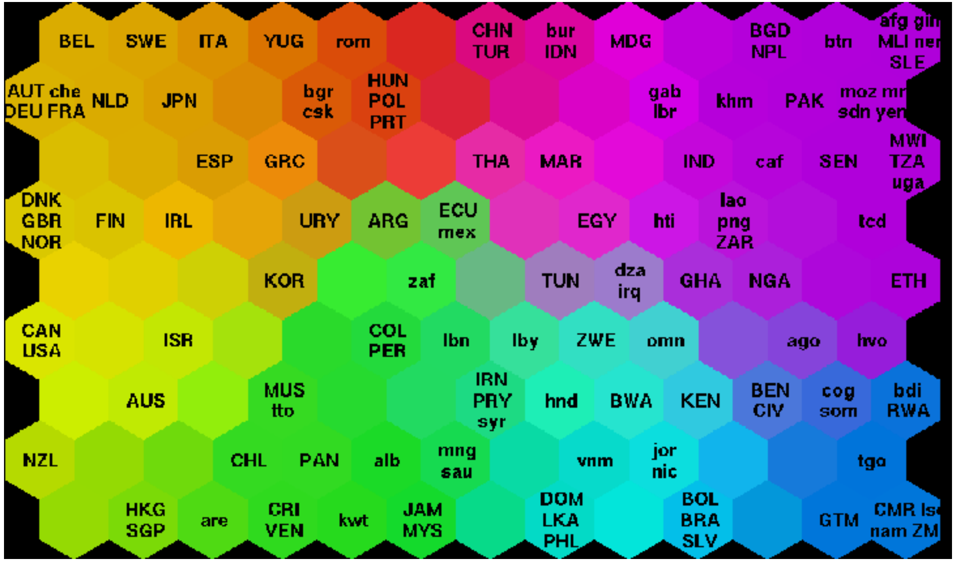
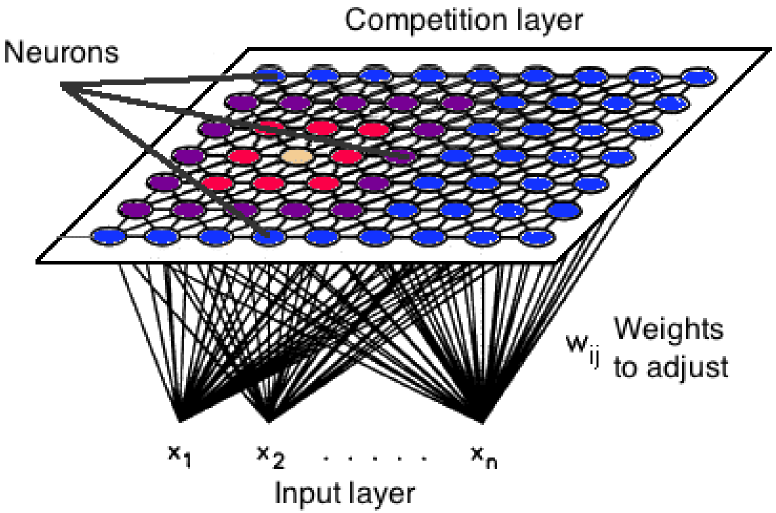


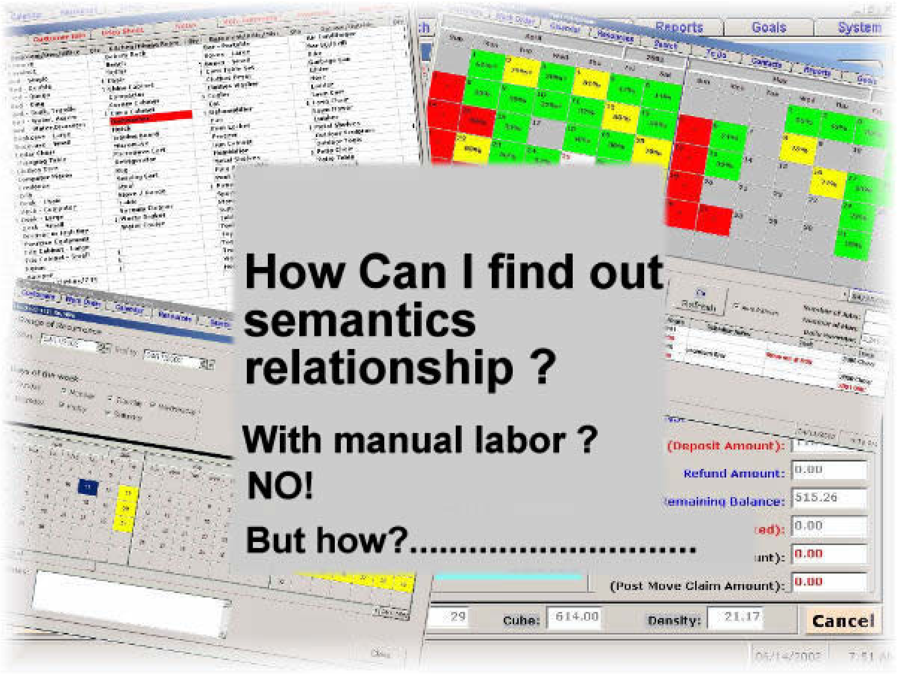


Self Organizing Map (SOM)

Kazi Shah Nawaz Ripon
ksripon@ntnu.com

Motivation

- How to find out semantics relationship among lots of information without manual labor?

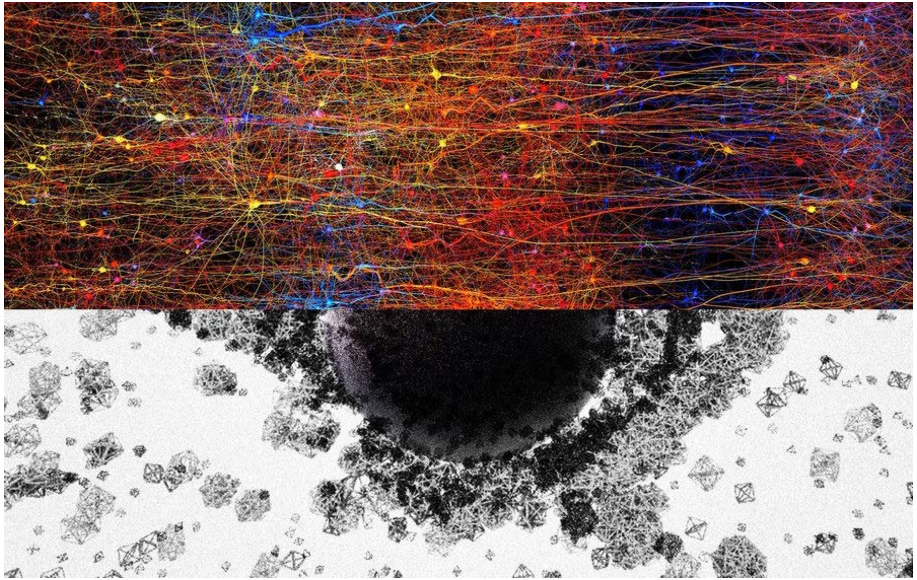


Motivation

- Humans can categorize data using less than 1 percent of the original information [1].

[1] Georgia Institute of Technology. "How the brain can handle so much data." ScienceDaily. www.sciencedaily.com/releases/2015/12/151215160649.htm (accessed October 30, 2017).

Motivation



- "How do we make sense of so much data around us, of so many different types, so quickly and robustly?"
- our brain creates neural structures with **up to 11 dimensions** when it processes information (Blue Brain Project [1]).

[1] https://bluebrain.epfl.ch/

Self-organizing Maps

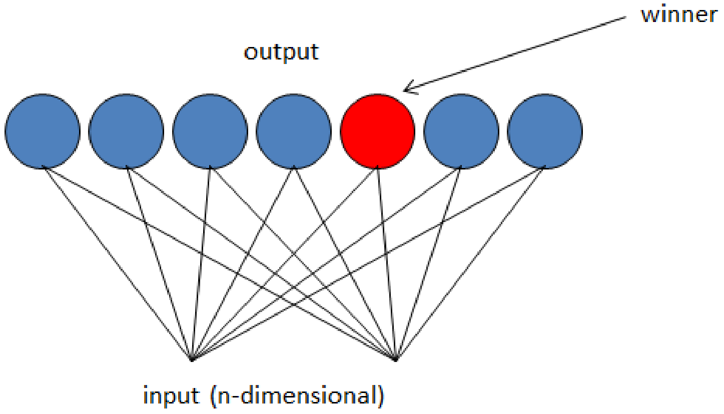
- Unsupervised learning neural network
- Maps multidimensional data onto a 2 dimensional grid
- Use a neighborhood function to preserve the topological properties of the input space.

Background

- Supervised training → Target output for each input pattern.
- Unsupervised training → Networks learn to form their own classifications of the training data without external help.
    - Common features.
    - Follow the neuro-biological organization of the brain.

Unsupervised Competitive Learning

- SOMs are based on unsupervised competitive learning.
  - Winner-takes all neuron.
  - lateral inhibition connections
- The basic idea of competitive learning was introduced in the early 1970s.
- Ideas first introduced by C. von der Malsburg (1973), developed and refined by T. Kohonen (1982)
- Biological basis: ‘brain maps’
- Primarily used for organization and visualization of complex data



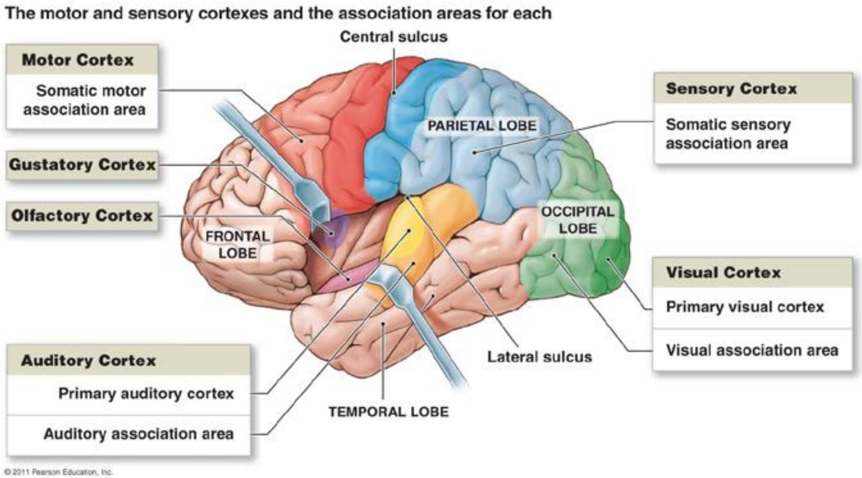
Topographic Maps

- Different sensory inputs (motor, visual, auditory, etc.) are mapped onto corresponding areas of the cerebral cortex in an **orderly fashion**.

- This **topographic map**, has two important properties:

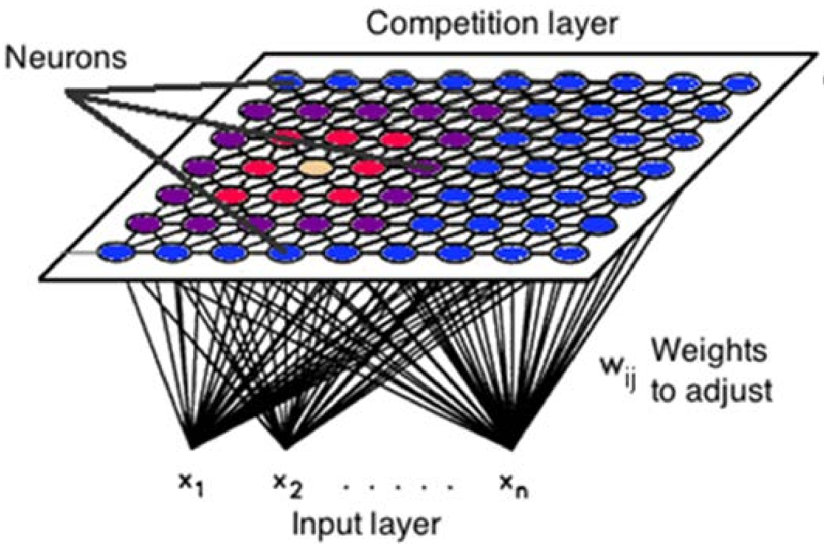
1. At each stage of representation, or processing, each piece of incoming information is kept in its proper context/neighbourhood.

2. Neurons dealing with closely related pieces of information are kept close together so that they can interact via short synaptic connections.



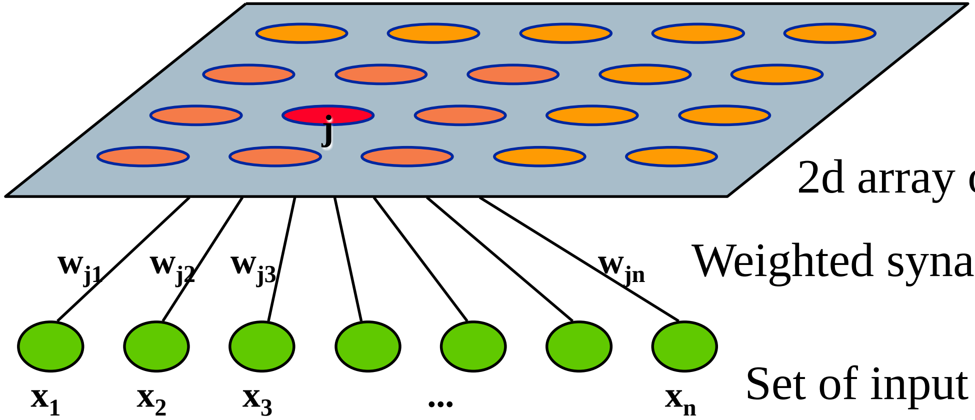
SOM: Network Architecture

- Two layers of units
  - Input: *n* units (length of training vectors)
  - Output: *m* units (# 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

- Input patterns are shown to all neurons simultaneously
- Competitive learning: the neuron with the largest response is chosen
- Selected neuron activated together with ‘neighbourhood’ neurons
- Adaptive process changes weights to more closely resemble inputs



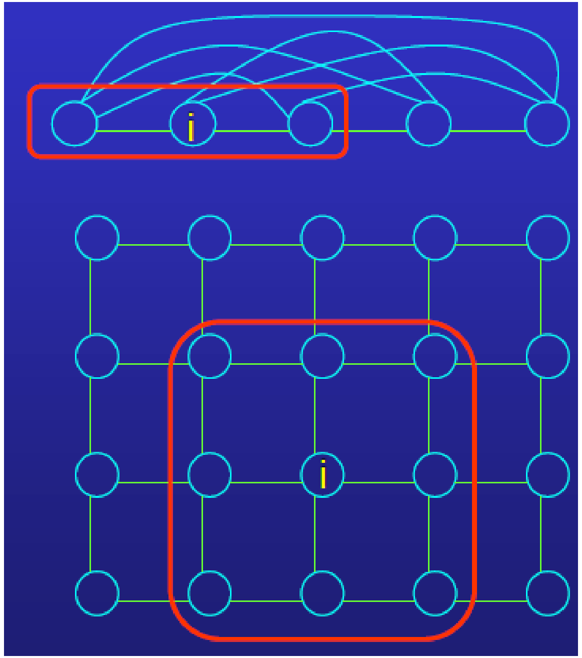
2d array of neurons

Weighted synapses

Set of input signals
(**connected to *all* neurons in lattice**)

Common Output-layer Structures

- Each grid point represents a output node
- The grid is initialized with random vectors



One-dimensional

(completely interconnected for determining "winner" unit)

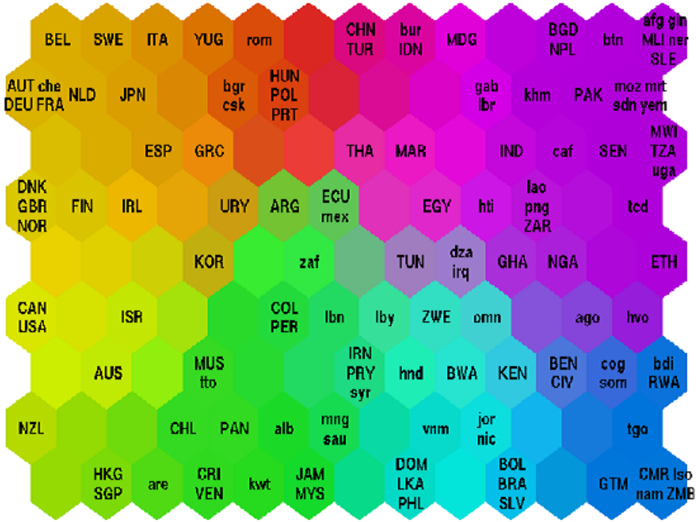
Two-dimensional

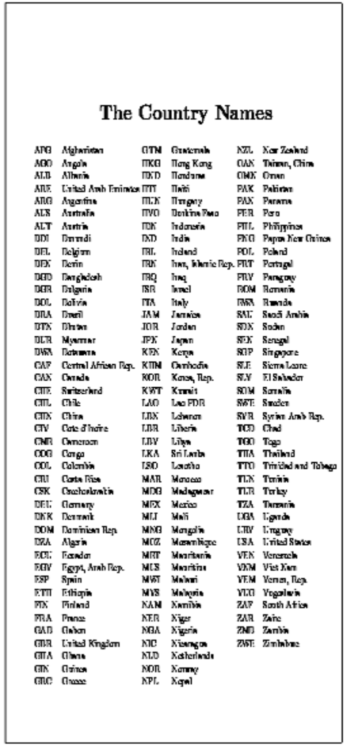
(connections omitted, only neighborhood relations shown [green])

Neighborhood of neuron *i*

SOM – Result Example

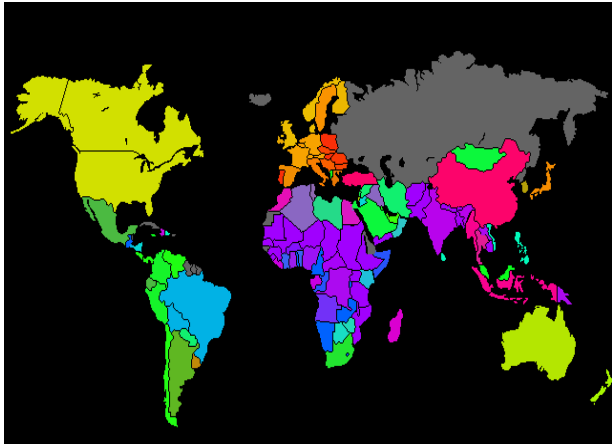
‘Poverty map’ based on 39 indicators from World Bank statistics (1992)





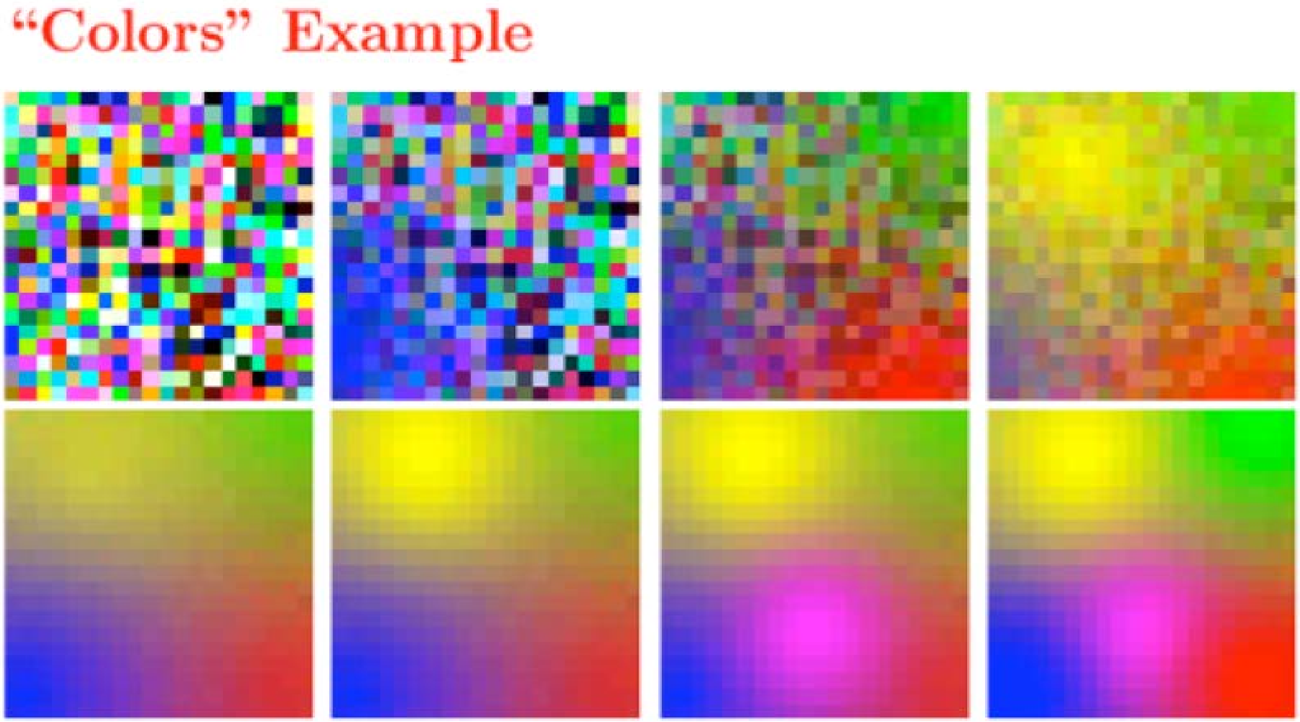
The Country Names

AFG	Afghanistan	GTM	Guatemala	NZL	New Zealand
AGO	Angola	HKG	Hong Kong	OAN	Taiwan, China
ALB	Albania	HND	Honduras	OMN	Oman
ARE	United Arab Emirates	HTI	Haiti	PAK	Pakistan
ARG	Argentina	HUN	Hungary	PAN	Panama
AUS	Australia	HVO	Burkina Faso	PER	Peru
AUT	Austria	IDN	Indonesia	PHL	Philippines
BDI	Burundi	IND	India	PNG	Papua New Guinea
BEL	Belgium	IRL	Ireland	POL	Poland
BEN	Benin	IRN	Iran, Islamic Rep.	PRT	Portugal
BGD	Bangladesh	IRQ	Iraq	PRY	Paraguay
BGR	Bulgaria	ISR	Israel	ROM	Romania
BOL	Bolivia	ITA	Italy	RWA	Rwanda
BRA	Brazil	JAM	Jamaica	SAU	Saudi Arabia
BTN	Bhutan	JOR	Jordan	SDN	Sudan
BUR	Myanmar	JPN	Japan	SEN	Senegal
BWA	Botswana	KEN	Kenya	SGP	Singapore
CAF	Central African Rep.	KHM	Cambodia	SLE	Sierra Leone
CAN	Canada	KOR	Korea, Rep.	SLV	El Salvador
CHE	Switzerland	KWT	Kuwait	SOM	Somalia
CHL	Chile	LAO	Lao PDR	SWE	Sweden
CHN	China	LBN	Lebanon	SYR	Syrian Arab Rep.
CIV	Cote d'Ivoire	LBR	Liberia	TCD	Chad
CMR	Cameroon	LBY	Libya	TGO	Togo
COG	Congo	LKA	Sri Lanka	THA	Thailand
COL	Colombia	LSO	Lesotho	TTO	Trinidad and Tobago
CRI	Costa Rica	MAR	Morocco	TUN	Tunisia
CSK	Czechoslovakia	MDG	Madagascar	TUR	Turkey
DEU	Germany	MEX	Mexico	TZA	Tanzania
DNK	Denmark	MLI	Mali	UGA	Uganda
DOM	Dominican Rep.	MNG	Mongolia	URY	Uruguay
DZA	Algeria	MOZ	Mozambique	USA	United States
ECU	Ecuador	MRT	Mauritania	VEN	Venezuela
EGY	Egypt, Arab Rep.	MUS	Mauritius	VNM	Viet Nam
ESP	Spain	MWI	Malawi	YEM	Yemen, Rep.
ETH	Ethiopia	MYS	Malaysia	YUG	Yugoslavia
FIN	Finland	NAM	Namibia	ZAF	South Africa
FRA	France	NER	Niger	ZAR	Zaire
GAB	Gabon	NGA	Nigeria	ZMB	Zambia
GBR	United Kingdom	NIC	Nicaragua	ZWE	Zimbabwe
GHA	Ghana	NLD	Netherlands		
GIN	Guinea	NOR	Norway		
GRC	Greece	NPL	Nepal		

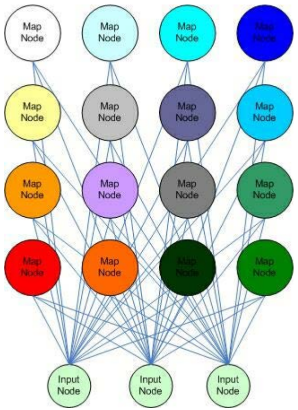


SOM – Result Example

Map representation of 5 initial samples: blue, yellow, red, green, magenta



SOM – Algorithm Overview



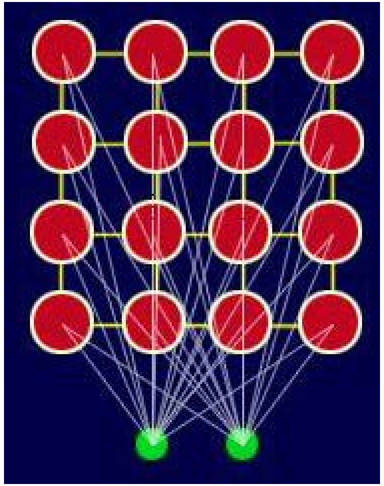
1. Randomly initialise all weights
2. Select input vector $\boldsymbol{x} = [x_1, x_2, x_3, \ldots, x_n]$
3. Compare $\boldsymbol{x}$ with weights $\boldsymbol{w}_j$ for each neuron j to determine winner
4. Update winner so that it becomes more like $\boldsymbol{x}$, together with the winner's *neighbours*

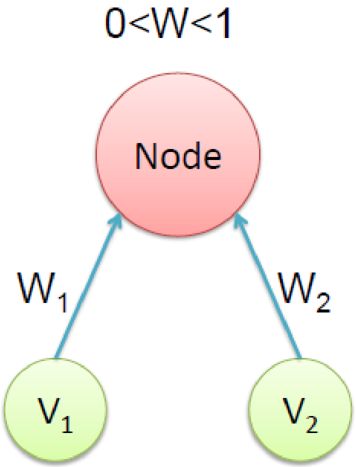
$$w_{ij}(n+1) = w_{ij}(n) + \eta(n)[x_i - w_{ij}(n)]$$

5. Adjust parameters: *learning rate* & *neighbourhood function*
6. Repeat from (2) until the map has converged (no noticeable changes in the weights / pre-defined no. of training cycles)

NB: Learning rate generally decreases with time: $0 < \eta(n) \leq \eta(n-1) \leq 1$

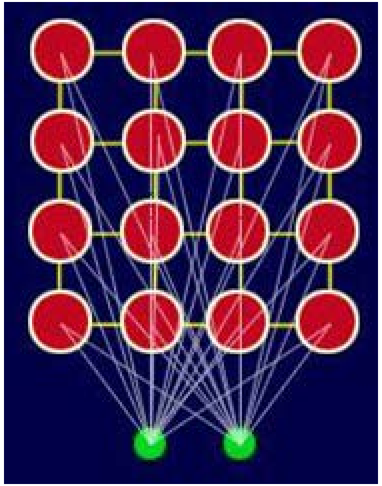
Initialisation





- Prior to training, each node's weights must be initialized
- Randomly initialise the weight vectors $\boldsymbol{w}_j$ for all nodes j
- Typically these will be set to small standardized random values.

Choose A Random Vector



TEMP	HUMIDITY
85	85
80	90
83	78
70	96
68	80
65	70
64	65
72	95
69	70
75	80
75	70
72	90
81	75
...	...

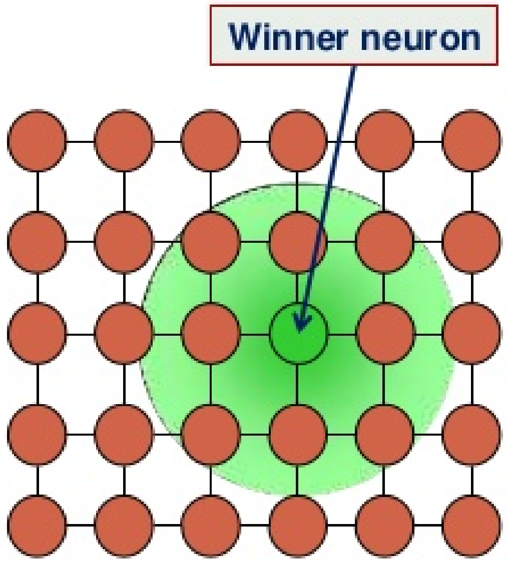
- A vector is chosen at random from the set of training data and presented to the lattice.

Calculating the Best Matching Unit (BMU)

- Calculating the BMU can be done differently among the node's weights (***W_1, W_2, … , W_n***) and the input vector's values (***V_1, V_2, … , V_n***):
  - Nearest neighbor
  - Farthest neighbor
  - Distance between means
  - Distance between medians
- Most common method is Euclidean distance.

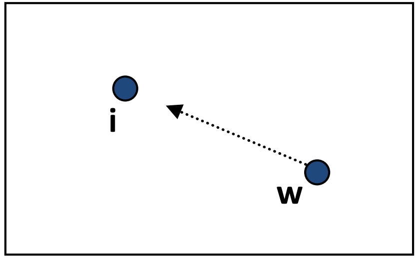
$$\sqrt{\sum_{i=0}^{n} x_i^{\,2}}$$

- More than one contestant, choose randomly

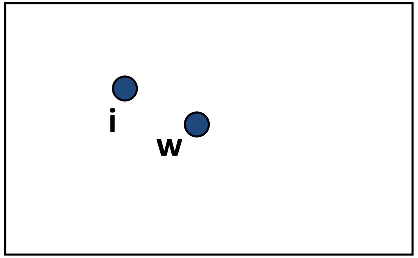


Winner - BMU

- Move the weight vector $\boldsymbol{w}$ of the winning neuron towards the input $\boldsymbol{i}$



Before learning



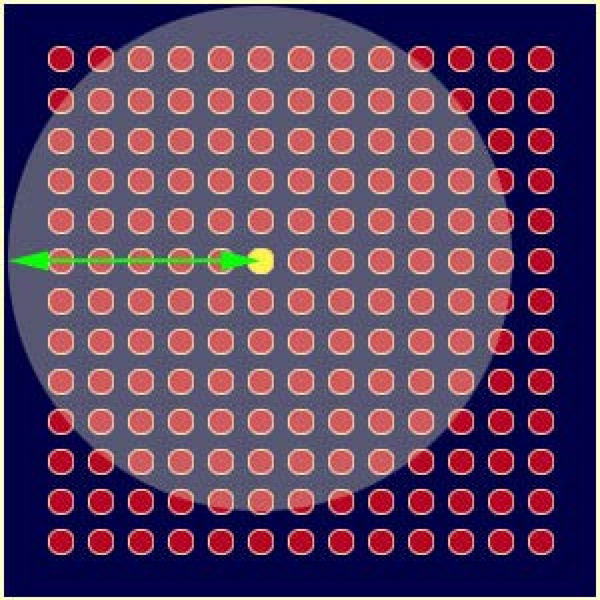
After learning

Determining BMU Neighborhood

- To calculate which of the other nodes are within the BMU's neighbourhood.
  - All these nodes will have their weight vectors adjusted in the next step.

- The area of the neighbourhood shrinks over time until eventually the neighborhood is just the BMU itself.

- Several Methods.
  - Static
  - Exponential
  - Linear



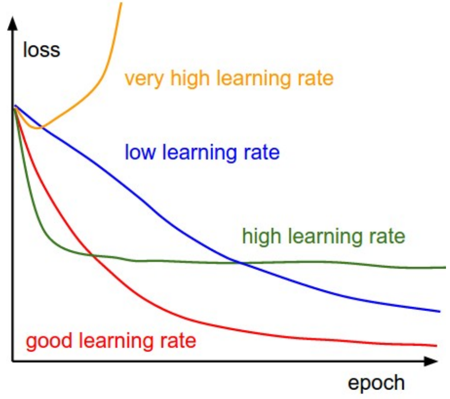
Exponential Decay Function

Time (iteration of the loop)

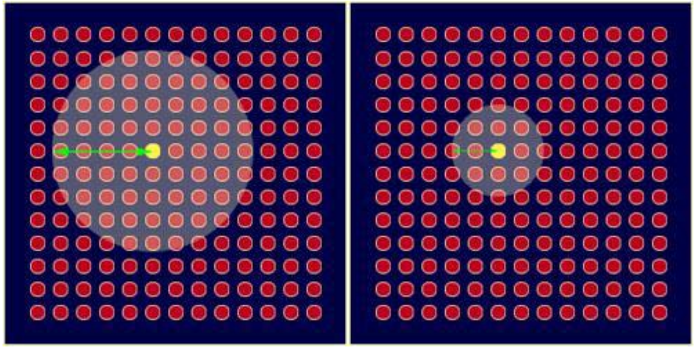
$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right) \qquad t = 1,2,3\ldots$$

Time constant

Width of neighborhood at time t_0

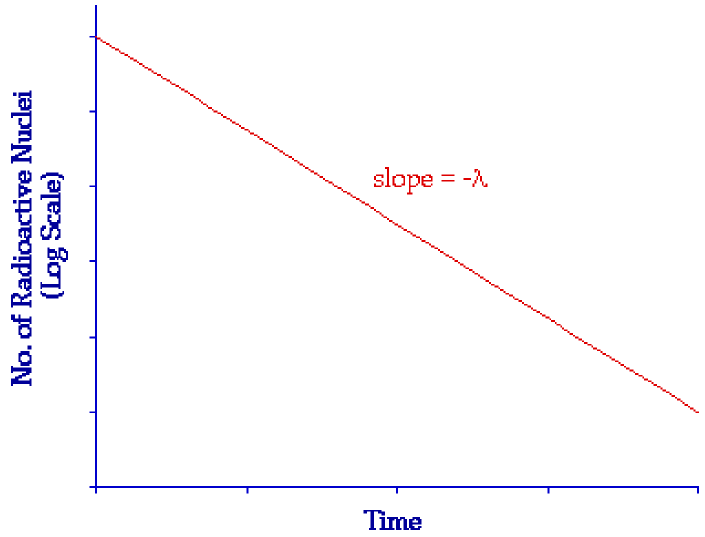


neighborhood shrinks on each iteration

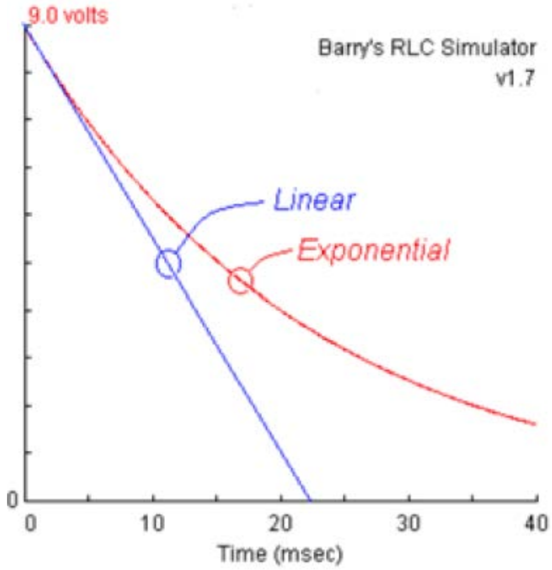


Linear Decay Function

$$\sigma(t) = \sigma_0 + \lambda t\,, \qquad \lambda \langle 0, \quad t = 1,2,3...$$

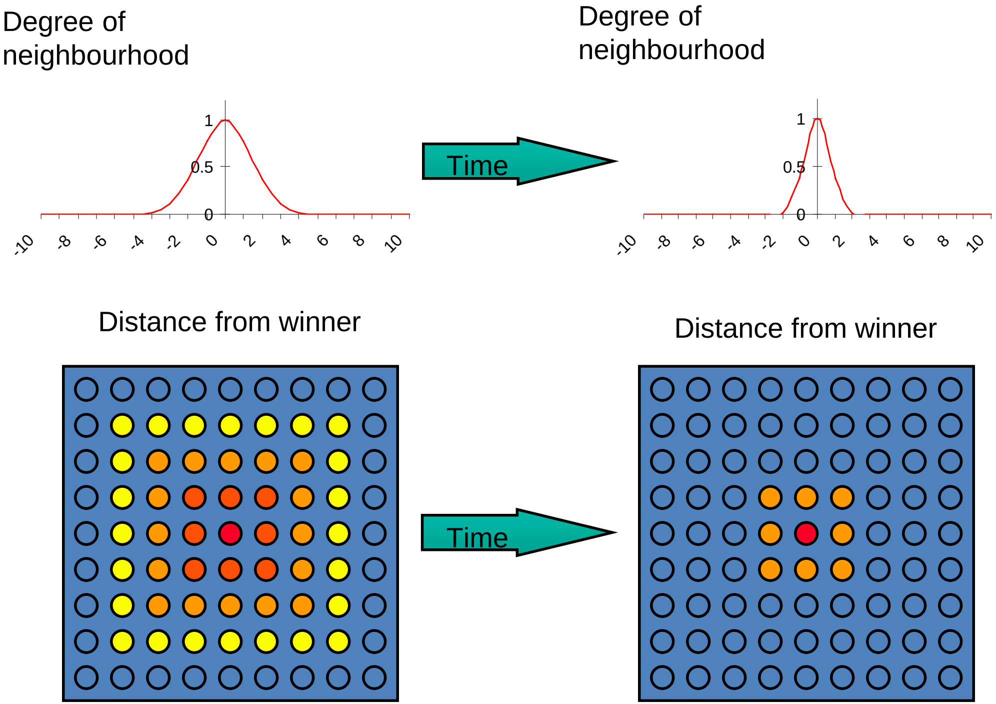


Linear vs Exponential



how does a capacitor discharge?

Neighbourhood Function



Adjusting the Weights

- Every node within the BMU's neighbourhood (including the BMU) has its weight vector adjusted as:

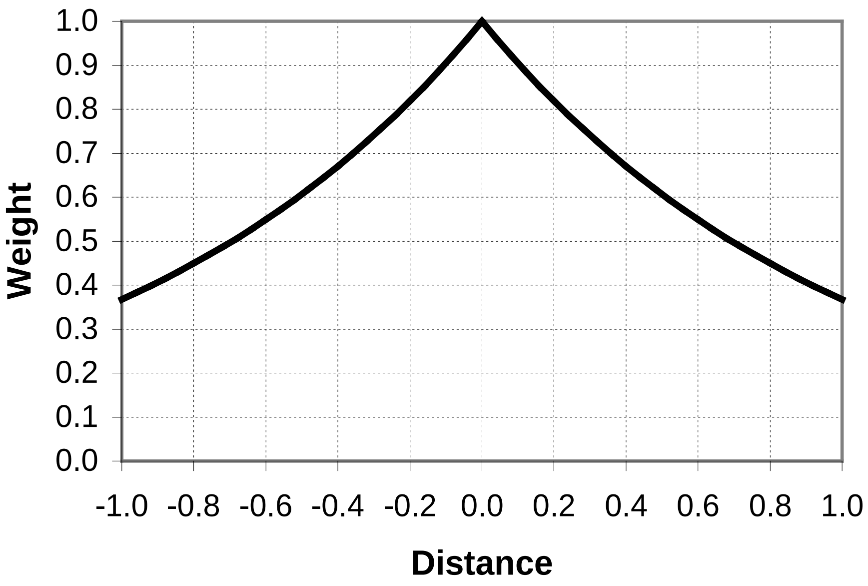
$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) h_{ij(x)} [x_i - w_{ij}(t)]$$

A B C D

Current learning rate (B) $\times$ Degree of neighbourhood with respect to winner (C) $\times$
Difference between current weights and input vector (D)
to the current weights (A)”

Weighting of the Neighborhood ($h_{ij(x)}$)

Weighting decreases exponentially



–x-axis shows distance from winning node

–y-axis shows ‘degree of neighbourhood’ (max. 1)

Adjusting Learning Rate

- The learning rate is also an exponential decay function.
  - This ensures that the SOM will converge.

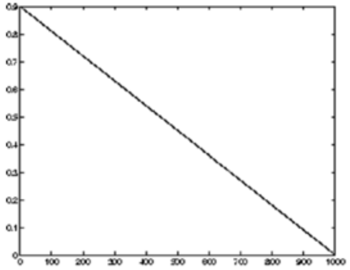
Time (iteration of the loop)

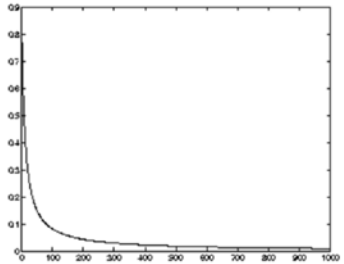
λ

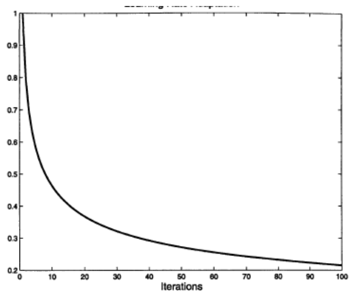
$$\eta(t) = \eta_0 \exp\left(-\frac{t}{\lambda}\right) \qquad t = 1,2,3...$$

Time constant

Width of neighborhood at time t_0







Adjusting Learning Rate

- Linear

$$\alpha(t) = \frac{1}{t}$$

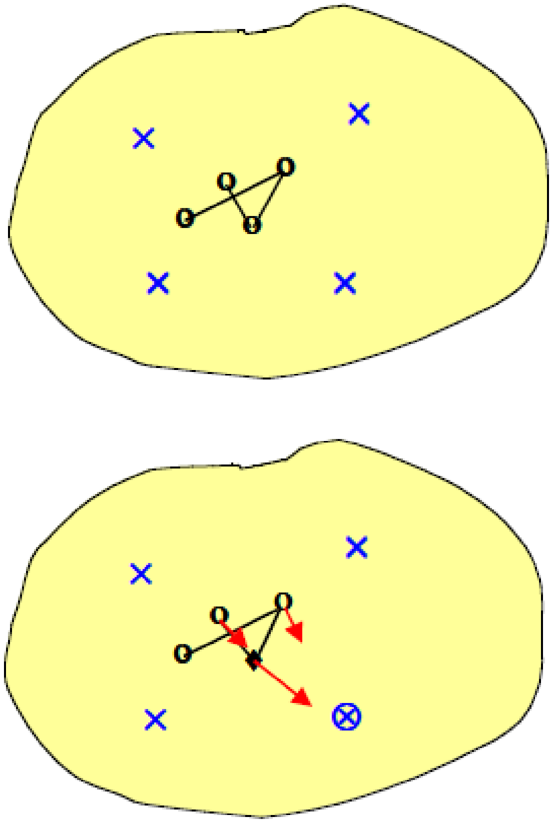
- Inverse-of-time

$$\alpha(t,T) = \left(1 - \frac{t}{T}\right)$$

- Power Series

$$\alpha(t,T) = (0.005)^{\frac{t}{T}}$$

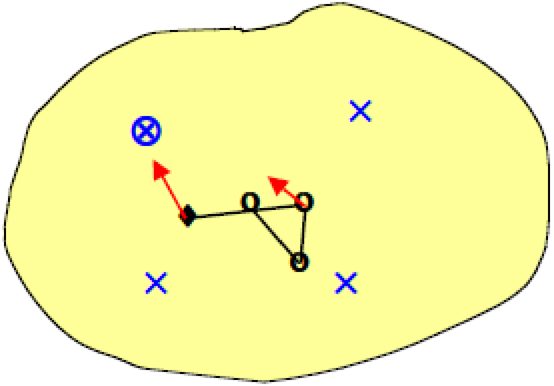
Visualizing the Self Organization Process [1]

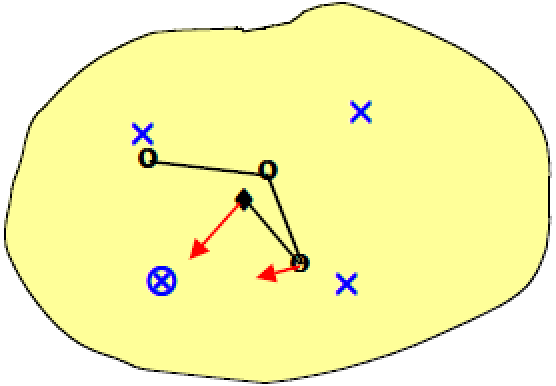


- 4 data points (**x**) in continuous 2D input space.
- Goal: to map four points in a discrete 1D output space.
- The output nodes map to points in the input space (**o**).
- Randomly pick one of the data points for training (⊗).
- The closest output point represents the winning neuron (♦).

[1] http://www.cs.bham.ac.uk/~jxb/NN/l16.pdf

Visualizing the Self Organization Process: Next Steps





Example

- The animals should be ordered by SOM.
- And the animals will be described with their attributes(size, living space).

Size:
small=0 medium=1 big=2

Living space:
Land=0 Water=1 Air=2

e.g. Mouse = (0/0)

	Mouse	Lion	Horse	Shark	Dove
Size	small	medium	big	big	small
Living space	Land	Land	Land	Water	Air
	(0/0)	(1/0)	(2/0)	(2/1)	(0/2)

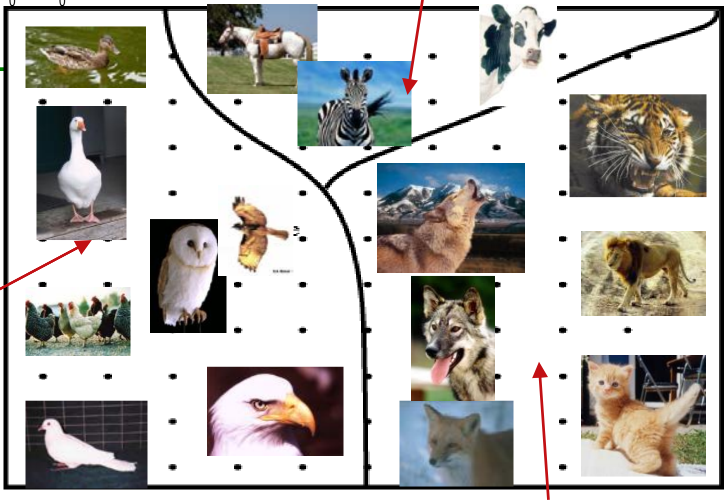
Example

Animal names and their attributes

		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
is	Small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	Medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	Big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	Mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	Feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	Hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	Run	0	0	0	0	0	0	0	0	1	1	0	1	1	1		
	Fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0		
	Swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0		

A grouping according to similarity has emerged

peaceful



birds

hunters

[Teuvo Kohonen 2001] Self-Organizing Maps; Springer;

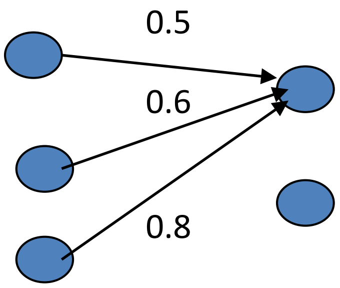
Example

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.

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:

$$w_{ij}(n+1) = w_{ij}(n) + 0.5[x_i - w_{ij}(n)]$$

$$0.65 = 0.5 + 0.5(0.8 - 0.5)$$

$$0.65 = 0.6 + 0.5(0.7 - 0.6)$$

$$0.6 \quad = 0.8 + 0.5(0.4 - 0.8)$$

$$\begin{bmatrix} 0.65 & 0.4 \\ 0.65 & 0.2 \\ 0.60 & 0.5 \end{bmatrix}$$

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.

Another 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

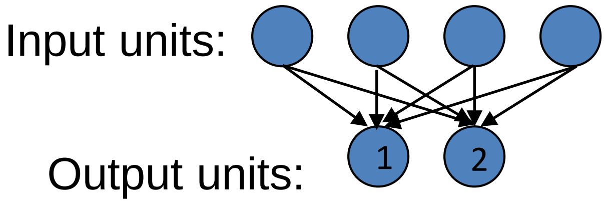
  i1: (1, 1, 0, 0)

  i2: (0, 0, 0, 1)

  i3: (1, 0, 0, 0)

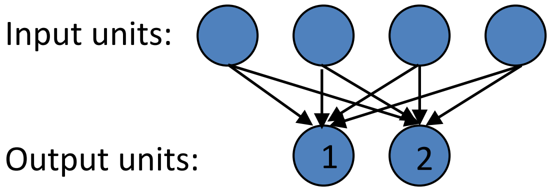
  i4: (0, 0, 1, 1)

Network Architecture



What should we expect as outputs?

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.
- Initial weight matrix

  (random values between 0 and 1)

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

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

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
    - $d^2 = (.2-1)^2 + (.6-1)^2 + (.5-0)^2 + (.9-0)^2 = 1.86$
  - Unit 2 weights
    - $d^2 = (.8-1)^2 + (.4-1)^2 + (.7-0)^2 + (.3-0)^2 = .98$
  - Unit 2 wins
  - Weights on winning unit are updated

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

  - Giving an updated weight matrix:

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

Second Weight Update

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

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

- 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

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

  - Giving an updated weight matrix:

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

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

$$= [.92 \quad .76 \quad .28 \quad .12] + 0.6([1 \; 0 \; 0 \; 0] - [.92 \quad .76 \quad .28 \quad .12]) =$$

$$[.97 \;\; .30 \;\; .11 \;\; .05]$$

  - 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

$$= [.08 \quad .24 \quad .20 \quad .96] + 0.6([0 \;\; 0 \;\; 1 \;\; 1] - [.08 \quad .24 \quad .20 \quad .96]) =$$

$$[.03 \;\; .10 \;\; .68 \;\; .98]$$

  - 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

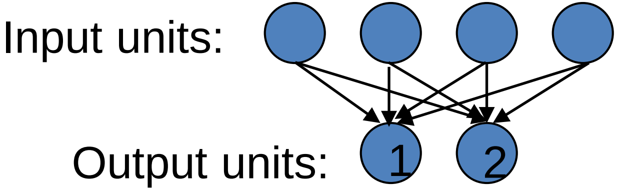
After many iterations (epochs) through the data set:

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

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)



Weights

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

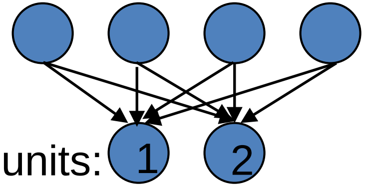
What clusters do the data samples fall into?

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

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

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

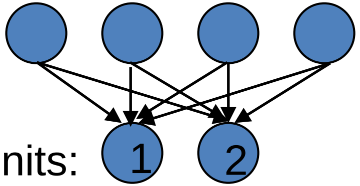
Training samples

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

Solution

Input units:

Output units:



Weights

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

- **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
    - $(0\text{-}1)^2 + (0\text{-}.5)^2 + (1\text{-}0)^2 + (1\text{-}0)^2 = 1+.25+1+1=3.25$

Parameter Setup

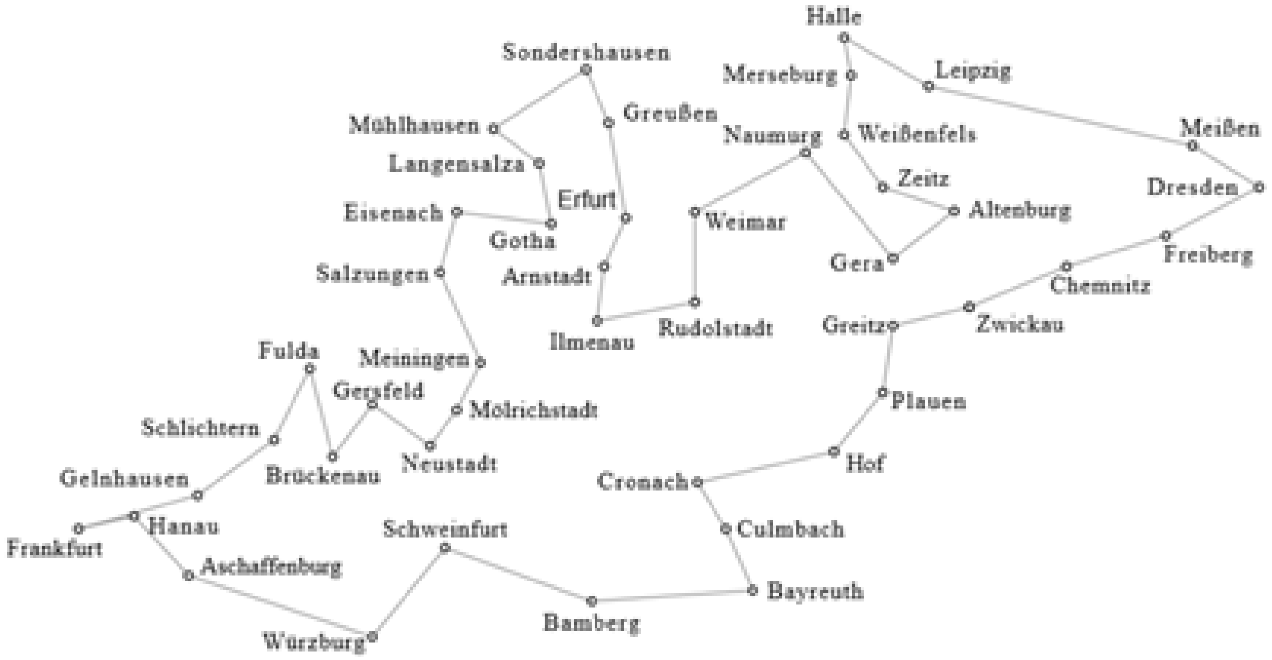
- Number of iterations *T*
  - Convergence of SOM is rather slow $\Rightarrow$ Should be set as high as possible
  - Roughly 100-1000 iterations at minimum.

- Size of the initial neighborhood σ_0
  - Small enough to allow local adaption.
  - $\sigma_0 = 0$ indicates no neighbor structure

- Maximum learning rate $\eta(t_0)$
  - Higher values have mostly random effect.
  - Most critical are the final stages

Optimal choices of σ_0 and $\eta(t_0)$ highly correlated.

SOM in TSP



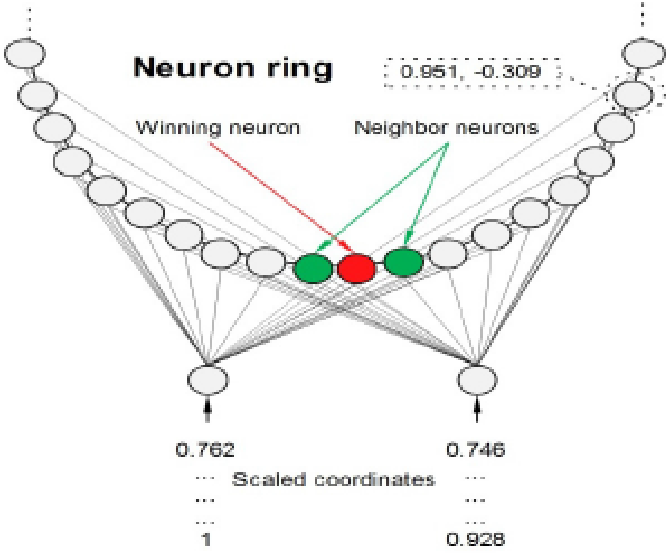
Halle
Sondershausen
Merseburg
Leipzig
Greußen
Mühlhausen
Naumurg
Weißenfels
Meißen
Langensalza
Zeitz
Erfurt
Dresden
Eisenach
Weimar
Altenburg
Gotha
Gera
Freiberg
Salzungen
Arnstadt
Chemnitz
Greitz
Zwickau
Rudolstadt
Ilmenau
Fulda
Meiningen
Gersfeld
Mölrichstadt
Plauen
Schlichtern
Neustadt
Hof
Gelnhausen
Brückenau
Cronach
Hanau
Schweinfurt
Culmbach
Frankfurt
Aschaffenburg
Bayreuth
Bamberg
Würzburg

SOM in TSP

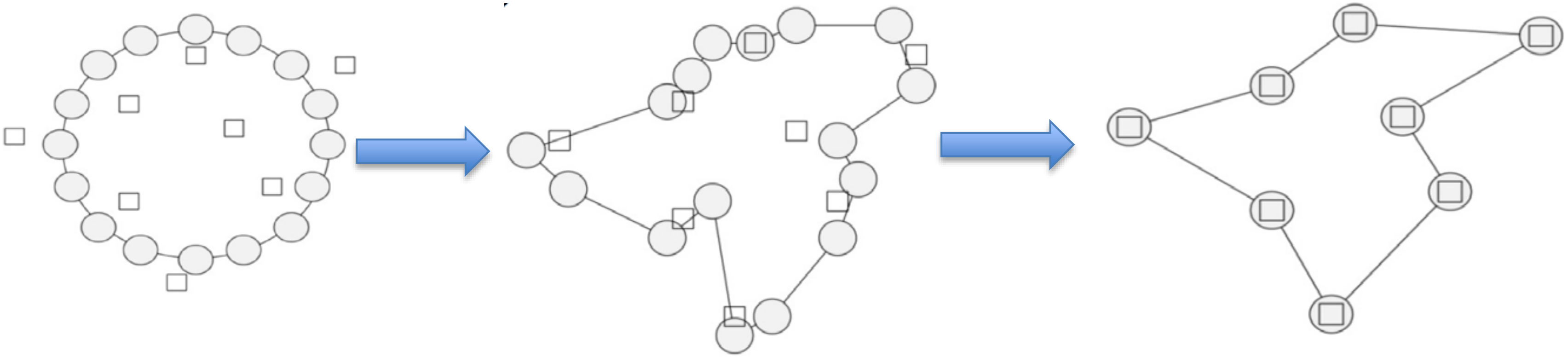
- **Input layer:**
  - A two-dimensional input.
  - *defines the coordinates of the cities in the two dimensional Euclidian space.*

- Output layer:
  - m (# of cities) output neurons.

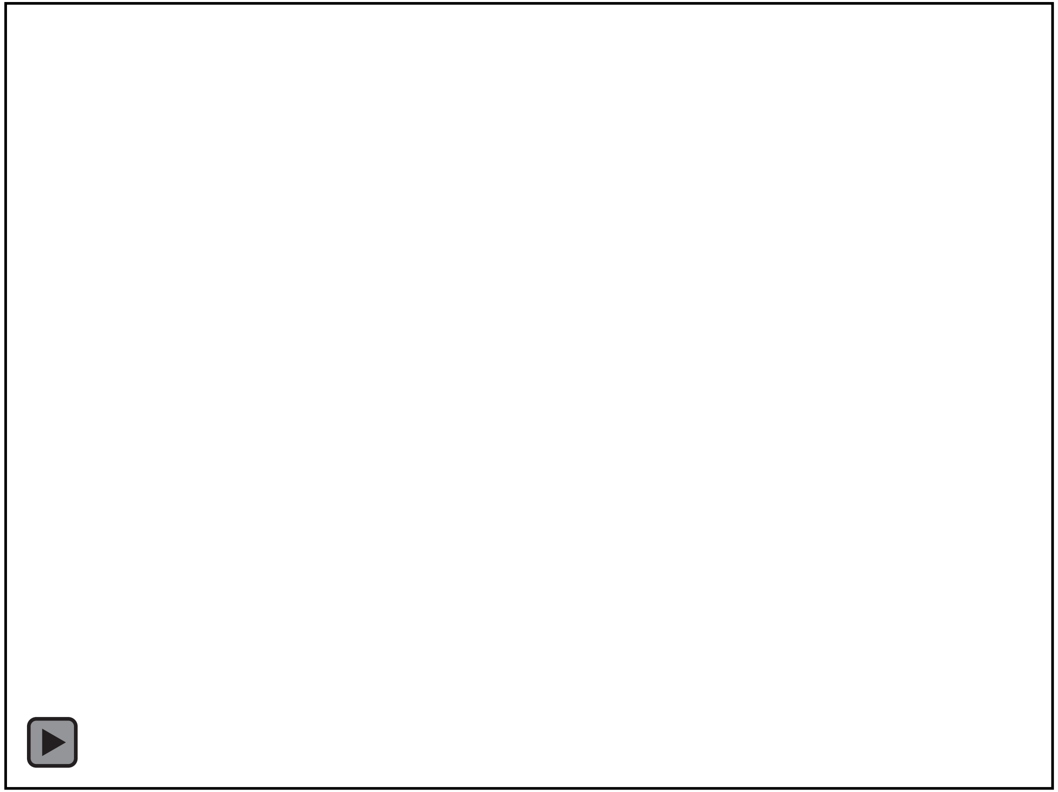
- Topology:
  - One dimensional/Ring



SOM in TSP



SOM in TSP [1]



[1] www.youtube.com/watch?v=8tnxgfE6glI