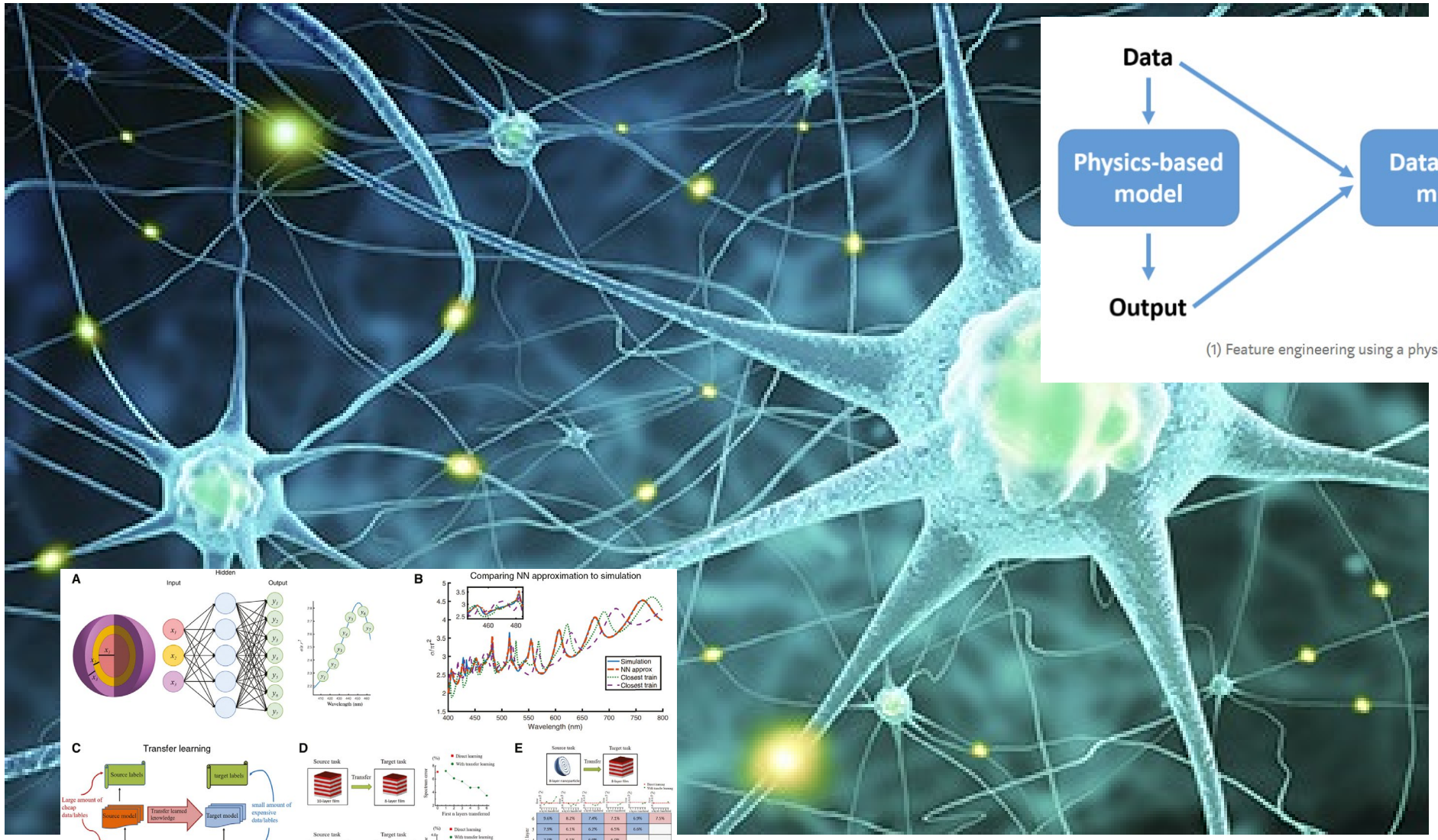
Learning to Compete



https://towardsdatascience.com/physics-guided-neural-networks-pgnns-8fe9dbad9414

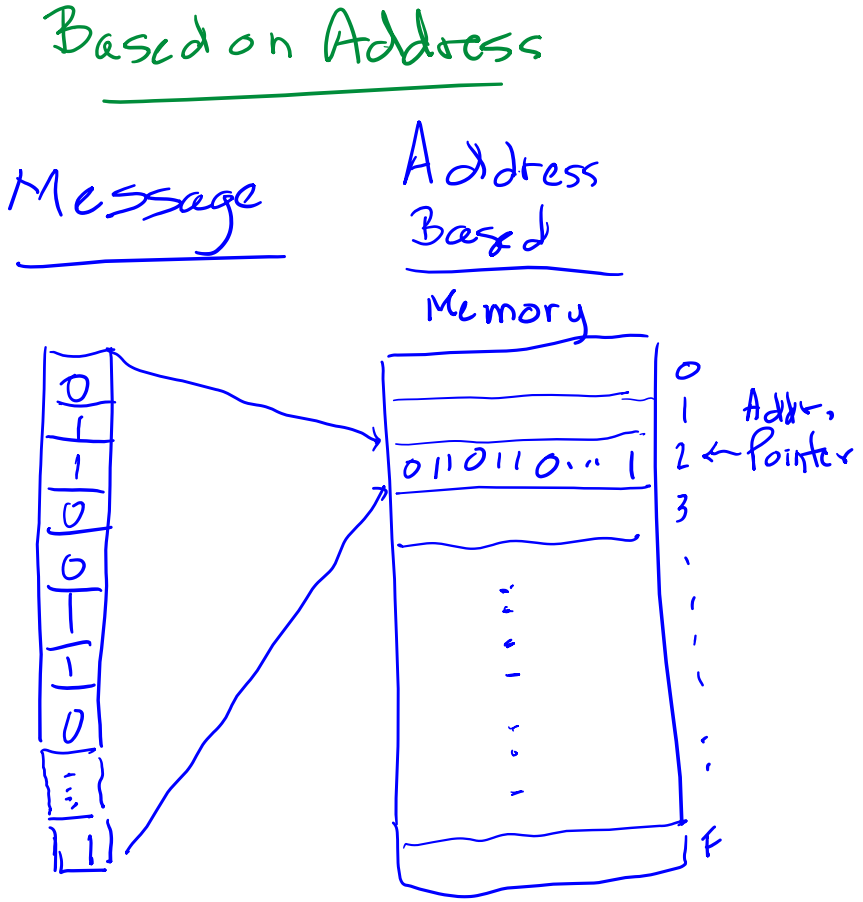
https://www.neuraldump.net/2016/03/introduction-to-neural-networks/

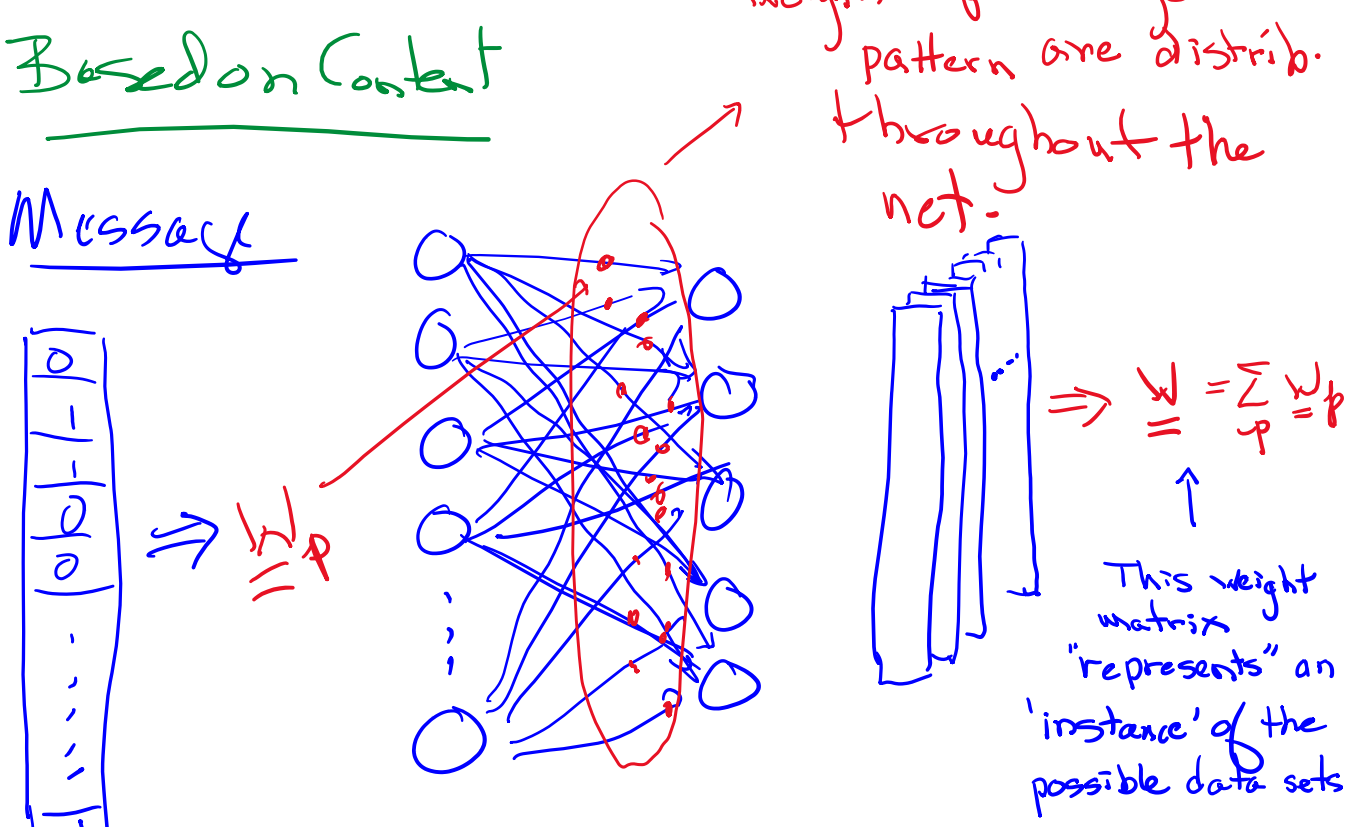
Learning Goals for Lectures 11/12

- **Competing with fixed weights**
- **Self-Organizing Maps (SOM)**
- **Counter-propagation network (CPN)**

First, some questions addressed:

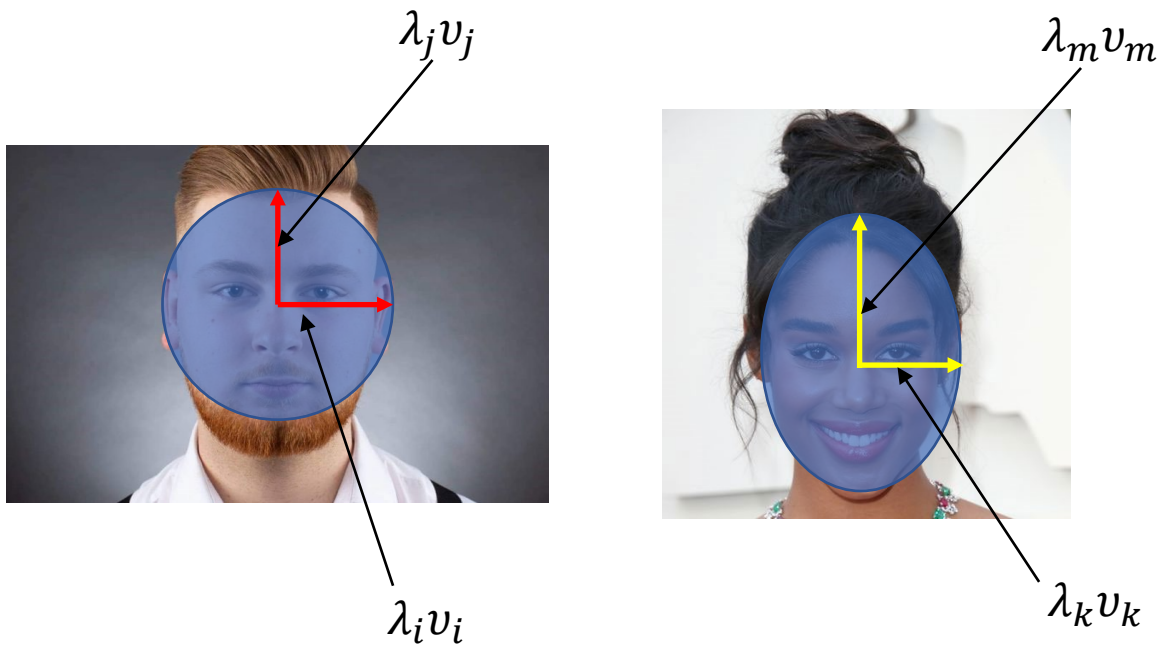
- **In lecture 7 we discussed storing and retrieving based on content rather than storage address and that information is stored in weights throughout the system. You used the term 'instantiation'. In computer science classes that term is used to define creating a new instance of an object. How does this translate to instantiation in NNs we discussed in class?**





First, some questions addressed:

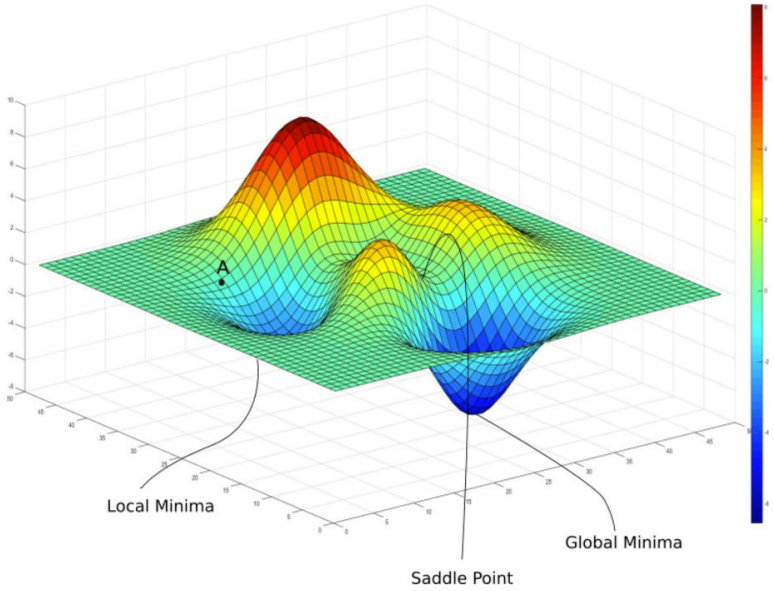
- For the recurrent linear associator I understand why calculating the eigenvector is significant for pattern association, however I do not quite understand how it is applied to a physical example of decision making. Would you please explain how this works in recognizing a pattern as a given input such as telling similar image apart?



First, some questions addressed:

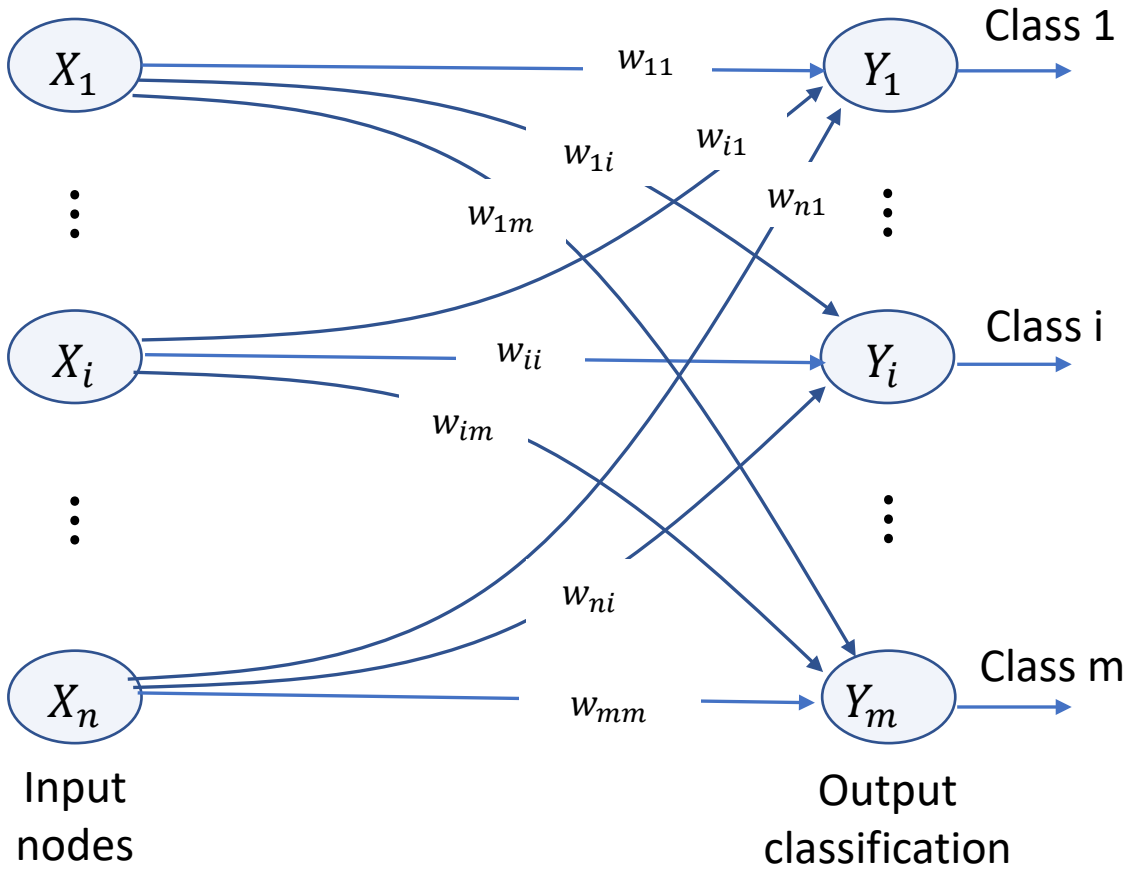
- **How do we make sure the gradient descent algorithm (delta rule) or other rule approaches like Hebb converge to the global minima? How do we know which algorithm is better?**

  - Hebb rule (correlation rule): simple, but limited. For our purposes, at this point in time, we assume that the weights will converge if the data are linearly independent and orthogonal and provide perfect recall. If the data are correlated, the weights will converge, but will be subject to cross-talk. The results "…may still be satisfactory."
  - Gradient descent: The delta rule (activation function = identity function) and extended delta rule (continuously differentiable activation function - not limited to identity function). These two rules are based upon minimizing the squared error = cost function. MSE guarantees local minimum. Global?



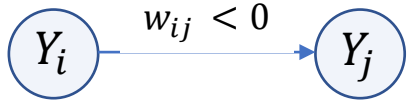
Neural Networks Based on Competition

- **Competition is an important feature for Neural Networks**
  - Competition between neurons has been observed in biological nerve systems
  - Competition is important in solving many problems
- **To classify an input pattern into one of m classes**
  - Simple case: one class node has output 1, all other 0
  - Most cases: often more than one class node will have a non-zero output
    - If these class-nodes competed with each other, maybe only one would eventually win (**winner-takes-all**). The winner represents the computed classification of the input data.

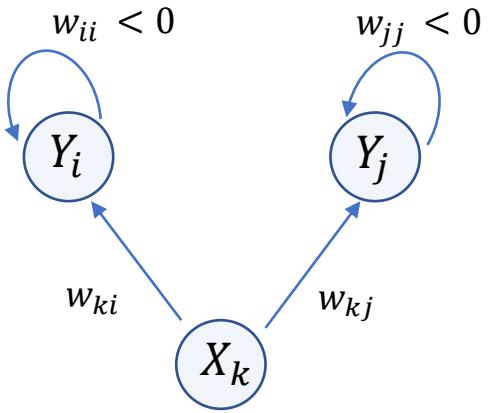


Neural Networks Based on Competition

- **Winner-takes-all (WTA)**
  - Among all competing nodes, only one will win and all others will lose
  - We primarily deal with single winner WTA, but multiple-winners WTA is possible for some particular applications
  - Easiest way to realize WTA – have an external, central arbitrator (algorithm) to decide the winner by comparing the current outputs of the competitors
- **Ways to realize competition in NNs**
  - Lateral inhibition (MAXNET, Mexican hat)
    - Output of each node feeds to the others through inhibitory connections (i.e. negative weights)
  - Resource competition
    - Output of x_k is distributed to y_i and y_j proportional to w_{ki} and w_{kj}, as well as y_i and y_j
    - Self decay

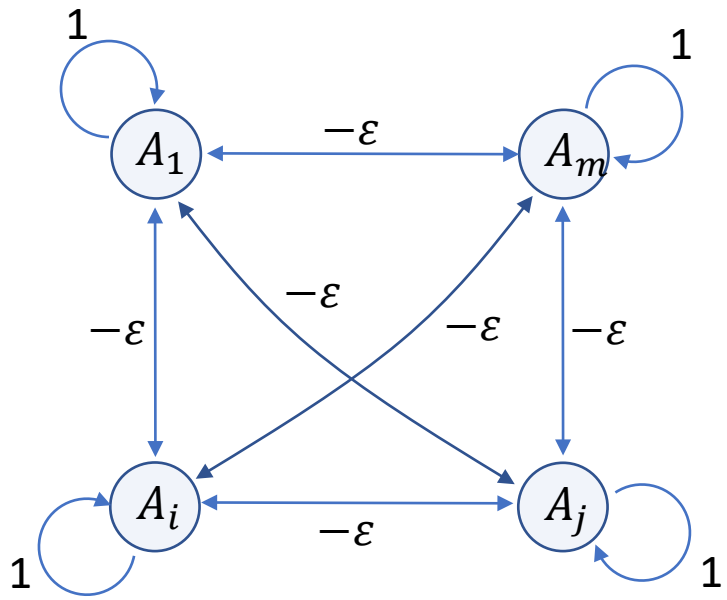


- **Learning methods in competitive networks**
  - Competitive learning
  - Kohonen learning (self-organizing map (SOM))
  - Counter-propagation net
  - Adaptive resonance theory (ART, chapter 5)



Fixed-weight Competitive Nets

- **Maxnet**
  - **Lateral inhibition between competitors**



Competitive Layer
(Winner-take-all)

Weights:

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

Activation function:

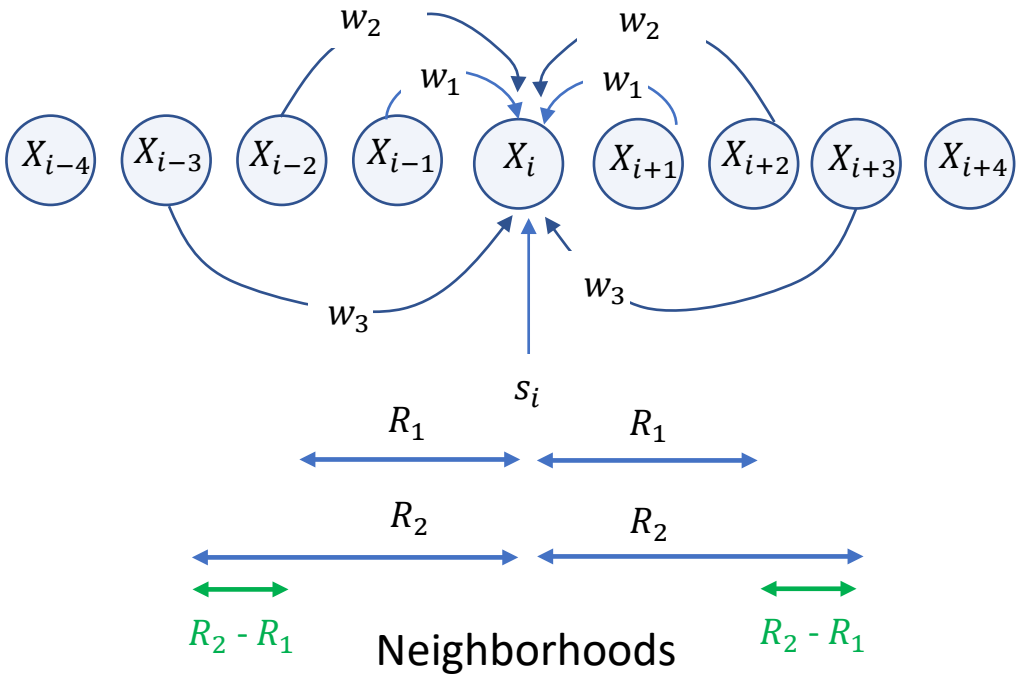
$$f(x) = \begin{cases} x, & if\ x > 0 \\ 0, & otherwise \end{cases}$$

Notes:

- Competition: iterative process until the net converges (at most one node with a positive activation)
- $0 < \varepsilon < {}^{1}/_{m}$, where m is the number of competing nodes
  - ε too small: slow convergence
  - ε too large: may suppress all nodes

Fixed-weight Competitive Nets

- **Mexican Hat**
  - Contrast-enhancing subnet
  - Each neuron is connected with excitatory links to a number of "cooperative neighbors"
  - Each neuron is connected with inhibitory links to a number of "competitive neighbors"



$R_2 =$ Total range of connected weights
$R_1 =$ Range of positive weights (cooperative neighbors)
$R_2 - R_1 =$ *Range of negative weights (competitive neighbors)*

Weights:

$$w_{ij} = \begin{cases} c_1, & if\ dist(i,j) \le R_1 \\ c_2, & if\ R_1 < dist(i,j) \le R_2 \\ 0, & if\ dist(i,j) > R_2 \end{cases}$$

Activation function:

$$f(x) = \begin{cases} max, & if\ x > max \\ x, & if\ 0 \le x \le \max \\ 0, & if\ x < 0 \end{cases}$$

$$x_i = f\left[s_i(t) + \sum_k x_k w_{i+k}(t-1)\right]$$

Fixed-weight Competitive Nets

- **Mexican Hat Algorithm (external signal at initialization only):**
  - Step 0: Initialize t_max, R_1, and R_2 to reasonable values, and $\mathbf{x}_{old} = \mathbf{0}$

$$w_k = C_1 \text{ for } k = 0, \ldots, R_1 \; (C_1 > 0)$$
$$w_k = C_2 \text{ for } k = R_1 + 1, \ldots, R_2 \; (C_2 < 0)$$

  - Step 1: Set $\mathbf{x} = \mathbf{s}$; set $\mathbf{x}_{old}$ (R_2+1:R_2+n) = $\mathbf{x}$(1:n), set iter counter t = 1.
  - Step 2: While t < t_max, do Steps 3 – 7
    - Step 3: Compute activation levels input to each node (i = 1:n):

$$x_i = C_1 \sum_{k=-R_1}^{R_1} x\_old_{i+k} + \{ C_2 \sum_{k=-R_2}^{-R_1-1} x_{old_{i+k}} + C_2 \sum_{k=R_1+1}^{R_2} x_{old_{i+k}} \}$$

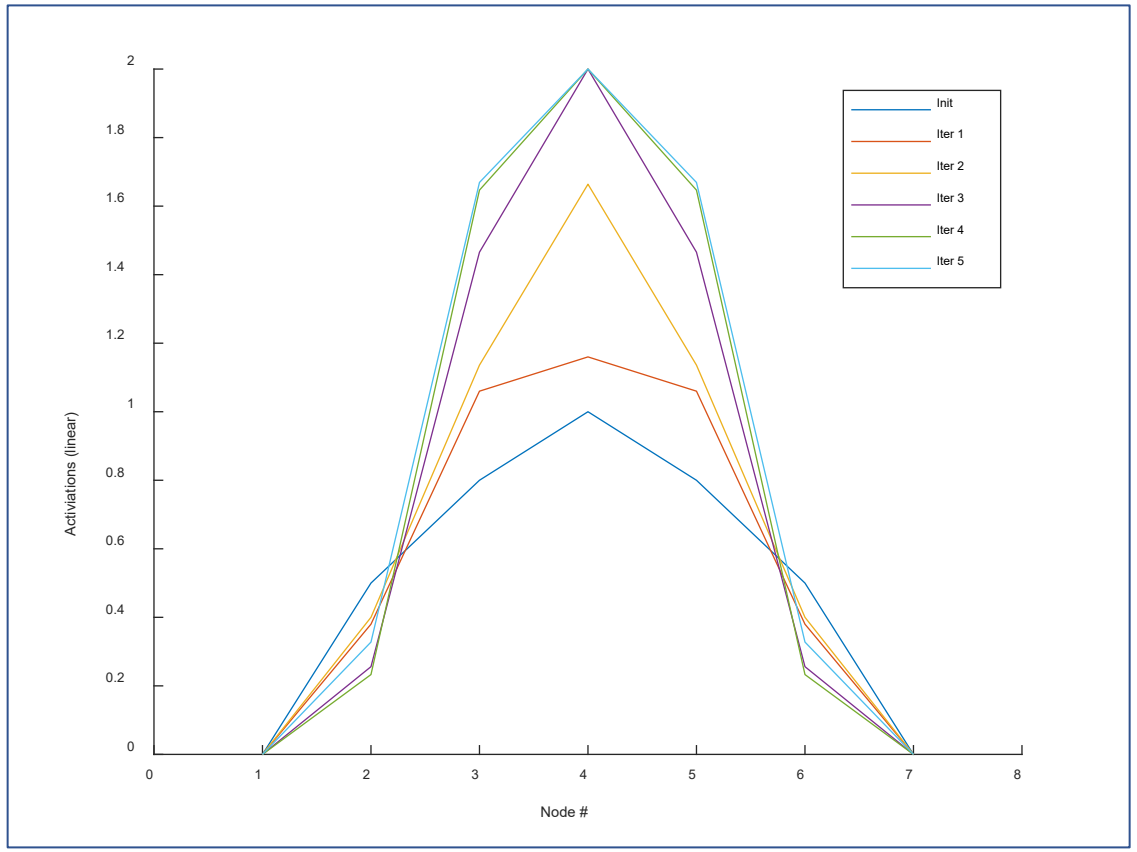
    - Step 4: Apply activation function:

$$x_i = \min\left(x_{max}, \max(0, x_i)\right) \; (i = 1{:}n)$$

    - Step 5: Save current activations: $x_{old}(R_2 + 1{:}R_2 + n) = x(1{:}n)$
    - Step 6: Increment iter counter; t += 1;
    - Step 7: Test stopping condition: If t < t_max then continue; else stop.

Example: Mexican Hat

- Matlab implementation of example (see code in Module “MH_example.m”)



Activations:

```
x =

         0
    0.3278
    1.6691
    2.0000
    1.6691
    0.3278
         0
```

- This type of network can be utilized as a neighborhood function as part of self-organizing maps

Hamming Net

- **Hamming net uses the concept of Hamming distance to implement competition**
  - We know that ***Hamming distance*** (d) = # of bits in disagreement between two (n-dimensional) vectors = n (# of bits) - # of bits in agreement (a)
    - For bipolar vectors:

$$\boldsymbol{x} \cdot \boldsymbol{y} = a - d = (agreement - disagreement)$$

and since n = # of bits in the vector,

$$\mathrm{d} = n - a$$

thus,

$$\boldsymbol{x} \cdot \boldsymbol{y} = 2a - n$$

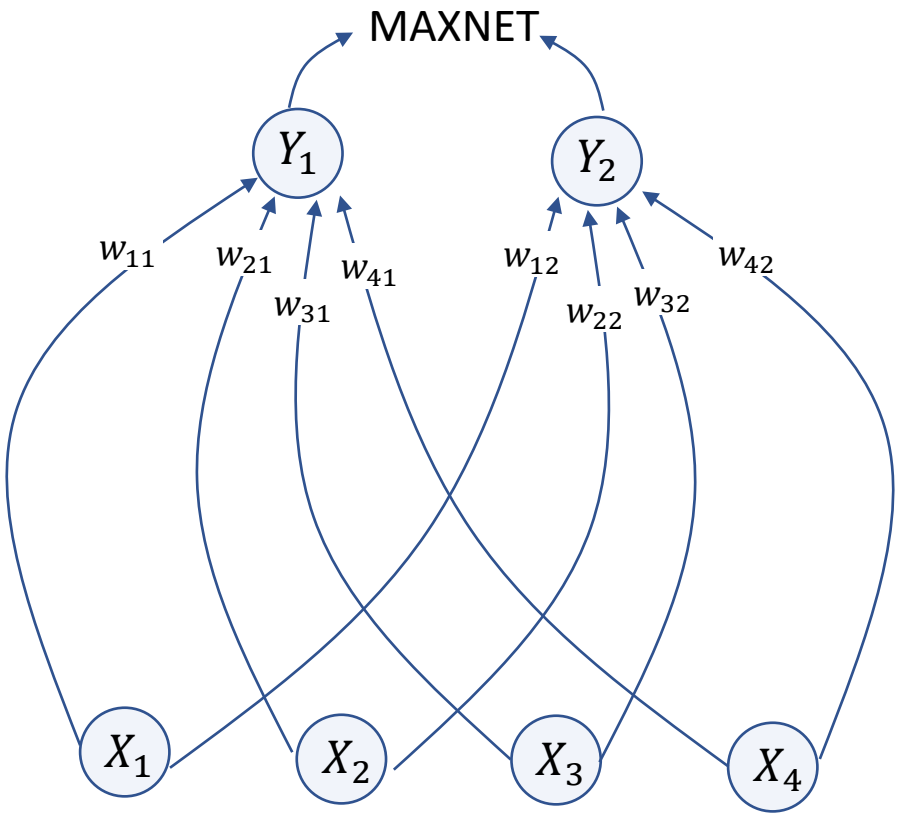
or:

$$a = 0.5\,(\boldsymbol{x} \cdot \boldsymbol{y} + n)$$

- The above suggests setting the weights in the net to be proportional to ½ the exemplar vector and setting the value of the bias to be ½ n.
- In other words, you can use this type of network to compare the input vector to several exemplar vectors and pick the one that is the best match using a competitive network layer (i.e. upper layer).
- Can be used for object classification

Hamming Net Architecture

- **Hamming net uses the concept of Hamming distance to implement a competition**



- Given a set of m bipolar exemplar vectors: **e**(1), **e**(2), …, **e**(m), the network is used to find the closest match to the bipolar input vector
- The net input y_in to Y gives the number of components in which the input vector and the exemplar vector agree.
- **e**(1) is associated with Y_1
- **e**(2) is associated with Y_2, etc.
- n = number of input nodes, number of input elements
- m = number of output nodes, number of exemplar vectors
- Storage capacity = m

Fixed-weight Competitive Nets

- **Hamming Net Algorithm:**
  - Step 0: Store m exemplar vectors by initializing weights

$$w_{ij} = 0.5\, \boldsymbol{e}_i(j);\ i = 1{:}n,;\ j = 1{:}m$$
$$b_j = 0.5\, n,;\ j = 1{:}m$$

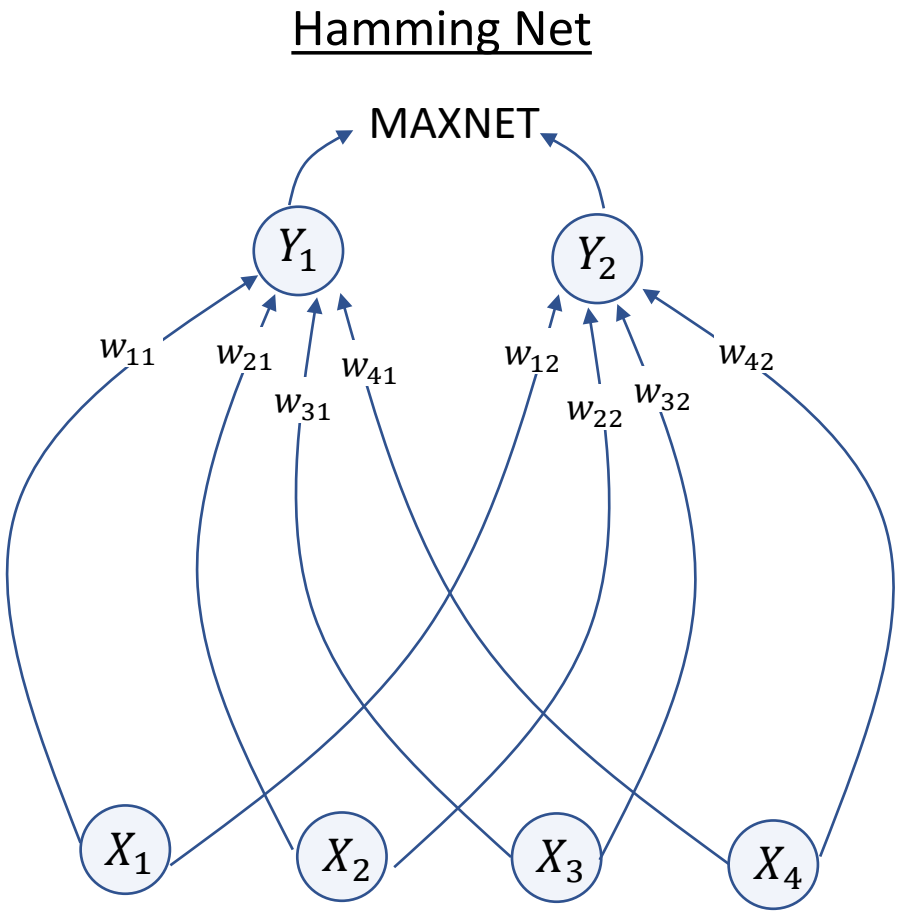
  - Step 1: For each vector, do Steps 2 - 4
    - Step 2: Compute the net input to each unit Y_j

$$y\_in_j = b_j + \sum_i x_i w_{ij};\ j = 1{:}m$$
$$= \frac{1}{2}\left(\boldsymbol{x} \cdot \boldsymbol{e}_j + n\right) = \boldsymbol{a}_j$$

    - Step 3: Initialize activations for MAXNET:

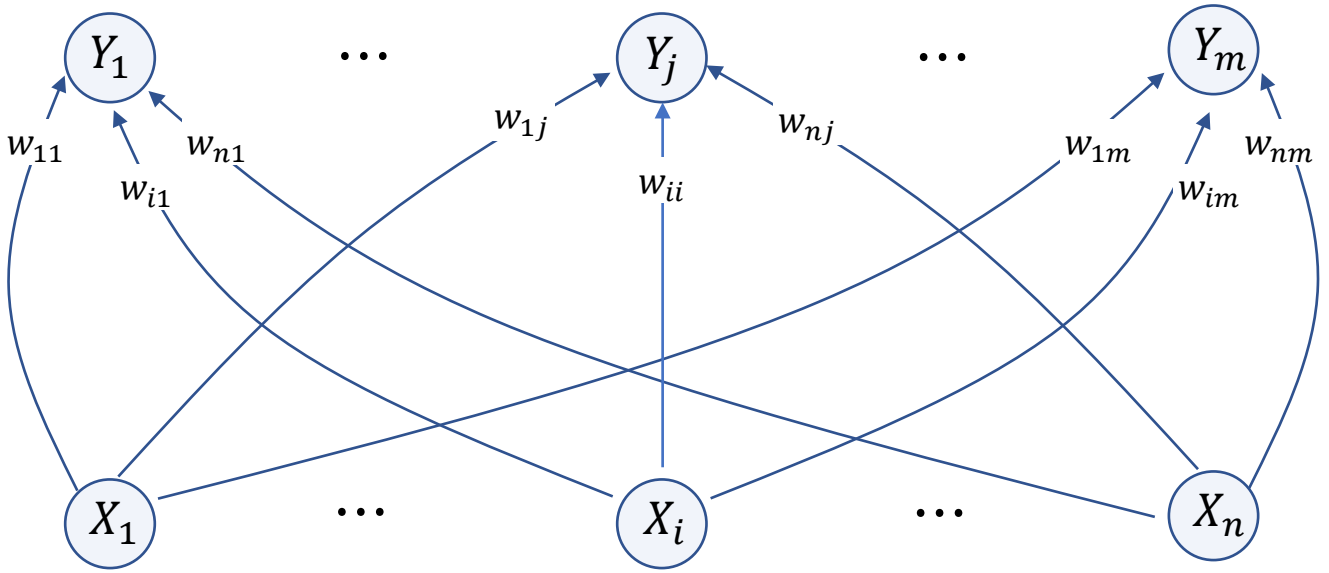
$$y_j(0) = y_{in\,j};\ j = 1{:}m$$

    - Step 4: MAXNET is executed to find the best match



Competitive Networks: Self-Organizing Maps (SOMs)

- **Unsupervised learning**
- **Goal:**
  - Learn to form classes/clusters of exemplars/sample patterns according to similarities
  - Patterns in a cluster would have similar features
  - No prior knowledge as to what features are important for classification or how many classes
- **Architecture:**
  - Output nodes $Y_1,\ldots,Y_m$ can represent m classes
  - Theses output nodes compete with each other (WTA realized either by an external algorithm or by lateral inhibitions as in MAXNET).



Competitive Learning

- **Training**
  - Train the network so that the weight vector $\mathbf{w}_{ij}$ associated with $\mathbf{Y}_j$ becomes representative of the class of input patterns $\mathbf{Y}_j$ is supposed to represent
  - Two phase unsupervised learning

1. Competing phase:

- Apply an input vector **x** randomly chosen from sample set
- Compute the output for all nodes: $\mathbf{y}_j = \mathbf{x}^T\mathbf{w}_{ij}$ (if appropriate)
- Determine the winner (NOTE: winner is not given in training samples so this is unsupervised)

2. Rewarding phase:

- The winner is rewarded by updating its weights (weights associated with all other output nodes are not updated)

- This two-phase procedure is repeated many times with a gradually decreasing learning rate until all the weights are stabilized.

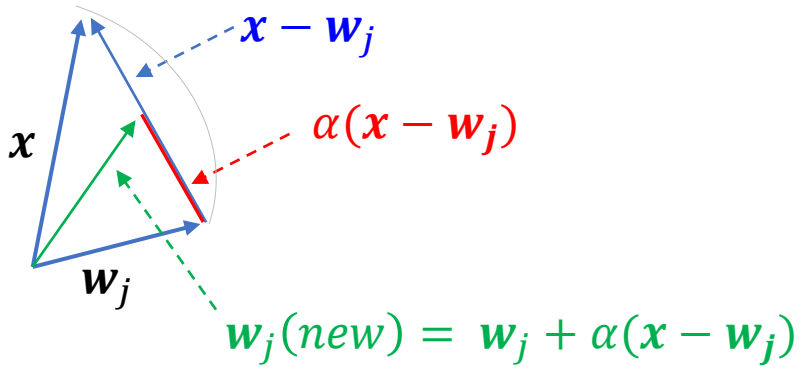
Competitive Learning

- **Two approaches to weight updates:**

$$\boldsymbol{w}_j(new) = \boldsymbol{w}_j(old) + \Delta \boldsymbol{w}_j$$

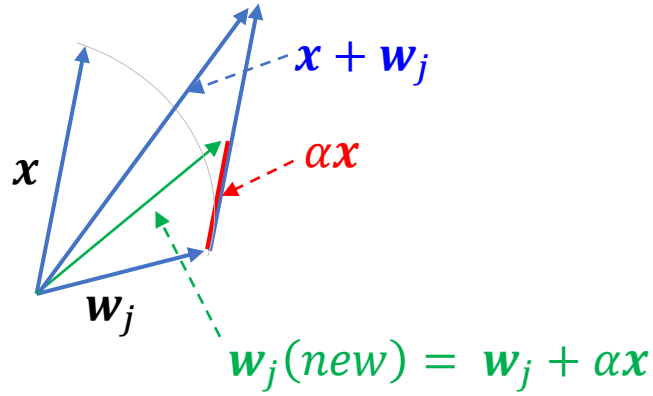
- **Method 1:**

$$\Delta \boldsymbol{w}_j = \alpha(\boldsymbol{x} - \boldsymbol{w}_j)$$



- **Method 2:**

$$\Delta \boldsymbol{w}_j = \alpha \boldsymbol{x}$$

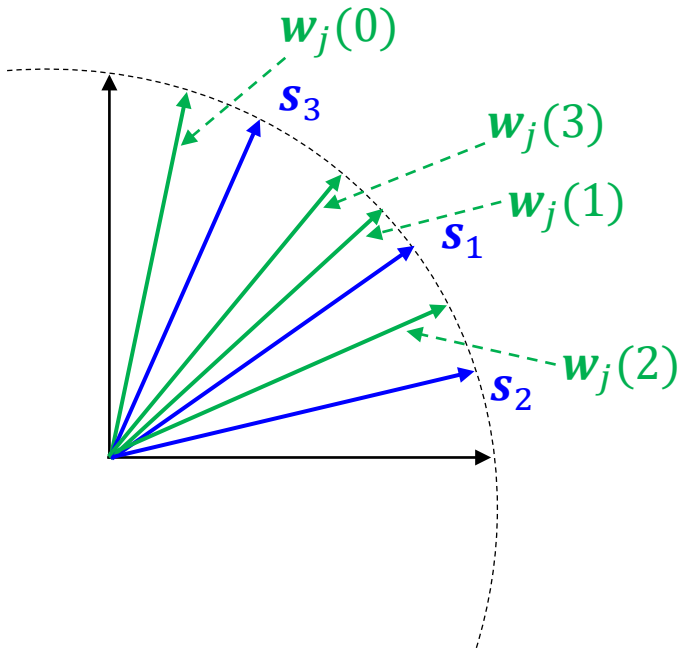


- **For each approach w_j is moved closer to x**
- **Normalize the weight vector to unit length after each update:** $\boldsymbol{w}_j = \dfrac{\boldsymbol{w}_j}{\|\boldsymbol{w}_j\|}$

Competitive Learning

- This process can be interpreted as w_j moving towards the center of a cluster of sample vectors during updates

- Three exemplar vectors: s_1, s_2, s_3
- Initial weight vector $w_j(0)$



Competitive Learning Example (SOMs)

- **A simple example of competitive learning**
  - 4 vectors of dimension n=4 in 2 classes (P = 4, n = 4, m = 2)

$$\boldsymbol{s}_1^T = (1, 1, 0, 0)$$
$$\boldsymbol{s}_2^T = (0, 0, 0, 1)$$
$$\boldsymbol{s}_3^T = (1, 0, 0, 0)$$
$$\boldsymbol{s}_4^T = (0, 0, 1, 1)$$

  - Initialization:

$$\alpha = 0.6 \qquad \boldsymbol{W} = \begin{bmatrix} .2 & .8 \\ .6 & .4 \\ .5 & .7 \\ .9 & .3 \end{bmatrix}$$

  - Train with $\mathbf{s}_1$:

$$D_1 = \|\boldsymbol{s}_1 - \boldsymbol{w}_1\|^2 = (.2-1)^2 + (.6-1)^2 + (.5-0)^2 + (.9-0)^2 = 1.86$$

$$D_2 = \|\boldsymbol{s}_1 - \boldsymbol{w}_2\|^2 = (.8-1)^2 + (.4-1)^2 + (.7-0)^2 + (.3-0)^2 = 0.98 \quad \checkmark \textit{ class 2 wins}$$

$$\boldsymbol{w}_2 = \begin{bmatrix} .8 \\ .4 \\ .7 \\ .3 \end{bmatrix} + 0.6 \left\{ \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} .8 \\ .4 \\ .7 \\ .3 \end{bmatrix} \right\} = \begin{bmatrix} .92 \\ .76 \\ .28 \\ .12 \end{bmatrix} \rightarrow \boldsymbol{W} = \begin{bmatrix} .2 & .92 \\ .6 & .76 \\ .5 & .28 \\ .9 & .12 \end{bmatrix}$$

Competitive Learning Example (SOMs)

- **Training with $\mathbf{s}_2 = (0, 0, 0, 1)^T$, reveals class 1 wins this time**

$$\boldsymbol{W} = \begin{bmatrix} .08 & .92 \\ .24 & .76 \\ .20 & .28 \\ .96 & .12 \end{bmatrix}$$

  - Repeating for $\mathbf{s}_3$ and $\mathbf{s}_4$ (the end of the first iteration through vectors) results in:

$$\boldsymbol{W} = \begin{bmatrix} .032 & .970 \\ .096 & .300 \\ .680 & .110 \\ .980 & .048 \end{bmatrix}$$

  - Reduce learning rate:

$$\alpha(new) = 0.5 \ * \alpha(old) = 0.5 * 0.6 = 0.30$$

  - Repeat training. After 100 iterations, the weight matrix becomes:

$$\boldsymbol{W} = \begin{bmatrix} 6.7e-17 & 1.000000 \\ 2.0e-16 & .4900000 \\ .5100000 & 2.3e-16 \\ 1.000000 & 1.0e-16 \end{bmatrix} \rightarrow \boldsymbol{W} = \begin{bmatrix} 0.0 & 1.0 \\ 0.0 & 0.5 \\ 0.5 & 0.0 \\ 1.0 & 0.0 \end{bmatrix}$$

$\boldsymbol{w}_1 = average\ of\ vectors\ in\ cluster\ 1$
$\boldsymbol{w}_2 = average\ of\ vectors\ in\ cluster\ 2$

$$\boldsymbol{W}^T\boldsymbol{s}_1 = \begin{bmatrix} 0.0 \\ 1.5 \end{bmatrix} = C_2$$

$$\boldsymbol{W}^T\boldsymbol{s}_2 = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} = C_1$$

$$\boldsymbol{W}^T\boldsymbol{s}_3 = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix} = C_2$$

$$\boldsymbol{W}^T\boldsymbol{s}_4 = \begin{bmatrix} 1.5 \\ 0.0 \end{bmatrix} = C_1$$

Competitive Learning Example (SOMs)

- **Observations**

➢ In an ideal situation, when the learning is completed, each weight vector is close to the centroid of a group or cluster of sample input vectors

➢ To stabilize the calculation of w, the learning rate (α) may be reduced slowly towards zero during the learning process (we demonstrated this in the examples).

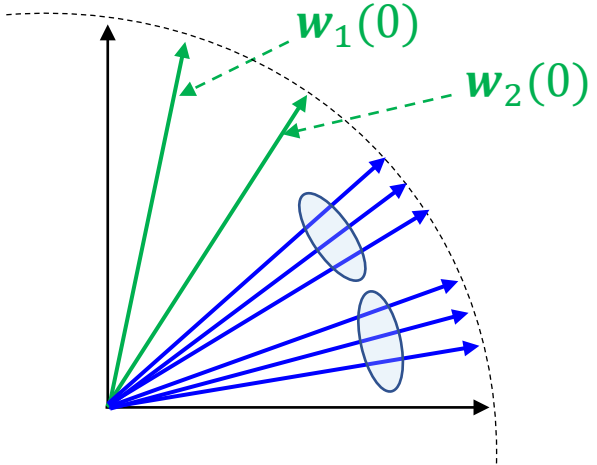
➢ Number of output nodes:

- ✓ If too few: several clusters may be combined into one class
- ✓ If too many: over classification (splitting vectors between clusters)
- ✓ Adaptive Resonance Theory (ART) model allows dynamic addition/removal of output nodes

➢ Initial w:

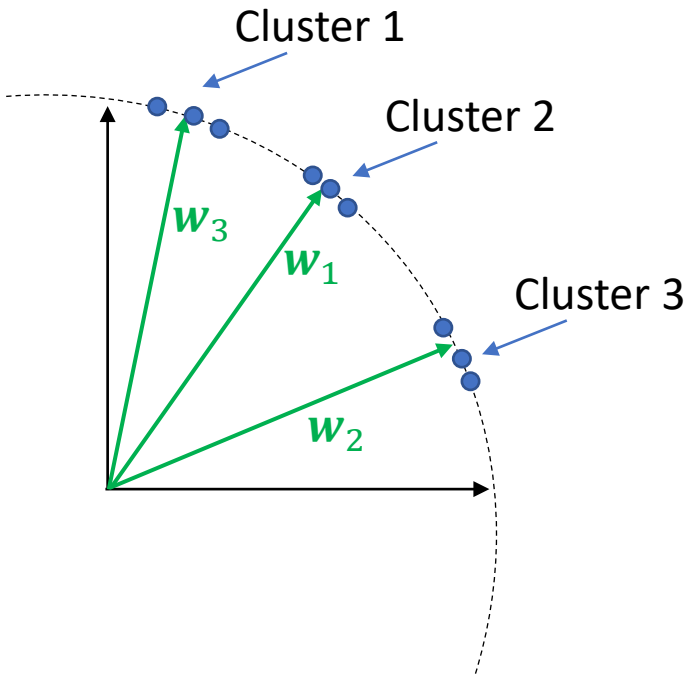
- ✓ Training samples known to be in distinct classes (if the information is known ahead of time)
- ✓ Random (however poor choices may cause anomalies)

- $\mathbf{w}_1$ will always win no matter which cluster the sample vector belongs to
- $\mathbf{w}_2$ is stuck and will not participate in the learning
- Attempts to unstick weight updates is to temp. shut off nodes which have a very high winning rate



Kohonen Self-Organizing Maps (SOMs)

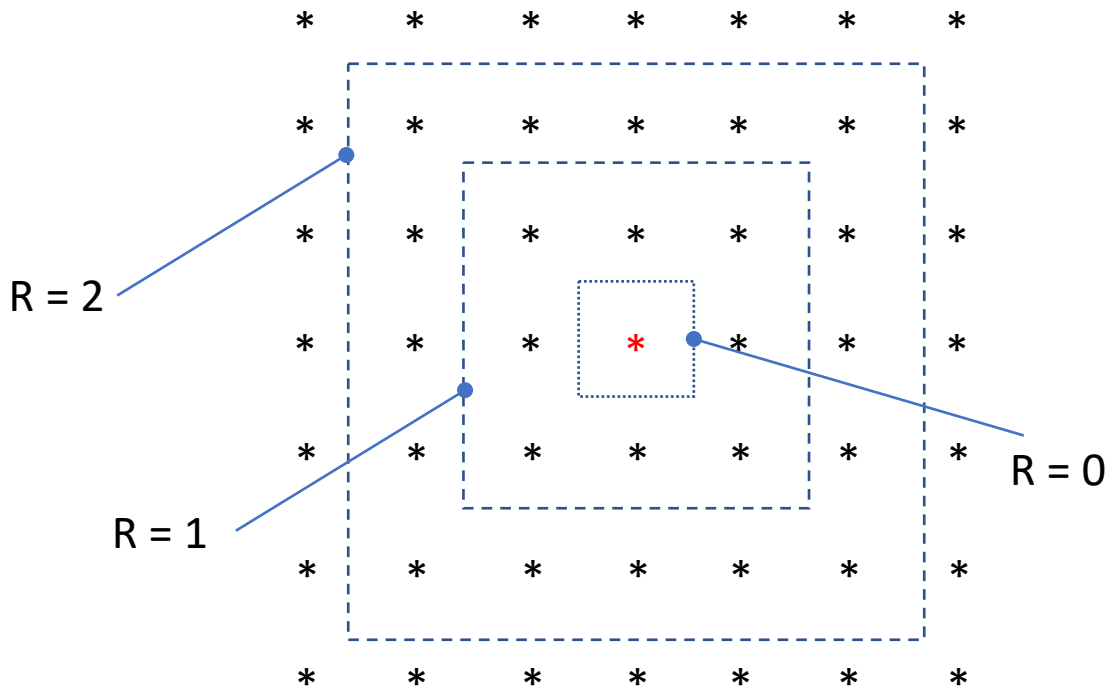
- **Competitive learning (Kohonen 1982) is a special case of SOMs**
- **In competitive learning:**
  - The network is trained to organize the input vector space into subspaces/classes/clusters
  - Each output node corresponds to one class
  - The output nodes are not ordered, i.e. mapping is random

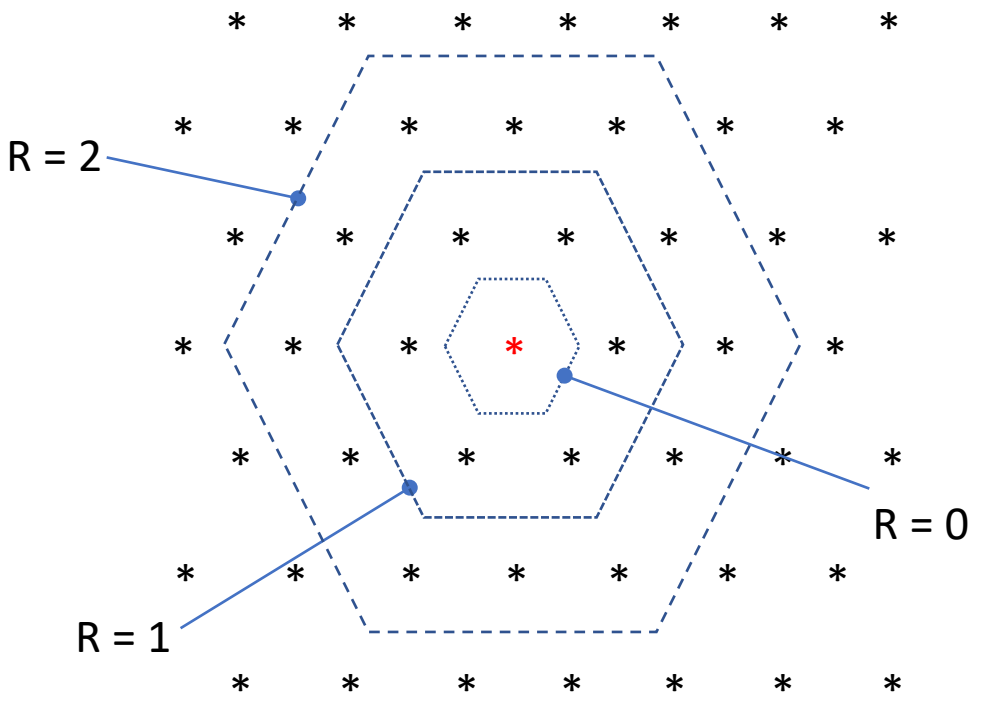


- The topological order of the three clusters is 1, 2, 3
- The order of their maps to the output nodes is: 3, 1, 2
- The mapping does not preserve the topological order of the training vectors

Kohonen Self-Organizing Maps (SOMs)

- **Topological mapping**
  - A mapping that preserves the neighborhood relationships between input vectors (topology preserving or feature preserving)
  - If $\mathbf{x}_1$ and $\mathbf{x}_2$ are two neighboring input vectors (by some distance metric), their corresponding winning output nodes (class mappings) i and j must also be close to each other in some fashion
  - One dimensional: line or ring, node i has neighbors i $\pm$ R
  - Two dimensional:
    - Rectangular grid – node (i,j) has neighbors [(i, j $\pm$ R), (i $\pm$ R, j)]
    - Hexagonal grid: 6 neighbors





Kohonen Self-Organizing Maps (SOMs)

- **Biological motivation**
  - The brain is organized in many places in such a way that different sensory inputs are represented by topologically ordered computational maps.
  - Mapping two dimensional continuous inputs from sensor organs (eyes, ears, skin, etc.) to two dimensional discrete outputs in the nervous system
    - Retinotopic map: from eye (retina) to visual cortex
    - Tonotopic map: from the ear to the auditory cortex
  - These maps preserve the topological orders of the input – inputs such as tactile, visual, and acoustic are mapped to different areas of the cerebral cortex in a topologically ordered fashion.
  - Thus, the neurons transform input signals into a *place-coded probability distribution that* represents the computed values of parameters by sites of maximum relative activity within the map.
  - Biological evidence suggests that the connections in these maps are not entirely pre-programmed or pre-wired prior to birth. Learning must occur after the person is born to create the necessary connections for appropriate topological mapping

Self-Organizing Maps (SOMs)

- **General SOM Architecture – Kohonen does not use this activation calculation**
  - Output layer:
    - Each node represents a class of inputs
    - Node activation:

$$y_j = \boldsymbol{x}^T \boldsymbol{w}_j = \sum_i w_{ij} x_i$$

    - Neighborhood relation is defined over these nodes.
    - Each node cooperates with all of its neighbors within distance R and competes with all other nodes.
    - Cooperation and competition of these nodes can be realized by the Mexican Hat model

$$R = 0: all\ nodes\ are\ competitors\ (no\ cooperation) \rightarrow random\ map$$

$$R > 0: implies\ topology\ preservation\ mapping$$

Kohonen Self-Organizing Maps (SOMs)

- **Kohonen SOM Learning**
  1. Initialize $\mathbf{w}_j$ for all of the output nodes and α to a small value
  2. For a randomly selected input sample/exemplar **x** determine the winning output node *J* either:

$$D(j) = \|\boldsymbol{x} - \boldsymbol{w}_J\|^2 = \sum_i (w_{ij} - x_i)^2 \; is \; a \; minimum$$

  3. For all output nodes (j) with $|J - j| \leq R$, update the weight

$$w_{ij}(new) = w_{ij}(old) + \alpha(\boldsymbol{x} - w_{ij})$$

  4. Periodically reduce α and R slowly.
  5. Repeat 2 – 4 until network is stabilized

*NB: The connection weights do not multiply the input signal values unless the dot product measure of similarity is being used.

Observations

1. Initial weights: small random values from $[-\varepsilon, \varepsilon]$

2. Reduction of α:

- Linear: $\alpha(t + \Delta t) = \alpha(t) - \Delta\alpha$
- Geometric: $\alpha(t + \Delta t) = \beta\alpha(t)$
- Δt may be ≥ 1

3. Reduction of R: $R(t + \Delta t) = R(t) - 1, while\ R(t) > 0$

- Reduction should be much slower than the reduction of α
- R can be a constant throughout the learning process

4. Effect of learning

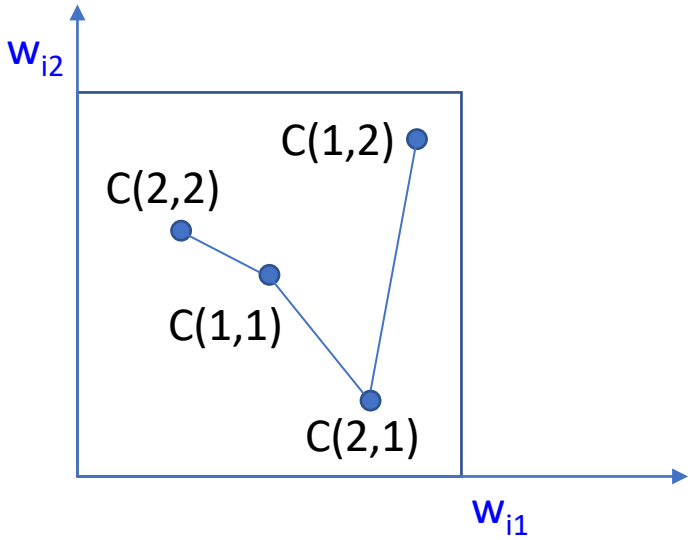
- For each input **x** the weight vector of all J's close neighbors is pulled closer to **x** (i.e. within radius of R)

5. Eventually if w_j becomes closer to $w_{j\pm1}$ the classes they represent are also similar

6. Large initial values of R may be needed for certain circumstances

Topology of Kohonen Map

- **Input vector: (x,y) pairs**
- **Output vector: line or 2D grid using some neighborhood radius**
- **You can represent the topology of the output nodes by using the weight vector (2D) as the (i,j) grid position for each node and connect each node with a line.**
- **Connect neighboring output nodes example:**
  - Output nodes: (1,1) (1,2) (2,1) (2,2)
  - Weight vectors: (.5,. 5) (.7, .2) (.9, .9) (.3, .6)



Classical Optimization Problem: Traveling Salesman Problem (TSP)

- Given a set of cities to visit where each city is only visited once and then returning to the original city, find the shortest path (optimization)
- Each city is represented as its coordinate position in a 2D grid = input vector
- Output nodes, C_j, form a 1D SOM, each node corresponds to a city
- Initially, $C_1, \ldots, C_n$, are initialized with random weight vectors
- During the learning process, a winner node C_j given an input (x,y) of city s_i, has its associated weights w_j moved towards (x,y) and that of its neighbors within the radius R.
- As a result, C_{j-1} *and* C_{j+1} will subsequently be more likely to win with input vectors which are similar to (x,y) (cities close to i).
- At the end of the process, if a node j represents a city i, it would end up having its neighbors represent cities similar to city i.
- This can be viewed as a concurrent greedy algorithm

Classical Optimization Problem: Traveling Salesman Problem (TSP)

- **Inputs:**
  - city A = (.41, .43), city B = (.25, .15), …, city J = (.62, .28)
- **Outputs:**
  - node 1 = rand(x,y), node 2 = rand(x,y), …, node 10 = rand(x,y)

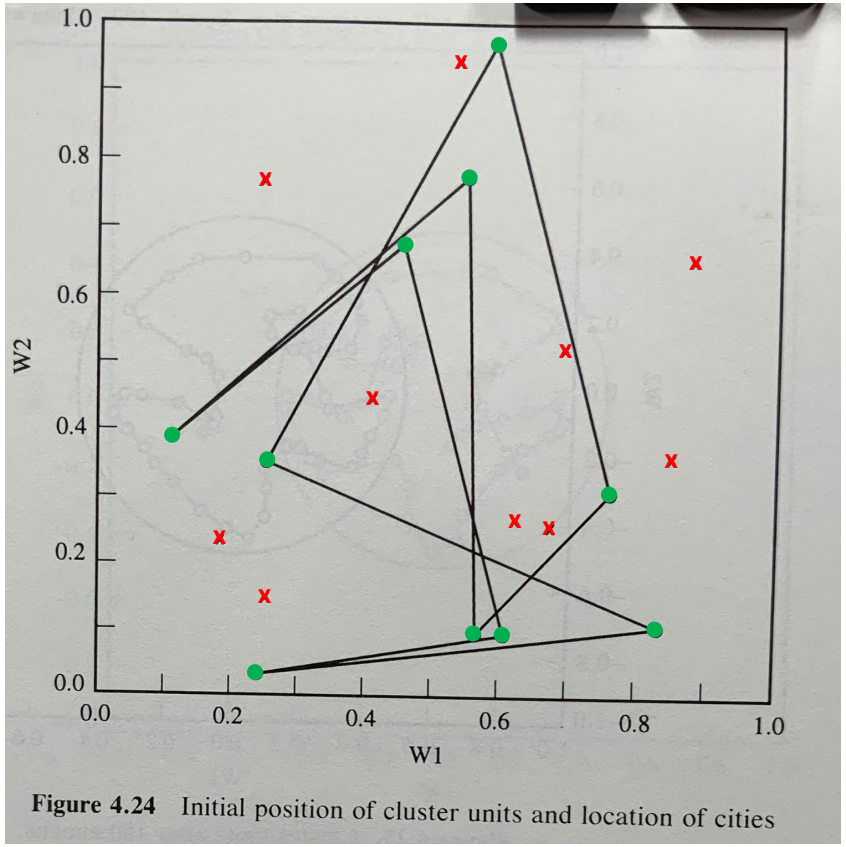
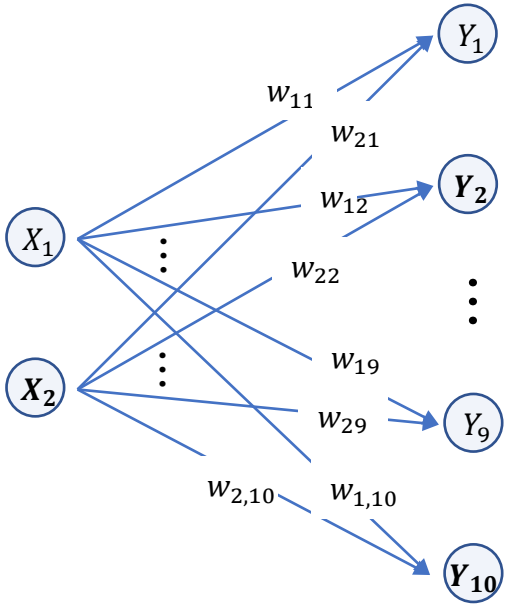
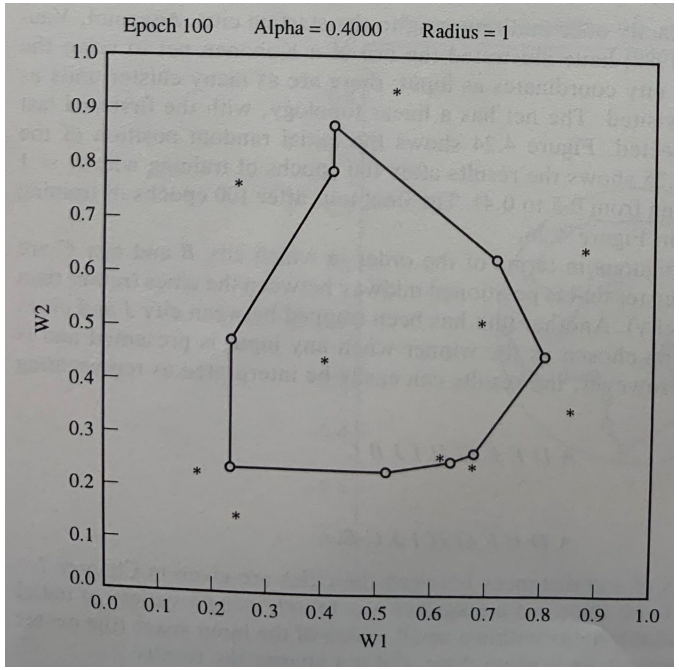


Figure 4.24 Initial position of cluster units and location of cities

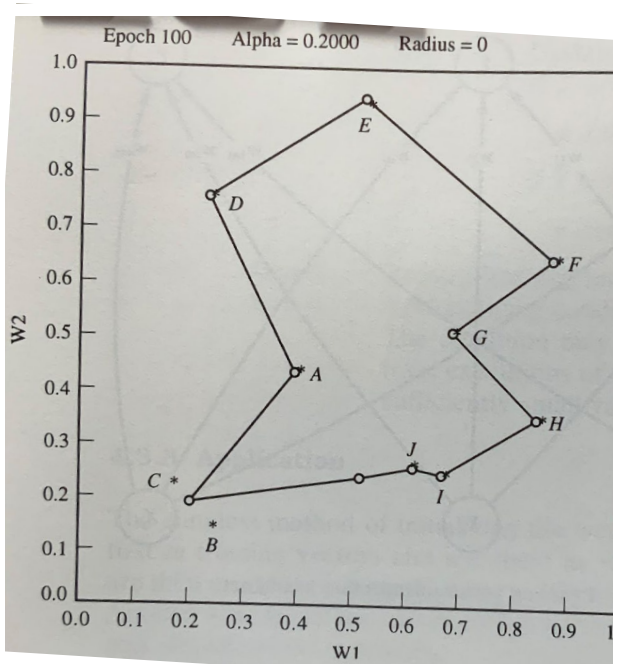


Classical Optimization Problem: Traveling Salesman Problem (TSP)

- **Kohonen SOM with $\alpha = 0.4, R = 1,\ 100\ iters$:**



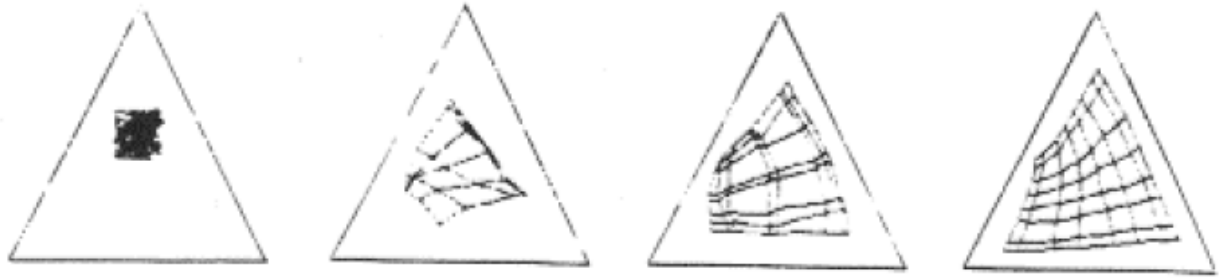
- **Kohonen SOM with $\alpha = 0.2, R = 0,\ 100\ iters$:**

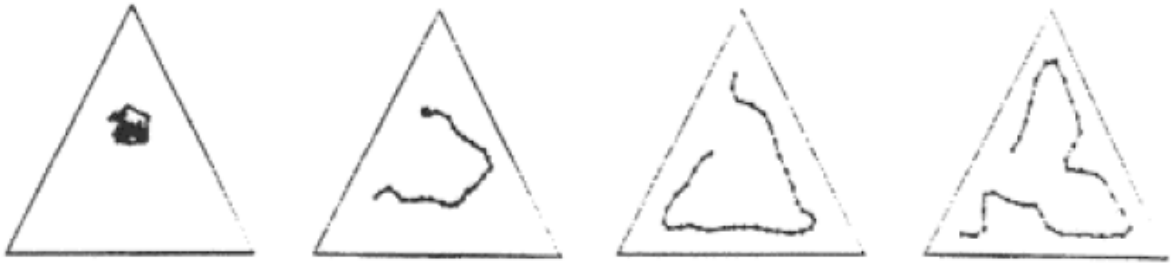


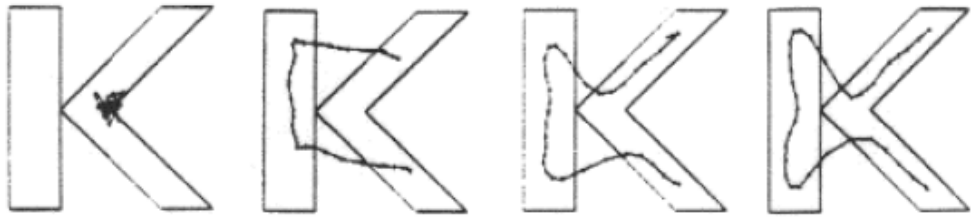
Two possible solutions:

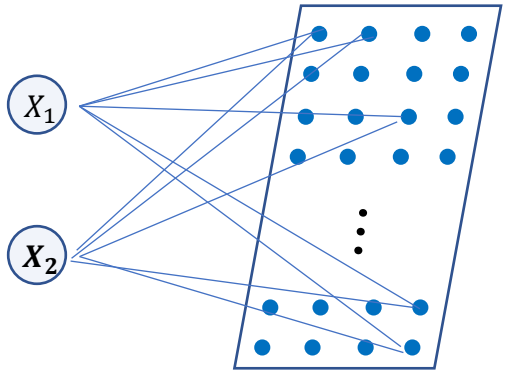
- *A D F G H I J B C*
- *A D F G H I J C B*

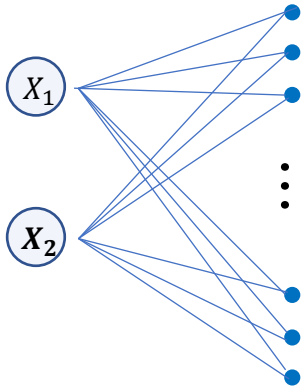
Kohonen SOM: Other examples









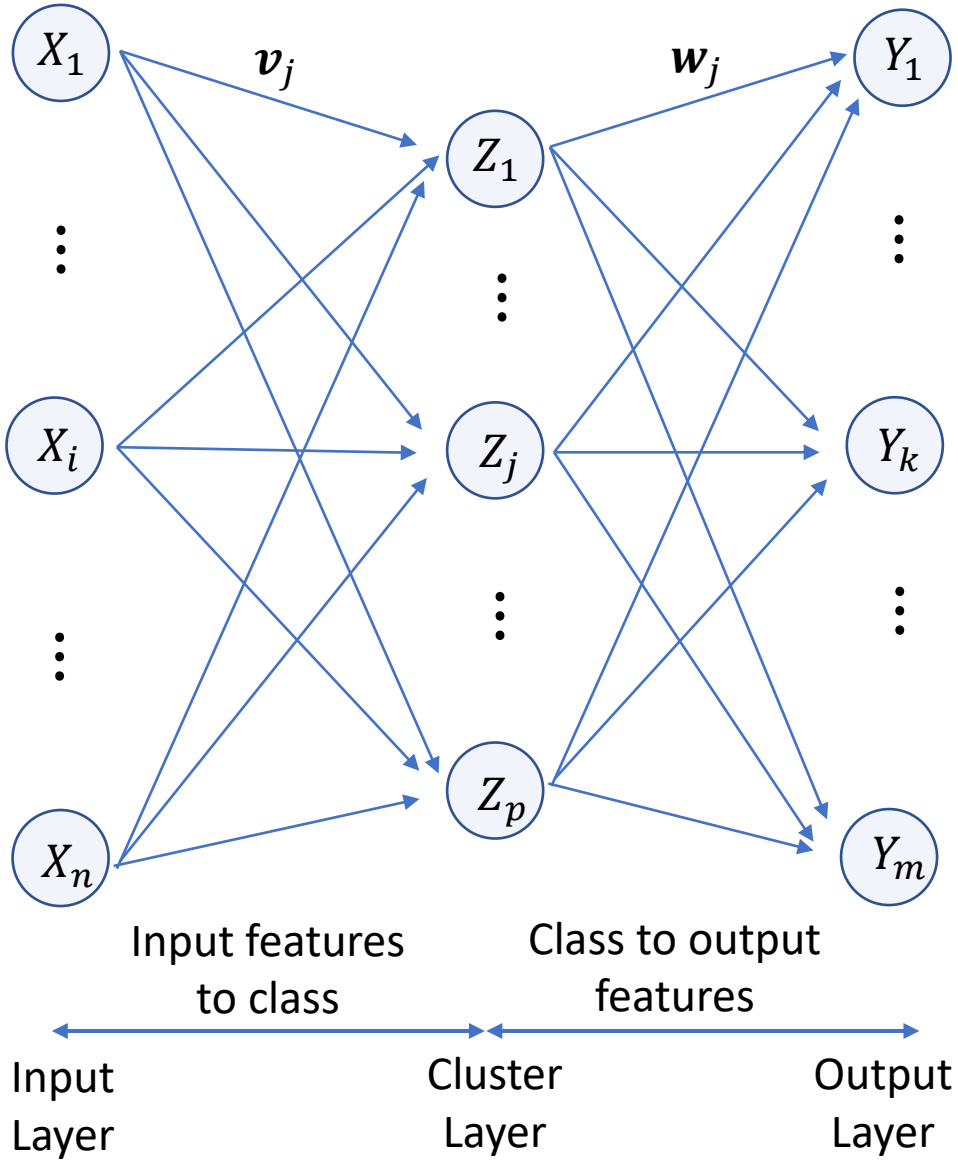


Counter-Propagation Network (CPN)

- **Multi-layer networks using a combo of input, clustering, and output layers**
- **Counterpropagation networks are used for: data compression, function approximation, pattern association**
- **Technique uses an adaptive look-up table approach to handle large data sets**
- **Purpose: fast and coarse approximation of vector mapping $y = \phi(x)$**
  - Not to map any x to is $\phi(x)$ with a given precision
  - Input vectors are divided into clusters/classes
  - Each cluster of x's has one output y which is ideally the average of $\phi(x)$ for all x in that class

Counter-Propagation Network (CPN)

- **General architecture for simple Forward Only CPN**



Forward-only CPN

- **Learning occurs in two phases**
  - Training sample **x**:**y** where $\boldsymbol{y} = \phi(\boldsymbol{x})$ is the precise mapping
  - Phase 1: Unsupervised

    $\mathbf{v}_j$ is trained by competitive learning to become the representative vector of a cluster of input vectors **x** (use sample x only)

    1. For a chosen **x**, feedforward to determine the winning Z_j
    2. $v_{ij}(new) = v_{ij}(old) + \alpha(x_i - v_{ij}(old))$
    3. Reduce α, then repeat Steps 1 and 2 until the stopping condition is met

  - Phase 2: Supervised

    $\mathbf{w}_j$ is trained by delta rule to be an average output of $\phi(\boldsymbol{x})$ where **x** is an input vector that causes Z_j to win (use both **x** and **y**).

    1. For a chosen **x**, feedforward to determine the winning $\mathbf{Z}_j$
    2. $v_{ij}(new) = v_{ij}(old) + \alpha(x_i - v_{ij}(old))$ (optional)
    3. $w_{ik}(new) = w_{jk}(old) + \alpha(y_k - w_{jk}(old))$
    4. Reduce α, then repeat Steps 1 and 2 until the stopping condition is met

Forward-only CPN

- **After training, the network works like a look-up of a math table**
  - For any input ***x***, find a region where ***x*** falls (represented by the winning *Z* node)
  - Use the region as the index to look-up the table for the function value.
  - CPN works in multi-dimensional input space
  - More cluster nodes (*Z*), more accurate mapping

Full CPN

- If both $\boldsymbol{y} = \phi(x)$ and its inverse function $\boldsymbol{x} = \phi^{-1}(\boldsymbol{y})$ exist, we can establish bi-directional approximation
- Provides an efficient method of representing a large number of vector pairs **x:y** by adaptively constructing a look-up table. It produces an approx. $x^*: y^*$ based on input of an x-vector (with no y), y (with no x) or with both x:y available.
- Two pairs of weight matrices:

  $\boldsymbol{V}(\boldsymbol{x}\ to\ Z)$ and $\boldsymbol{U}(Z\ to\ \boldsymbol{y})$ for approximate mapping of $\boldsymbol{x}\ to\ \boldsymbol{y} = \phi(\boldsymbol{x})$

  $\boldsymbol{W}(\boldsymbol{y}\ to\ Z)$ and $\boldsymbol{T}(Z\ to\ \boldsymbol{x})$ for approximate mapping of $\boldsymbol{y}\ to\ \boldsymbol{x}^* = \phi(\boldsymbol{x})$

- When **x**:**y** is applied ($\boldsymbol{x}$ to X and $\boldsymbol{y}$ to Y), they can jointly determine the winner J or separately Z_{Jx}, Z_{Jy}

Full CPN

Architecture of Counter Propagation Network.