Unsupervised Learning

Using ANNs

Unsupervised Learning

- If correct I/O association is ***not*** provided
- A number of samples are imposed
- What does an ANN do with samples?
  - Network topology
    - Layers and connection
  - Learning rules used
    - familiarity, principal component analysis, feature mapping, etc.
  - Learning paradigm
    - Competitive vs. cooperative
  - Update
    - Batch (off-line) update vs. interactive (on-line) update

Again, the Recurring Theme of

- Finding to which a sample belongs
  - Belong to everyone
  - Belong to only one
  - Belong to a small group of classes
- How a sample affects class statistics
  - Global weighted update
  - Competitive update
  - Collaborative update

Issues

- Network topology
  - Does multiple layers help?
    - Training mechanism?
    - Separation of functionalities?
  - But lateral connections are often important
- Update rules
  - Firing of neurons is instantaneous upon receiving inputs
  - Cf with k-mean which is batch

Learning Rules

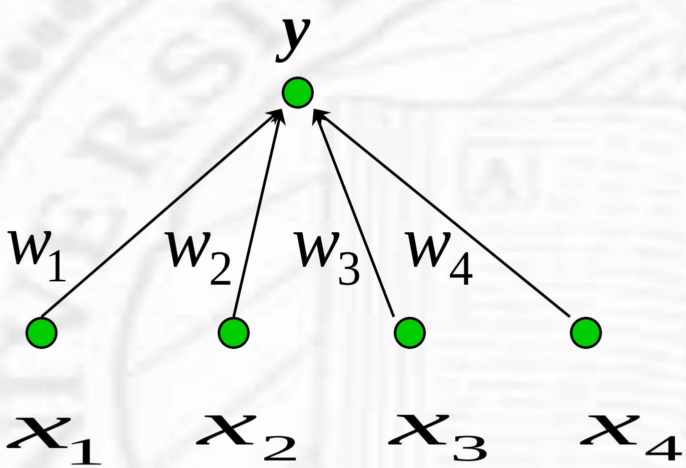
- Even though we can use the same networks, we have to be careful about the learning rules
- Rules that require backpropagation of error (knowing the correct I/O association) are not applicable
- E.g., Use Hebb rules instead
  - Reward for correlated pre- and post- firing

Learning Paradigm

- Global learning
  - Nice guy, democratic approach
- Cooperative
  - Try to maintain some kind of local structure with a radial basis attention function
- Competitive learning
  - Playground bully approach
    - Mine only
    - Not only it is mine, stay far away as possible

Simplest case

- One linear unit with Hebb's learning rules



Similarity measure

$$y = \sum_i w_i x_i = \mathbf{w}^{\mathbf{t}}\mathbf{x} = \mathbf{x}^{\mathbf{t}}\mathbf{w}$$

$$\Delta w_i = \eta y x_i \qquad \boxed{\Delta \mathbf{w} = \eta y \mathbf{x}}$$

– weights are adjusted to be "similar" to inputs

– more frequent input patterns dominate

– pattern "familiarity" is learned

Simplest case

- At equilibrium (with a lot of patterns observed and weight vectors do not change significantly)

$$y = \sum_i w_i x_i = \mathbf{w}^{\mathbf{t}}\mathbf{x}$$

$$\Delta w_i = y x_i = \sum_j w_j x_j x_i = 0$$

$$\mathbf{Cw} = 0, \mathbf{C} = \mathbf{xx}^t, C_{ij} = x_i x_j$$

  - implies that **w** is the eigenvector of matrix **C** with zero eigenvalue
  - but this cannot be stable!

$$\Delta \mathbf{w} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \begin{bmatrix} x_1 & \dots & x_d \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix} = \mathbf{Cw}$$

Simplest case

- Train a network with the same pattern over and over again
  - weights will go to infinity, dominated by the eigenvectors with the largest eigenvalues

$$
\begin{aligned}
&\mathbf{w}' = \mathbf{w} + \mathbf{C}\mathbf{w} \\
&\mathbf{w}'' = \mathbf{w}' + \mathbf{C}\mathbf{w}' = \mathbf{w} + \mathbf{C}\mathbf{w} + \mathbf{C}^2\mathbf{w} \\
&\ldots \\
&\mathbf{w}^{(n)} = \mathbf{w} + \mathbf{C}\mathbf{w} + \ldots + \mathbf{C}^n\mathbf{w} \qquad \mathbf{w} = \sum_i a_i \mathbf{u}_i \\
&\approx a_1 \lambda_1^{\,n} \mathbf{u}_1
\end{aligned}
$$

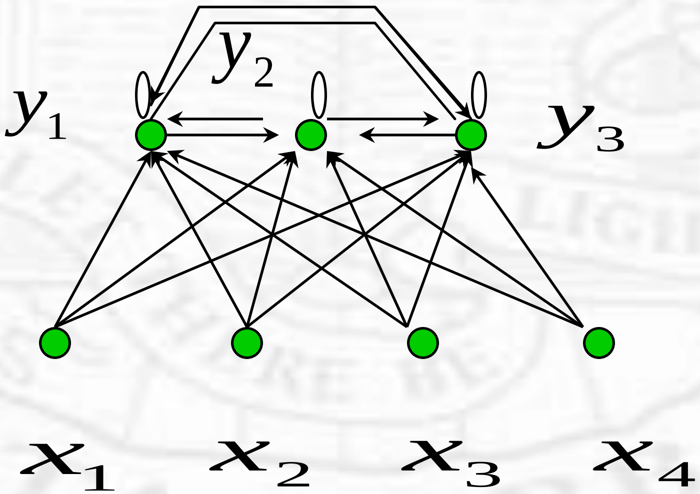
Oja's learning rule

$$\Delta w_i = \eta y(x_i - y w_i)$$

- similar learning effect as Hebb's rule
- If the weight already confirms to the pattern, don't learn
- without divergence of weight vector
- weight vector converges to the maximal eigenvector
- can be generalized to locate other eigenvectors (principal component analysis)

Unsupervised Competitive Learning

- Clustering or categorizing data
- Only one output active (winner-take-all)
- Lateral inhibition
- Each output neuron y for one class



Simple competitive learning

- one-layer network
- decision rule: (most) similar one learns

$$\mathbf{w}_{\mathbf{i}^*} \cdot \mathbf{x} \geq \mathbf{w}_{\mathbf{i}} \cdot \mathbf{x} (\text{for all} \mathrm{i})$$

$$|\mathbf{w}_{\mathbf{i}^*} - \mathbf{x}| \leq |\mathbf{w}_{\mathbf{i}} - \mathbf{x}| (\text{for all i}, |\mathbf{x}| = 1)$$

- update rule: closer to the input pattern

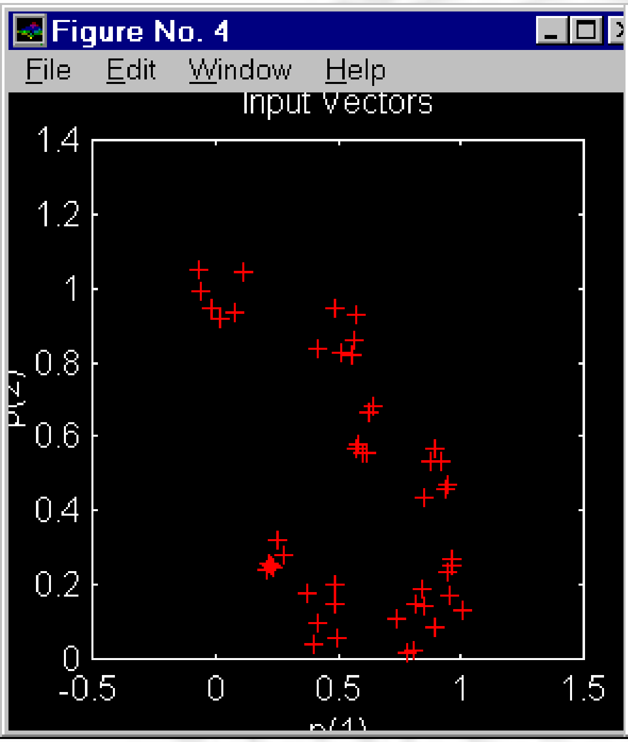
$$\Delta w_{i^* j} = \eta x_j^u$$

$$\Delta w_{i^* j} = \eta \left(\frac{x_j^u}{\sum_j x_j^u} - w_{i^* j} \right) \qquad \sum_j w_{i^* j} = 1$$

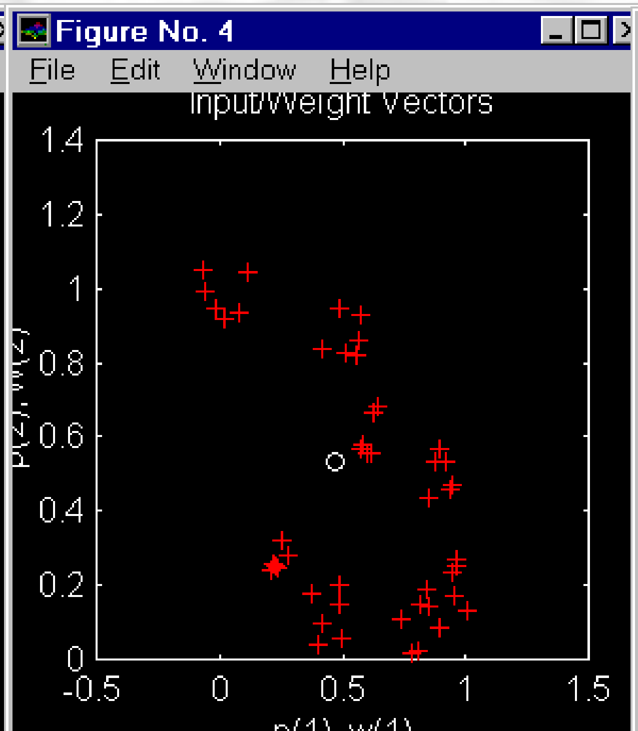
$$\Delta w_{i^* j} = \eta (x_j^u - w_{i^* j})$$

Competitive Learning Example

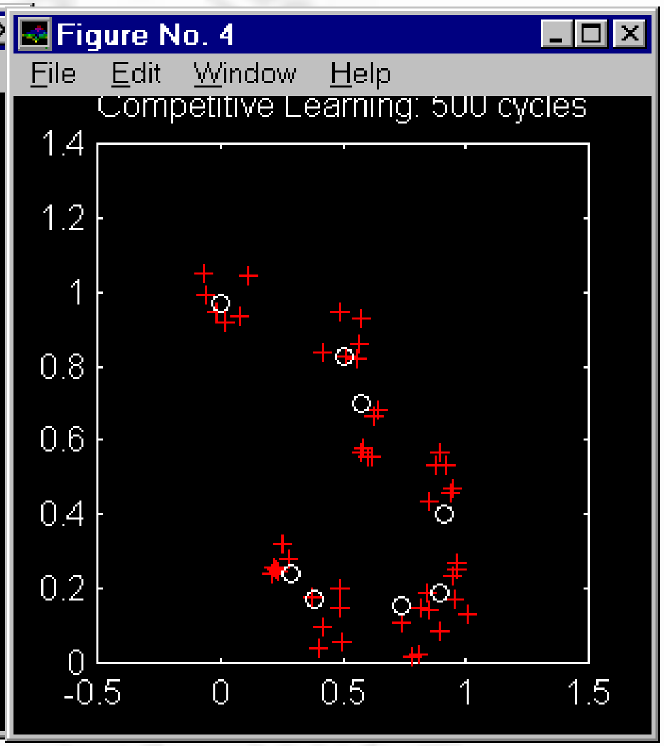
Input data



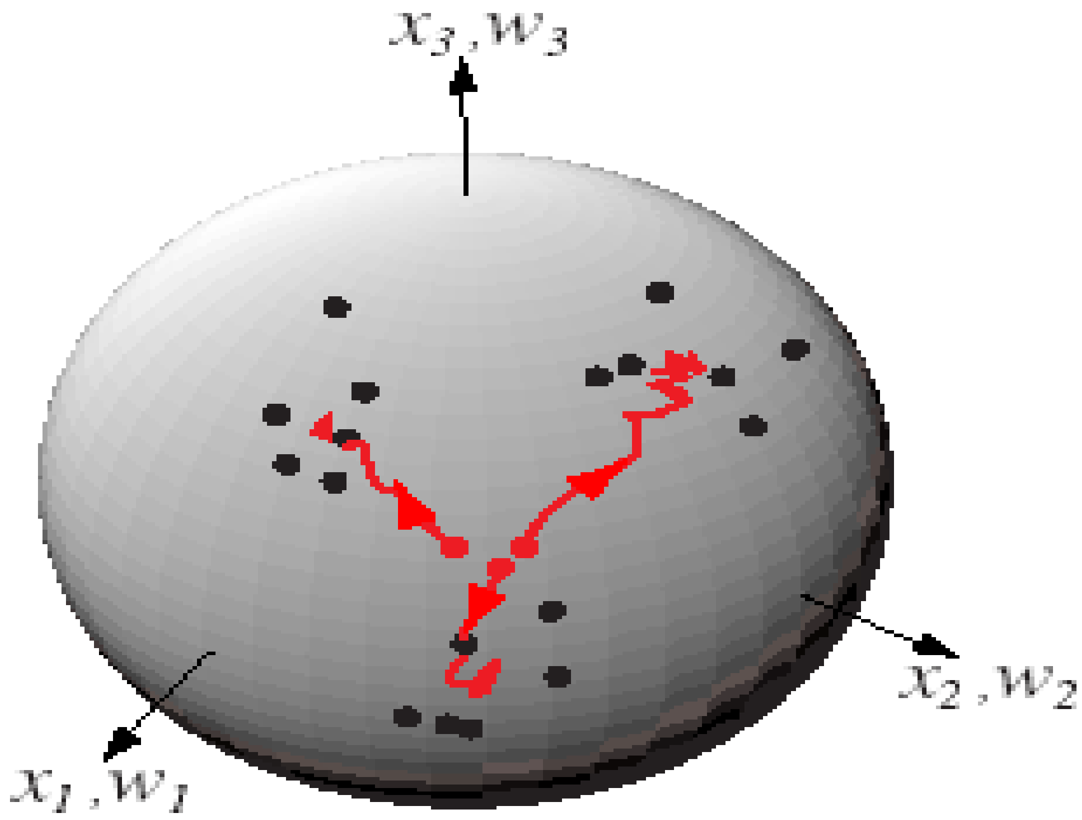
Initial placement



Final placement



More Examples



Vector Quantization

- A compression technique to represent input vectors with a smaller number of “code” (representative, prototype) vectors
- Standard decision rule + learning rule
- Learning Vector Quantization
  - standard decision rule +

$$\Delta w_{i^* j} = \begin{cases} \eta(x_j^u - w_{i^* j}) & \text{correct class} \\ \boxed{-\eta(x_j^u - w_{i^* j})} & \text{in correct class} \end{cases}$$

Playground bully, push others away

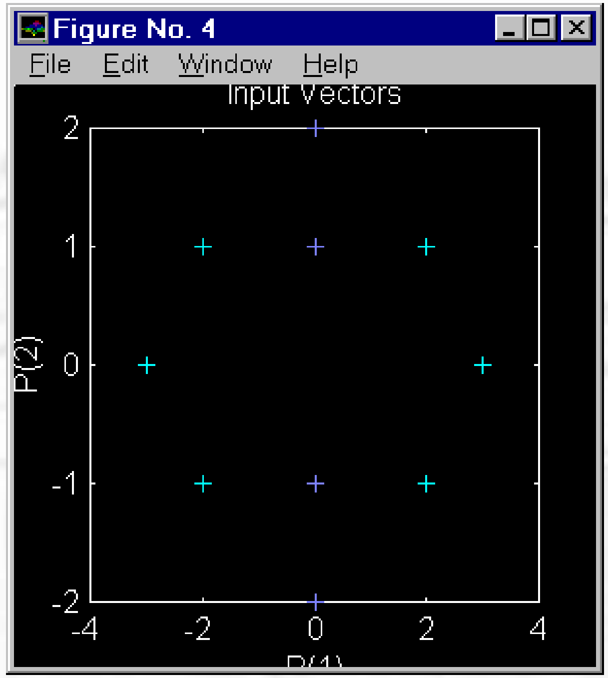


Figure No. 4
File Edit Window Help
Input Vectors
P(2)
2
1
0
-1
-2
-4
-2
0
2
4

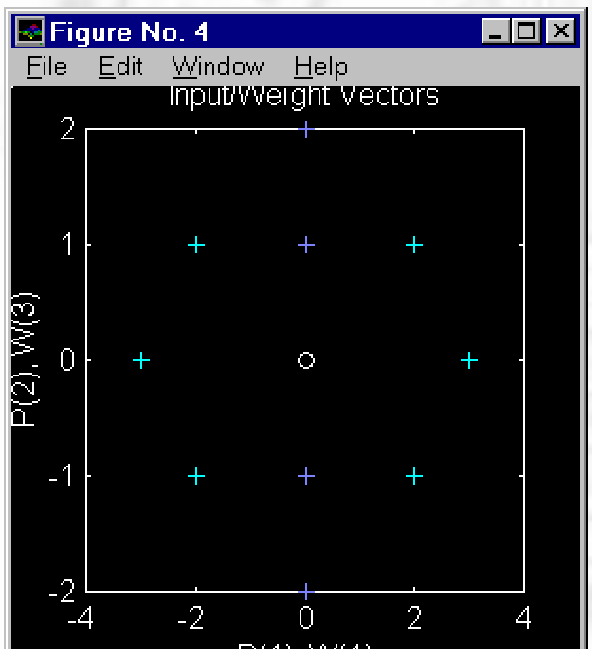


Figure No. 4
File Edit Window Help
Input/Weight Vectors
P(2), W(3)
2
1
0
-1
-2
-4
-2
0
2
4

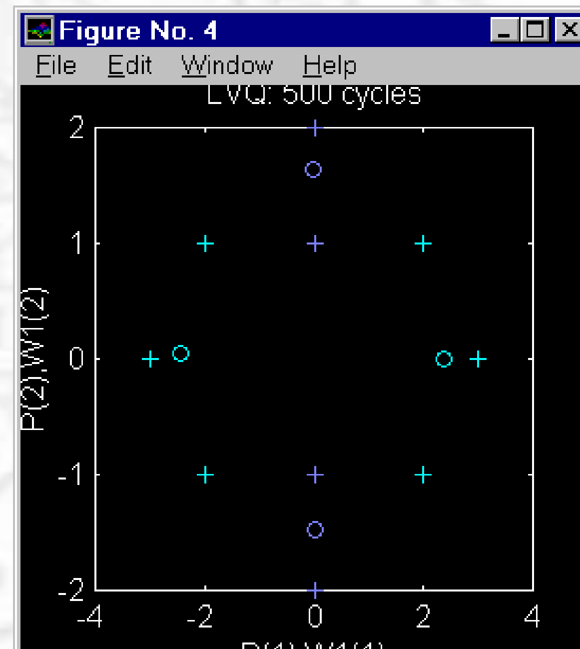
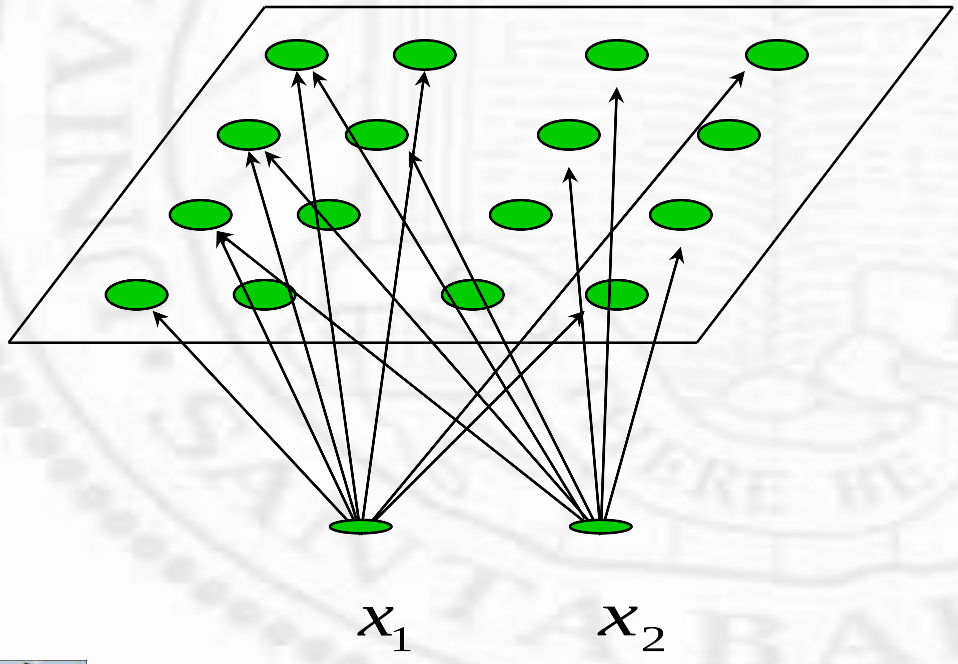


Figure No. 4
File Edit Window Help
LVQ: 500 cycles
P(2),W1(2)
2
1
0
-1
-2
-4
-2
0
2
4

Feature Mapping

- A topology preserving map
- Similar inputs map to outputs which are close-by



Kohonen Map (Self-Organizing Map)

- Preserve neighborhood relations
- Decision rule

$$| w_{i^*} - x | \leq | w_i - x | \text{(for all i)}$$

- Update rule
  - initially, the neighborhood is large
  - gradually the neighborhood narrows down

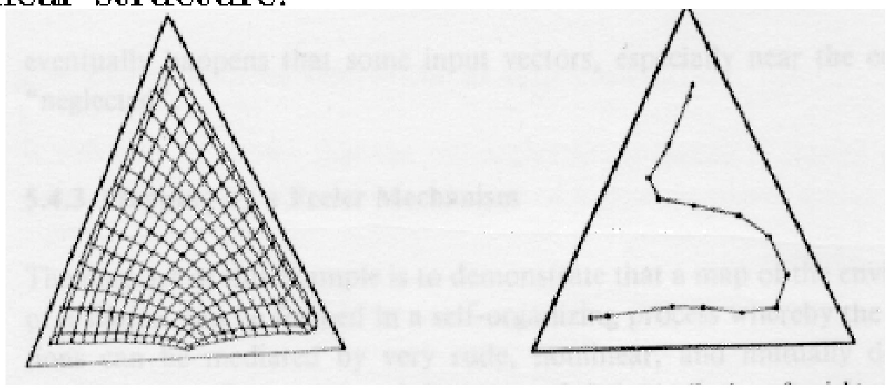
$$\Delta w_{ij} = \boxed{\eta}\boxed{\Lambda}(i, i^*)(x_j - w_{ij})$$

$$\Lambda(i, i^*) = e^{-|r_i - r_{i^*}|^2 / 2\sigma^2}$$

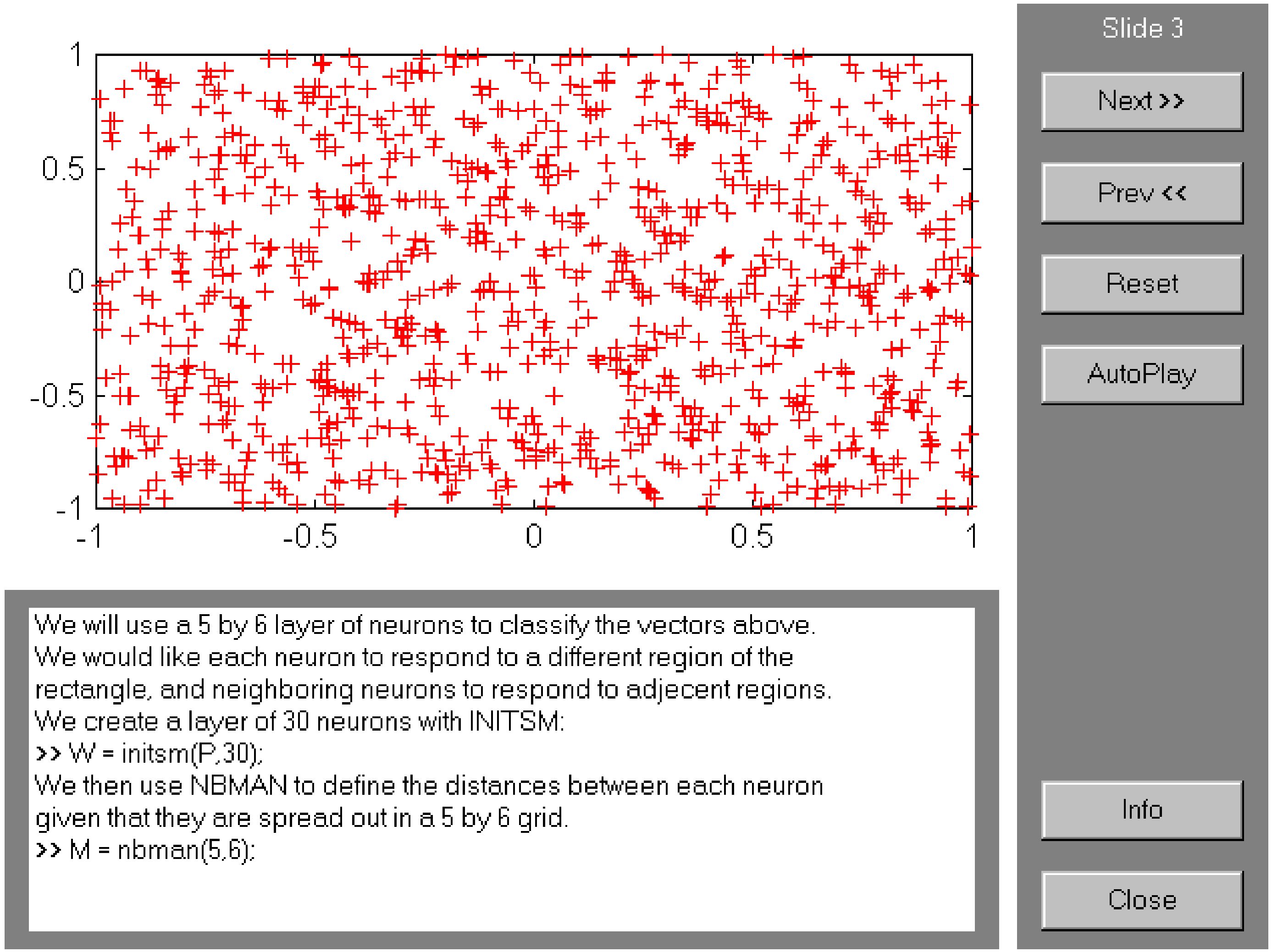
- Learning rate
- Neighborhood size
- Both drop through iterations

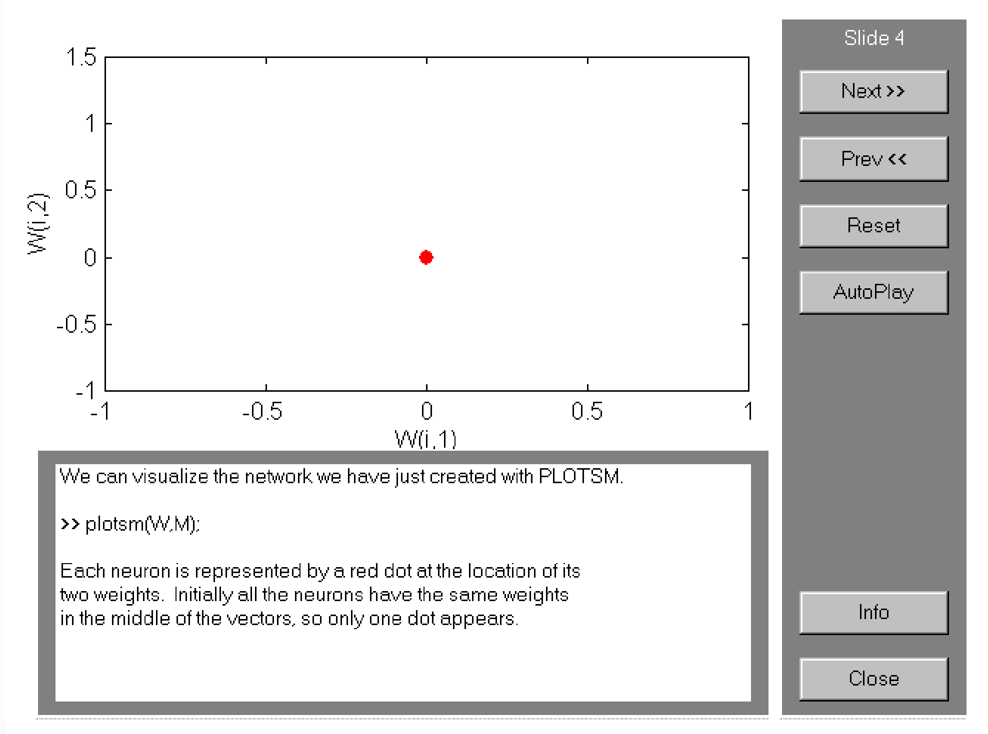
Two examples:

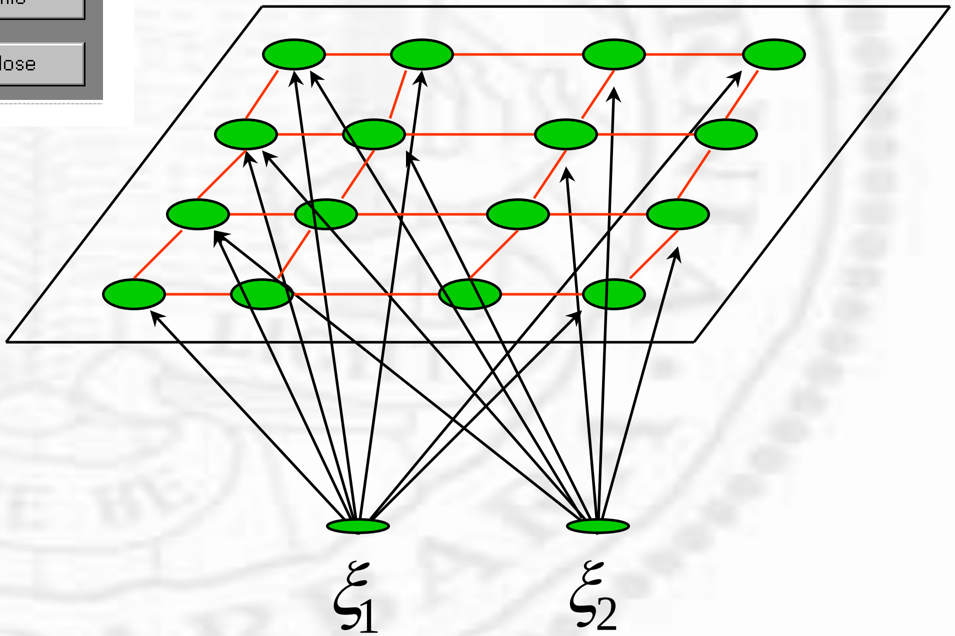
Suppose the data samples are uniformly distributed within the triangle, and each time a point inside the triangle is randomly selected as the training example. Suppose that the algorithm starts with a set of neurons, whose states are chosen at the mass center of the triangle with small perturbations. The figure below shows the final states of the neurons, one starts with neighborhood of lattice structure, and the other with linear structure.

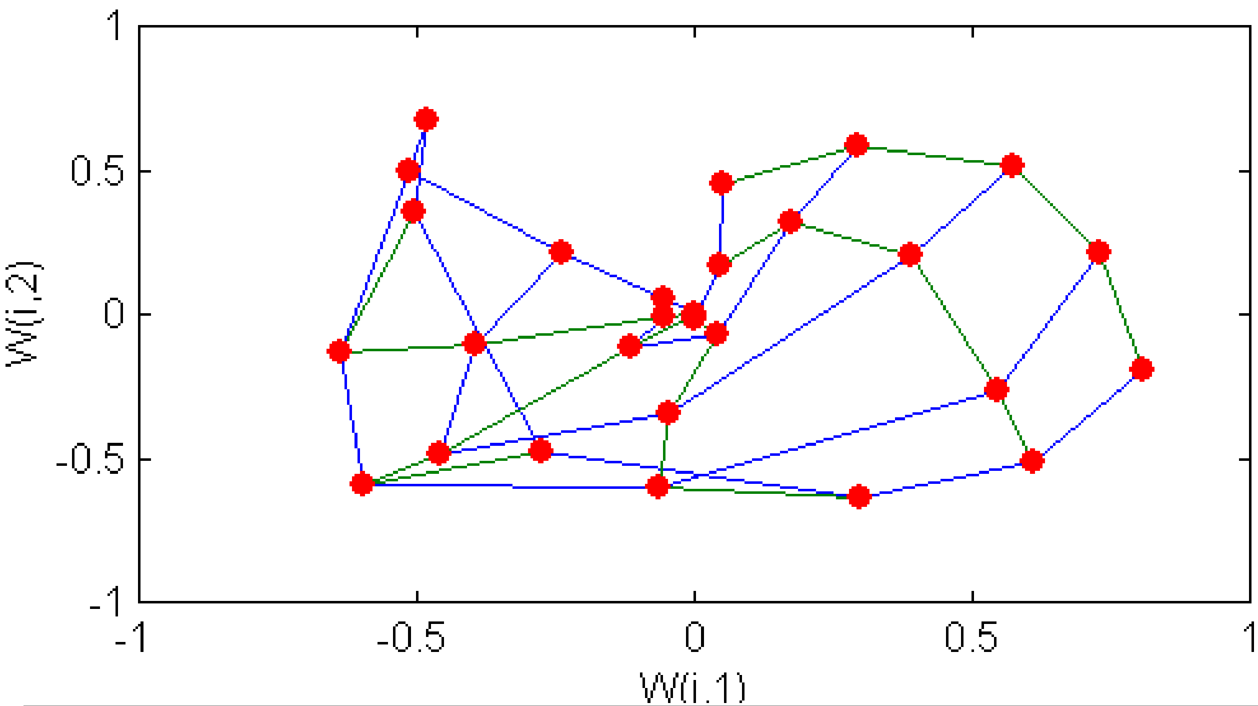


SOM Example


Slide 3
Next >>
Prev <<
Reset
AutoPlay
Info
Close
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1
We will use a 5 by 6 layer of neurons to classify the vectors above.
We would like each neuron to respond to a different region of the
rectangle, and neighboring neurons to respond to adjecent regions.
We create a layer of 30 neurons with INITSM:
>> W = initsm(P,30);
We then use NBMAN to define the distances between each neuron
given that they are spread out in a 5 by 6 grid.
>> M = nbman(5,6);


1.5
1
0.5
0
-0.5
-1
W(i,2)
-1
-0.5
0
0.5
1
W(i,1)
We can visualize the network we have just created with PLOTSM.
>> plotsm(W,M);
Each neuron is represented by a red dot at the location of its
two weights. Initially all the neurons have the same weights
in the middle of the vectors, so only one dot appears.
Slide 4
Next >>
Prev <<
Reset
AutoPlay
Info
Close


ξ_1
ξ_2



We train the map on the vectors using TRAINSM, displaying the maps progress every 20 epochs for 250 epochs.

```
>> tp = [20 250];
>> W = trainsm(W,M,P,tp);
```

Note that the layer of neurons has begun to self-organize so that each neuron now classifies a different region of the input space, and adjecent (connected) neurons respond to adjecent regions.

Slide 5

Next >>

Prev <<

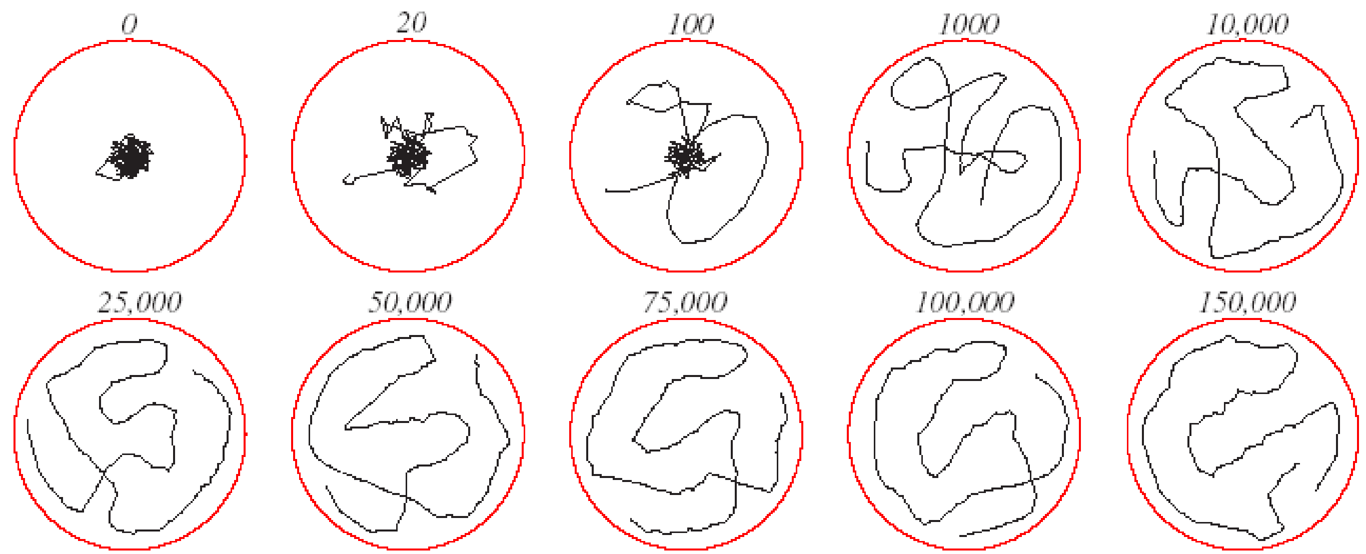
Reset

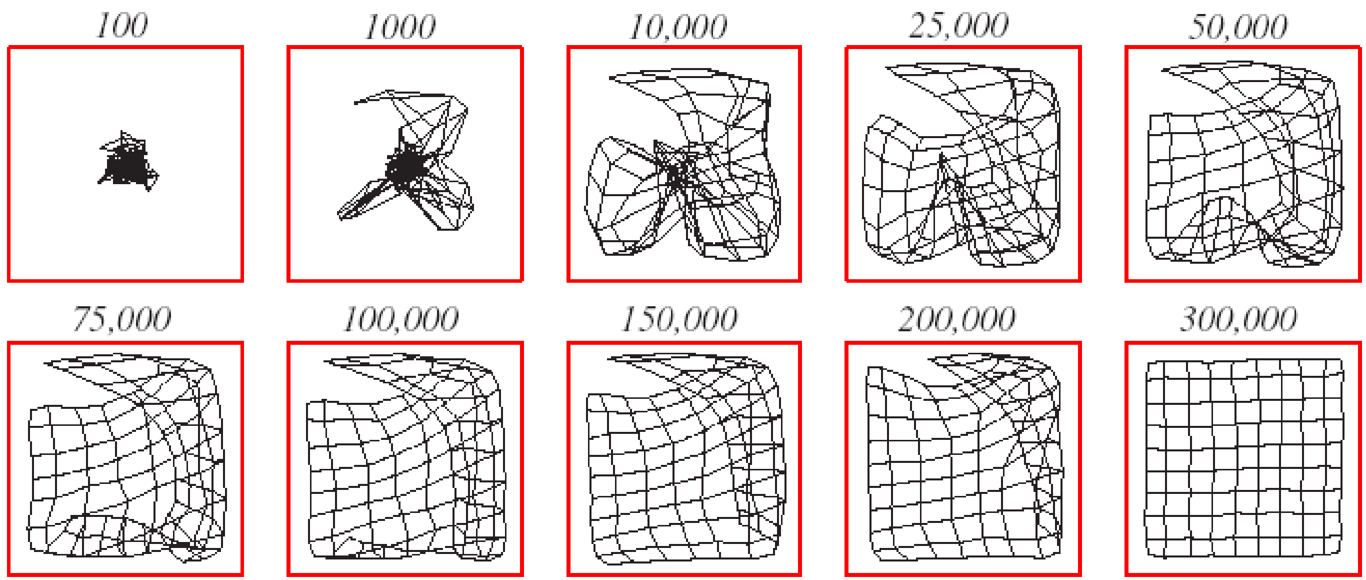
AutoPlay

Info

Close

More Examples





More Examples

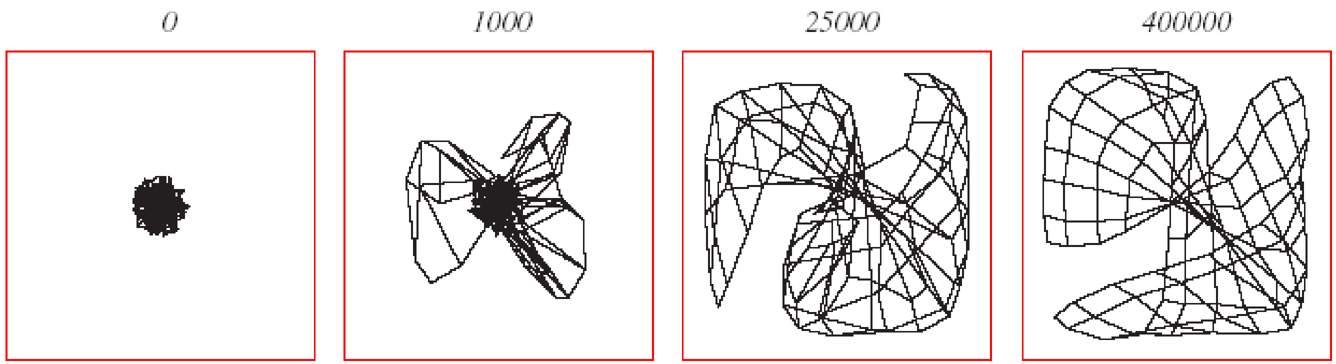


FIGURE 10.32. Some initial (random) weights and the particular sequence of patterns (randomly chosen) lead to kinks in the map; even extensive further training does not eliminate the kink. In such cases, learning should be restarted with randomized weights and possibly a wider window function and slower decay in learning. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

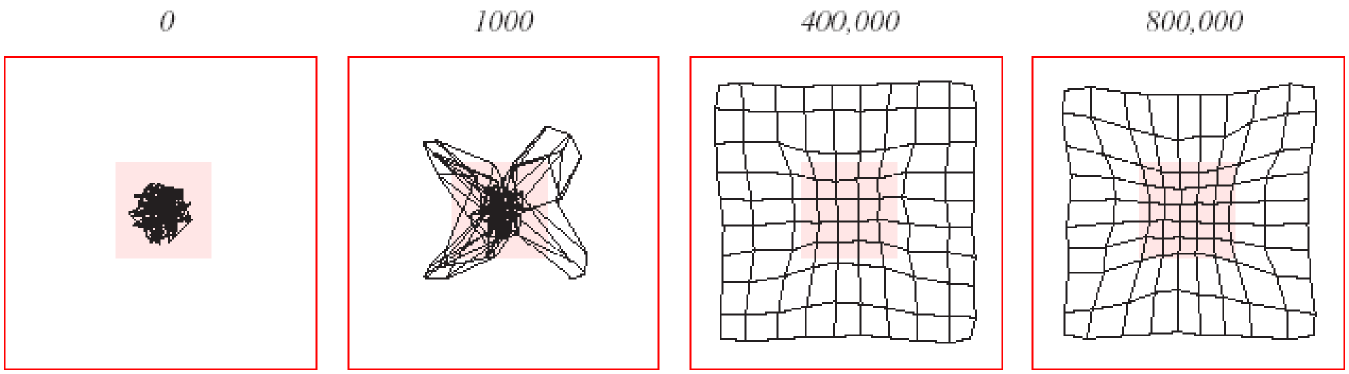
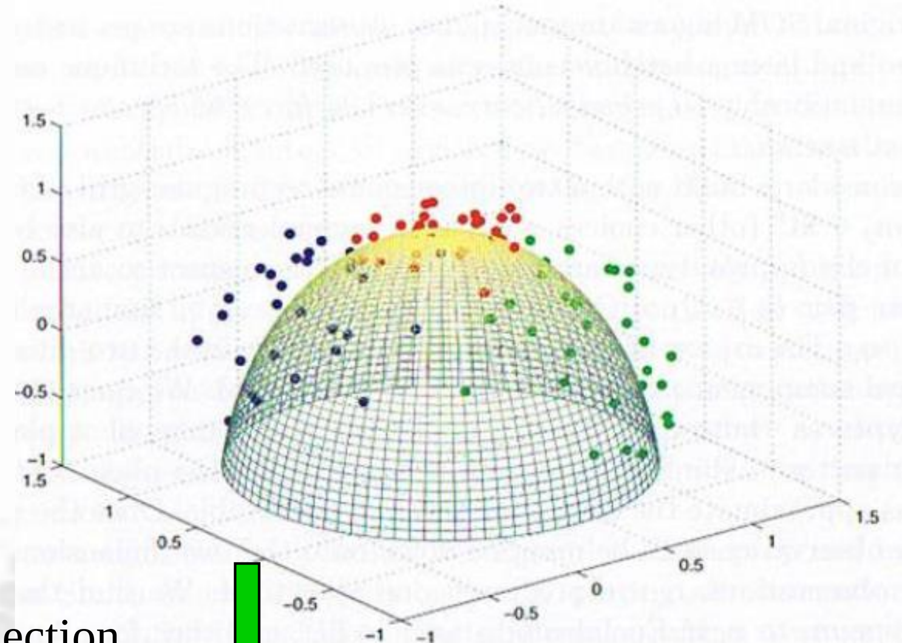
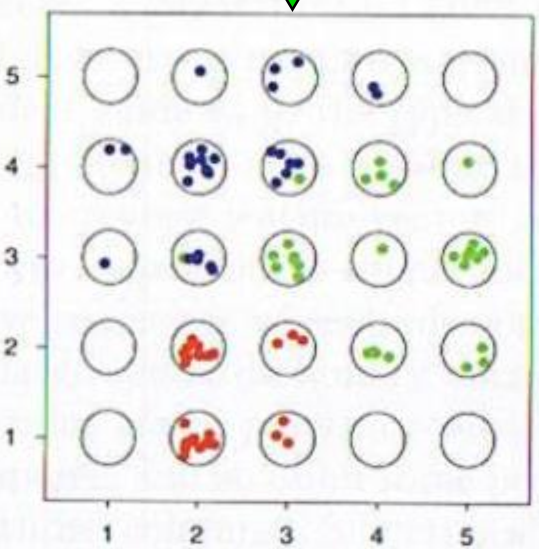


FIGURE 10.33. As in Fig. 10.31 except that the sampling of the input space was not uniform. In particular, the probability density for sampling a point in the central square region (pink) was 20 times greater than elsewhere. Notice that the final map devotes more nodes to this center region than in Fig. 10.31. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

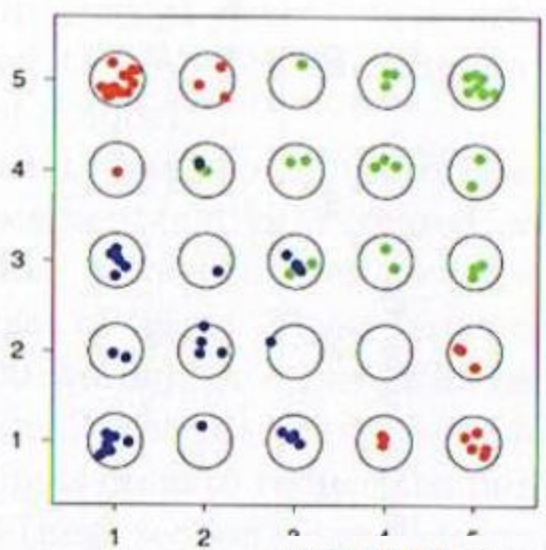


Three data clusters in 3D

Projection
From 3D to 2D



SOM



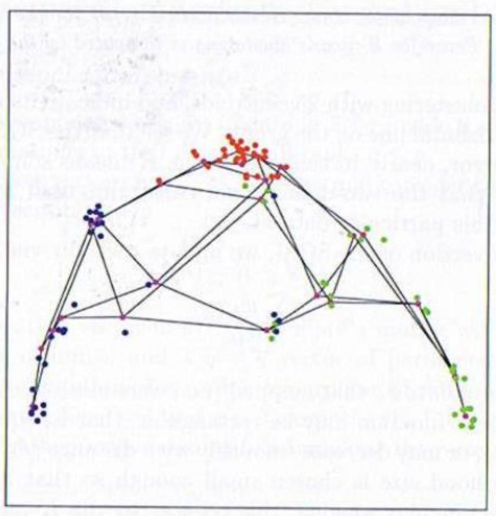


FIGURE 14.17. *Wiremesh representation of the fitted SOM model in $\mathbb{R}^3$. The lines represent the horizontal and vertical edges of the topological lattice. The double lines indicate that the surface was folded diagonally back on itself in order to model the red points. The cluster members have been jittered to indicate their color, and the purple points are the node centers.*

Traveling salesman problem

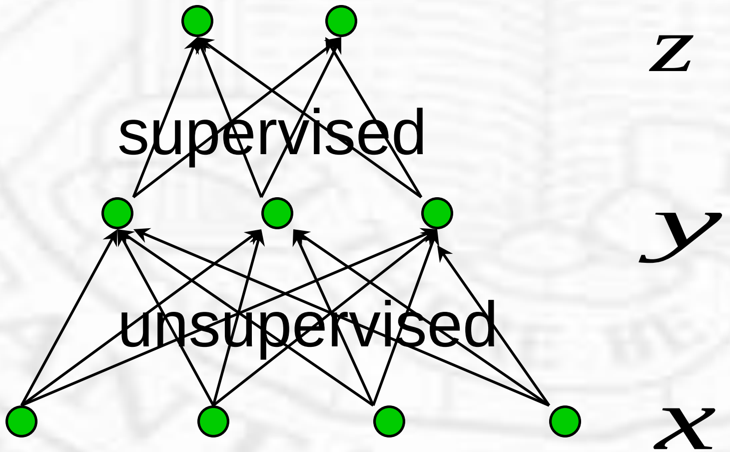
- Mapping from a plane to a 1D ring
- Modified Kohenen algorithm
- Standard decision rule + update rule:
  - 1st term: pulling weight to a particular city
  - 2nd term: minimize inter-city distance

$$\Delta w_i = \eta\left(\sum_u \Lambda^u(i)(x^u - w_i) + k(w_{i+1} - 2w_i + w_{i-1})\right)$$

$$\Lambda^u(i) = \frac{e^{-|x^u - w_i|^2/2\sigma^2}}{\sum_j e^{-|x^u - w_j|^2/2\sigma^2}}$$

Hybrid Learning Schemes

- Improved speed
- Satisfactory performance
- Unsupervised layer: clustering (divide input space in a Voronoi tessellation)
- Supervised layer: key-value lookup



- Example 1:
  - input to hidden layer: competitive learning
  - hidden to output layer: general delta rule

$$| w_{i^*} - x | \le | w_i - x | (for all i)$$

$$\Delta w_{ij} = \eta (z_i^u - O_i^u) y_j$$

- Example 2 (radial basis function):
  - input to hidden layer:

$$g_i(x) = \frac{e^{-(x-u_i)^2 / 2\sigma_i^2}}{\sum_j e^{-(x-u_j)^2 / 2\sigma_j^2}}$$