Self Organizing Maps

Unsupervised Learning

In **unsupervised competitive learning** the neurons take part in some competition for each input. The winner of the competition and sometimes some other neurons are allowed to change their weights

- In **simple competitive learning** only the winner is allowed to learn (change its weight).
- In **self-organizing maps** other neurons in the neighborhood of the winner may also learn.

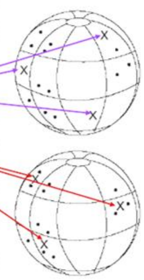
Competitive Neural Networks

- Competitive networks
  - cluster
  - encode
  - classify

  data by identifying
  - vectors which logically belong to the same category
  - vectors that share similar properties
- Competitive learning algorithms use competition between lateral neurons in a layer (via lateral interconnections) to provide selectivity (or localization) of the learning process

Example of CL

- Three clusters of vectors (denoted by solid dots) distributed on the unit sphere
- Initially randomized codebook vectors (crosses) move under influence of a competitive learning rule to approximate the centroids of the clusters
- Competitive learning schemes use codebook vectors to approximate centroids of data clusters



Principle of Competitive Learning

- Given a sequence of stochastic vectors $X_k \in \Re^n$ drawn from a possibly unknown distribution, each pattern X_k is compared with a set of initially randomized weight vectors $W_j \in \Re^n$ and the vector W_J which best matches X_k is to be updated to match X_k more closely

Inner Product vs Euclidean Distance Based Competition

- Inner Product

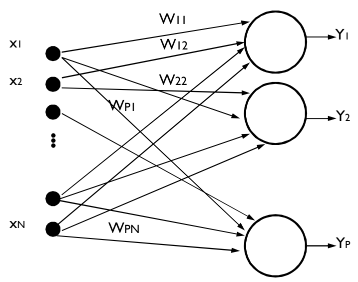
$$y_J = \max_j \{X_k^T W_j\}$$

- Euclidean Distance Based Competition

$$\|X_k - W_J\| = \min_j \{\|X_k - W_j\|\}$$

Simple Competitive Learning

N inputs units
P output neurons
P x N weights



$Y_i = 1 \text{ or } 0$

Learning

- Starting with small random weights, at each step:
- a new input vector is presented to the network
- all fields are calculated to find a winner
- $\vec{W}_{i^*}$ is updated to be closer to the input
- Using standard competitive learning equ.

$$\Delta W_{i^*j} = \eta(X_j - W_{i^*j})$$

Example

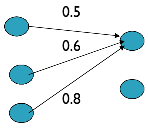
An SOM network with three inputs and two cluster units is to be trained using the four training vectors:

[0.8 0.7 0.4], [0.6 0.9 0.9], [0.3 0.4 0.1], [0.1 0.1 02] and

initial weights

$$\begin{bmatrix} 0.5 & 0.4 \\ 0.6 & 0.2 \\ 0.8 & 0.5 \end{bmatrix}$$

weights to the first cluster unit



The initial radius is 0 and the learning rate η is 0.5 . Calculate the weight changes during the first cycle through the data, taking the training vectors in the given order.

DJSCOE\BE(Extc)SemVII\NNFL\Vishakha Kelkar

Solution

The Euclidian distance of the input vector 1 to cluster unit 1 is:

$$d_1 = (0.5-0.8)^2 + (0.6-0.7)^2 + (0.8-0.4)^2 = 0.26$$

The Euclidian distance of the input vector 1 to cluster unit 2 is:

$$d_2 = (0.4-0.8)^2 + (0.2-0.7)^2 + (0.5-0.4)^2 = 0.42$$

Input vector 1 is closest to cluster unit 1 so update weights to cluster unit 1:

$$\left.\begin{aligned} w_{ij}(n+1) &= w_{ij}(n) + 0.5[x_i - w_{ij}(n)] \\ 0.65 &= 0.5 + 0.5(0.8-0.5) \\ 0.65 &= 0.6 + 0.5(0.7-0.6) \\ 0.6 &= 0.8 + 0.5(0.4-0.8) \end{aligned}\right\} \begin{bmatrix} 0.65 & 0.4 \\ 0.65 & 0.2 \\ 0.60 & 0.5 \end{bmatrix}$$

DJSCOE\BE(Extc)SemVII\NNFL\Vishakha Kelkar

Solution

The Euclidian distance of the input vector 2 to cluster unit 1 is:

$$d_1 = (0.65-0.6)^2 + (0.65-0.9)^2 + (0.6-0.9)^2 = 0.155$$

The Euclidian distance of the input vector 2 to cluster unit 2 is:

$$d_2 = (0.4-0.6)^2 + (0.2-0.9)^2 + (0.5-0.9)^2 = 0.69$$

Input vector 2 is closest to cluster unit 1 so update weights to cluster unit 1 again:

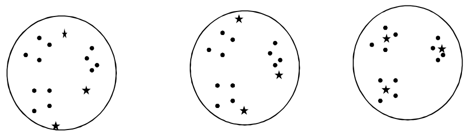
$$\left.\begin{aligned} w_{ij}(n+1) &= w_{ij}(n) + 0.5[x_i - w_{ij}(n)] \\ 0.625 &= 0.65 + 0.5(0.6-0.65) \\ 0.775 &= 0.65 + 0.5(0.9-0.65) \\ 0.750 &= 0.60 + 0.5(0.9-0.60) \end{aligned}\right\} \begin{bmatrix} 0.625 & 0.4 \\ 0.775 & 0.2 \\ 0.750 & 0.5 \end{bmatrix}$$

Repeat the same update procedure for input vector 3 and 4 also.

DJSCOE\BE(Extc)SemVII\NNFL\Vishakha Kelkar

Result

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



DJSCOE\BE(Extc)SemVII\NNFL\Vishakha Kelkar

Self Organized Map (SOM)

- The **self-organizing map** (SOM) is a method for unsupervised learning, based on a grid of artificial neurons whose weights are adapted to match input vectors in a training set.
- It was first described by the Finnish professor Teuvo Kohonen and is thus sometimes referred to as a **Kohonen map**.
- SOM is one of the most popular neural computation methods in use, and several thousand scientific articles have been written about it. SOM is especially good at producing visualizations of high-dimensional data.

Self Organizing Maps (SOM)

- SOM is an unsupervised neural network technique that approximates an unlimited number of input data by a finite set of models arranged in a grid, where neighbor nodes correspond to more similar models.
- The models are produced by a learning algorithm that automatically orders them on the two-dimensional grid along with their mutual similarity.

Features Of SOM ?

- Unsupervised Learning
- Clustering
- Classification
- Monitoring
- Data Visualization
- Potential for combination between SOM and other neural network (MLP-RBF)

SOM: interpretation

- **Each** SOM **neuron** can be seen as representing a cluster containing all the input examples which are mapped to that neuron.
- For a given input, the output of SOM is the neuron with weight vector most similar (with respect to Euclidean distance) to that input.

Simple Model

- Network has inputs and outputs
- There is no feedback from the environment ➔ no supervision
- The network updates the weights following some learning rule, and finds patterns, features or categories within the inputs presented to the network

Self Organizing Networks

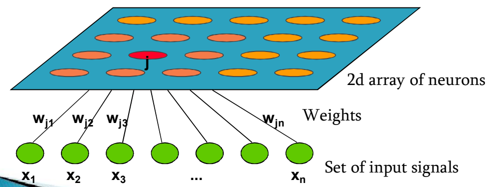
- Discover significant patterns or features in the input data
- Discovery is done without a teacher
- Synaptic weights are changed according to local rules
- The changes affect a neuron's immediate environment until a final configuration develops

Network Architecture

- Two layers of units
  - Input: *n* units (length of training vectors)
  - Output: *m* units (number of categories)
- Input units fully connected with weights to output units
- Intralayer (lateral) connections
  - Within output layer
  - Defined according to some topology
  - Not weights, but used in algorithm for updating weights

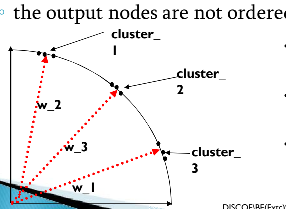
SOM - Architecture

- Lattice of neurons ('nodes') accept and respond to set of input signals
- Responses compared; 'winning' neuron selected from lattice
- Selected neuron activated *together with 'neighbourhood' neurons*
- Adaptive process changes weights to more closely inputs



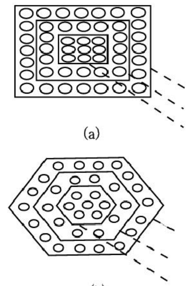
Self-Organizing Maps (SOM)

- Competitive learning (Kohonen 1982) is a special case of SOM (Kohonen 1989)
- In competitive learning,
  - the network is trained to organize input vector space into subspaces/classes/clusters
  - each output node corresponds to one class
  - the output nodes are not ordered: ***random map***



- The topological order of the three clusters is 1, 2, 3
- The order of their maps at output nodes are 2, 3, 1
- The map does not preserve the topological order of the training vectors

Measuring distances between nodes



(a)

(b)

- Distances between output neurons will be used in the learning process.
- It may be based upon:

a) Rectangular lattice
b) Hexagonal lattice

- Let d(i,j) be the distance between the output nodes i,j
- d(i,j) = 1 if node j is in the first outer rectangle/hexagon of node i
- d(i,j) = 2 if node j is in the second outer rectangle/hexagon of node i
- And so on..

More about SOM learning

- Upon repeated presentations of the training examples, the weight vectors of the neurons tend to follow the distribution of the examples.
- This results in a topological ordering of the neurons, where neurons adjacent to each other tend to have similar weight vectors.
- The input space of patterns is mapped onto a discrete output space of neurons.

SOM – Learning Algorithm

1. Randomly initialise all weights
2. Select input vector $\mathbf{x} = [x_1, x_2, x_3, \dots, x_n]$ from training set
3. Compare $\mathbf{x}$ with weights $\mathbf{w}_j$ for each neuron j to

$$d_j = \sum_i (w_{ij} - x_i)^2$$

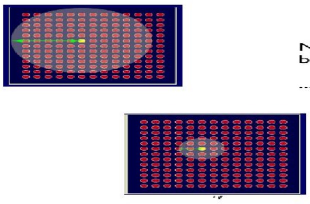
4. determine winner
   find unit j with the minimum distance
5. Update winner so that it becomes more like $\mathbf{x}$, together with the winner also upadate winner's *neighbours* within the radius

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

6. Adjust parameters: learning rate & 'neighbourhood function'
7. Repeat from (2) until …

Note that: Learning rate generally decreases with time:

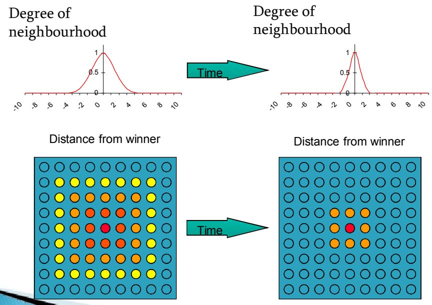
$$0 < \eta(n) \leq \eta(n-1) \leq 1$$



$N_{13}(1) = \{8, 12, 13, 14, 18\}$ ← Two possible neighborhoods of neuron 13

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

Neighbourhood function



UPDATE RULE

$$w_j(n+1) = w_j(n) + \eta(n)\, h_{ij(x)}(n)\, \big(\mathrm{x} - w_j(n)\big)$$

exponential decay update of the learning rate:

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{T_2}\right)$$

Two-phases learning approach

- Self-organizing or ordering phase. The learning rate and spread of the Gaussian neighborhood function are adapted during the execution of SOM, using for instance the exponential decay update rule.
- Convergence phase. The learning rate and Gaussian spread have small fixed values during the execution of SOM.

Ordering Phase

- Self organizing or ordering phase:
  - Topological ordering of weight vectors.
  - May take 1000 or more iterations of SOM algorithm.
- Important choice of the parameter values. For instance
  - $\eta(n)$: $\eta_0 = 0.1$ $T_2 = 1000$
    $\Rightarrow$ decrease gradually $\eta(n) \geq 0.01$
  - $h_{ji(x)}(n)$:
- With this parameter setting initially the neighborhood of the winning neuron includes almost all neurons in the network, then it shrinks slowly with time.

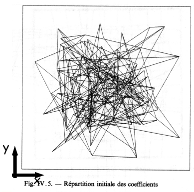
Convergence Phase

- Convergence phase:
  - Fine tune the weight vectors.
  - Must be at least 500 times the number of neurons in the network $\Rightarrow$ thousands or tens of thousands of iterations.
- Choice of parameter values:
  - $\eta(n)$ maintained on the order of 0.01.
  - Neighborhood function such that the neighbor of the winning neuron contains only the nearest neighbors. It eventually reduces to one or zero neighboring neurons.

Illustration of learning for Kohonen maps

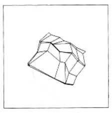
Inputs: coordinates (x,y) of points drawn from a square

Display neuron j at position x_j,y_j where its s_j is maximum

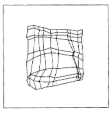


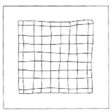
Random initial positions

100 inputs

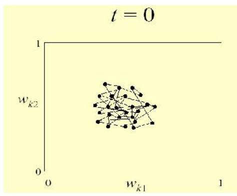


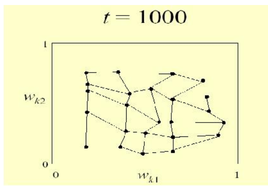
200 inputs

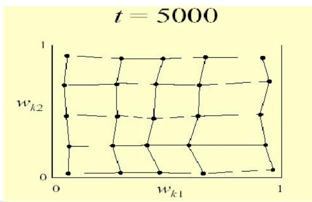




1000 inputs







Examples

- A simple example of competitive learning
  - 6 vectors of dimension 3 in 3 classes, node ordering: B – A – C

    $\boldsymbol{i_1 = (1.1, 1.7, 1.8)}$ $\quad$ $\boldsymbol{i_2 = (0, 0, 0)}$

    $\boldsymbol{i_3 = (0, 0.5, 1.5)}$ $\quad$ $\boldsymbol{i_4 = (1, 0, 0)}$

    $\boldsymbol{i_5 = (0.5, 0.5, 0.5)}$ $\quad$ $\boldsymbol{i_6 = (1, 1, 1)}$
  - Initialization: $\eta = 0.5$, weight matrix: $W(0) = \begin{bmatrix} w_A: & 0.2 & 0.7 & 0.3 \\ w_B: & 0.1 & 0.1 & 0.9 \\ w_C: & 1 & 1 & 1 \end{bmatrix}$
  - D(t) = 1 for the first epoch, = 0 afterwards
  - Training with $i_1 = (1.1, 1.7, 1.8)$

    determine winner: squared Euclidean distance between i_1 and w_j

    $d_{A,1}^2 = (1.1-0.2)^2 + (1.7-0.7)^2 + (1.8-0.3)^2 = 4.1$

    $d_{B,1}^2 = 4.4, \; d_{C,1}^2 = 1.1$

    - C wins, since D(t) = 1, weights of node C and its neighbor A are updated, but not w_B

Examples

$$W(0) = \begin{bmatrix} w_A: & 0.2 & 0.7 & 0.3 \\ w_B: & 0.1 & 0.1 & 0.9 \\ w_C: & 1 & 1 & 1 \end{bmatrix} \qquad W(1) = \begin{bmatrix} w_A: & 0.65 & 1.2 & 1.05 \\ w_B: & 0.1 & 0.1 & 0.9 \\ w_C: & 1.05 & 1.35 & 1.4 \end{bmatrix}$$

$$W(15) = \begin{bmatrix} w_A: & 0.83 & 0.77 & 0.81 \\ w_B: & 0.47 & 0.23 & 0.30 \\ w_C: & 0.61 & 0.95 & 1.34 \end{bmatrix}$$

	$W(0)$	$W(15)$
$\lvert w_A - w_B \rvert$	$= 0.85$	1.28
$\lvert w_B - w_C \rvert$	$= 1.28$	1.75
$\lvert w_C - w_A \rvert$	$= 1.22$	0.80

- Observations:
  - Distance between weights of non-neighboring nodes (B, C) increase
  - Input vectors switch allegiance between nodes, especially in the early stage of training

t	A	B	C
1 – 6	(3,5)	(2,4)	(1,6)
7 – 12	(6)	(2,4,5)	(1,3)
13 – 16	(6)	(2,4,5)	(1,3)

Another Self-Organizing Map (SOM) Example

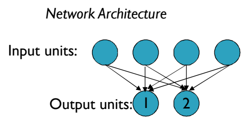
- From Fausett (1994)
- n = 4, m = 2
  - More typical of SOM application
  - Smaller number of units in output than in input; *dimensionality reduction*
- Training samples

  i1: (1, 1, 0, 0)

  i2: (0, 0, 0, 1)

  i3: (1, 0, 0, 0)

  i4: (0, 0, 1, 1)



What should we expect as outputs?

What are the Euclidean Distances Between the Data Samples?

- Training samples

  i1: (1, 1, 0, 0)

  i2: (0, 0, 0, 1)

  i3: (1, 0, 0, 0)

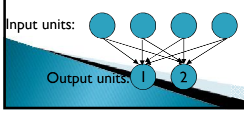
  i4: (0, 0, 1, 1)

	i1	i2	i3	i4
i1	0			
i2		0		
i3			0	
i4				0

Euclidean Distances Between Data Samples

- Training samples
  - i1: (1, 1, 0, 0)
  - i2: (0, 0, 0, 1)
  - i3: (1, 0, 0, 0)
  - i4: (0, 0, 1, 1)

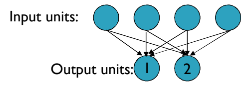
	i1	i2	i3	i4
i1	0			
i2	3	0		
i3	1	2	0	
i4	4	1	3	0



What might we expect from the SOM?

DJSCOE\BE(Extc)SemVII\NNFL\Vishakha Kelkar

Example Details

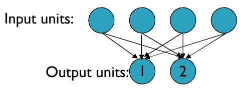


- Training samples
  i1: (1, 1, 0, 0)
  i2: (0, 0, 0, 1)
  i3: (1, 0, 0, 0)
  i4: (0, 0, 1, 1)
- With only 2 outputs, neighborhood = 0
  - Only update weights associated with winning output unit (cluster) at each iteration
- Learning rate
  $\eta(t) = 0.6;\ 1 <= t <= 4$
  $\eta(t) = 0.5\ \eta(1);\ 5 <= t <= 8$
  $\eta(t) = 0.5\ \eta(5);\ 9 <= t <= 12$
  etc.

Problem: Calculate the weight updates for the first four steps

DJSCOE\BE(Extc)SemVII\NNFL\Vishakha Kelkar

Example Details



Initial Weights

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .2 & .6 & .5 & .9 \\ .8 & .4 & .7 & .3 \end{bmatrix}$$

$$d^2 = (\text{Euclidean distance})^2 = \sum_{k=1}^{n} (i_{l,k} - w_{j,k}(t))^2$$

Weight update: $$w_j(t+1) = w_j(t) + \eta(t)(i_l - w_j(t))$$

Problem: Calculate the weight updates for the first four steps

DJSCOE\BE(Extc)SemVII\NNFL\Vishakha Kelkar

First Weight Update

i1: (1, 1, 0, 0)
i2: (0, 0, 0, 1)
i3: (1, 0, 0, 0)
i4: (0, 0, 1, 1)

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .2 & .6 & .5 & .9 \\ .8 & .4 & .7 & .3 \end{bmatrix}$$

- Training sample: i1
  - Unit 1 weights
    - $d^2 = (.2\text{-}1)^2 + (.6\text{-}1)^2 + (.5\text{-}0)^2 + (.9\text{-}0)^2 = 1.86$
  - Unit 2 weights
    - $d^2 = (.8\text{-}1)^2 + (.4\text{-}1)^2 + (.7\text{-}0)^2 + (.3\text{-}0)^2 = .98$
  - Unit 2 wins
  - Weights on winning unit are updated

$$new-unit2-weights = [.8\ \ .4\ \ .7\ \ .3] + 0.6([1\ 1\ 0\ 0] - [.8\ \ .4\ \ .7\ \ .3]) =$$
$$[.92\ .76\ .28\ .12]$$

  - Giving an updated weight matrix:

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .2 & .6 & .5 & .9 \\ .92 & .76 & .28 & .12 \end{bmatrix}$$

DJSCOE\BE(Extc)SemVII\NNFL\Vishakha Kelkar

Second Weight Update

i1: (1, 1, 0, 0)
i2: (0, 0, 0, 1)
i3: (1, 0, 0, 0)
i4: (0, 0, 1, 1)

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .2 & .6 & .5 & .9 \\ .92 & .76 & .28 & .12 \end{bmatrix}$$

- Training sample: i2
  - Unit 1 weights
    - $d^2 = (.2-0)^2 + (.6-0)^2 + (.5-0)^2 + (.9-1)^2 = .66$
  - Unit 2 weights
    - $d^2 = (.92-0)^2 + (.76-0)^2 + (.28-0)^2 + (.12-1)^2 = 2.28$
  - Unit 1 wins
  - Weights on winning unit are updated

$$new-unit1-weights = [.2 \quad .6 \quad .5 \quad .9] + 0.6([0 \quad 0 \quad 0 \quad 1] - [.2 \quad .6 \quad .5 \quad .9]) = [.08 \quad .24 \quad .20 \quad .96]$$

  - Giving an updated weight matrix:

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .08 & .24 & .20 & .96 \\ .92 & .76 & .28 & .12 \end{bmatrix}$$

Third Weight Update

i1: (1, 1, 0, 0)
i2: (0, 0, 0, 1)
i3: (1, 0, 0, 0)
i4: (0, 0, 1, 1)

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .08 & .24 & .20 & .96 \\ .92 & .76 & .28 & .12 \end{bmatrix}$$

- Training sample: i3
  - Unit 1 weights
    - $d^2 = (.08-1)^2 + (.24-0)^2 + (.2-0)^2 + (.96-0)^2 = 1.87$
  - Unit 2 weights
    - $d^2 = (.92-1)^2 + (.76-0)^2 + (.28-0)^2 + (.12-0)^2 = 0.68$
  - Unit 2 wins
  - Weights on winning unit are updated

$$new-unit2-weights = [.92 \quad .76 \quad .28 \quad .12] + 0.6([1 \quad 0 \quad 0 \quad 0] - [.92 \quad .76 \quad .28 \quad .12]) = [.97 \quad .30 \quad .11 \quad .05]$$

  - Giving an updated weight matrix:

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .08 & .24 & .20 & .96 \\ .97 & .30 & .11 & .05 \end{bmatrix}$$

Fourth Weight Update

i1: (1, 1, 0, 0)
i2: (0, 0, 0, 1)
i3: (1, 0, 0, 0)
i4: (0, 0, 1, 1)

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .08 & .24 & .20 & .96 \\ .97 & .30 & .11 & .05 \end{bmatrix}$$

- Training sample: i4
  - Unit 1 weights
    - $d^2 = (.08-0)^2 + (.24-0)^2 + (.2-1)^2 + (.96-1)^2 = .71$
  - Unit 2 weights
    - $d^2 = (.97-0)^2 + (.30-0)^2 + (.11-1)^2 + (.05-1)^2 = 2.74$
  - Unit 1 wins
  - Weights on winning unit are updated

$$new-unit1-weights = [.08 \quad .24 \quad .20 \quad .96] + 0.6([0 \quad 0 \quad 1 \quad 1] - [.08 \quad .24 \quad .20 \quad .96]) = [.03 \quad .10 \quad .68 \quad .98]$$

  - Giving an updated weight matrix:

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} .03 & .10 & .68 & .98 \\ .97 & .30 & .11 & .05 \end{bmatrix}$$

Applying the SOM Algorithm

Data sample utilized

time (t)	1	2	3	4	D(t)	η(t)
1	Unit 2				0	0.6
2		Unit 1			0	0.6
3			Unit 2		0	0.6
4				Unit 1	0	0.6

'winning' output unit

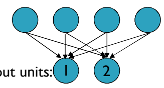
After many iterations (epochs) through the data set:

$$\begin{array}{l} \text{Unit 1:} \\ \text{Unit 2:} \end{array} \begin{bmatrix} 0 & 0 & .5 & 1.0 \\ 1.0 & .5 & 0 & 0 \end{bmatrix}$$

Did we get the clustering that we expected?

Training samples
i1: (1, 1, 0, 0)
i2: (0, 0, 0, 1)
i3: (1, 0, 0, 0)
i4: (0, 0, 1, 1)

Input units:



Output units:

Weights

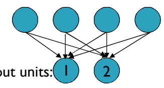
Unit 1: $\begin{bmatrix} 0 & 0 & .5 & 1.0 \end{bmatrix}$
Unit 2: $\begin{bmatrix} 1.0 & .5 & 0 & 0 \end{bmatrix}$

What clusters do the data samples fall into?

Solution

Training samples
i1: (1, 1, 0, 0)
i2: (0, 0, 0, 1)
i3: (1, 0, 0, 0)
i4: (0, 0, 1, 1)

Input units:



Output units:

Weights

Unit 1: $\begin{bmatrix} 0 & 0 & .5 & 1.0 \end{bmatrix}$
Unit 2: $\begin{bmatrix} 1.0 & .5 & 0 & 0 \end{bmatrix}$

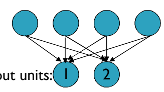
- Sample: i1
  - Distance from unit1 weights
    - $(1\text{-}0)^2 + (1\text{-}0)^2 + (0\text{-}.5)^2 + (0\text{-}1.0)^2 = 1+1+.25+1=3.25$
  - Distance from unit2 weights
    - $(1\text{-}1)^2 + (1\text{-}.5)^2 + (0\text{-}0)^2 + (0\text{-}0)^2 = 0+.25+0+0=.25$ (winner)
- Sample: i2
  - Distance from unit1 weights
    - $(0\text{-}0)^2 + (0\text{-}0)^2 + (0\text{-}.5)^2 + (1\text{-}1.0)^2 = 0+0+.25+0$ (winner)
  - Distance from unit2 weights
    - $(0\text{-}1)^2 + (0\text{-}.5)^2 + (0\text{-}0)^2 + (1\text{-}0)^2 = 1+.25+0+1=2.25$

d^2 = (Euclidean distance)^2 = $\sum_{k=1}^{n} (i_{l,k} - w_{j,k}(t))^2$

Solution

Training samples
i1: (1, 1, 0, 0)
i2: (0, 0, 0, 1)
i3: (1, 0, 0, 0)
i4: (0, 0, 1, 1)

Input units:



Output units:

Weights

Unit 1: $\begin{bmatrix} 0 & 0 & .5 & 1.0 \end{bmatrix}$
Unit 2: $\begin{bmatrix} 1.0 & .5 & 0 & 0 \end{bmatrix}$

- Sample: i3
  - Distance from unit1 weights
    - $(1\text{-}0)^2 + (0\text{-}0)^2 + (0\text{-}.5)^2 + (0\text{-}1.0)^2 = 1+0+.25+1=2.25$
  - Distance from unit2 weights
    - $(1\text{-}1)^2 + (0\text{-}.5)^2 + (0\text{-}0)^2 + (0\text{-}0)^2 = 0+.25+0+0=.25$ (winner)
- Sample: i4
  - Distance from unit1 weights
    - $(0\text{-}0)^2 + (0\text{-}0)^2 + (1\text{-}.5)^2 + (1\text{-}1.0)^2 = 0+0+.25+0$ (winner)
  - Distance from unit2 weights
    - $(0\text{-}1)^2 + (0\text{-}.5)^2 + (1\text{-}0)^2 + (1\text{-}0)^2 = 1+.25+1+1=3.25$

d^2 = (Euclidean distance)^2 = $\sum_{k=1}^{n} (i_{l,k} - w_{j,k}(t))^2$