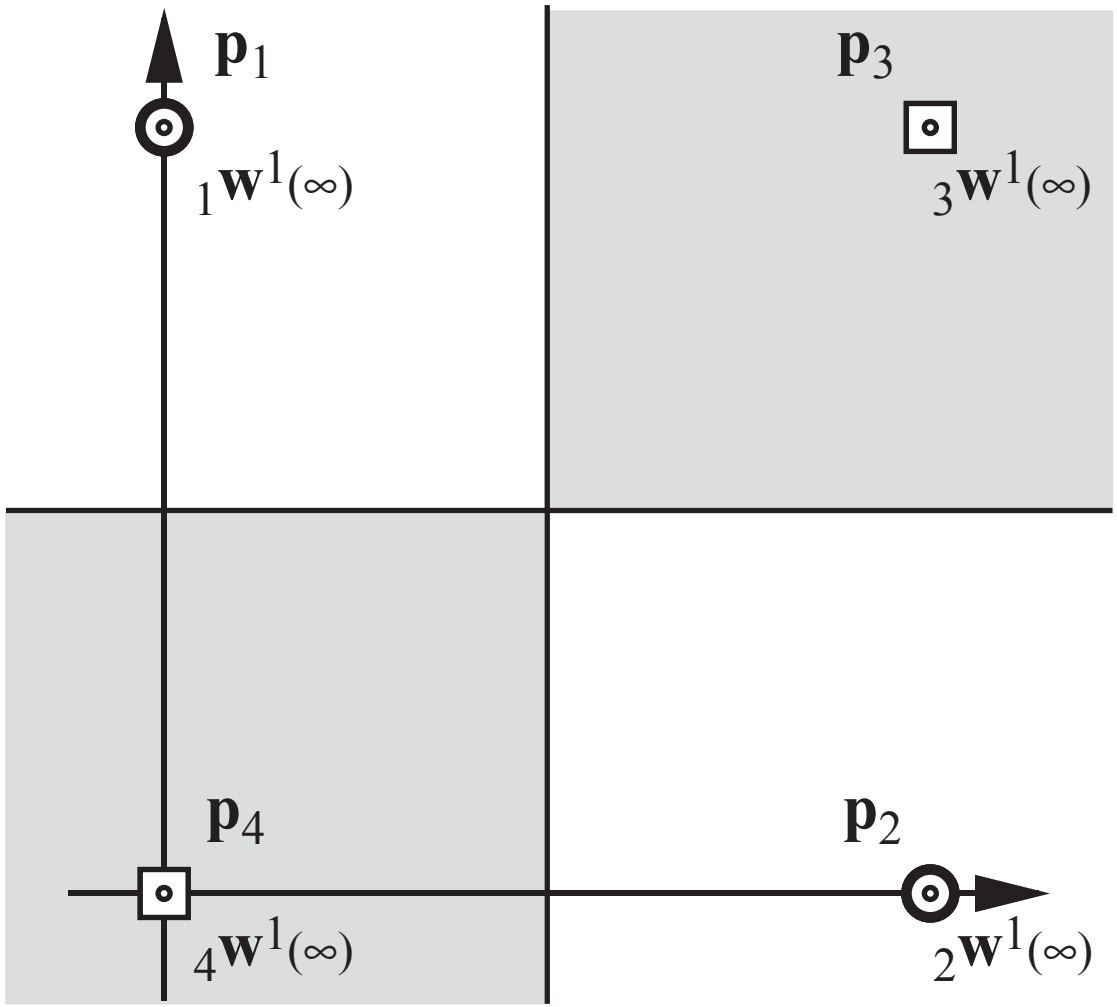
$$_1\mathbf{w}^1(1) = {}_1\mathbf{w}^1(0) + \alpha(\mathbf{p}_1 - {}_1\mathbf{w}^1(0))$$

$$_1\mathbf{w}^1(1) = \begin{bmatrix} 0.25 \\ 0.75 \end{bmatrix} + 0.5\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.25 \\ 0.75 \end{bmatrix}\right) = \begin{bmatrix} 0.125 \\ 0.875 \end{bmatrix}$$

# Figure


$\mathbf{p}_1$
$\mathbf{p}_3$
${}_1\mathbf{w}^1(1)$
${}_1\mathbf{w}^1(0)$
${}_2\mathbf{w}^1(0)$
${}_3\mathbf{w}^1(2)$
${}_3\mathbf{w}^1(0)$
${}_4\mathbf{w}^1(0)$
$\mathbf{p}_4$
$\mathbf{p}_2$

# Final Decision Regions



# LVQ2

If the winning neuron in the hidden layer incorrectly classifies the current input, we move its weight vector away from the input vector, as before. However, we also adjust the weights of the closest neuron to the input vector that does classify it properly. The weights for this second neuron should be moved toward the input vector.

When the network correctly classifies an input vector, the weights of only one neuron are moved toward the input vector. However, if the input vector is incorrectly classified, the weights of two neurons are updated, one weight vector is moved away from the input vector, and the other one is moved toward the input vector. The resulting algorithm is called **LVQ2**.

# LVQ2 Example