CISC/CMPE452/CISC874/COGS 400

Unsupervised Learning II

Ch. 5 - Textbook

Farhana Zulkernine

Adaptive Resonance Theory

- Adaptive resonance theory (ART) networks *allow the number of clusters to vary with problem size*.
  - *Adaptively* adds new cluster prototypes as new nodes.
- Invented by Stephen Grossberg in 1976.
- The user can control the dissimilarity between members of the same cluster by selecting a *vigilance parameter*.
- ART networks undergo *unsupervised learning* to associate patterns with prototypes.
- *'Resonance'* refers to the matching process.

ART1 Network

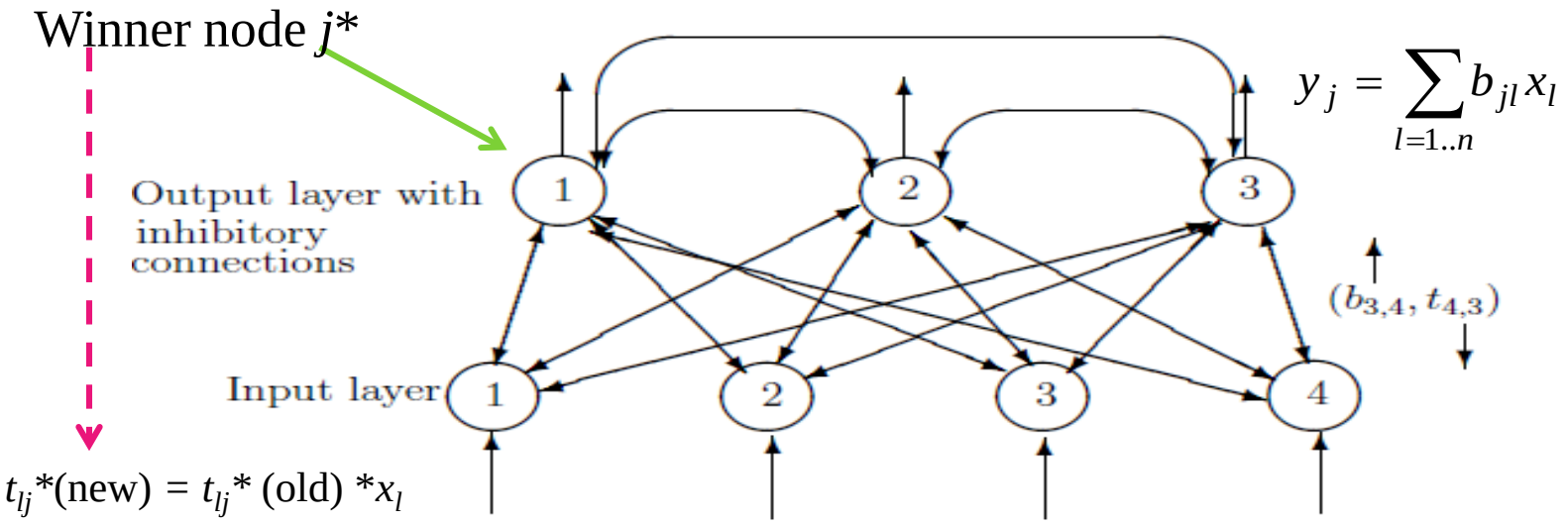


Figure 5.10: ART1 network.

If similar, modify weights to bring closer

- Each input layer node corresponds to an input vector component.
- *Bidirectional* connection – feedforward and feedback.
- Each outer layer node represents a cluster centroid.

ART1 – How it works

- Each input pattern is presented several times, and may be associated with different clusters before the network stabilizes.
  - At each successive presentation, network weights are modified.
- Matching succeeds if the pattern returned in response is sufficiently similar to the input pattern.
- Otherwise, signals travel back and forth between the output layer and the input layer ("resonance") until a match is discovered.
- If no satisfactory match is found, a new cluster is formed around the new input vector (adaptive).
- **ART1** restricts all inputs to be **binary** valued.

Figure 5.11 Algorithm for updating weights in ART1

Initialize each $t_{\ell,j}(0) = 1,\ b_{j,\ell}(0) = \frac{1}{n+1}$

while the network has not stabilized **do**

1. Let A contain all nodes;
2. For a randomly chosen input vector x, compute $y_j = b_j \cdot x$ for each $j \in A$.
3. **repeat**
   (a) Let j^* be a node in A with largest y_j.
   (b) Compute $s* = (s_1^*, \ldots, s_n^*)$ where $s_\ell^* = t_{\ell,j^*}\ x_l$ ← Signal sent back to input
   (c) If $\frac{\sum_{\ell=1}^{n} s_\ell^*}{\sum_{\ell=1}^{n} x_\ell} \leq \rho$ then: remove j^* from set A ← Not enough similarity to input
   else: associate x with node j^* and update weights:

Only weight of j* is updated

$$b_{j^*,\ell}(\text{new}) = \frac{t_{\ell,j^*}(\text{old})\ x_\ell}{0.5 + \sum_{\ell=1}^{n} t_{\ell,j^*}(\text{old}) x_\ell} \quad (1)$$

$$t_{\ell,j^*}(\text{new}) = t_{\ell,j^*}(\text{old}) x_\ell$$

x and t are binary values

   until A is empty or x is associated with some node
4. If A is empty, create new node with weight vector using eq. 1 for $t_{l,j^*}(\text{old})=1$

end while

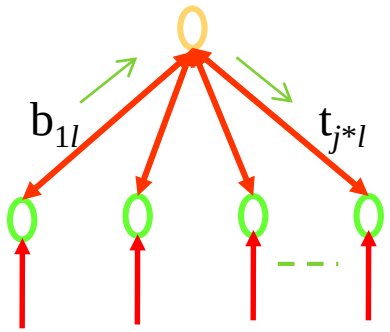
Example 5.6

Let $\rho = 0.7$, $n = 7$, and input vectors $\{(1,1,0,0,0,0,1), (0,0,1,1,1,1,0), (1,0,1,1,1,1,0), (0,0,0,1,1,1,0), (1,1,0,1,1,1,0)\}$. For the first node, let $t_{\ell,1}(0) = 1$, $b_{1,\ell}(0) = \frac{1}{8}$.

For the first input vector

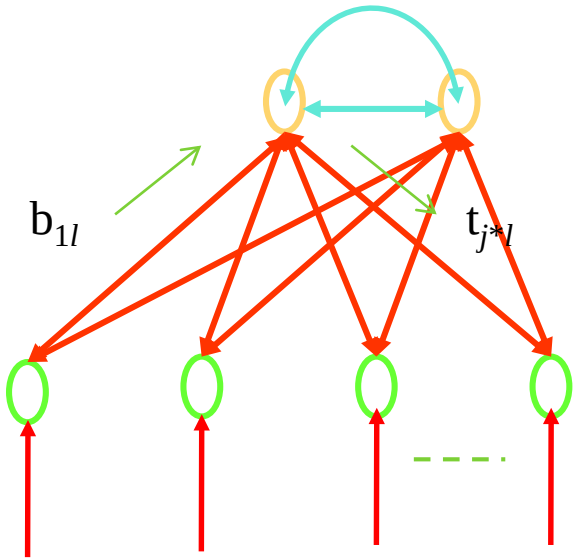
$$y_1 = \frac{1}{8} \times 1 + \frac{1}{8} \times 1 + \frac{1}{8} \times 0 + \cdots + \frac{1}{8} \times 0 + \frac{1}{8} \times 1 = \frac{3}{8},$$

and y_1 is declared the uncontested winner. Since



$$\frac{\sum_{\ell=1}^{7} t_{\ell,1}\, x_\ell}{\sum_{\ell=1}^{7} x_\ell} = \frac{3}{3} = 1 > 0.7,$$

Example 5.6 (cont…)



$$b_{1,\ell}(1) = \begin{cases} \frac{1}{0.5+3} = \frac{1}{3.5} & \text{for } \ell = 1, 2, 7; \\ 0 & \text{otherwise.} \end{cases}$$

Likewise,

$$t_{\ell,1}(1) = t_{\ell,1}(0)x_\ell.$$

$$B(1) = \left[\begin{array}{ccccccc} \frac{1}{3.5} & \frac{1}{3.5} & 0 & 0 & 0 & 0 & \frac{1}{3.5} \end{array} \right]^T$$

$$T(1) = \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right]^T.$$

For the second sample $(0, 0, 1, 1, 1, 1, 0)$, $y_1 = 0$, but $\sum_\ell t_{\ell,1}x_\ell / \sum_\ell x_\ell = 0 < \rho$. A new node is generated, with

$$b_{j^*,\ell}(\text{new}) = \frac{t_{\ell,j^*}(\text{old})\ x_\ell}{0.5 + \sum_{\ell=1}^{n} t_{\ell,j^*}(\text{old})x_\ell}$$

$$t_{\ell,j^*}(\text{new}) = t_{\ell,j^*}(\text{old})x_\ell$$

$$B(2) = \left[\begin{array}{ccccccc} \frac{1}{3.5} & \frac{1}{3.5} & 0 & 0 & 0 & 0 & \frac{1}{3.5} \\ 0 & 0 & \frac{1}{4.5} & \frac{1}{4.5} & \frac{1}{4.5} & \frac{1}{4.5} & 0 \end{array} \right]^T$$

$$T(2) = \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array} \right]^T.$$

Considering $t_{l,j\_new}(0)=1$

Observations

- Samples may switch allegiance: a sample previously activating node 2 now activates node 3 as same input is presented repeatedly.
- When the initial winner in the competition (with highest y) fails the vigilance test, another node may subsequently satisfy the vigilance test.
- Top-down weights are modified by computing intersections with the *input vector*, so that the *number of 1's gradually decreases or remains the same*. That is also the reason for initializing each top-down weight to 1.
- Weights to output layer *b* represent the cluster centroids.
- Given sufficient number of nodes, "outliers" that ought not to belong to any cluster will also be assigned separate nodes.

Topologically Organized Networks

- There is a **variation of Kohonen Learning** which combines competitive learning with topological structuring such that adjacent nodes tend to have similar weight vectors. It is called **Self-Organizing Maps (SOM)**, or sometimes called **Kohonen** or **Self-Organizing Feature Maps (SOFM)**.
- Self-organizing because learning ensures that weight vectors represent the cluster center and also **maintain proximity in Euclidean space** even though initial values are arbitrary.

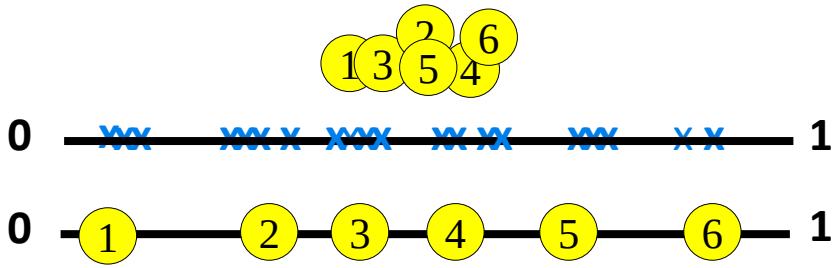
Features

- Competitive learning requires **inhibitory connections among all nodes** at Kohonen layer; topological structuring requires that ***each node also has excitatory connections to a small number of nodes***.
- Topology is specified in terms of a **neighbourhood relation** among nodes
  - Can be expressed in terms of indices and links among prototypes.
  - Distance is measured in terms of # of links.
- Learning proceeds by moving the winner node as well as its neighbours towards the presented input sample.

Applications

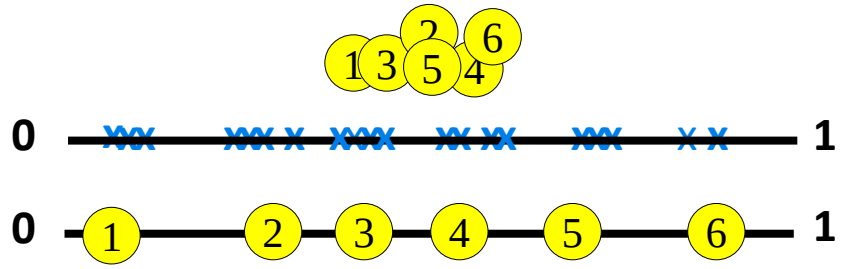
- Clustering – Each weight vector represents a cluster centroid.
- Vector Quantization – Each weight vector represents a codebook vector averaging the dimensions of all input vectors.
- Approximating Probability Distribution – The centroids in a given region is roughly proportional to the number of input vectors in that region.

E.g. Approx. probability dist. using single dimensional topology



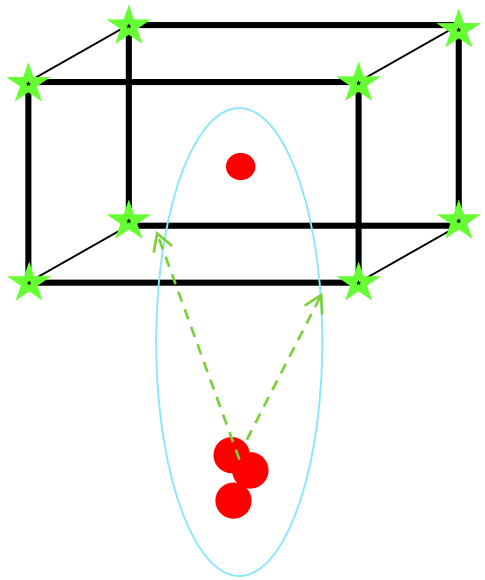
Applications (cont…)

- Proximity of prototypes: Suppose we want to interpolate the outputs of the prototypes - that is, we need to know which is closest, and the one(s) that is next closest.
- Note, that since the initial values of the prototype locations are random, there is no way to predict their order.

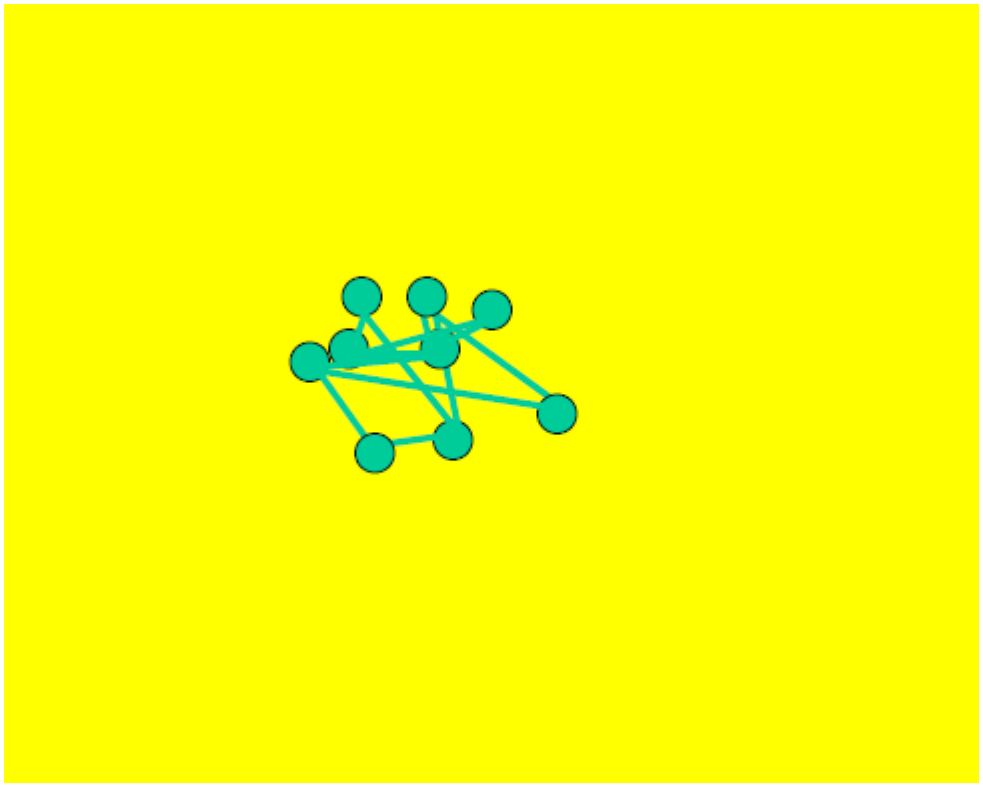


Example

- Eight 3-dimensional samples at the corners of the unit cube are presented, and one node is initially in the cube while all others are far away, only the node inside the cube is the winner for all samples.
- If the neighbourhood relation for the inner node does include other nodes, then those nodes are pulled towards the cube.
- If computation continues long enough, some of them eventually become winners for some inputs.



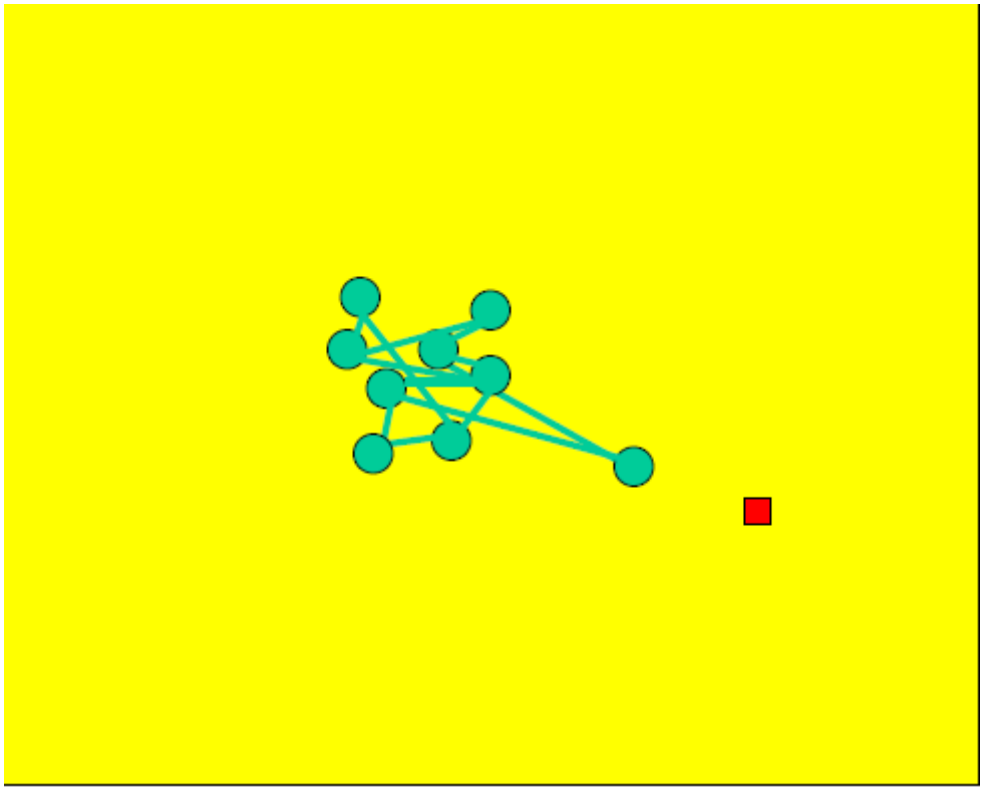
How it works



When we depict the prototypes, we display the adjacent elements in the organization of prototypes by *connecting them with lines*.

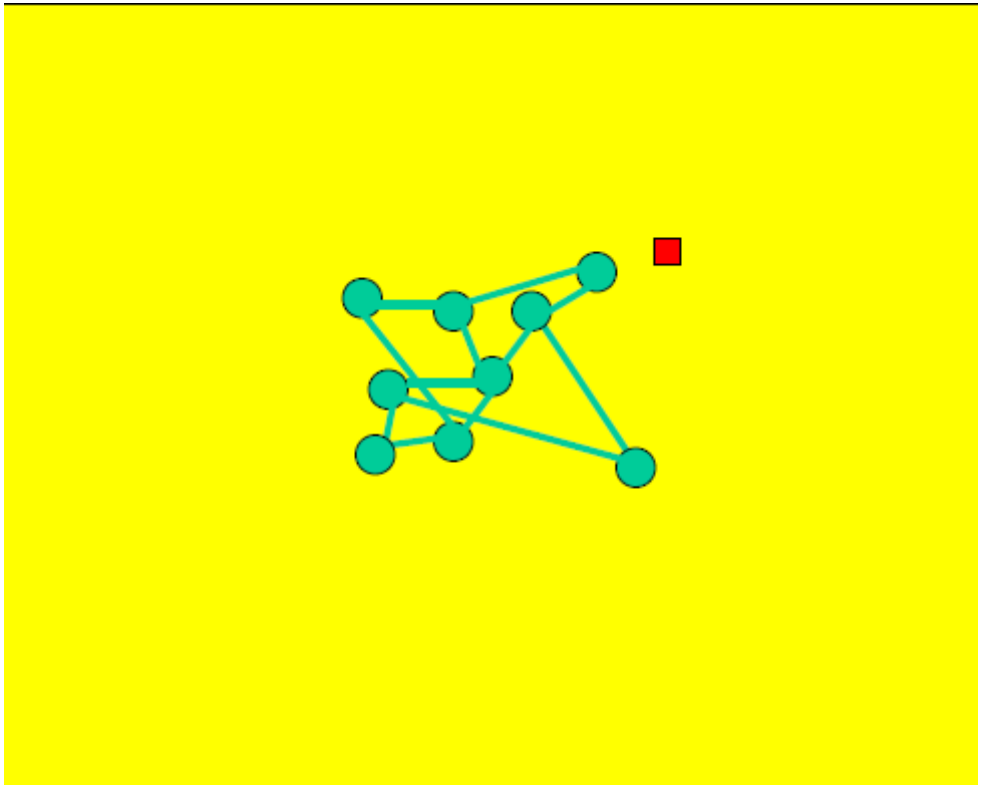
- Initially, each prototype has a random value within a small area in the center.

How (cont…)



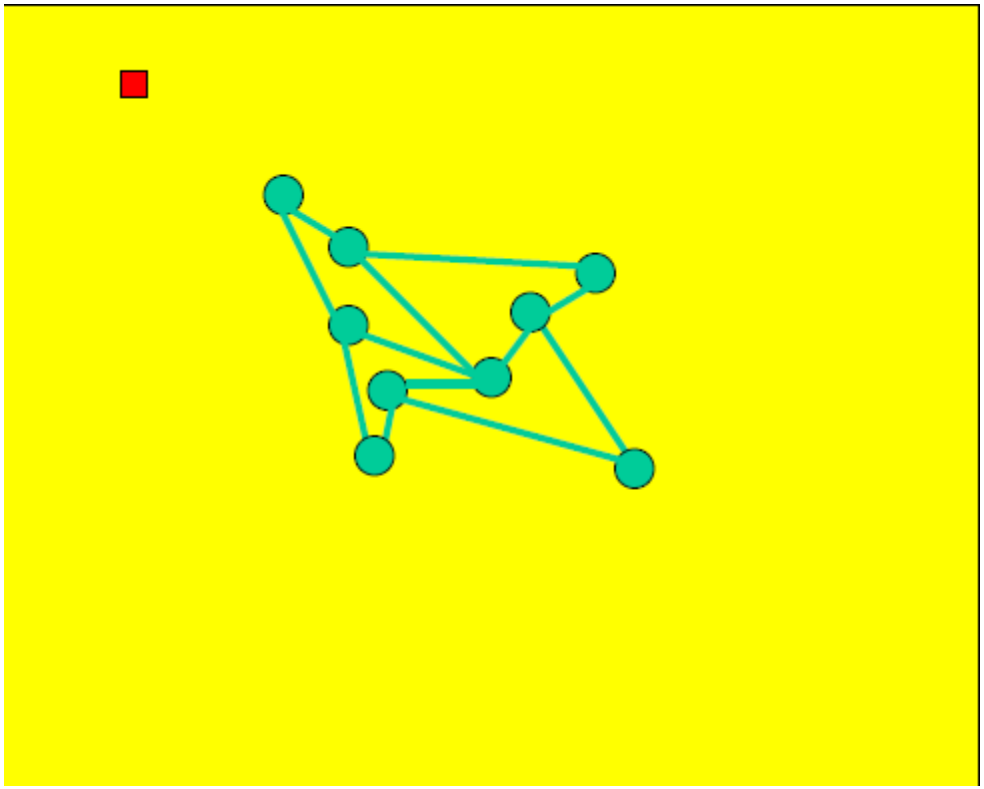
- The first training sample pulls a few connected prototypes towards it.

How (cont…)

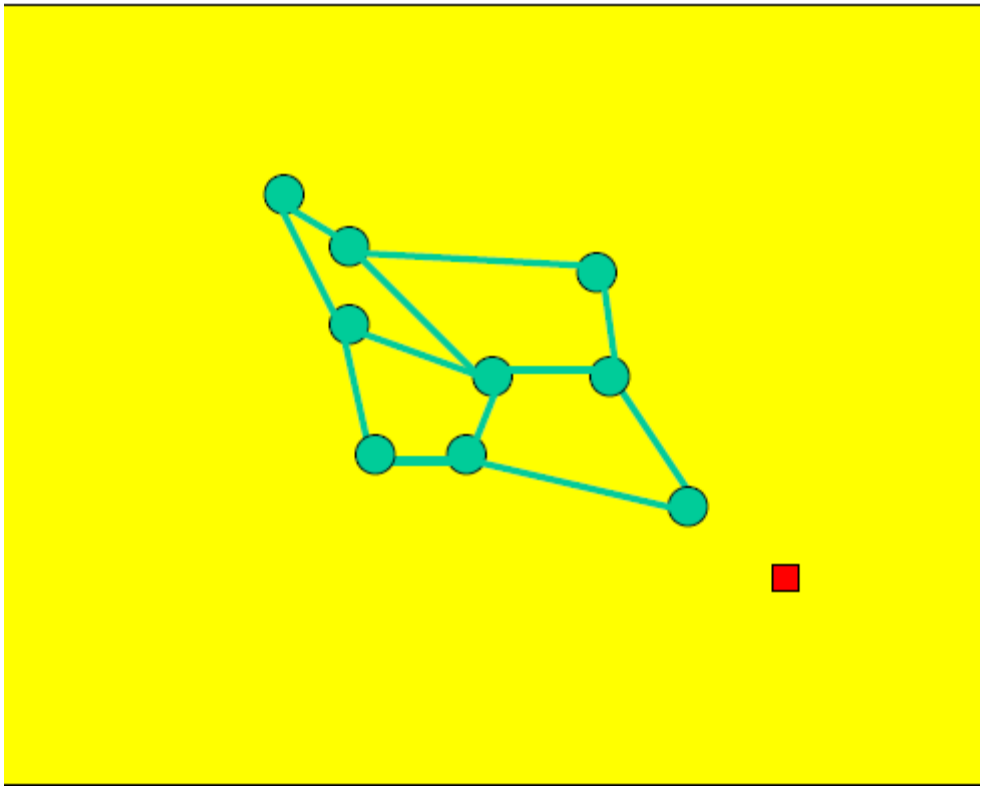


- Each subsequent training trial continues to spread out the array of prototypes.

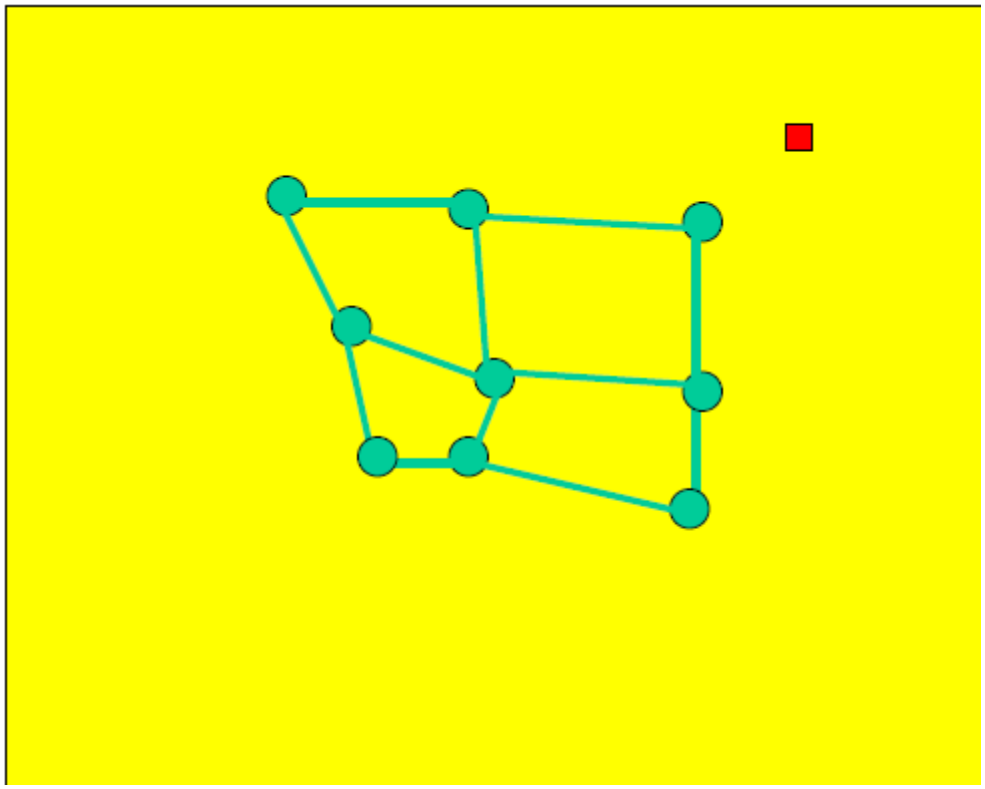
How (cont…)



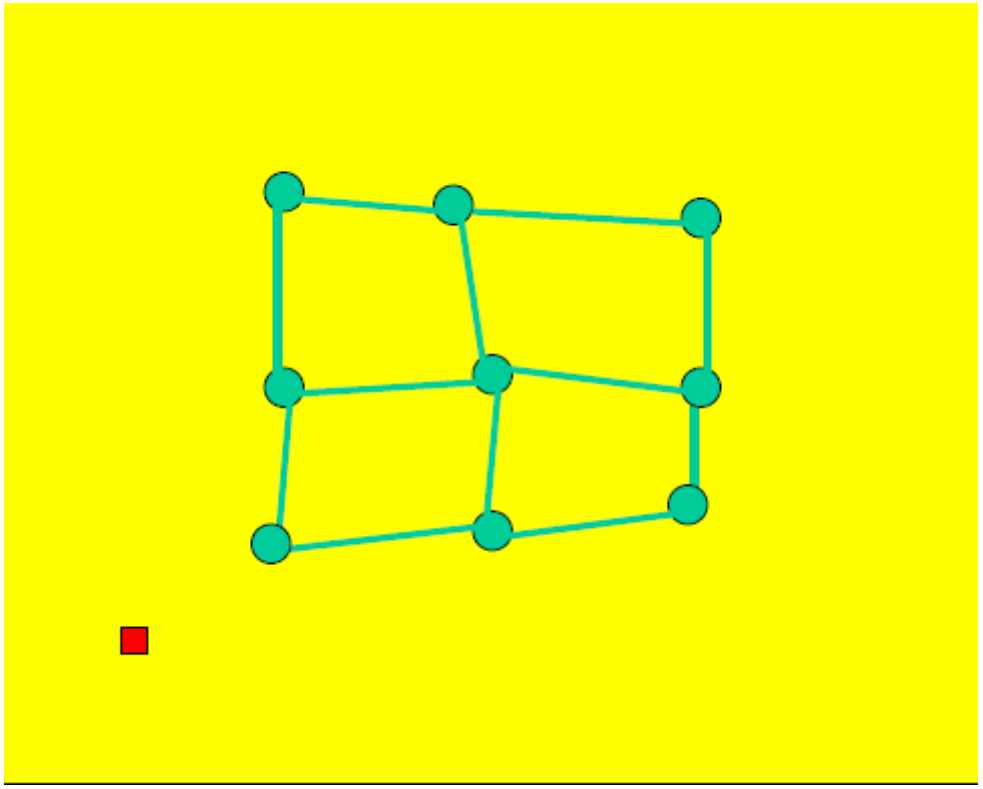
How (cont…)



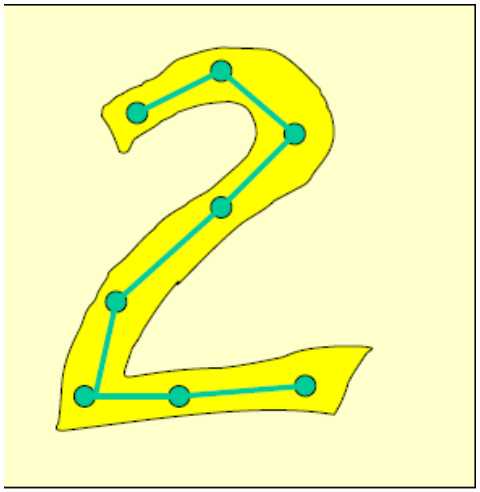
Example (cont…)



Example (cont…)

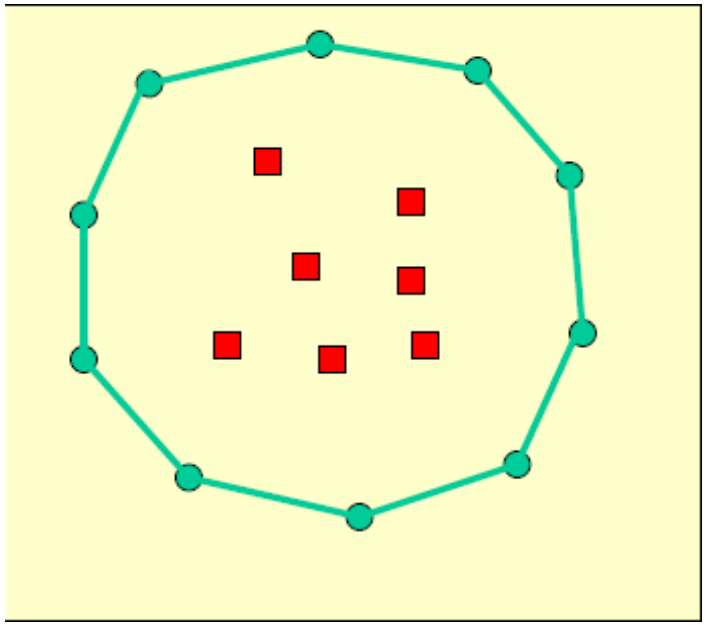


Example Application
Hand Drawn Characters



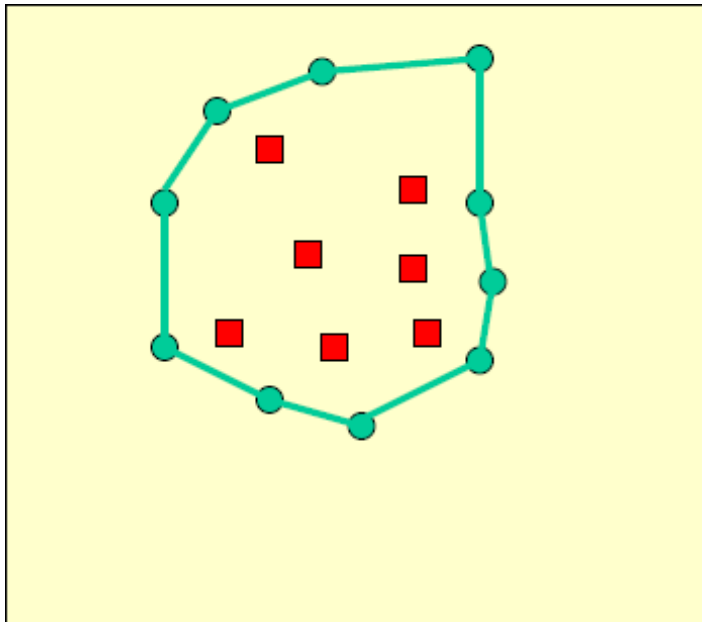
- Given a digitized hand drawn character, start a 1-D array of prototypes somewhere within the character.
- Generate uniform random samples within the character.
- The result is that the prototypes form into the skeleton.
- The determination of the skeleton is a big step towards being able to identify the character.

Example Application
Travelling Salesperson Problem



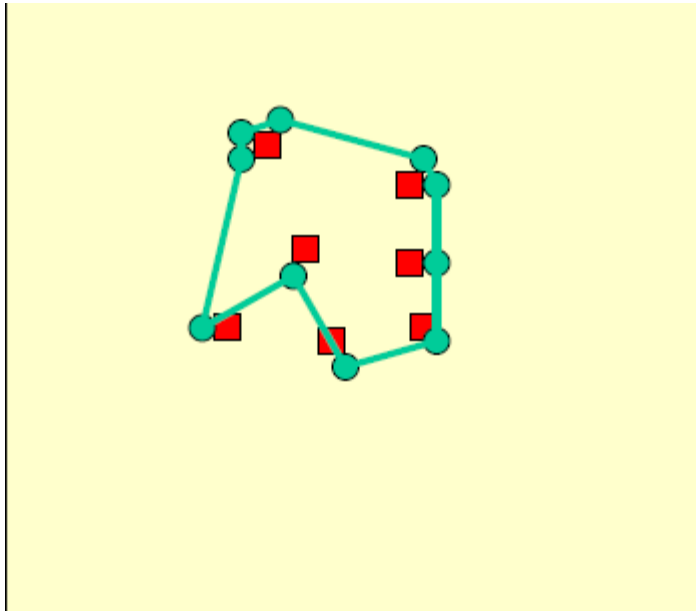
- Given a set of *n* cities (■) to visit we start off with a set of *m* prototypes (●), *m* > *n*.
- The prototypes are organized in a 1-D (end-around) array, initially positioned roughly in a circle that contains the set of cities.
- Then we generate training samples that are the locations of the cities.

Example TSP (cont…)



- As the city locations are used as sample points, the prototypes start to move in towards the cities, but still retain their order.

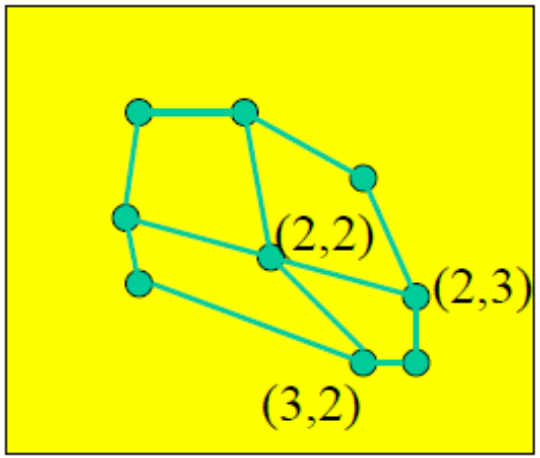
Example TSP (cont…)



- Until each prototype settles onto a city (and every city will then have at least one prototype)
- Then merge prototypes for the same city, and a path through the cities remains.

Distance Measures

- Consider, for example, a 2D index on prototypes, giving a 3x3 array.
- For Euclidean distance $|w(2,2) - w(2,3)| > |w(3,2) - w(2,3)|$
- But using neighborhood distance $D((2,2),(2,3)) < D((3,2),(2,3))$
- Objective: After clustering, prototypes (cluster centres) would represent the inputs, and at the same time maintain the indexing so that adjacent prototypes would have adjacent indices. So from the indices we can find adjacent prototypes.



When we depict the prototypes, we display the adjacent elements in the organization of prototypes by *connecting them with lines*.

Multi-Dimensional Map

- In practice, the SOM often converges quickly to ordered maps, but there is no guarantee that this will occur.
- An SOM‘s execution can be viewed as consisting of two phases,
  1. "volatile“ phase - prototypes search for niches to move into,
  2. "sober" phase - consists of nodes settling into cluster centroids in the vicinity of positions found in the earlier phase.
- The sober phase converges, but the emergence of an ordered map depends on the result of the volatile phase.
- A topologically ordered configuration is reachable, but the network may move out of it, as in general, there is no energy function that is optimized (no gradient descent to reach minimum energy state) in the weight update rule.

Example

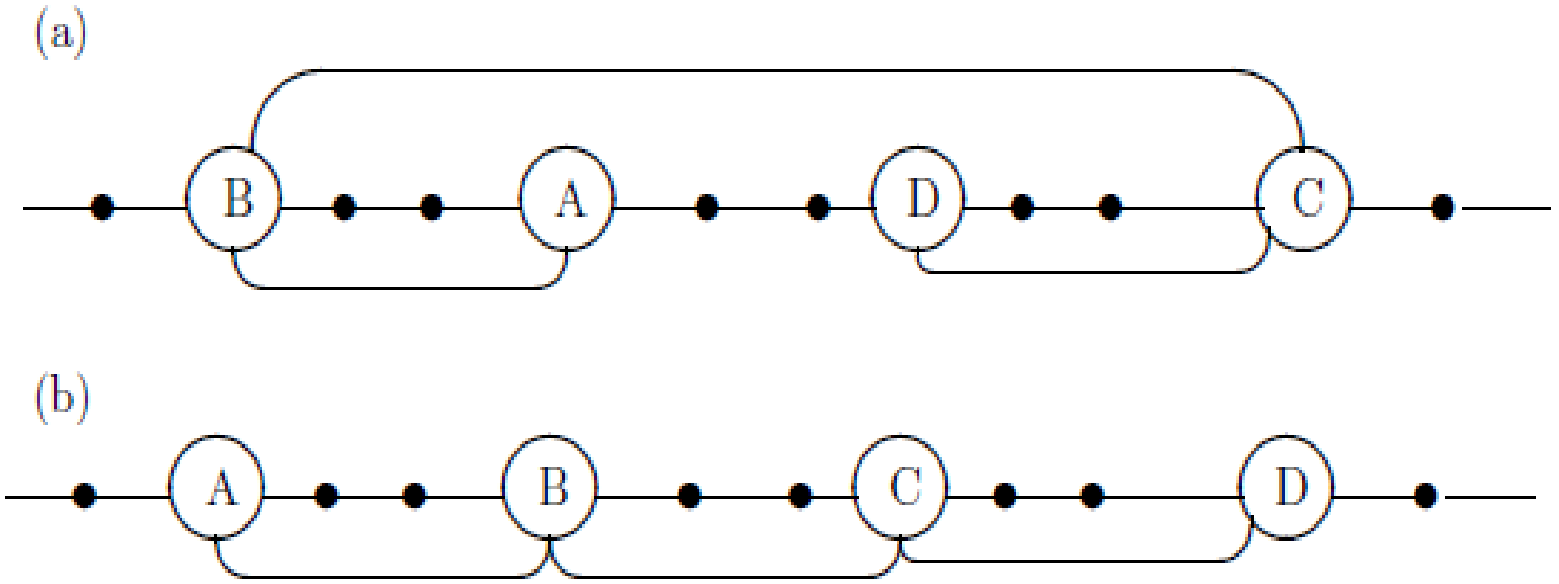


Figure 5.16: Emergence of an "ordered" map. The dark circles represent data points in one-dimensional space, and nodes are identified as A, B, C, and D, with connections indicating linear topology. (a) depicts initial positions of nodes before applying SOM learning algorithm, and (b) shows final positions of nodes, with topologically adjacent nodes having similar weights.

Example – Topology Adjacency

- Consider the network presented next with vertices at the corners of a square. If input patterns are repeatedly chosen near the center of the square but slightly closer to w1, then the nodes previously at w_0 and w_2 will eventually move closer to each other than to node w_3.
- **Topological adjacency no longer guarantees proximity of weight vectors**.

Example

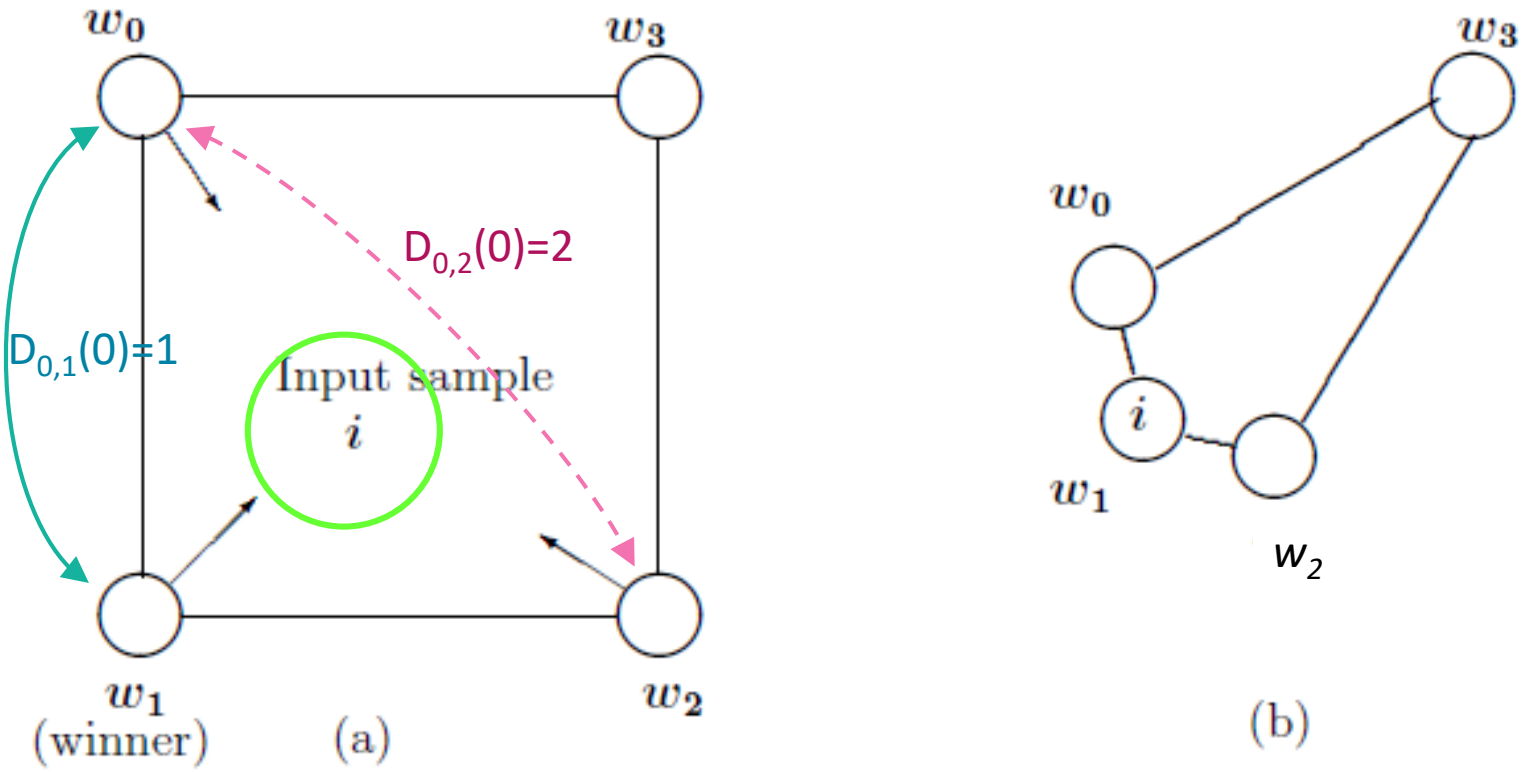


Figure 5.17: A self-organizing network in which the weight vectors (w_0, w_2) of topologically non-adjacent nodes move closer to each other than to the weight vector of an adjacent node (w_3), on repeated presentation of an input sample for which w_1 is the winner: (a) initial positions of weight vectors; (b) final positions of weight vectors.

Topological Distance

- Topological vicinity is defined by the *topological distance $D(t)$* between nodes.
  - The neighbourhood $N_j(t)$ contains nodes that are within a distance of $D(t)$ from node j at time t, where $D(t)$ decreases with time.
  - $D(t)$ does NOT refer to Euclidean distance in input space, it refers only to the length of the path connecting two nodes for the pre-specified topology chosen for the network.
- To make topological adjacency represent proximity of weight vectors, *decrease neighborhood D and learning rate with time*.

Weight Change Rule

- If $N_j(t) = \{j\} \cup \{\text{neighbours of } j \text{ at time } t\}$ where j is the winner node, and i is the input vector presented to the network at time t, then the weight change rule is:

$$w_\ell(t+1) = \begin{cases} w_\ell(\mathrm{t}) + \eta(t)\ (i - w_\ell(\mathrm{t})), & \text{if } \ell \in \mathcal{N}_j(t) \\ w_\ell(\mathrm{t}), & \text{if } \ell \notin \mathcal{N}_j(t) \end{cases}$$

- Weights vectors often become **ordered**, i.e., topological neighbours become associated with weight vectors that are near each other in the input space.

Kohonen's SOM Learning algorithm

Select network topology (neighbourhood relation);

Initialize weights randomly, and select $D(0) > 0$;

while computational bounds are not exceeded do

1. Select an input sample i_l
2. Find the output node j^* with minimum $\sum_{k=1..n} (i_{lk}(t)-w_{jk}(t))^2$
3. Update weights to all nodes within a topological distance of $D(t)$ from j^*, using

   $$w_j(t+1) = w_j(t) + \eta(t)(i_l(t) - w_j(t));$$

   where $0 < \eta(t) < \eta(t\text{-}1) < 1$ and $j \in N_{j^*}(t)$;

   Weights of non-neighboring nodes are left unchanged.
4. Reduce $D(t)$ and η after a time interval t'

end while

SOM

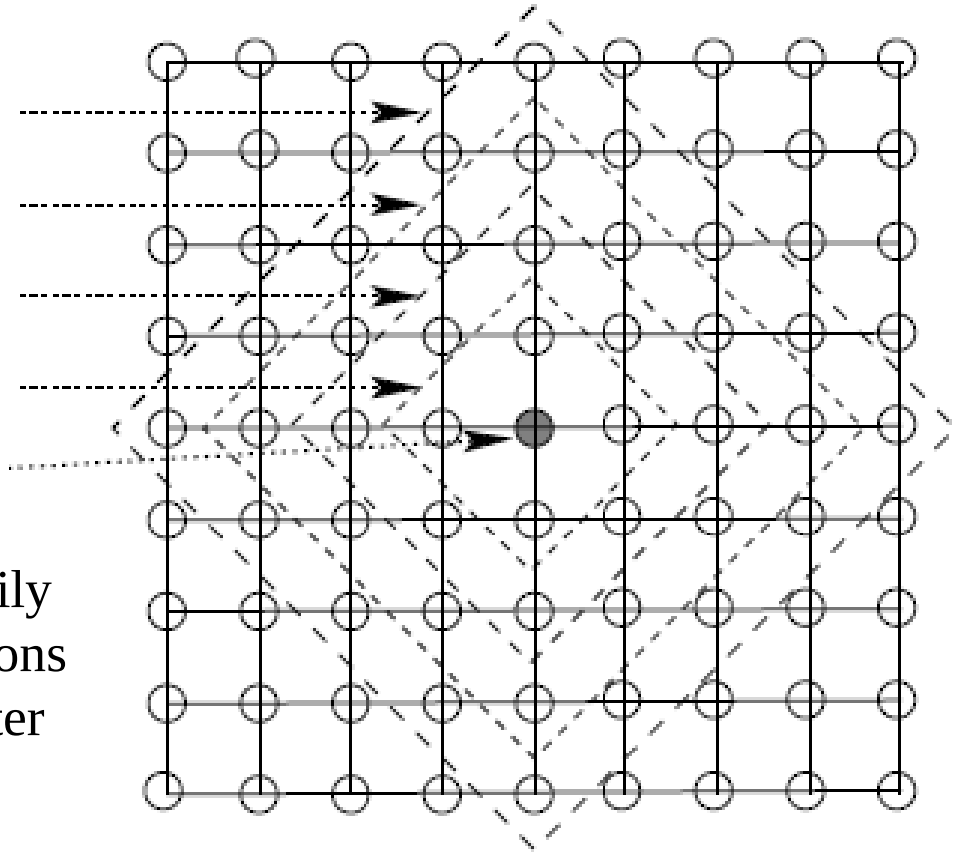
Neighbourhood for $0 \leq t < 10$

Neighbourhood for $10 \leq t < 20$

Neighbourhood for $20 \leq t < 30$

Neighbourhood for $30 \leq t < 40$

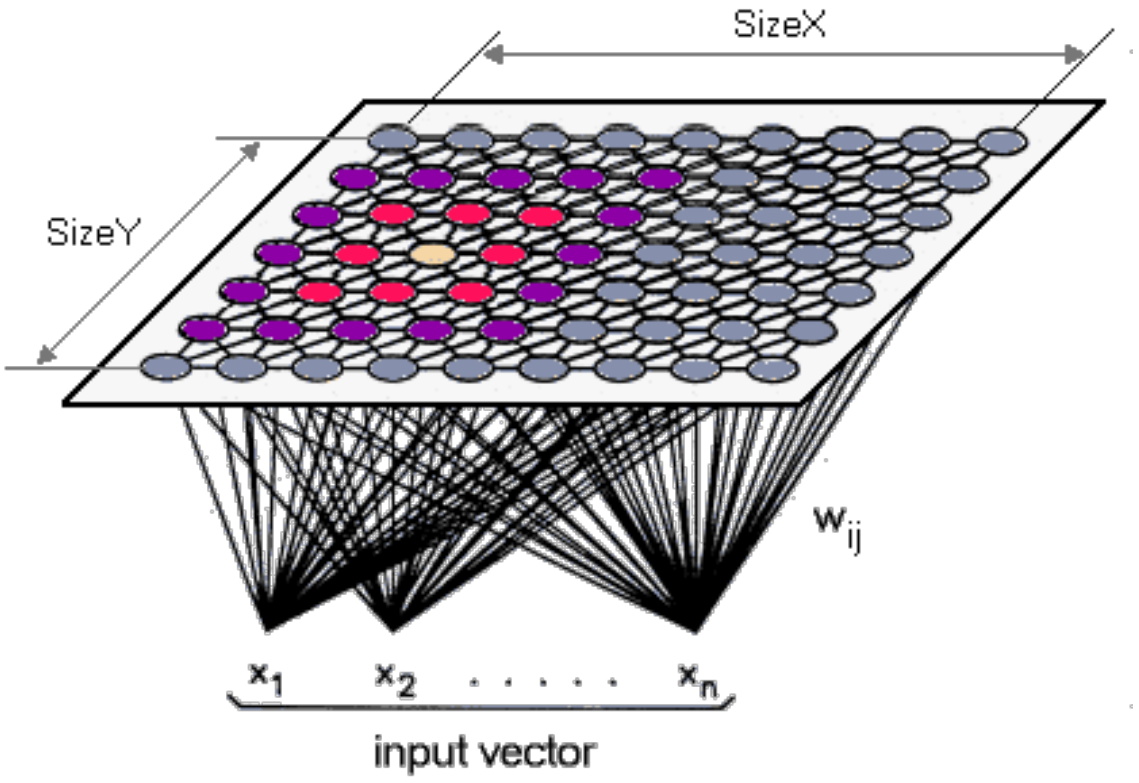
After $t > 39$



The SOM algorithm with steadily shrinking neighbourhood relations has been shown to perform better than networks with static neighbourhood functions.

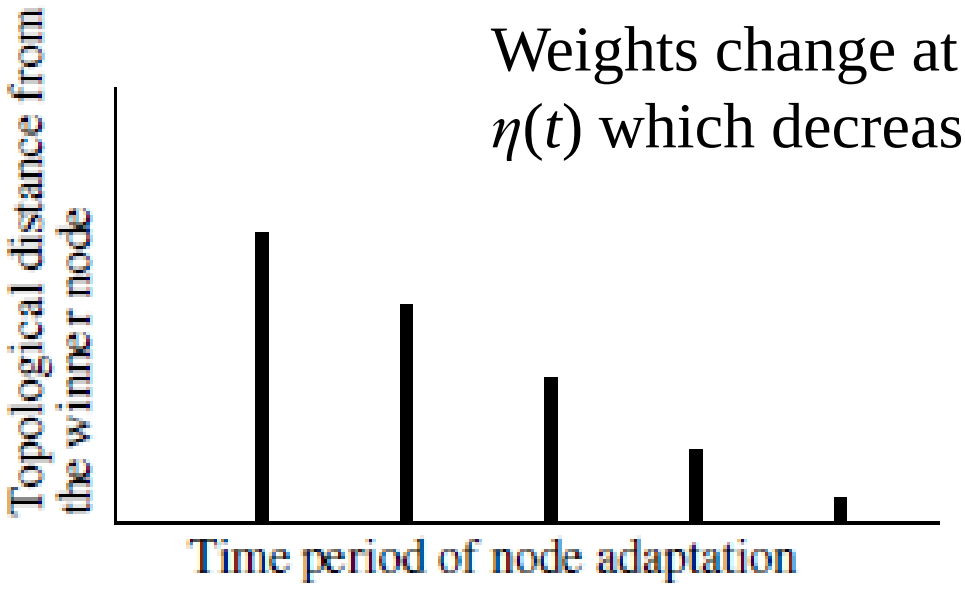
Figure: Time-varying neighbourhoods of a node in a SOM network with grid topology: $D(t) = 4$ & $\eta = 0.1$ for $0 \leq t < 10$, $D(t) = 3$ & $\eta = 0.08$ for $10 \leq t < 20$, $D(t) = 2$ & $\eta = 0.06$ for $20 \leq t < 30$, $D(t) = 1$ & $\eta = 0.04$ for $30 \leq t < 40$, and $D(t) = 0$ & $\eta = 0.02$ for $t > 39$.

SOM/SOFM Network



Gighub https://github.com/yogonza524/SOM

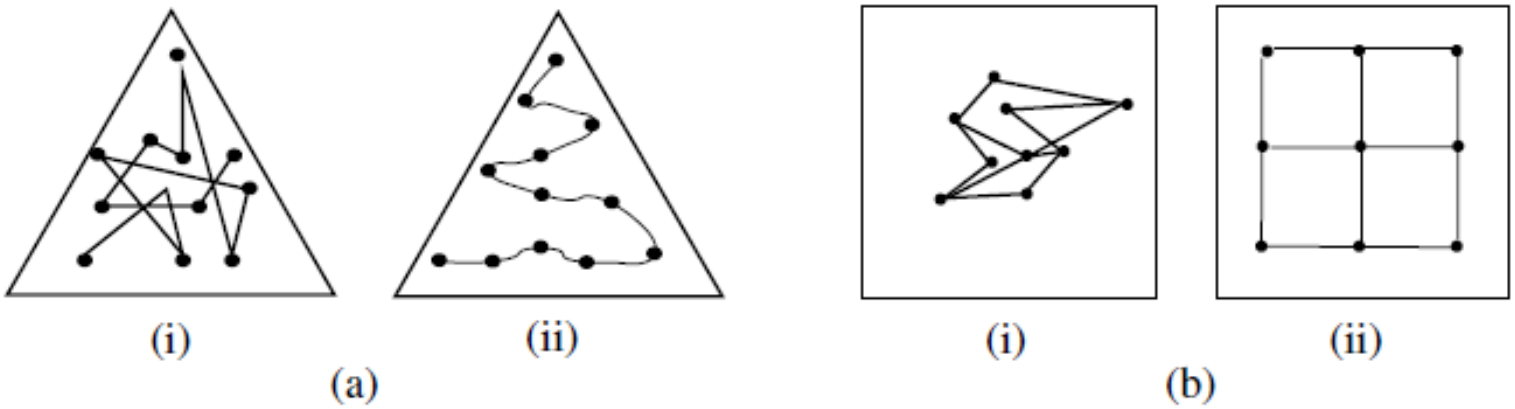
SOM



Weights change at time t at a rate $\eta(t)$ which decreases with time

Figure: Shrinkage of the neighbourhood of the winner node in topologically organized SOM networks, as time increases.

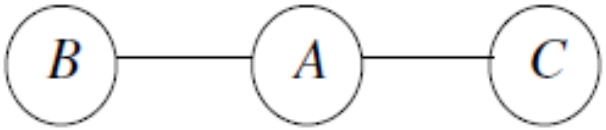
Example



Weight vectors (in the input space) before and after the SOM learning algorithm is applied, (a) using a 1-dimensional linear network topology for 2-dimensional input data uniformly distributed in a triangular region; and (b) using a 2-dimensional grid network topology for 2-dimensional input data uniformly distributed in a rectangular region.
Weight vectors are initially located randomly, as shown in (a)(i) and (b)(i). By the end of the learning process, the weights become “ordered”, as shown in (ii) of each example.

Example 5.7

For the same data as in example 5.2, assume a 3- node network with node A adjacent to nodes B and C respectively.



$\{i_1 = (1.1, 1.7, 1.8),\ i_2 = (0, 0, 0), i_3 = (0, 0.5, 1.5),\ i_4 = (1, 0, 0),$
$i_5 = (0.5, 0.5, 0.5),\ i_6 = (1, 1, 1)\}.$

$$W(0) = \begin{pmatrix} w_A: & 0.2 & 0.7 & 0.3 \\ w_B: & 0.1 & 0.1 & 0.9 \\ w_C: & 1 & 1 & 1 \end{pmatrix}.$$

Let $D(t) = 1$ for one initial round of training set presentations (until $t = 6$), and $D(t) = 0$ thereafter.

Example 5.7 (cont…)

$t = 1$:
Sample presented: $i_1 = (1.1, 1.7, 1.8)$.
Squared Euclidean distance between A and i_1:
$d^2_{A1} = (1.1 - 0.2)^2 + (1.7 - 0.7)^2 + (1.8 - 0.3)^2 = 4.1$. Similarly, $d^2_{B1} = 4.4$ and $d^2_{C1} = 1.1$.
C is the "winner" since $d^2_{C1} < d^2_{A1}$ and $d^2_{C1} < d^2_{B1}$.
Since $D(t) = 1$ at this stage, the weights of both C and its neighbor A are updated according to the Equation 5.2. e.g.,

$$w_{A,1}(1) = w_{A,1}(0) + \eta(1) \cdot (x_{1,1} - w_{A,1}(0)) = 0.2 + (0.5)(1.1 - 0.2) = 0.65$$

The resulting weight matrix is:

$$W(1) = \begin{pmatrix} \boldsymbol{w}_A: & 0.65 & 1.2 & 1.05 \\ \boldsymbol{w}_B: & 0.1 & 0.1 & 0.9 \\ \boldsymbol{w}_C: & 1.05 & 1.35 & 1.4 \end{pmatrix}.$$

Note that the weights attached to B are not modified since B falls outside the neighborhood of the winner node C.

Example 5.7 (cont…)

$t = 7$:
η is now reduced to 0.25, and the neighborhood relation shrinks, so that only the winner node is updated henceforth. Sample presented: i_1. C is the winner; only w_C is updated.

$$w_C : \quad 0.84 \quad 1.07 \quad 1.19.$$

$t = 13$:
η is further reduced to 0.1, i_1 is presented.

$$w_C : \quad 0.68 \quad 1.00 \quad 1.32.$$

Example 5.7 (cont…)

$t = 14$:
Sample presented: i_2. Weights of the winner node B are updated.

$$w_B : \quad 0.47 \quad 0.23 \quad 0.30.$$

$t = 15$:
Sample presented: i_3. Winner is C and w_C is updated. At this stage, the weight matrix is given by

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

At the beginning of the training process, the Euclidean distances between various nodes were given by:

$$|w_A - w_B| = 0.85, \ |w_B - w_C| = 1.28, \ |w_A - w_C| = 1.22.$$

Example 5.7 (cont…)

The training process increases the relative distance between non-adjacent nodes (B, C) ,while A remains roughly in between B and C:

$$|w_A - w_B| = 1.28, \ |w_B - w_C| = 1.75, \ |w_A - w_C| = 0.80.$$

Note that samples switch allegiance between nodes, especially in the early phase of computation. Computation continues until the same nodes continue to be the winners for the same input patterns.