ONLY GOD

1402/ 2023

mansoori@shirazu.ac.ir

Neural Network & Deep Learning

Single-Layer Competitive Network

CSE & IT Department
School of ECE
Shiraz University

Competitive Networks

mansoori@shirazu.ac.ir

NNs based on competition

- Self-organization
  - A network seeks to find patterns or regularity in input data without supervision
- Unsupervised learning
  - No target information provided in training set
  - Attempt to discover special patterns from available data without using external help
- Competition
  - Net is forced to make a decision intrinsically on the most responsive neuron to any pattern

Competition Forms

mansoori@shirazu.ac.ir

Two competition forms

- Winner-takes-all
  - Only one neuron will have a non-zero output signal after completing competition
- On-center, off-surround
  - Each neurons has a number of cooperative neighbors and some competitive neighbors to do contrast enhancement

Clustering Networks

mansoori@shirazu.ac.ir

Competition-based NNs

- Fixed-weight nets: Hamming net
- Adjustable-weight nets: SOM, LVQ, ART

In clustering net

- No. of input units = no. of input vector components
- No. of output units = no. of clusters to be formed
- Weight vector of each output unit (cluster) is a representative or exemplar vector for input patterns belonging to that cluster
- During training, net finds output unit that best matches input vector, then its weights vector is adjusted

Competition

mansoori@shirazu.ac.ir

Methods of determining the closest weight vector ($\vec{w}$) to pattern vector ($\vec{x}$)

- Euclidean distance: $||\vec{x} - \vec{w}||^2$
  - Winning unit has the smallest distance
- Dot product: $\vec{x}.\vec{w} = \vec{x}^T\vec{w}$
  - Winning unit has the largest product

For weights of unit length, two methods act equally

$$||\vec{x} - \vec{w}||^2 = (\vec{x} - \vec{w})^T(\vec{x} - \vec{w}) = ||\vec{x}||^2 + ||\vec{w}||^2 - 2\vec{x}^T\vec{w}$$
$$= 2(1 - \vec{x}.\vec{w})$$

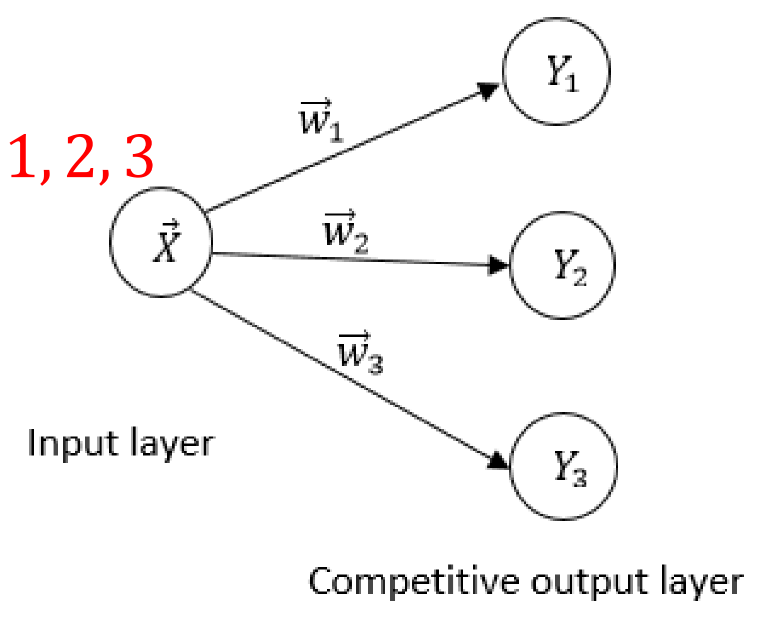
mansoori@shirazu.ac.ir

Competitive Learning

Competitive Learning

mansoori@shirazu.ac.ir

- Winner-takes-all approach
  - Only winner is allowed to learn (change its weight)
- Supposing all training vectors, $\vec{x}$, belong to three clusters
- We need a net with three competitive neurons for three clusters
  - Neuron Y_i for cluster C_i, $i = 1, 2, 3$



Competitive Learning

mansoori@shirazu.ac.ir

- Neuron Y_i should respond strongly to training vectors in cluster C_i, so $\vec{w}_i$ must be a good template for C_i
- For each input vector $\vec{x}$, find the winner neuron by competition:
  - Euclidean distance:

    $y_i = y\_in_i = \|\vec{x} - \vec{w}_i\|^2,$

    $k = \arg\min_i y_i \Rightarrow Y_k$: winner unit
  - Dot product:

    $y_i = y\_in_i = \vec{x}.\vec{w}_i$

    $k = \arg\max_i y_i \Rightarrow Y_k$: winner unit
- Weight $\vec{w}_k$ is closest to $\vec{x}$ $\Rightarrow$ try to align $\vec{w}_k$ on $\vec{x}$

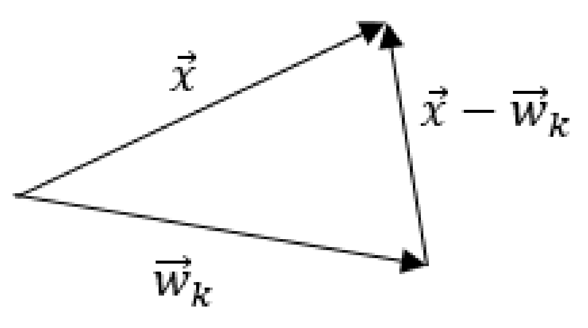
Competitive Learning

mansoori@shirazu.ac.ir

To obtain these alignments:

- $\vec{x}$ applies iteratively and weight $\vec{w}_k$ adjusts via rotating toward $\vec{x}$, by adding a fraction of $\vec{x} - \vec{w}_k$

$$\Delta\vec{w}_k = \alpha\,(\vec{x} - \vec{w}_k)$$



- If training patterns are well-clustered, a stable weight set can be found for each unit, otherwise weights might be unstable
- In stable solution, weight vector of each unit is a representative, template or average of training patterns in that cluster (as in *k*-means)

Competitive Learning

mansoori@shirazu.ac.ir

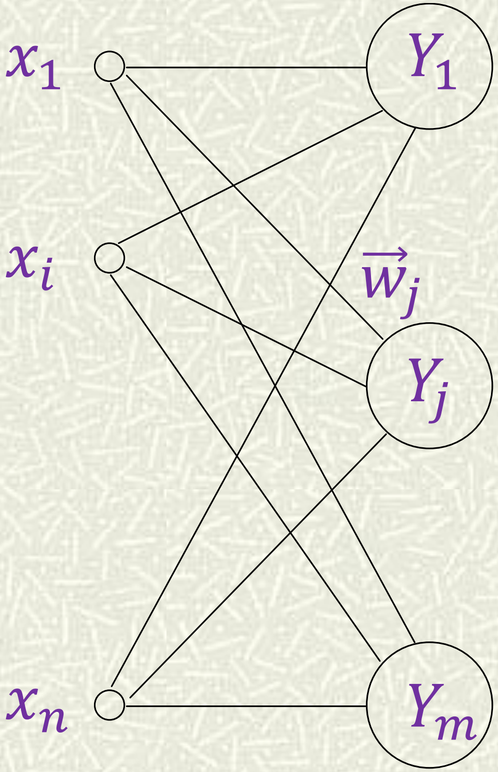
Network topology

- For winner-takes-all competitive learning, network consists of
  - A single layer of linear neurons
  - Each neuron is connected to all inputs

$$\vec{x} = [x_1 \dots x_n]^T \;,\; \vec{y} = [y_1 \dots y_m]^T$$

$$y_j = \vec{w}_j^{\,T} \vec{x}$$

$$W = [\vec{w}_1 \;\dots\; \vec{w}_m] \Longrightarrow \vec{y} = W^T \vec{x}$$



Training set: $X = [\vec{x}(1) \dots \vec{x}(P)]$

where $\vec{x}(q) = [x_1(q) \;\dots\; x_n(q)]^T$

Competitive Learning

mansoori@shirazu.ac.ir

Definition of winner

- Two criteria to define which neuron Y_k becomes winner of competition for input $\vec{x}$
  - Euclidean distance:

    $$k = \mathrm{win}(W, \vec{x}) \equiv \forall_{j=1..m} \|\vec{x} - \vec{w}_k\|^2 \leq \left\|\vec{x} - \vec{w}_j\right\|^2$$

  - Dot product:

    $$k = \mathrm{win}(W, \vec{x}) \equiv \forall_{j=1..m} \vec{w}_k^{\,T} \vec{x} \geq \vec{w}_j^{\,T} \vec{x}$$

- Training set X will be partitioned into clusters $\{C_k, k = 1 \dots K\}$ according to competition made by network:

  $$C_k = \{\vec{x} \in X \mid \mathrm{win}(W, \vec{x}) = k\}$$
  $$n_k = |C_k|$$

Competitive Learning

mansoori@shirazu.ac.ir

- Error function:

  $E(W) = \frac{1}{2P} \sum_{\vec{x} \in X} \|\vec{x} - \vec{w}_k\|^2$ where $k = \mathrm{win}(W, \vec{x})$

- Error function in terms of K clusters:

  $E(W) = \sum_{k=1}^{K} E_k(W)$

- Error function of cluster C_k:

  $E_k(W) = \frac{1}{2\,n_k} \sum_{\vec{x} \in C_k} \|\vec{x} - \vec{w}_k\|^2$

- Gradients of error function:

  $\nabla_{\vec{w}_k} E(W) = \nabla_{\vec{w}_k} E_k(W) = \frac{\partial}{\partial \vec{w}_k} E_k(W)$

  $= \frac{1}{2\,n_k} \frac{\partial}{\partial \vec{w}_k} \sum_{\vec{x} \in C_k} \|\vec{x} - \vec{w}_k\|^2 = -\frac{1}{n_k} \sum_{\vec{x} \in C_k} (\vec{x} - \vec{w}_k)$

Competitive Learning

mansoori@shirazu.ac.ir

- Gradient of error:

$$\nabla_{\vec{w}_k} E(W) = -\frac{1}{n_k} \sum_{\vec{x} \in C_k} (\vec{x} - \vec{w}_k) = -(\frac{1}{n_k} \sum_{\vec{x} \in C_k} \vec{x} - \frac{1}{n_k} \sum_{\vec{x} \in C_k} \vec{w}_k)$$
$$= -(\vec{m}_k - \vec{w}_k)$$

where $\vec{m}_k$ is mean of cluster C_k

- Using gradient descent:

$\Delta \vec{w}_k = -\alpha \frac{\partial}{\partial \vec{w}_k} E(W) = \alpha(\vec{m}_k - \vec{w}_k)$: in batch mode

- When learning algorithm converges, all gradients are zero:

$$-(\vec{m}_k - \vec{w}_k) = 0 \Rightarrow \vec{w}_k = \vec{m}_k = \frac{1}{n_k} \sum_{\vec{x} \in C_k} \vec{x}$$

- So, weight vectors of non-empty clusters have converged to mean of vectors in those clusters

mansoori@shirazu.ac.ir

SOM

mansoori@shirazu.ac.ir

Competitive Learning

- On-center, off-surround approach
  - Winner of competition and some neurons in its in neighborhood are allowed to learn (change weights)

- These competitive networks
  - Can organize groups of physically adjacent neurons in net which encode adjacent (proximity) patterns
  - This proximity defines a topography over a neural layer
  - These maps represent some features of input space
  - Such maps exist in many areas of cortex in animal brains

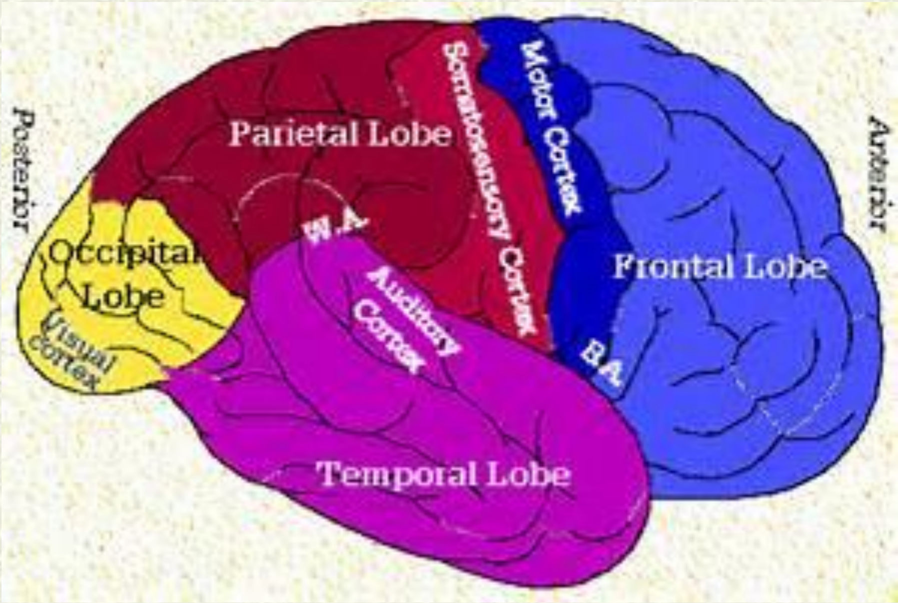
Biological Motivation of SOM

mansoori@shirazu.ac.ir

- A part of brain that contains many topographic maps is cerebral cortex (externally visible sheet of neural tissue)

- Is responsible for processing sensory information such as sound and vision
  - Visual cortex
  - Somatosensory cortex
  - Auditory cortex

Cytoarchitectural map



Characteristics of SOM

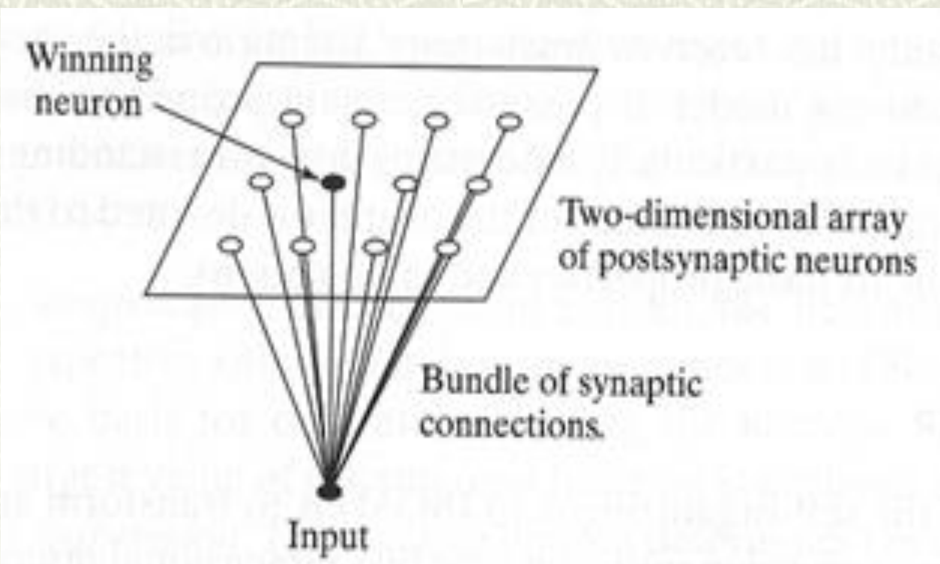
mansoori@shirazu.ac.ir

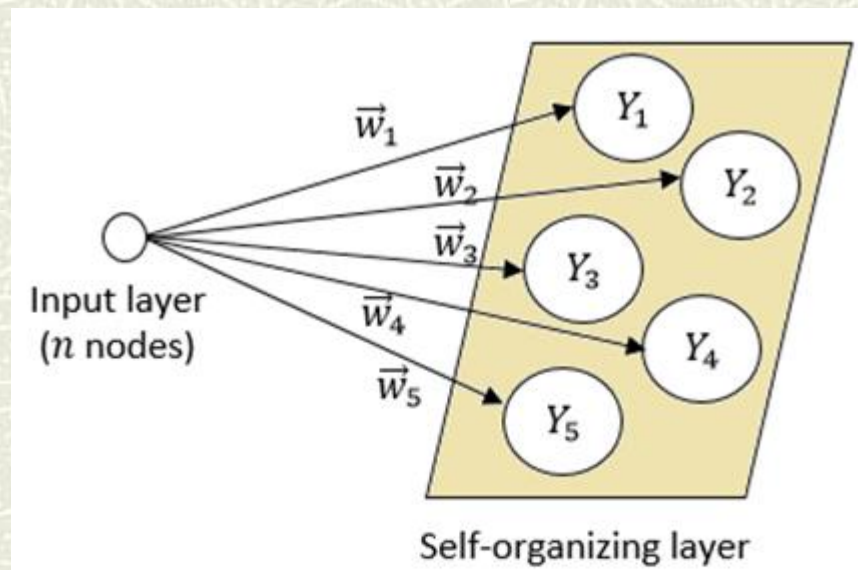
- Neurobiological hypothesis:
  - Structure self-organizes based on learning rules and system interaction
  - Axons physically maintain neighborhood relationships as they grow
- Kohonen examined problem of topographic map formation and developed algorithm of self-organizing feature map (SOFM or SOM)
- SOMs are neural network models for unsupervised learning, which combine a competitive learning principle with a topological structuring of neurons such that adjacent neurons tend to have similar weight vectors

Architecture of SOM

mansoori@shirazu.ac.ir

- Net architecture consists of an input layer and a self-organizing layer
- Input units are fully-connected to all neurons (a lattice) in competitive layer
- Neurons are arranged in some grid of fixed topology

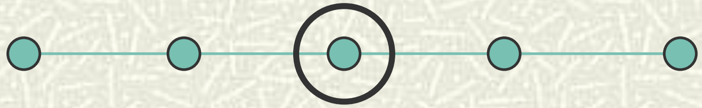




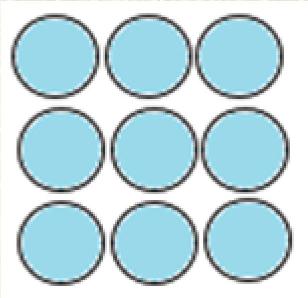
Architecture of SOM

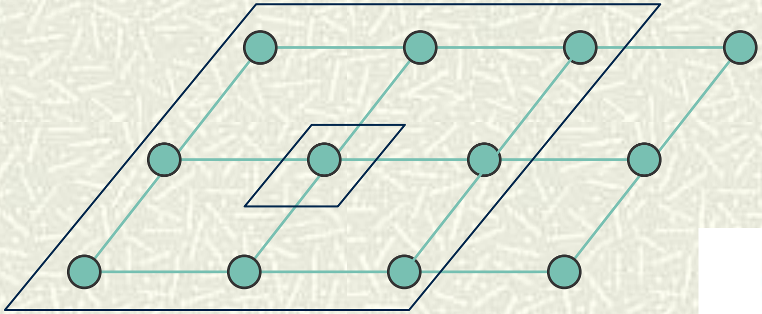
- For map formation, training should take place over spatial neighborhood of maximally active (winning) neuron within competitive layer (lattice) of net

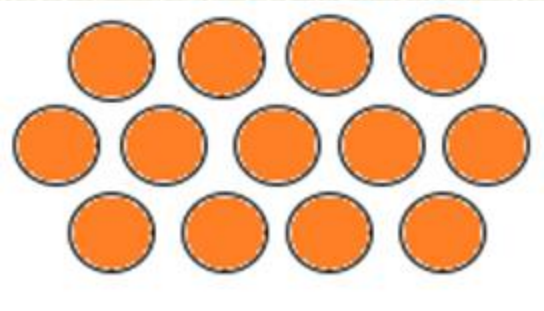
- Lattice topology determines neighborhood structure of neurons
  - 1D array of linear nodes



  - 2D array of rectangular grids (‘gridtop’)



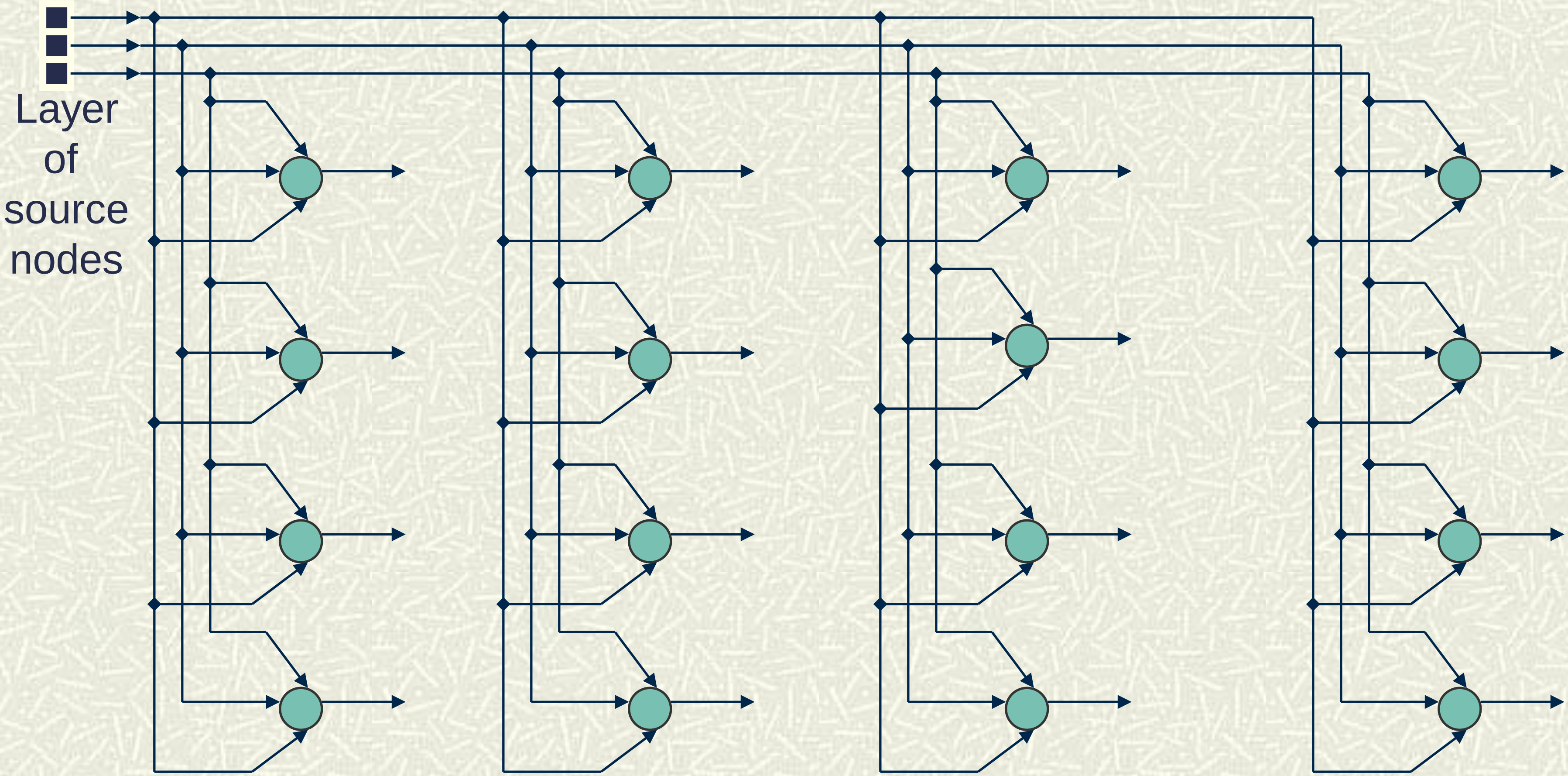




  - 2D array of hexagonal grids (‘hextop’)

Architecture of SOM

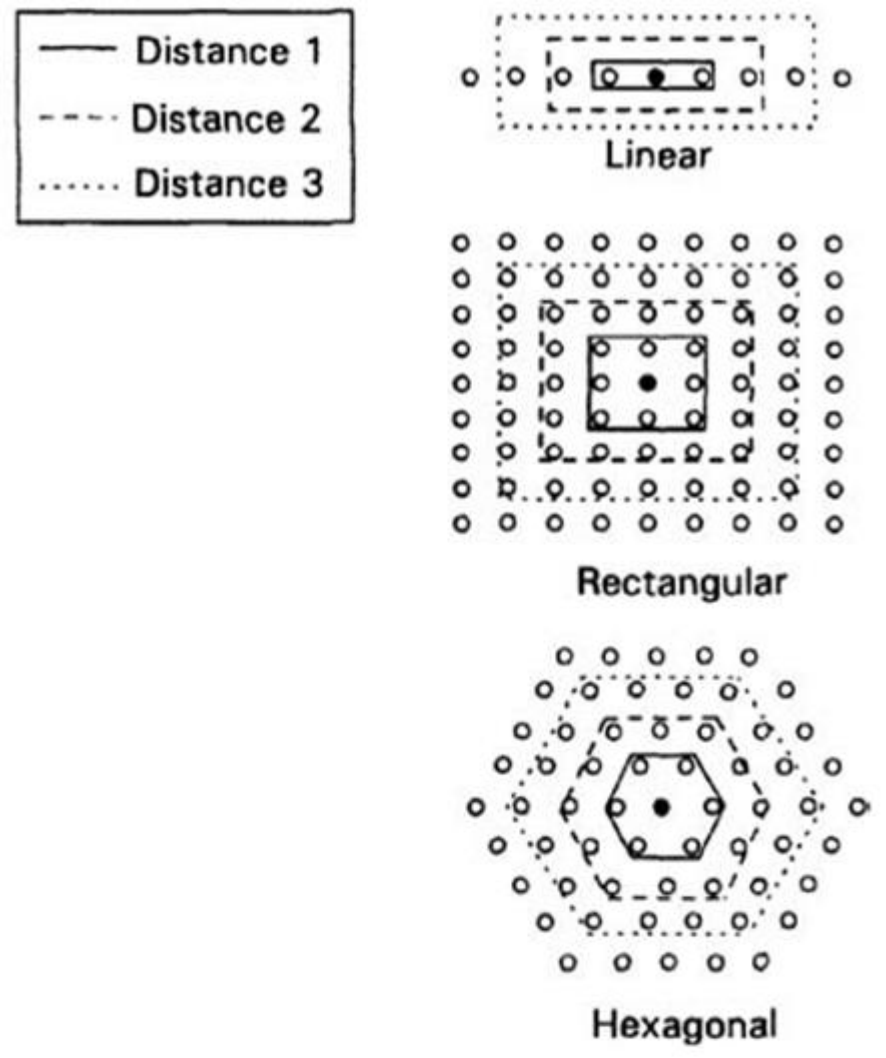
mansoori@shirazu.ac.ir



Architecture of SOM

mansoori@shirazu.ac.ir

- Neighborhoods are in radius (distance) of 1, 2 and 3
- No. of neighbors
  - In linear: 3, 5, 7
  - In rectangular: 9, 25, 49
  - In hexagonal: 7, 19, 37



Goals of SOM

mansoori@shirazu.ac.ir

- To find values for weight vectors of neurons, in such a way that adjacent neurons will have similar weight vectors
- For each input, output of net will be neuron whose weight vector is most similar to that input
- In this way, each weight vector of a neuron is center of a cluster containing all input examples which are mapped to that neuron

Algorithm of SOM1

mansoori@shirazu.ac.ir

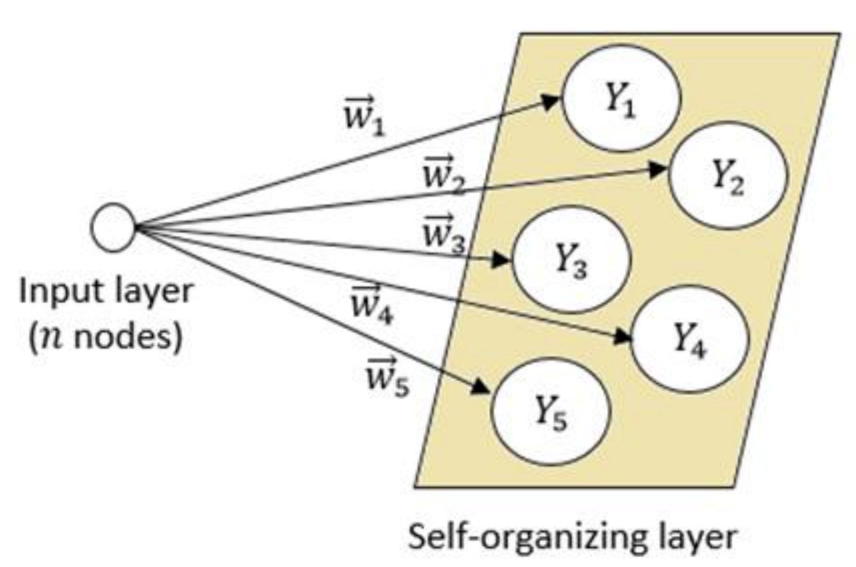
1. Initialization

   1.1. Weights, $\vec{w}_j\ ,\ (j = 1, \dots, m)$ (random values or using prior knowledge)

   1.2. Neighborhood topology (linear, rectangular or hexagonal)

   1.3. Neighborhood radius, R

   1.4. Learning rate, α



2. While weights changes are noticeable

Algorithm of SOM1

mansoori@shirazu.ac.ir

2. While weights changes are noticeable

   2.1. For each training input vector $\vec{x}$ (chosen randomly)

      2.1.1. Find the winning unit Y_k whose weight vector $\vec{w}_k$ is closest to $\vec{x}$, $k = \arg\min_j (\|\vec{x} - \vec{w}_j\|)$

      2.1.2. Update $\vec{w}_k$ and all $\vec{w}_j$ of units Y_j in neighbor of Y_k

$$\Delta\vec{w}_j = \begin{cases} \alpha(\vec{x} - \vec{w}_j), & \text{if } Y_j \text{ is neighbor of } Y_k \text{ in radius } R \\ 0\,, & \text{otherwise} \end{cases}$$

   2.2. Decrease learning rate (linearly or geometric)

   2.3. Reduce radius of neighborhood after a certain number of epochs

3. Stop

Operation of SOM1

mansoori@shirazu.ac.ir

Formation of map topography occurs in two phases

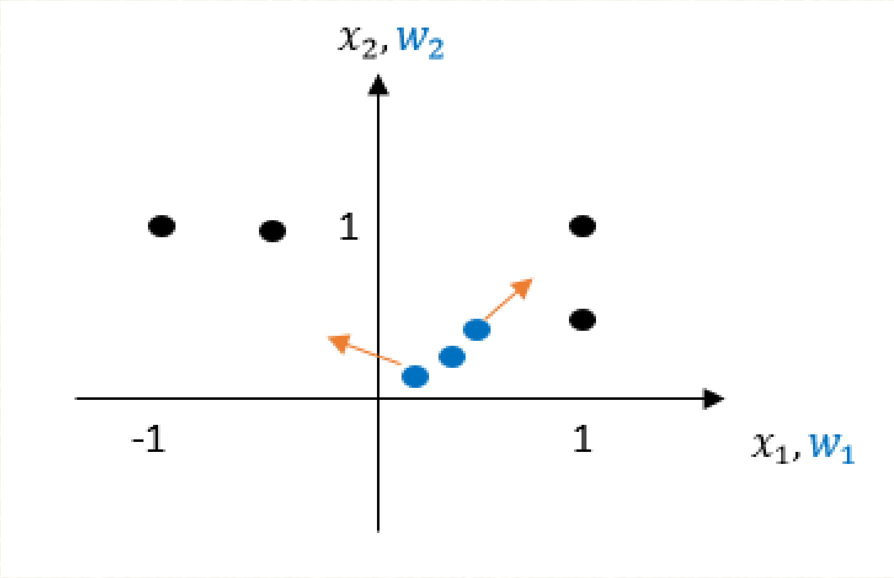
- Initial formation
  - Regional training over neighborhoods induces map formation (during first epoch, often)
  - Starting with a large neighborhood (covering about half of net)
    - Guarantees a global ordering
    - Avoids multi-encoding of a certain part of input space
- Final convergence
  - Training only the best match nodes makes small adjustments, so picking up finer details of input space

SOM1 Example

mansoori@shirazu.ac.ir

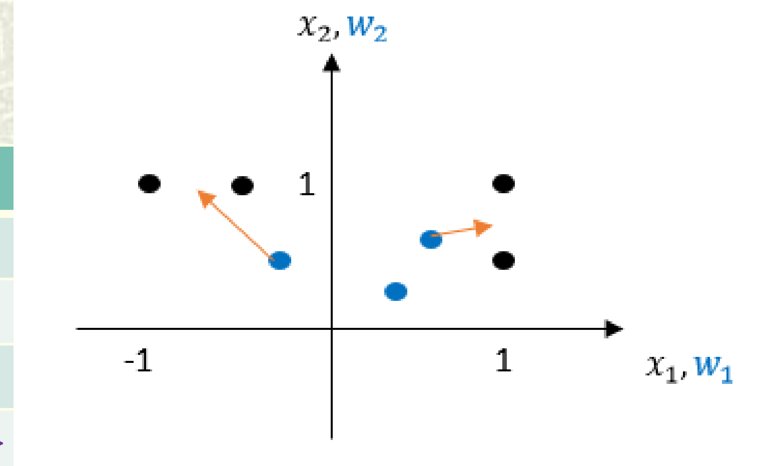
$\vec{s} = \{<1,1>, <1,0.5>, <-0.5,1>, <-1,1>\}$

$\alpha = 0.2,\ R = 0,\quad \vec{w}_1 =< 0.1, 0.3 >,\ \vec{w}_2 =< 0.4, 0.5 >,\ \vec{w}_3 =< 0.3, 0.4 >$



SOM1 Example

mansoori@shirazu.ac.ir



$\vec{w}_1$	$\vec{w}_2$	$\vec{w}_3$	$\vec{x}$
<0.10, 0.30>	<0.40, 0.50>	<0.3, 0.4>	<1,1>
<0.10, 0.30>	<0.52,0.60>	<0.3, 0.4>	<-1,1>
<-0.12, 0.44>	<0.52,0.60>	<0.3, 0.4>	<1,0.5>
<-0.12, 0.44>	<0.62,0.58>	<0.3, 0.4>	<-0.5,1>
<-0.20, 0.55>	<0.62,0.58>	<0.3, 0.4>	

$\|\vec{x}-\vec{w}_1\|$	$\|\vec{x}-\vec{w}_2\|$	$\|\vec{x}-\vec{w}_3\|$	k	$\Delta\vec{w}_k$
1.14	0.78	0.92	2	<0.12, 0.10>
1.30	1.57	1.43	1	<-0.22, 0.14>
1.12	0.49	0.71	2	<0.10, -0.02>
0.68	1.20	1.00	1	<-0.08, 0.11>

Learning in SOM

mansoori@shirazu.ac.ir

- Given an input pattern $\vec{x}$
- Find neuron Y_k which has closest weight vector by competition
- For each neuron Y_j in neighborhood $N(k)$ of winning neuron Y_k
  - Update weight vector $\vec{w}_j$ of neuron Y_j
- Neurons which are not in neighborhood of Y_k are left unchanged
- SOM algorithm
  - Starts with large neighborhood size; gradually reduces it
  - Gradually reduces learning rate

Learning in SOM

mansoori@shirazu.ac.ir

- Upon repeated presentations of training samples, weight vectors tend to follow distribution of samples
- This results in a topological ordering of neurons, where neurons adjacent to each other tend to have similar weights
- There are basically three essential processes:
  - competition
  - cooperation
  - weight adaption

Learning in SOM

mansoori@shirazu.ac.ir

Competition

- Competitive process: Find the best match of input vector $\vec{x}$ with weight vectors: $k = \underset{j}{\operatorname{argmin}}(\|\vec{x} - \vec{w}_j\|)$
- Input space of patterns is mapped onto a discrete output space of neurons by a process of competition among neurons of network

Cooperation

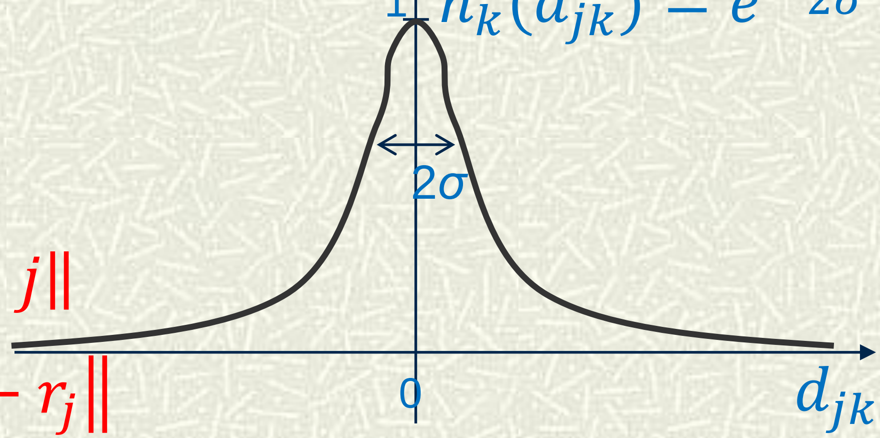
- Cooperative process: Winning neuron Y_k locates center of a topological neighborhood of cooperating neurons
- Topological neighborhood depends on lateral distance d_{jk} between winner neuron Y_k and neuron Y_j

mansoori@shirazu.ac.ir

Learning in SOM

Gaussian neighborhood function

$$h_k(d_{jk}) = e^{-\frac{d_{jk}^2}{2\sigma^2}}$$



d_{jk}: lateral distance

- In 1D lattice: $d_{jk} = \|k - j\|$
- In 2D lattice: $d_{jk} = \|r_k - r_j\|$

  r_j: position of neuron Y_j in 2D lattice

- σ (effective width): measures degree to which excited neurons in vicinity of winning neuron participate to learning
- Exponential decay update: $\sigma(n) = \sigma_0\, e^{-\frac{n}{T_1}}$, T_1: time constant

Learning in SOM

mansoori@shirazu.ac.ir

Weight adaption

- Applied to all neurons inside neighborhood $N(k)$ of winning neuron Y_k

$$\Delta\vec{w}_j = \eta y_j\, \vec{x} - g(y_j)\, \vec{w}_j$$

Hebbian term   forgetting term

$g(y_j) = \eta y_j$: scalar function of response y_j

$y_j = h_k(d_{jk})$: neighborhood function of neuron Y_k

$$\Delta\vec{w}_j = \eta h_k(d_{jk})(\vec{x} - \vec{w}_j)$$

- Exponential decay update: $\eta(n) = \eta_0\, e^{-\frac{n}{T_2}}$, T_2: time constant

Learning in SOM

mansoori@shirazu.ac.ir

Two phases of weight adaption

- **Self organizing (ordering) phase:**
  - Topological ordering of weight vectors
  - May take 1000 or more iterations of SOM algorithm

  - Important choice of parameter values:
    - $\eta_0 = 0.1$, $T_2 = 1000$ ==> learning rate decreases to $\eta(n) \geq 0.01$
    - Weight adaption, also, decreases gradually

    - σ_0 : Big enough ==> $T_1 = \frac{1000}{\log \sigma_0}$ ==> $h_k(.)$ decreases gradually
    - Initially neighborhood of winning neuron includes almost all neurons in network, then it shrinks slowly with iterations

Learning in SOM

mansoori@shirazu.ac.ir

Two phases of weight adaption

- **Convergence phase**:
  - Fine tune feature map
  - Must be at least 500 times number of neurons in network, so 1000s to 10000s of iterations
  - Choice of parameter values:
    - $\eta(n)$ maintained on order of 0.01
    - $h_k(.)$ contains only nearest neighbors of winning neuron Y_k ==> eventually reduces to 1 or 0 neighboring neurons

Algorithm of SOM

mansoori@shirazu.ac.ir

- Initialization: choose random small values for weight vectors such that $\vec{w}_j$ is different for all neurons Y_j

- Sampling: draw a sample example $\vec{x}$ from input space

- Similarity matching: find the best matching (winning) neuron Y_k as: $k = \underset{j}{\operatorname{argmin}}(\|\vec{x} - \vec{w}_j\|)$

- Updating: adjust synaptic weight vectors

$$\Delta\vec{w}_j = \eta h_k(d_{jk})(\vec{x} - \vec{w}_j)$$

- Continuation: go to sampling step until no noticeable changes in feature map are observed

SOM Example1

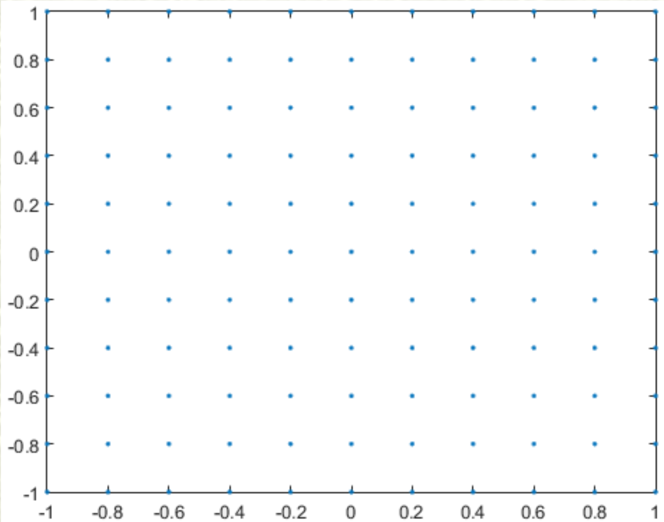
mansoori@shirazu.ac.ir

A 2D lattice driven by a 2D distribution

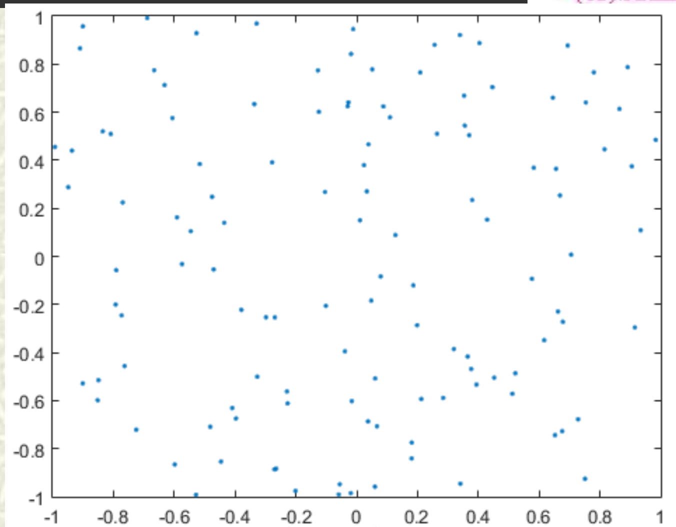
- 121 neurons arranged in a 2D lattice of 11x11 nodes
- Competitive neurons have same dimension of input space
- Input is bi-dimensional: $\vec{x} = (x_1, x_2)$ from a uniform distribution in a region defined by: $\{-1 \leq x_1, x_2 \leq +1\}$
- Weights are initialized with random values
- Neurons are visualized as changing positions in weight space as training takes place

SOM Example1

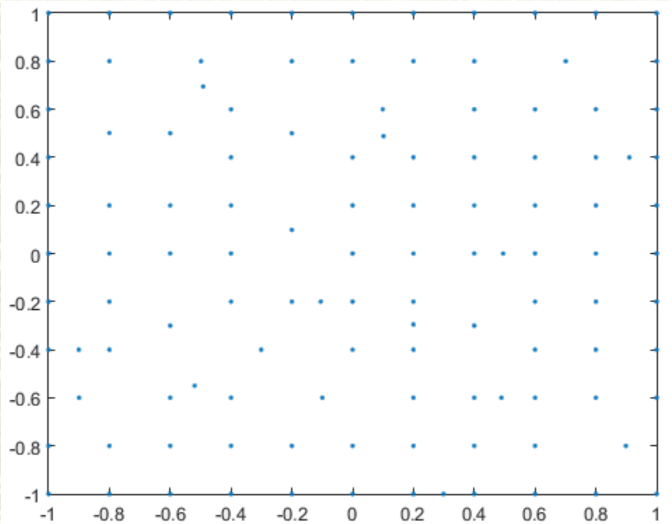
mansoori@shirazu.ac.ir



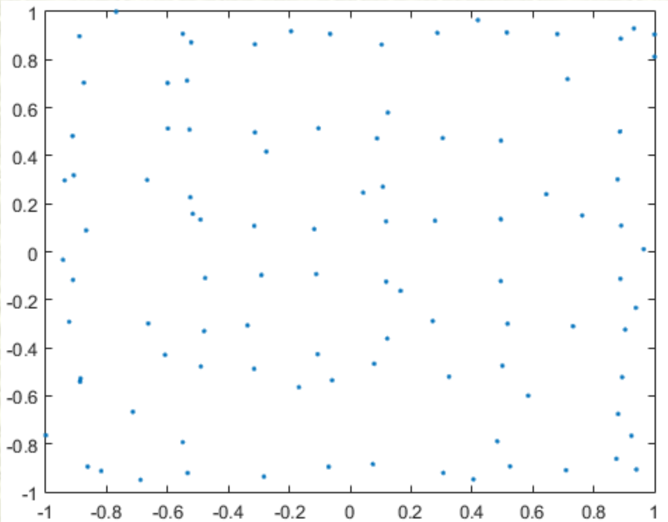
Input data distribution



Initial 2D lattice



Lattice after convergence phase



Lattice after ordering phase

SOM Example2

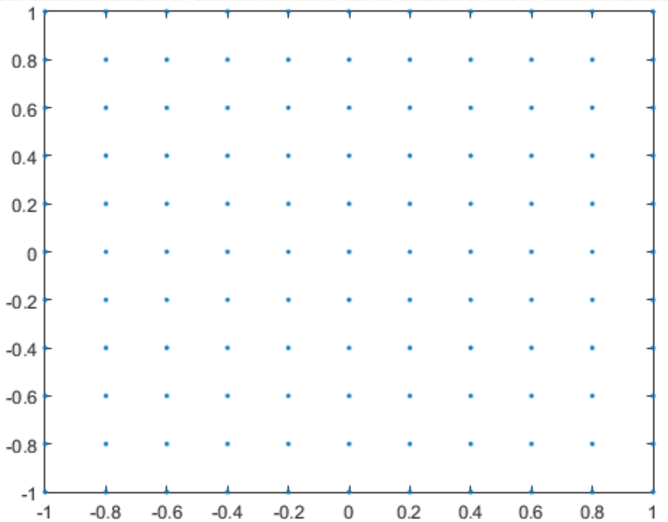
mansoori@shirazu.ac.ir

A 1D lattice driven by a 2D distribution

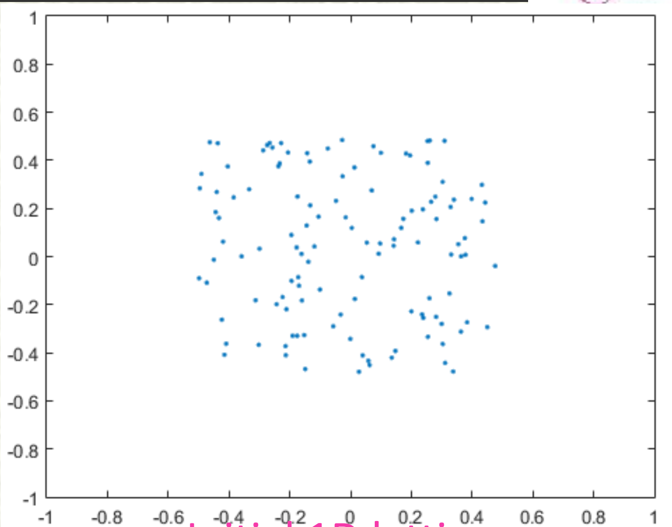
- 121 neurons arranged in a 1D lattice of 121 nodes
- Input is bi-dimensional: $\vec{x} = (x_1, x_2)$ from a uniform distribution in a region defined by: $\{-1 \leq x_1, x_2 \leq +1\}$
- Weights are initialized with random values
- Neurons are visualized as changing positions in weight space as training takes place

SOM Example2

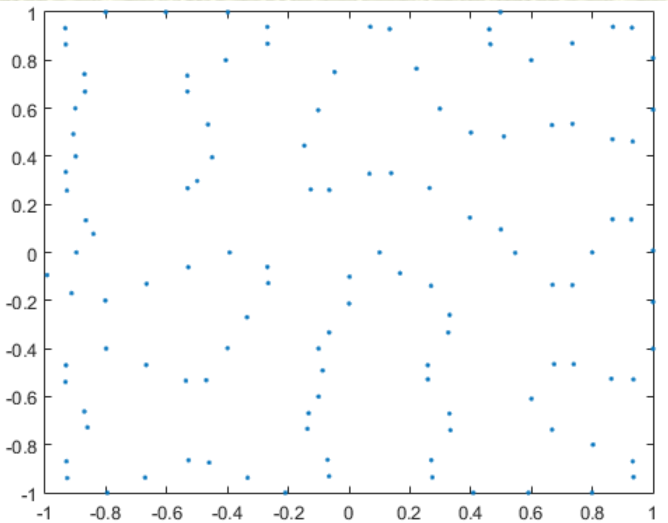
mansoori@shirazu.ac.ir



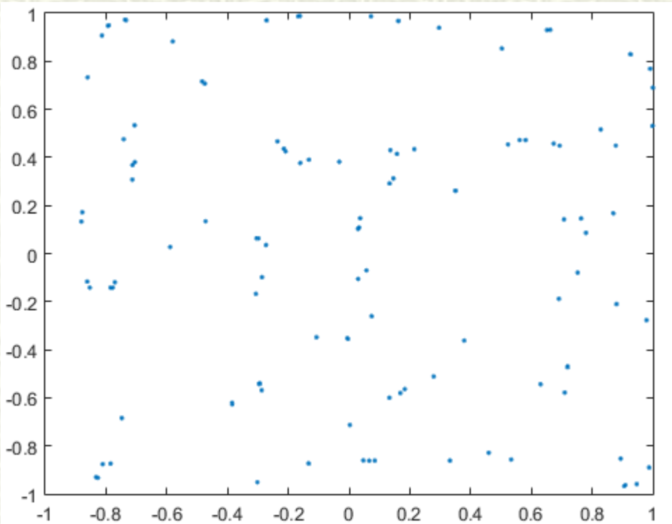
Input data distribution



Initial 1D lattice



Lattice after convergence phase



Lattice after ordering phase

mansoori@shirazu.ac.ir

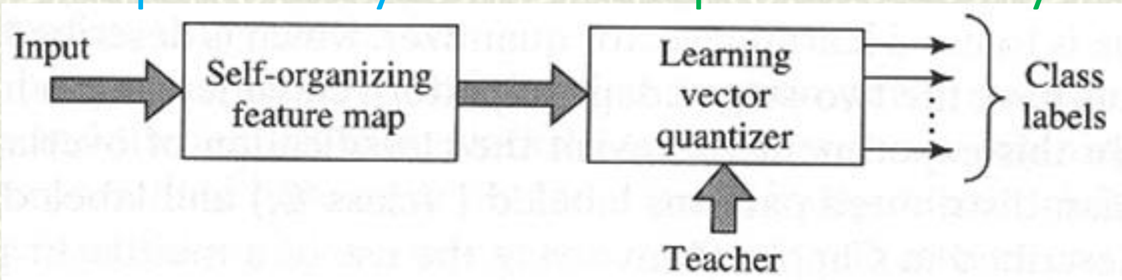
LVQ

Learning Vector Quantization (LVQ)

mansoori@shirazu.ac.ir

- Self-organizing encodes clusters in training data
- These neurons may become classifiers if they have class labels
- Class of each neuron can be assigned if training data have labels
- To improve performance of classifier, Kohonen proposed LVQ to supervisedly fine-tune class boundaries (classifier decision regions) of an SOM
- It can be divided into two parts:

competitive layer vector quantization layer



LVQ

mansoori@shirazu.ac.ir

- Vector quantizer as a clustering technique seeks to divide input space into areas that are assigned representative or code-book vectors

- SOM is a vector quantizer
  - SOM does input space division in efficient manner
  - Weight vector of each neuron is code-book vector of that cluster/class

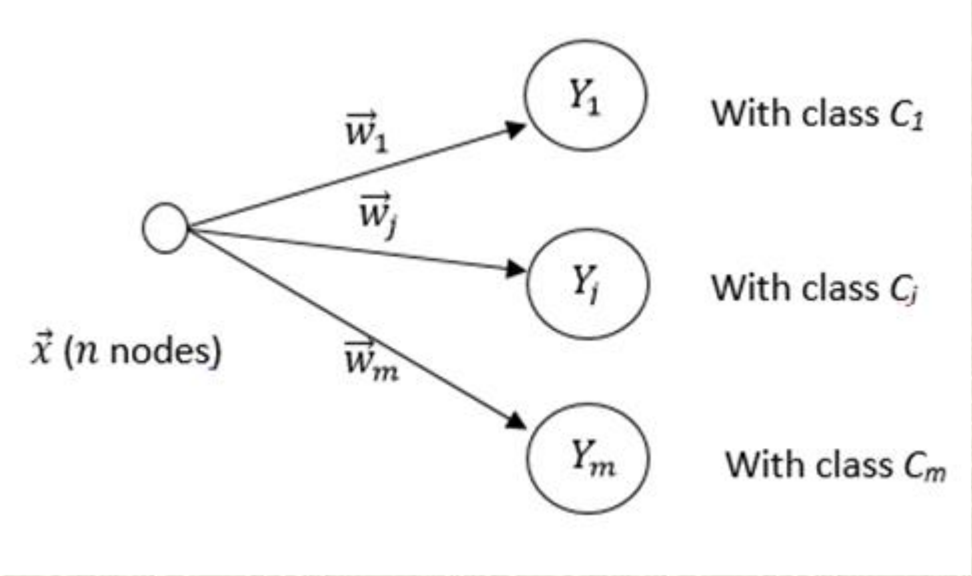
- Updating weight of winner neuron Y_k in LVQ1

$$\Delta\vec{w}_k = \begin{cases} \alpha(\vec{x} - \vec{w}_k), \text{if } \vec{x} \text{ truly classified: } \vec{w}_k \text{ moves toward } \vec{x} \\ -\alpha(\vec{x} - \vec{w}_k), \text{if } \vec{x} \text{ misclassified: } \vec{w}_k \text{ moves away } \vec{x} \end{cases}$$

LVQ

mansoori@shirazu.ac.ir

- After training, net classifies an input vector $\vec{x}$ by assigning it class of output neuron with weight vector closest to $\vec{x}$



Weight initialization methods in LVQ

1. Assigning initial weights and classes randomly, provided class of units are distinct

LVQ

mansoori@shirazu.ac.ir

Weight initialization methods in LVQ

2. Using first m training vectors of different classes for weights and remaining vectors for training

3. Using SOM
   - Repeatedly presenting patterns to net and finding maximally responding neuron (finding clusters and using their centers as initial weights of neurons)
   - Assigning class of each neuron according to a majority vote of patterns in that cluster

Algorithm of LVQ1

mansoori@shirazu.ac.ir

1. Initialize

   1.1. Weight vectors as code-book vectors, $\vec{w}_j$, and class label of neurons, $C_j \ (j = 1, \dots, m)$

   1.2. Learning rate, α

2. While weight changes are noticeable

Algorithm of LVQ1

mansoori@shirazu.ac.ir

2.1. For each training input vector $<\vec{s}; t>$ (chosen randomly)

2.1.1. Find the best matching (winning) unit Y_k whose weight vector $\vec{w}_k$ is closest to $\vec{s}$

$$k = \underset{j}{\operatorname{argmin}}(\|\vec{s} - \vec{w}_j\|)$$

2.1.2. Update $\vec{w}_k$ as

$$\Delta\vec{w}_k = \begin{cases} \alpha(\vec{s} - \vec{w}_k), & \text{if } t = C_k \\ -\alpha(\vec{s} - \vec{w}_k), & \text{if } t \neq C_k \end{cases}$$

2.2. Decrease learning rate

3. Stop

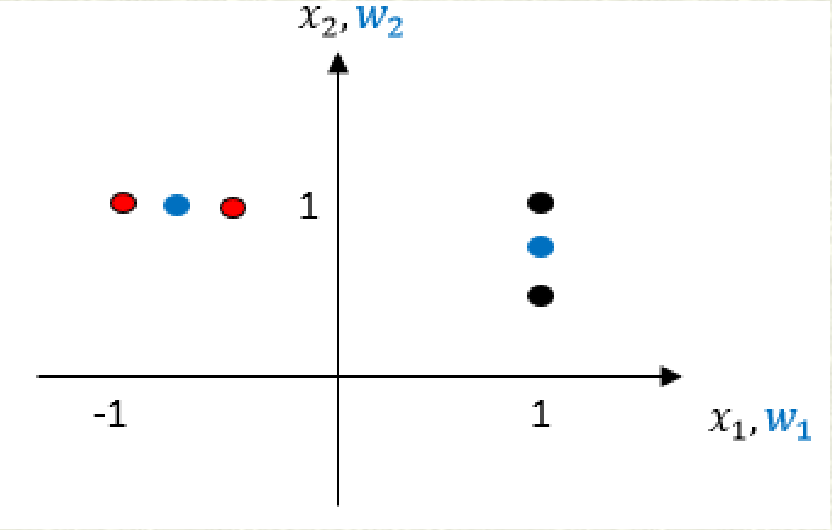
LVQ Example

mansoori@shirazu.ac.ir

$< \vec{s}, t >=< 1, 1; 1 >, < 1, 0.5; 1 >, < -0.5, 1; 2 >, < -1, 1; 2 >\}$

$\vec{w}_1 =< 1, 0.75 >$, $\vec{w}_2 =< -0.75, 1 >$,

$C_1 = 1,$ $C_2 = 2,$ $\alpha = 0.2,$



$\vec{w}_1$	$\vec{w}_2$	$\vec{s}$	t	$\|\vec{s}-\vec{w}_1\|$	$\|\vec{s}-\vec{w}_2\|$	k	C_k	$\Delta\vec{w}_k$
<1, 0.75>	<-0.75, 1>	<1,1>	1	0.25	1.75	1	1	<0,0.05>
<1, 0.8>	<-0.75, 1>	<-1,1>	2	2.01	0.25	2	2	<-0.05,0>
<1, 0.8>	<-0.8, 1>	<-0.5,1>	2	1.51	0.3	2	2	<0.06,0>
<1, 0.8>	<-0.74, 1>	<1,0.5>	1	0.3	1.81	1	1	<0,-0.06>
<1, 0.74>	<-0.74, 1>							

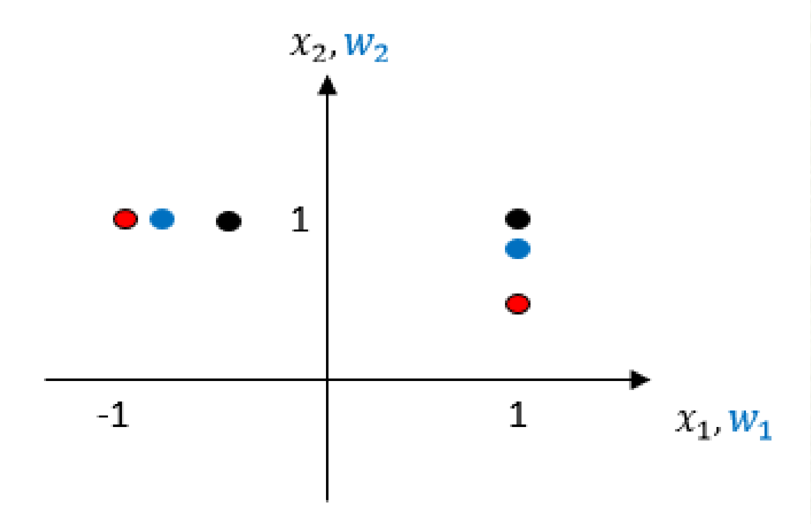
LVQ Example

mansoori@shirazu.ac.ir

$< \vec{s}, t > = < 1, 1; 1 >, < 1, 0.5; 2 >, < -0.5, 1; 1 >, < -1, 1; 2 >\}$

$\vec{w}_1 = < 1, 0.75 >$, $\vec{w}_2 = < -0.75, 1 >$,

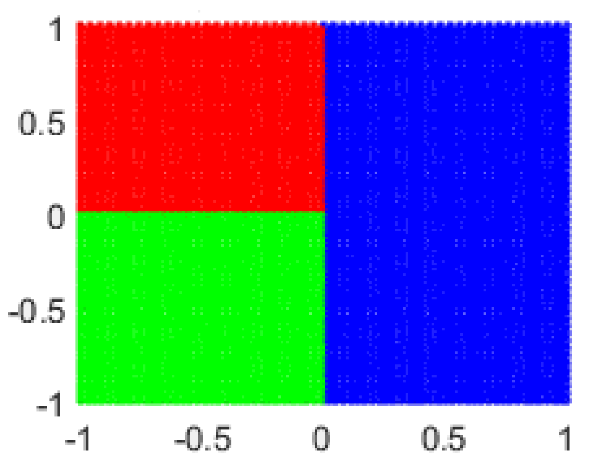
$C_1 = 1$, $C_2 = 2$, $\alpha = 0.2$,



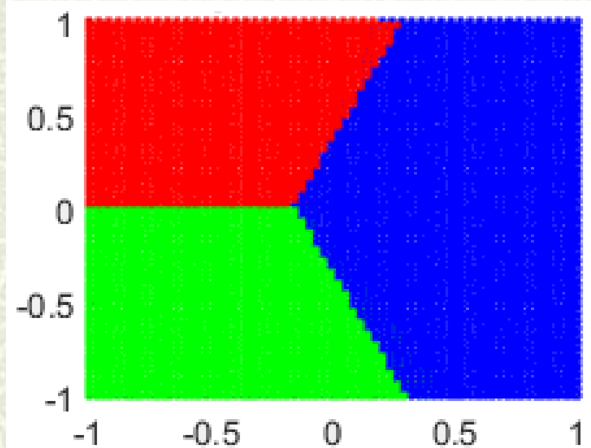
$\vec{w}_1$	$\vec{w}_2$	$\vec{s}$	t	$\|\vec{s} - \vec{w}_1\|$	$\|\vec{s} - \vec{w}_2\|$	k	C_k	$\Delta\vec{w}_k$
<1, 0.75>	<-0.75, 1>	<1,1>	1	0.25	1.75	1	1	<0,0.05>
<1, 0.8>	<-0.75, 1>	<-1,1>	2	2.01	0.25	2	2	<-0.05,0>
<1, 0.8>	<-0.8, 1>	<-0.5,1>	1	1.51	0.3	2	2	<-0.06,0>
<1, 0.8>	<-0.86, 1>	<1,0.5>	2	0.3	1.93	1	1	<0,0.06>
<1, 0.86>	<-0.86, 1>							

LVQ Examples

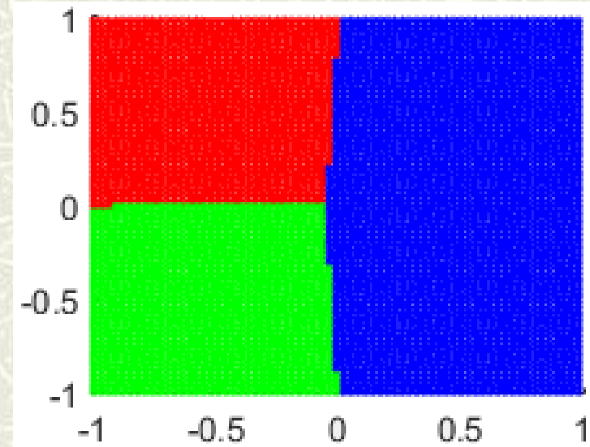
mansoori@shirazu.ac.ir



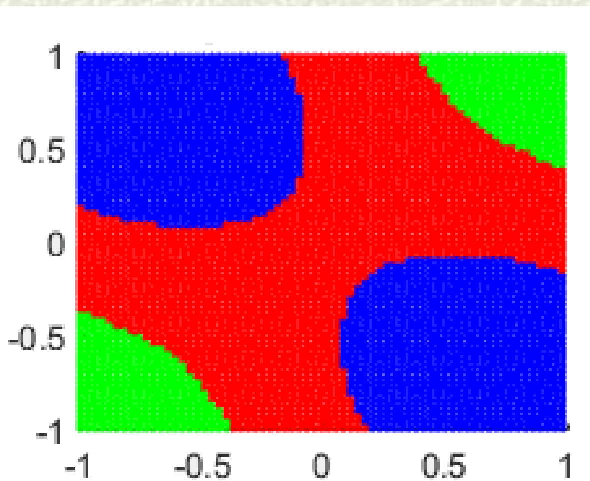
Input data distribution



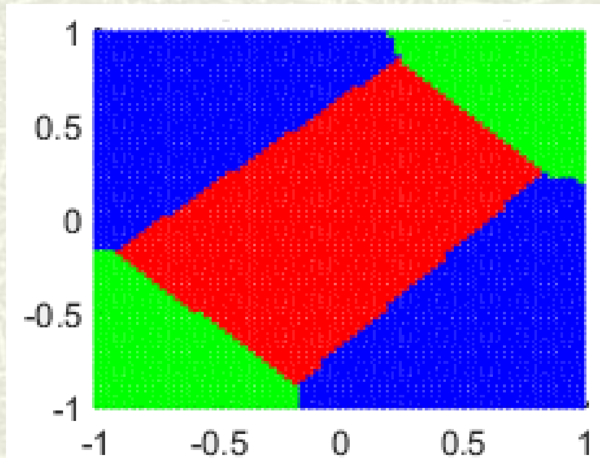
Output of 3 neurons



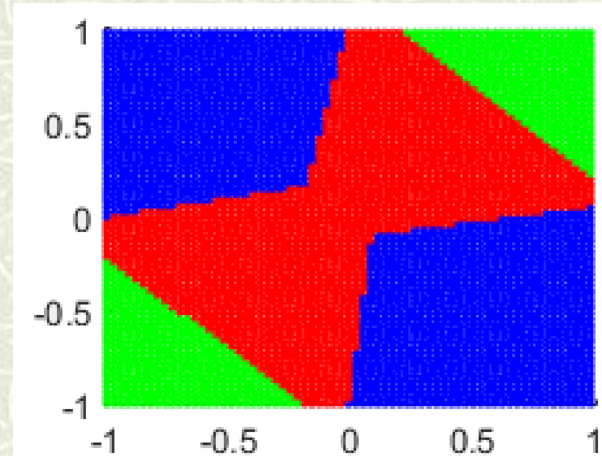
Output of 6 neurons



Input data distribution



Output of 5 neurons



Output of 10 neurons