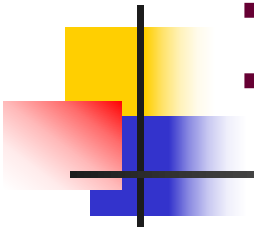
UNSUPERVISED LEARNING

INT301 Bio-computation, Week 10, 2025



Introduction

- So far, we mainly studied neural networks that learn from their environment in a **supervised manner**
- Neural networks can also learn in an unsupervised manner as well
- Unsupervised learning **discovers** significant features or patterns in the input data ***through general rules*** that operate locally
- Unsupervised learning networks typically consist of feed-forward connections and elements to facilitate 'local' learning

Hebbian Learning

- A simple principle was proposed by Hebb in 1949 in the context of biological neurons
- Hebbian principle

When a neuron repeatedly excites another neuron, then the threshold of the latter neuron is decreased, or the ***synaptic weight between the neurons is increased***, in effect increasing the likelihood of the second neuron to excite

- Hebbian learning rule $\Delta w_{ji} = \eta y_j x_i$
  - There is no desired or target signal required in the Hebb rule, hence it is ***unsupervised learning***

Hebbian Learning

- Consider the update of a single weight **w** (x and y are the pre- and post-synaptic activities)

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta x(n) y(n)$$

- For a linear activation function

$$\mathbf{w}(n+1) = \mathbf{w}(n)[1 + \eta x(n) x^{T}(n)]$$

  - **Weights increase without bounds.** If initial weight is negative, then it will increase in the negative. If it is positive, then it will increase in the positive range

- Hebbian learning is intrinsically unstable, unlike error-correction learning with BP algorithm

Hebbian Learning

- Consider a single linear neuron with *p* inputs

$$y = \mathbf{w}^T\mathbf{x} = \mathbf{x}^T\mathbf{w}$$

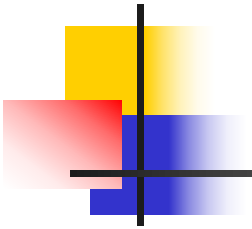
and

$$\Delta\mathbf{w} = \eta[x_1 y \;\; x_2 y \;\; \ldots \;\; x_p y]^T$$

- The dot product can be written as

$$y = |\mathbf{w}||\mathbf{x}| \cos(\alpha)$$

  - α = angle between vectors **x** and **w**
    - If α approaches 0 (**x** and **w** are 'close'), y is large
    - If α approaches 90 (**x** and **w** are 'far'), y is zero



Hebbian Learning

- A network trained with Hebbian learning creates a similarity measure (inner product) in its input space according to the information contained in the weights
  - The weights capture (memorizes) the information in the data during training
- During operation, when the weights are fixed, a large output y signifies that the present input is "similar" to the inputs **x** that created the weights during training

Oja's Rule

- The simple Hebbian rule causes the weights to increase (or decrease) without bounds
- The weights need to be normalized to one as

$$w_{ji}(n+1) = \frac{w_{ji}(n) + \eta x_i(n) y_j(n)}{\sqrt{\sum_i [w_{ji}(n) + \eta x_i(n) y_j(n)]^2}}$$

- Oja proves that, for small $\eta << 1$, the above normalization can be approximated as:

$$w_{ji}(n+1) = w_{ji}(n) + \eta y_j(n)[x_i(n) - y_j(n) w_{ji}(n)]$$

  - This is Oja's rule, or the normalized Hebbian rule
  - It involves a '**forgetting term**' that prevents the weights from growing without bounds

Oja's Rule

- It has been proved that using Lyapunov function analysis, Oja's rule converges asymptotically, unlike Hebbian rule which is unstable

- Oja's rule creates a ***principal component*** in the input space as the weight vector when applied to a single neuron

- How can we find other components in the input space with significant variance?

Dimensionality Reduction

- One approach to deal with high dimensional data is by reducing their dimensionality.
- Project high dimensional data onto a lower dimensional sub-space using linear or non-linear transformations.

$$\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{bmatrix} \xrightarrow[?]{\text{Reduce dimensionality}} \mathrm{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_k \end{bmatrix} \quad (\mathrm{K}<<\mathrm{N})$$

Oja's Rule

- How to find the projection onto orthogonal direction?
  - **Deflation method**: subtract the principal component from the input
- Oja's rule can be extended to extract multiple principal components

Oja's Rule

- Deflation procedure is adopted to compute the other eigenvectors
  - Assume that the first component is already obtained, compute the projection of the first eigenvector on the input

$$y = w_1^T x$$

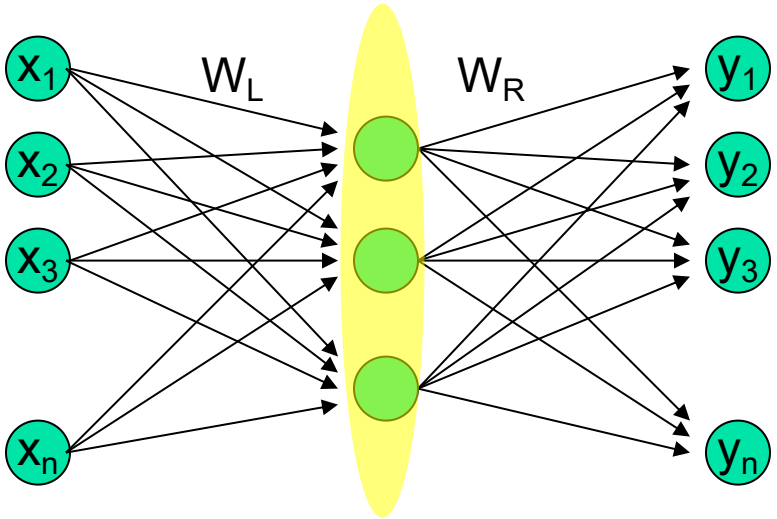
  - Generate the modified input as

$$\hat{x} = x - w_1 y = x - w_1 w_1^T x$$

  - Repeat Oja's rule on the modified data

PCA in Neural Networks

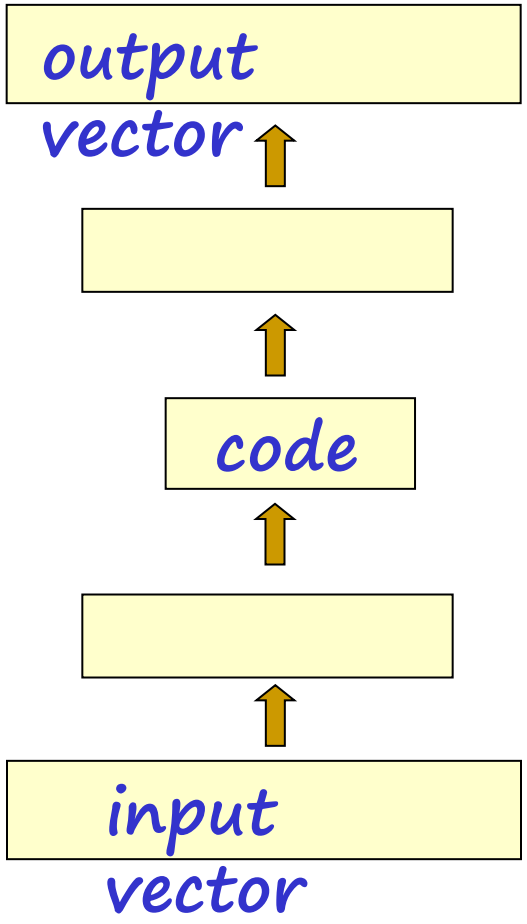
- **Multi-layer networks** with bottleneck layer



- Train using auto-associative output: e = x – y
- W_L spans the subspace of the first m principal eigenvectors.

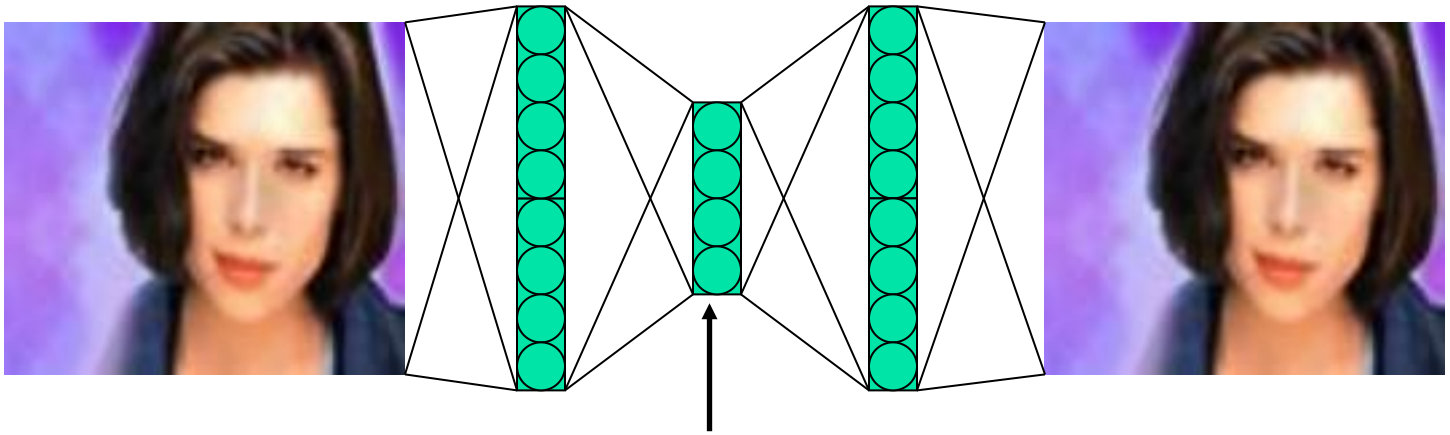
PCA in Neural Networks

- Using back-propagation for unsupervised learning
- Try to make the output be the same as the input in a network with a central bottleneck.
  - The activities of the hidden units in the bottleneck form an **efficient code**.
    - **The bottleneck does not have room for redundant features.**
  - Good for extracting independent features

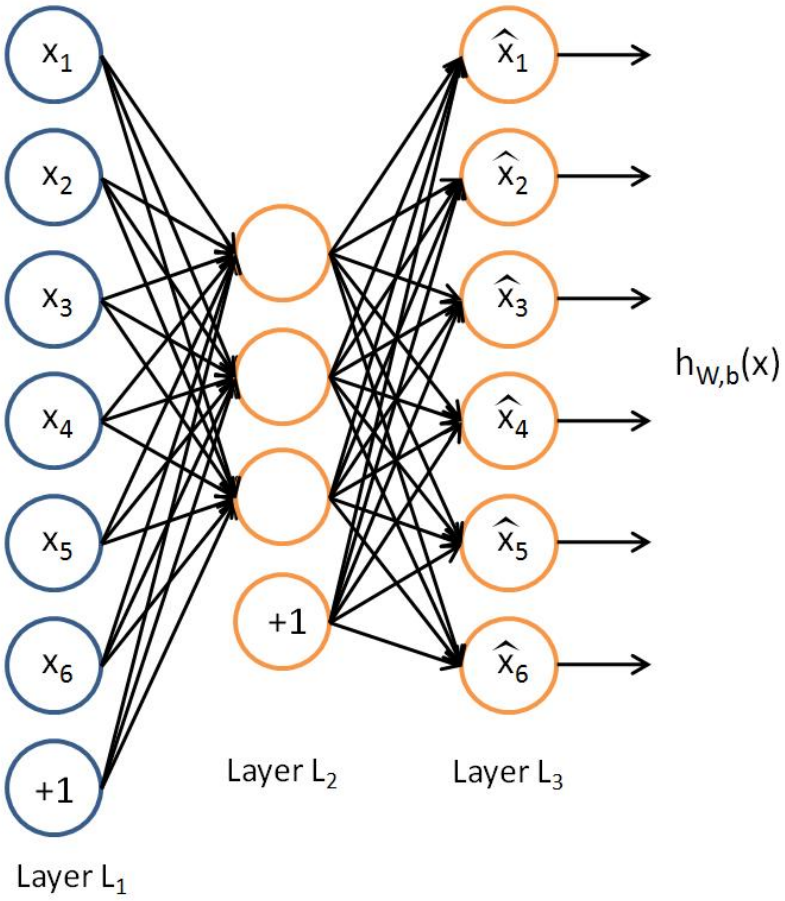


PCA in Neural Networks

- **Back-propagation algorithm can be used for unsupervised learning to discover significant features that characterise input patterns.**
- This can be achieved by learning the identity mapping, passing the data through a bottleneck: **auto-encoders**



Auto-encoders (Rumelhart 86)



An Autoencoder is a feedforward neural network that learns to predict the input itself in the output.

$$y^{(i)} = x^{(i)}$$

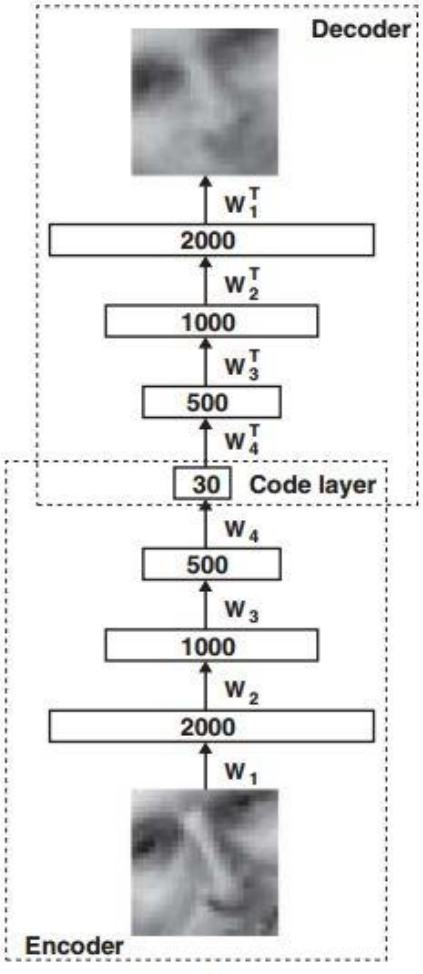
- The input-to-hidden part corresponds to an encoder
- The hidden-to-output part corresponds to a decoder.

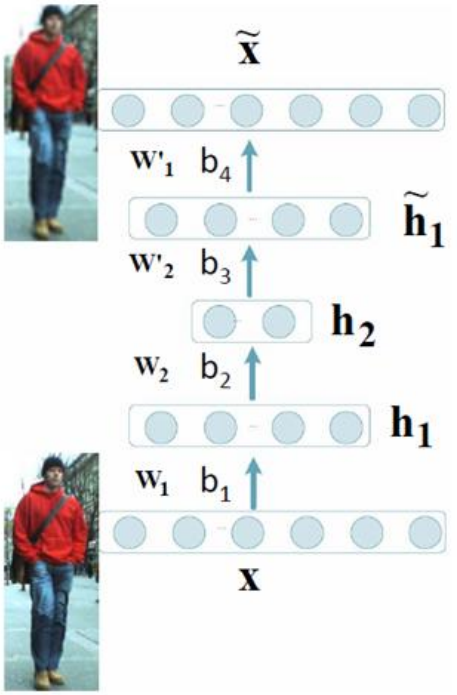
Auto-encoders (Rumelhart 86)

- To reproduce the input patterns at output layer
- Number of hidden layers and the sizes of the layers can vary
- Auto-encoder tends to find a data description which resembles the **PCA**; while small number of neurons in the bottleneck layer of the diabolo network acts as an ***information compressor*** **(code)**

Deep Auto-encoder (Hinton 06)

- A deep auto-encoder is constructed by extending the encoder and decoder of autoencoder with multiple hidden layers.





Encoding: $h_1 = \sigma(W_1 x + b_1)$

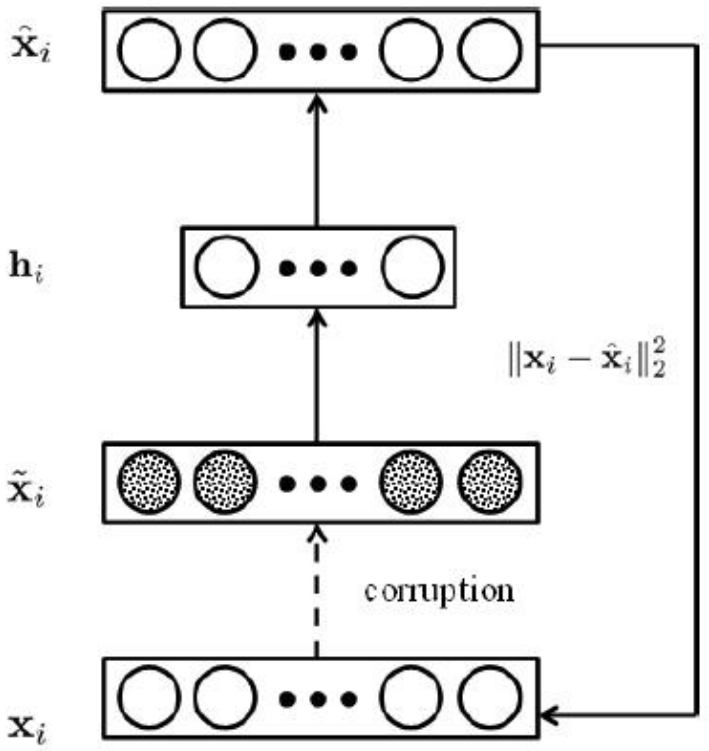
$h_2 = \sigma(W_2 h_1 + b_2)$

Decoding: $\tilde{h}_1 = \sigma(W'_2 h_2 + b_3)$

$\tilde{x} = \sigma(W'_1 h_1 + b_4)$

Denoising Auto-encoder (Vincent 08)

- By adding stochastic noise, it can force auto-encoder to learn more robust features.
- The loss function



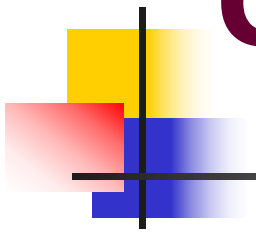
$$\mathbf{h}^{(\ell)} = \sigma(\mathbf{W}_1 \tilde{\mathbf{x}}^{(\ell)} + \mathbf{b}_1)$$
$$\hat{\mathbf{x}}^{(\ell)} = \sigma(\mathbf{W}_2 \mathbf{h}^{(\ell)} + \mathbf{b}_2)$$

$$\min_{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}_1,\mathbf{b}_2} \sum_{\ell} \|\mathbf{x}^{(\ell)} - \hat{\mathbf{x}}^{(\ell)}\|_2^2 + \lambda\left(\|\mathbf{W}_1\|_F^2 + \|\mathbf{W}_2\|_F^2\right)$$

Auto-encoders Network

- The network tries to reproduce the input in the output, inducing an short encoding in the hidden layer.
- This encoding retains the maximum amount of information about the input in a smaller dimensional space such that the input can be reconstructed.
- Auto-encoder networks can be used for dimensionality reduction, compression, etc.

Clustering Revisit



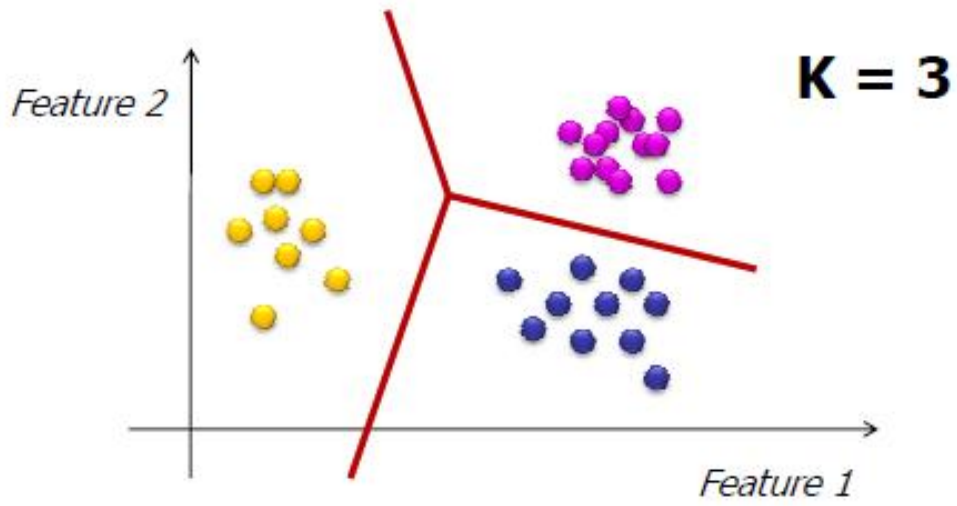
- Clustering analysis is the process of grouping a set of data into clusters
- A cluster is a collection of data points where each observation is
  - Similar to other observations in the same cluster;
  - Dissimilar to observations in other clusters
- Cluster analysis organizes data by abstracting the **underlying structure** either as a grouping of individuals, or as a hierarchy of groups.

Clustering Revisit

- These groupings are based on measured or perceived **similarities** among the patterns.
- Clustering is **unsupervised**. There are no category labels and other information about the source of data.
- Typical applications
  - As a stand-alone tool to get insight into data distribution
  - As a preprocessing step for other algorithms

K-means algorithm Revisit

- The k-means algorithm partitions the data into k *mutually exclusive clusters*



K-means algorithm Revisit

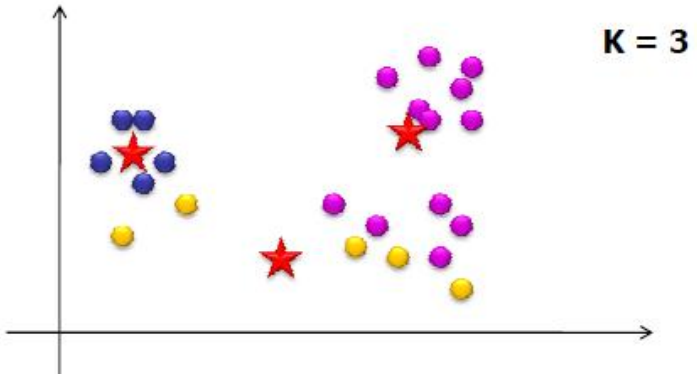
- Objective: minimize the sum of squared distance to its "**representative object**" in each cluster

$$\sum_{i=1}^{K} \sum_{X_j \in S_i} d^2(x_j, \mu_i)$$

S_i is the i-th cluster (i=1,2,…K)

μ_i is the i-th centroid of the points in cluster S_i

d is the distance function

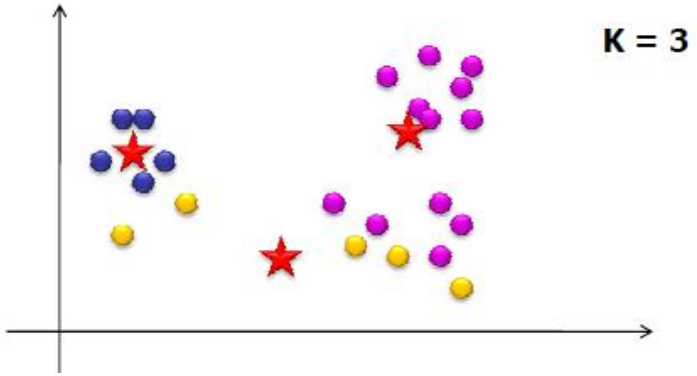


K-means algorithm Revisit

- Basically, the objective is to find the most *compact* partitioning of the data set into k partitions
  - Minimizing intra-cluster variance
  - Maximizing inter-cluster variance
- If we knew the cluster assignment of each point we could easily compute the centroid positions
- If we knew the centroid positions we could easily assign each point to a cluster
- But both are unknown

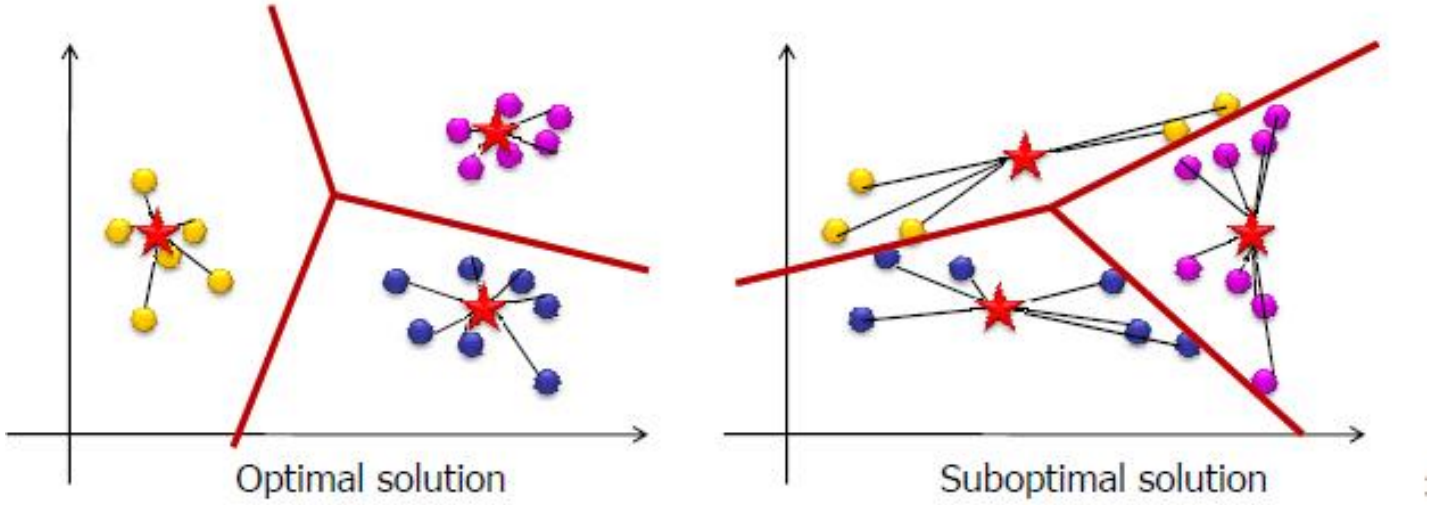
K-means algorithm Revisit

- Choose the number of clusters - K
- Randomly choose initial positions of K centroids
- Assign each of the points to the "nearest centroid" (depends on distance measure)
- Re-compute centroid positions
- If solution converges $\rightarrow$ Stop!



Things we need to consider

- Does the algorithm guarantee convergence to an optimal solution?

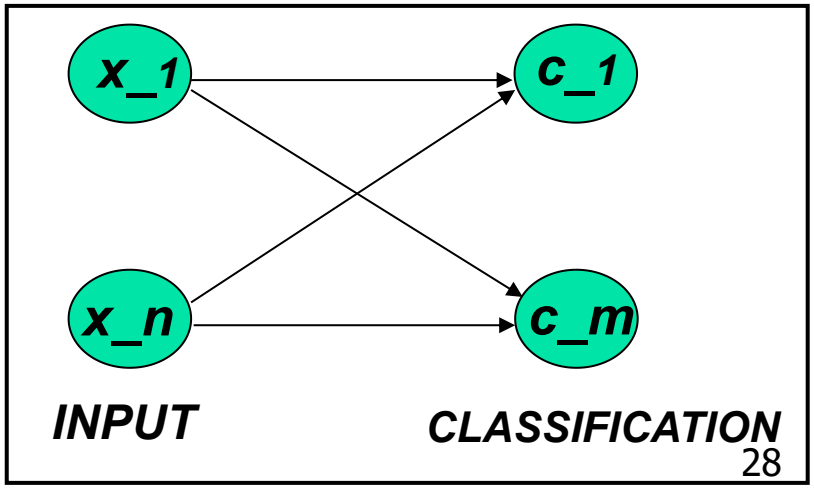


Unsupervised Competitive Learning

- In Hebbian networks, all neurons can "fire" at the same time
- **Competitive learning** means that only a single neuron from each group fires at each time step
- Output units compete with one another.
- These are **winner-takes-all (WTA)** units
- Competition is important for neural networks
  - Competition between neurons has been observed in biological nerve systems
  - Competition is important in solving many problems

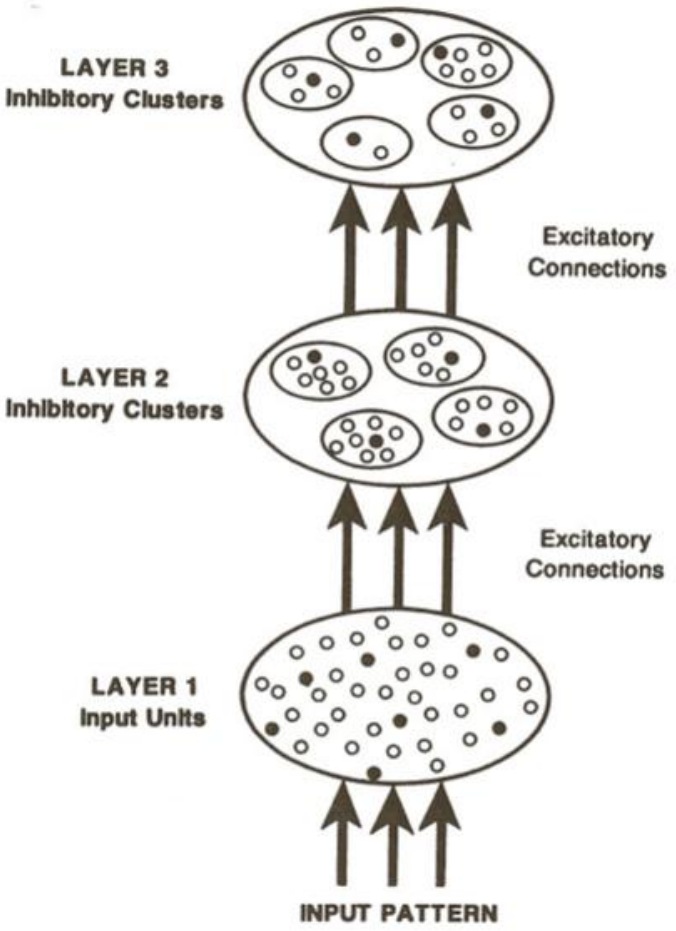
Unsupervised Competitive Learning

- To classify an input pattern into one of the m classes
  - idea case: only one node gives output 1, all the others are 0
  - however, usually more than one nodes have non-zero output
- If these class nodes compete with each other, maybe only one will win eventually and all others lose (winner-takes-all). The winner represents the computed classification of the input



Unsupervised Competitive Learning

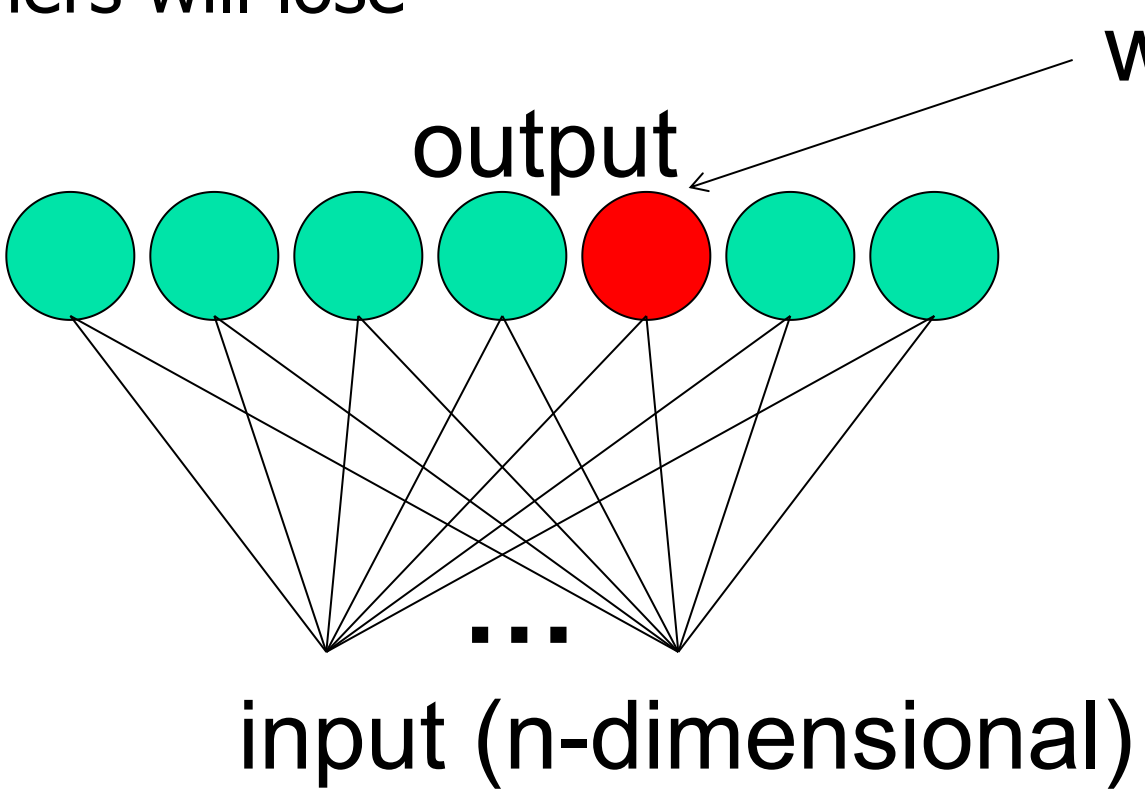
- Units (active or inactive) are represented in the diagram as dots
  - active units are represented by filled dots
  - inactive ones by open dots
- A unit in a given layer can
  - receive inputs from all of the units in the next lower layer
  - project outputs to all of the units in the next higher layer
- Connections between layers are excitatory
- Connections within layers are inhibitory
  - each layer consists of a set of clusters of mutually inhibitory units
  - the units within a cluster inhibit one another in such a way that only one unit per cluster may be active



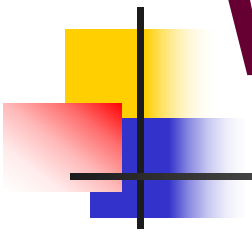
The architecture of competitive learning mechanism

Winner-takes-all (WTA)

- Among all competing nodes, only one will win and all others will lose



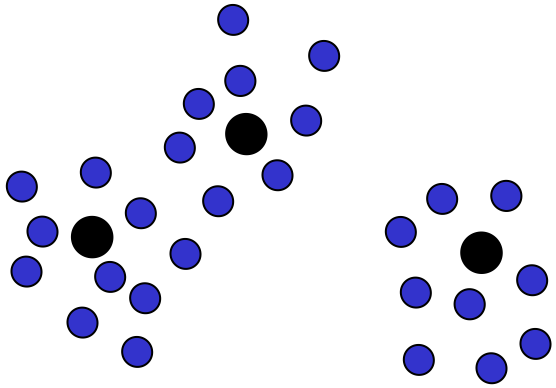
Winner-takes-all (WTA)



- Easiest way to realize WTA: have an external, central arbitrator (a program) to decide the winner by comparing the current outputs of the competitors (break the tie arbitrarily)
- This is biologically unsound (no such external arbitrator exists in biological nerve system)

Simple Competitive Learning

- initialize K prototype vectors
- present a single example
- identify the closest prototype, i.e., the so-called ***winner***
- move the winner even closer towards the example



Intuitively clear, plausible procedure

- places prototypes in areas with high density of data
- identifies the most relevant combinations of features

Simple Competitive Learning

- Winner:

$$h_j = \sum_i w_{ji} x_i$$

$$w_{j*} \cdot x \geq w_j \cdot x \qquad \forall j \neq j^*$$

Note: the inner product of two normal vectors is related to the cosine of the angle between them

Winner = output node whose incoming weights are the shortest Euclidean distance from the input vector

- Lateral inhibition

Simple Competitive Learning

- (One possible) update rule for all neurons:

$$\Delta w_{ji} = \eta y_j \left(x_i - w_{ji} \right)$$

$$\begin{cases} y_{j^*} = 1 \\ y_j = 0 \qquad if \qquad j \neq j^* \end{cases}$$

The neuron with largest activation is then adapted to be more likely the input that caused the excitation

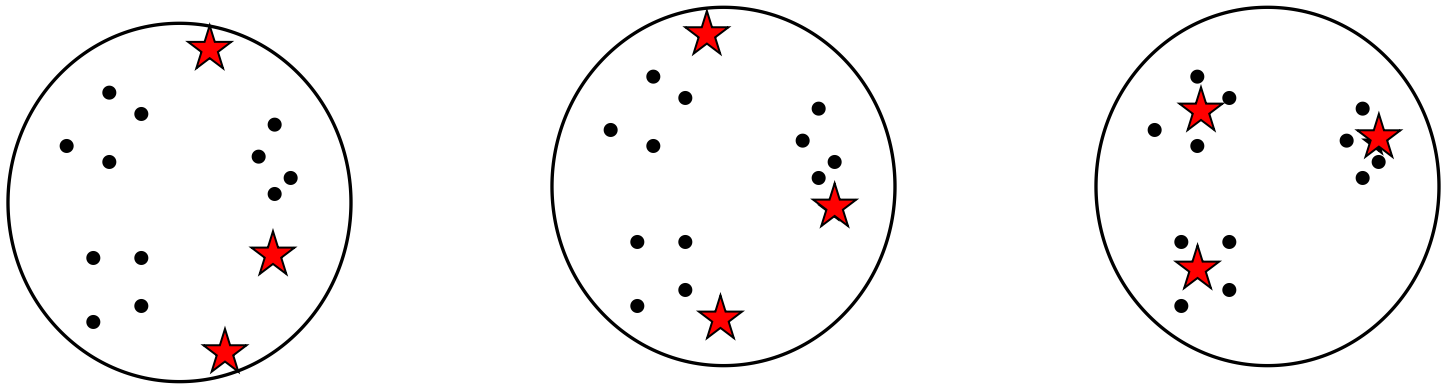
Only the incoming weights of the winner node are modified.

Some units may never or rarely become a winner, and so weight vector may not be updated→**DEAD UNIT**

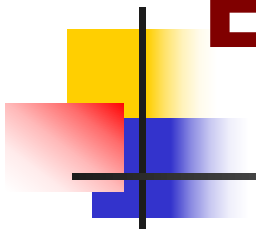
Simple Competitive Learning

- Each output unit moves to the center of mass of a cluster of input vectors ➔

clustering



Enforcing fairer competition



- Initial position of weight vector of an output unit may be in region with few (if any) patterns
- Some units may never or rarely become a winner, and so weight vector may not be updated, thus preventing it finding richer part of pattern space $\rightarrow$**DEAD UNIT**
- More efficient to ensure a fairer competition where each unit has an equal chance of representing some part of training data

Leaky learning

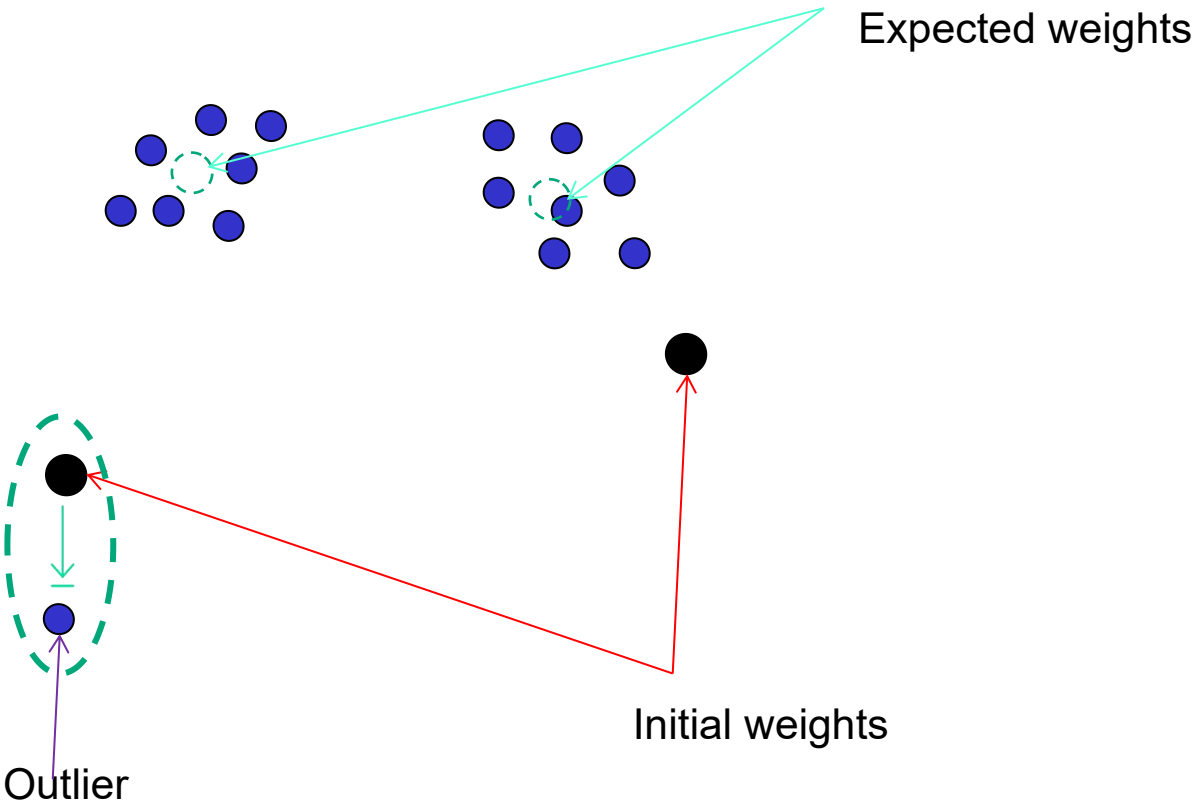
- Modify weights of both winning and losing units but at different learning rates

$$w(t+1) = w(t) + \begin{cases} \eta_w (x - w(t)) \\ \eta_L (x - w(t)) \end{cases}$$

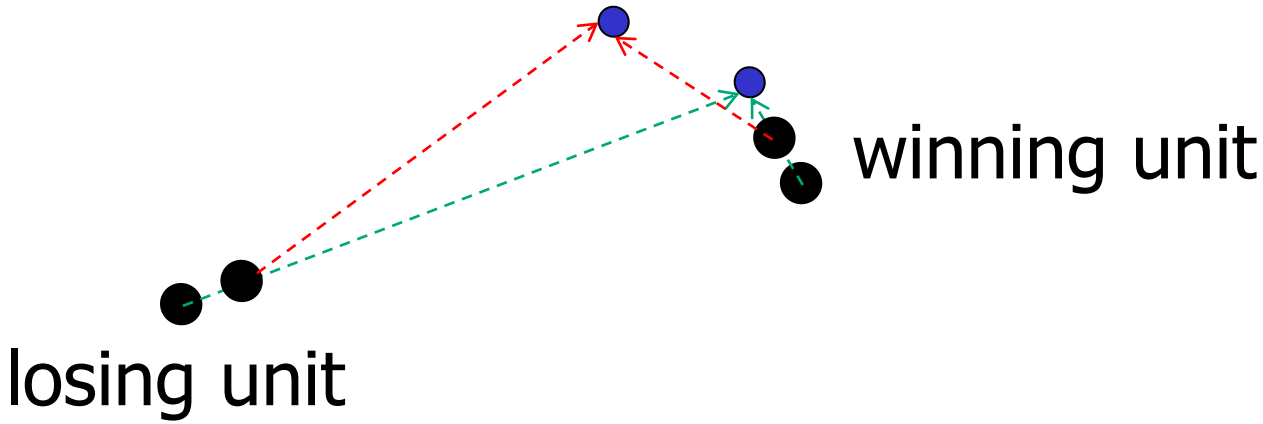
where $\eta_w \gg \eta_L$

which has the effect of slowly moving losing units towards denser regions pattern space.

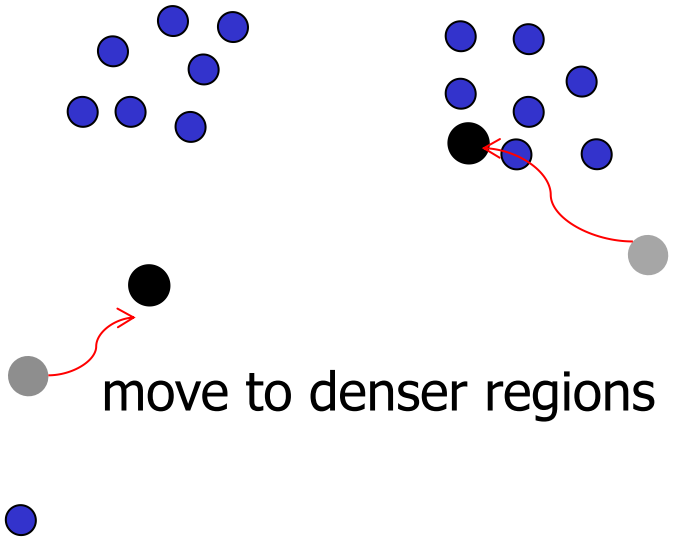
Leaky learning



Leaky learning



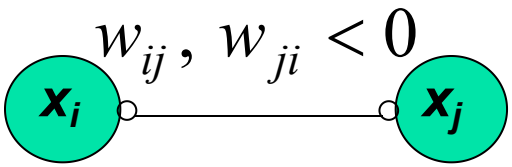
Leaky learning

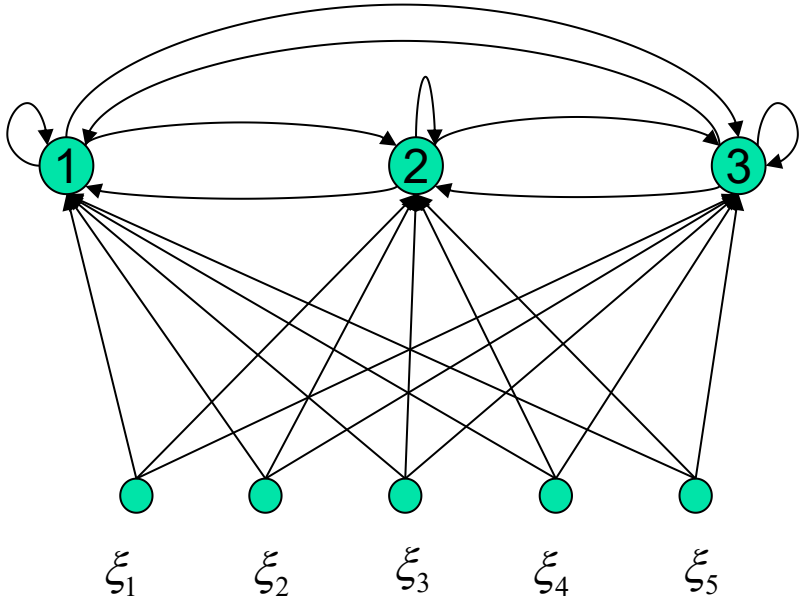


Maxnet

Lateral inhibition

output of each node feeds to others through inhibitory connections (with negative weights)





A specific competitive net that performs Winner Take All (WTA) competition is the **Maxnet**

Maxnet

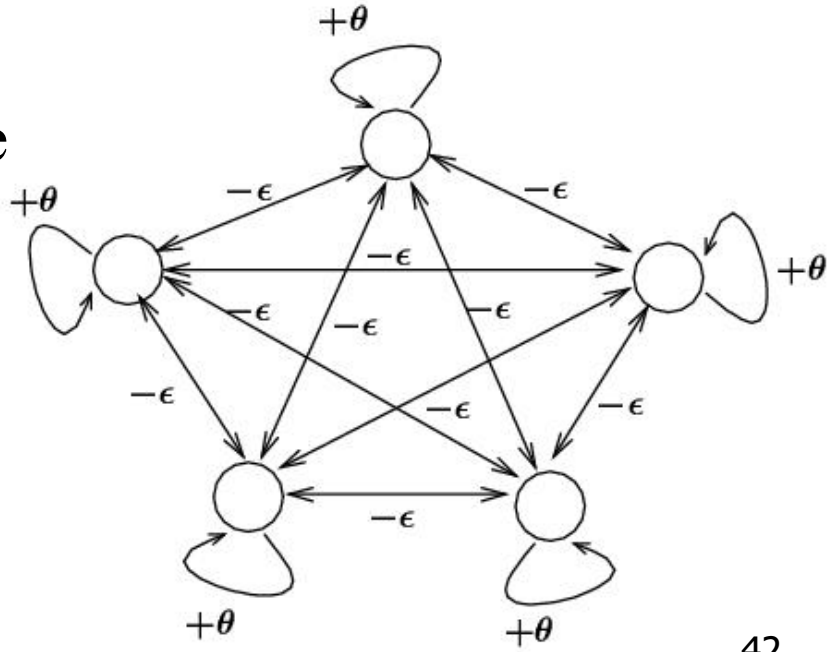
Maxnet

Lateral inhibition between competitors

$$\text{weights} \quad : w_{ji} = \begin{cases} \theta & \text{if } i = j \\ -\varepsilon & \text{otherwise} \end{cases}$$

$$\text{node function} \quad :$$

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

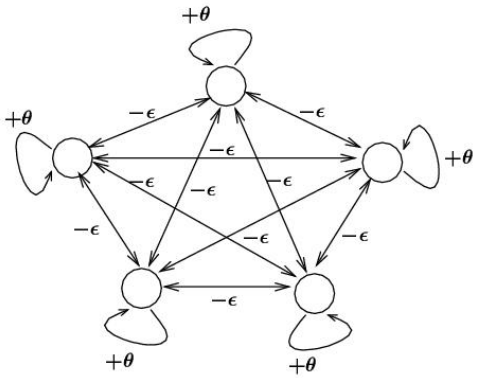


Maxnet

$$\text{weights} \quad : w_{ji} = \begin{cases} \theta & \text{if } i = j \\ -\varepsilon & \text{otherwise} \end{cases}$$

$$\text{node function} \quad :$$

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



Notes:

- Competition: iterative process until the net stabilizes (at most one node with positive activation)

  $0 < \varepsilon < 1/m$, where m is the # of competitors

- ε too small: takes too long to converge
- ε too big: may suppress the entire network (no winner)

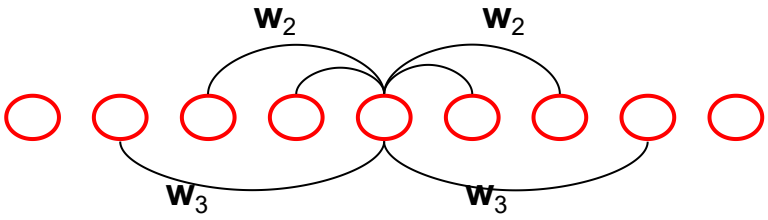
Mexican Hat Network

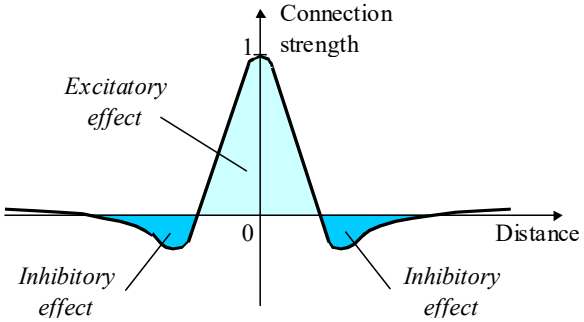
- Among all competing nodes, only one will win and all others will lose
  - The neuron with the largest activation level among all neurons in the output layer becomes the winner. This neuron is the only neuron that produces an output signal. The activity of all other neurons is suppressed in the competition.

- We mainly deal with single winner WTA, *but multiple winners WTA are possible*
  - The lateral 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 output layer.

Mexican Hat Network

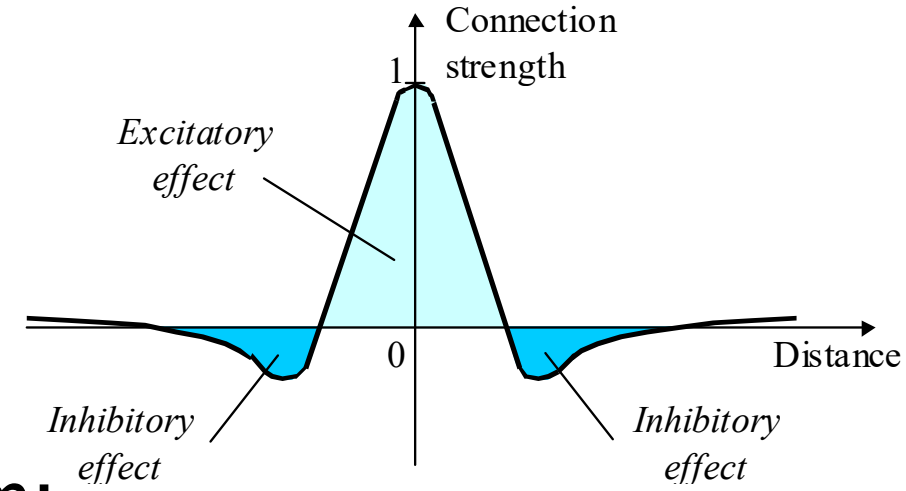
- **Architecture**: For a given node,
  - close neighbors: **cooperative** (mutually excitatory , w > 0)
  - distant neighbors: **competitive** (mutually inhibitory, w < 0)
  - too far away neighbors: irrelevant (w = 0)





- Need a definition of **distance (neighborhood):**
  - one dimensional: ordering by index (1,2,…n)
  - two dimensional: lattice

Mexican Hat Network



- Equilibrium:
  - negative input = positive input for all nodes
  - winner has the highest activation
  - its cooperative neighbors all have positive activations
  - its competitive neighbors all have negative (or zero) activations

Example

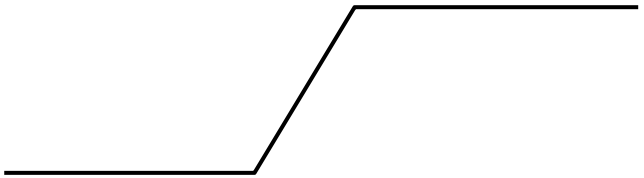
weights

$$w_{ij} = \begin{cases} c_1 & \text{if} \quad \text{distance}(i,j) < k \ (c_1 > 0) \\ c_2 & \text{if} \quad \text{distance}(i,j) = k \ (0 < c_2 < c_1) \\ c_3 & \text{if} \quad \text{distance}(i,j) > k \ (c_3 \le 0) \end{cases}$$

activation function

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

ramp function :



Vector Quantization
Application of competitive learning

- **Idea**
  - categorize a given set of input vectors into M classes using competitive learning algorithms
  - represent any vector just by the class into which it falls

- Important application of competitive learning (esp. in data compression)
  - divides entire pattern space into a number of separate subspaces
  - set of M units represent set of prototype vectors: **CODEBOOK**
  - new pattern x is assigned to a class based on its closeness to a prototype vector using Euclidean distances

Vector Quantization

- A codebook
  - a set of centroids/**codewords/codevector**):

$$\{m_1, m_2, \ldots, m_K\}$$

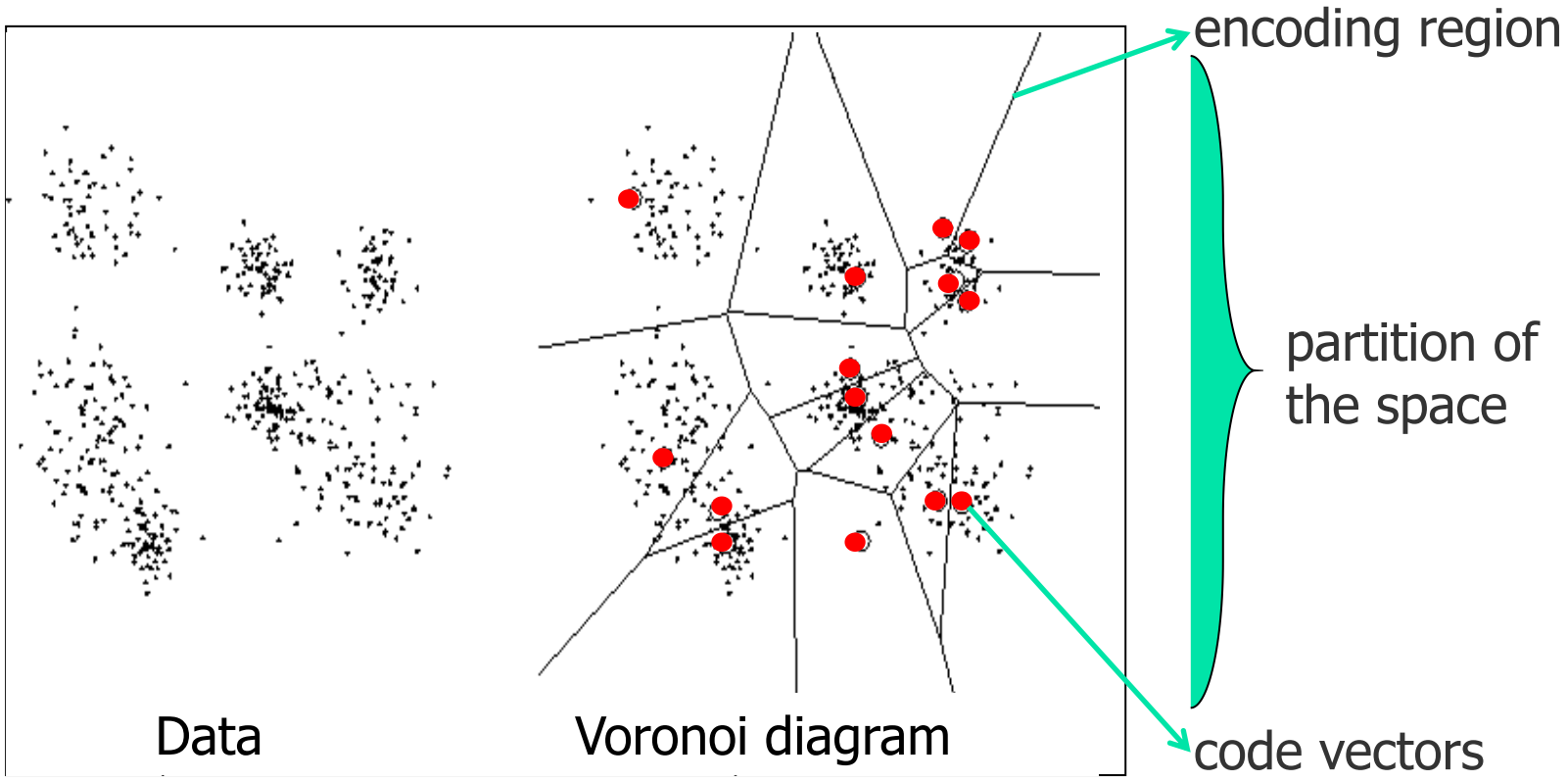
- A quantization function:

$$q(x_i) = m_k$$

Often, the nearest-neighbor function

- *K*-means can be used to construct the codebook

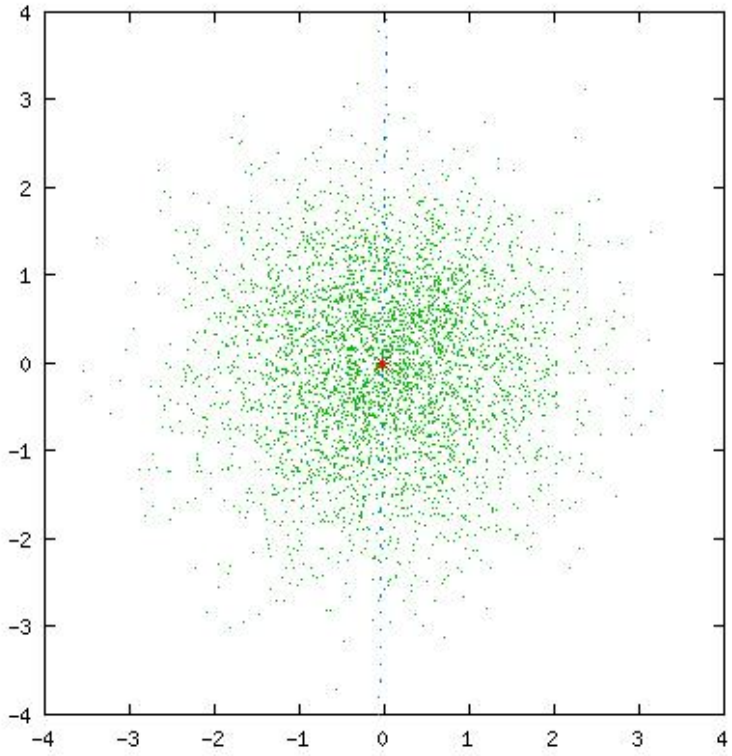
Vector Quantization



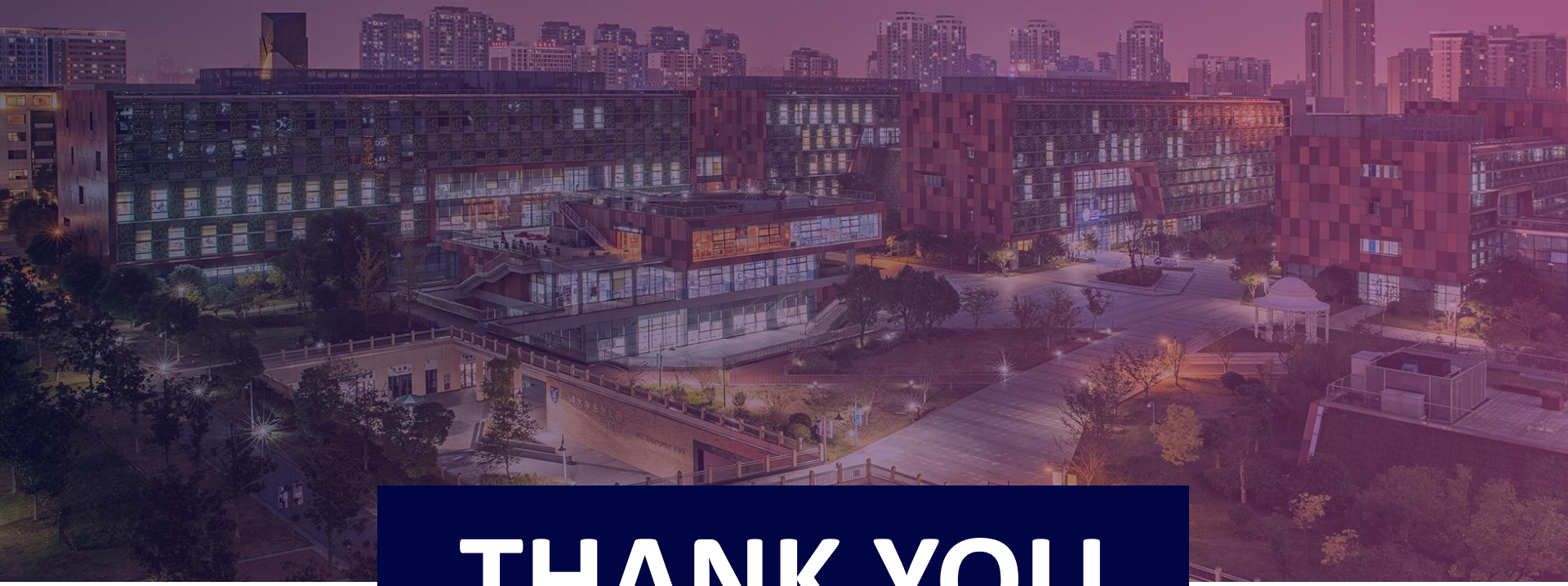
A Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. For each seed there is a corresponding region consisting of all points closer to that seed than to any other. These regions are called Voronoi cells.

Vector Quantization

- LBG design algorithm for VQ
  - NOT within the scope of this module



Two-dimensional VQ animation



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學