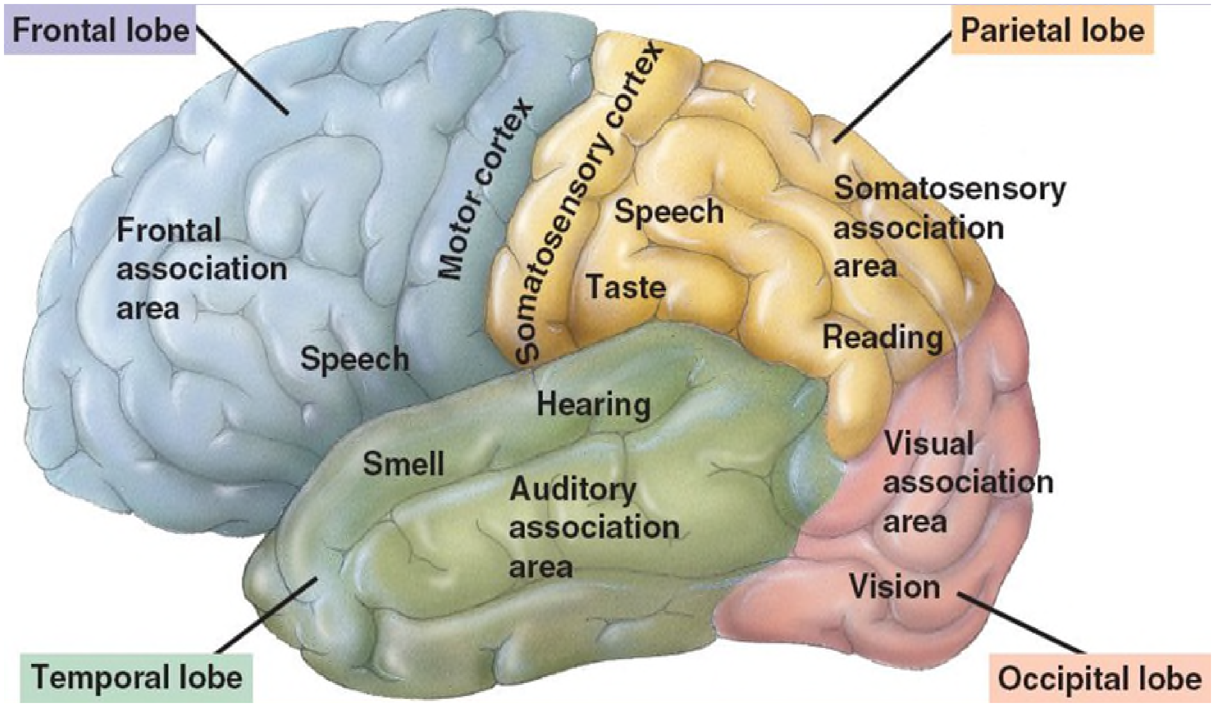
2. Self-Organizing Map Neural Network



The map of a human brain

Some of the example regions are:

- Motor cortex
- Visual cortex
- Auditory cortex
- Speech cortex

The brain map shows clearly that

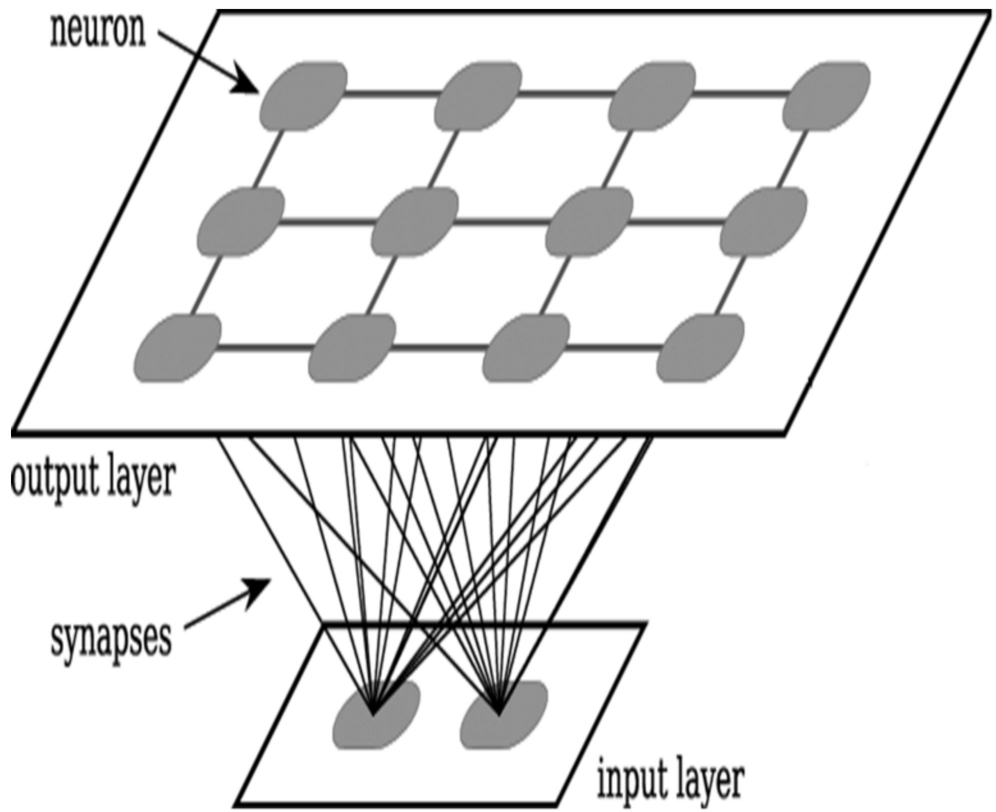
(1) Different sensory inputs (motor, visual, auditory etc.) are mapped to corresponding regions in an orderly fashion;

(2) These cortical maps are not entirely genetically pre-determined. Rather, they are sketched in during the early development of the nervous system.

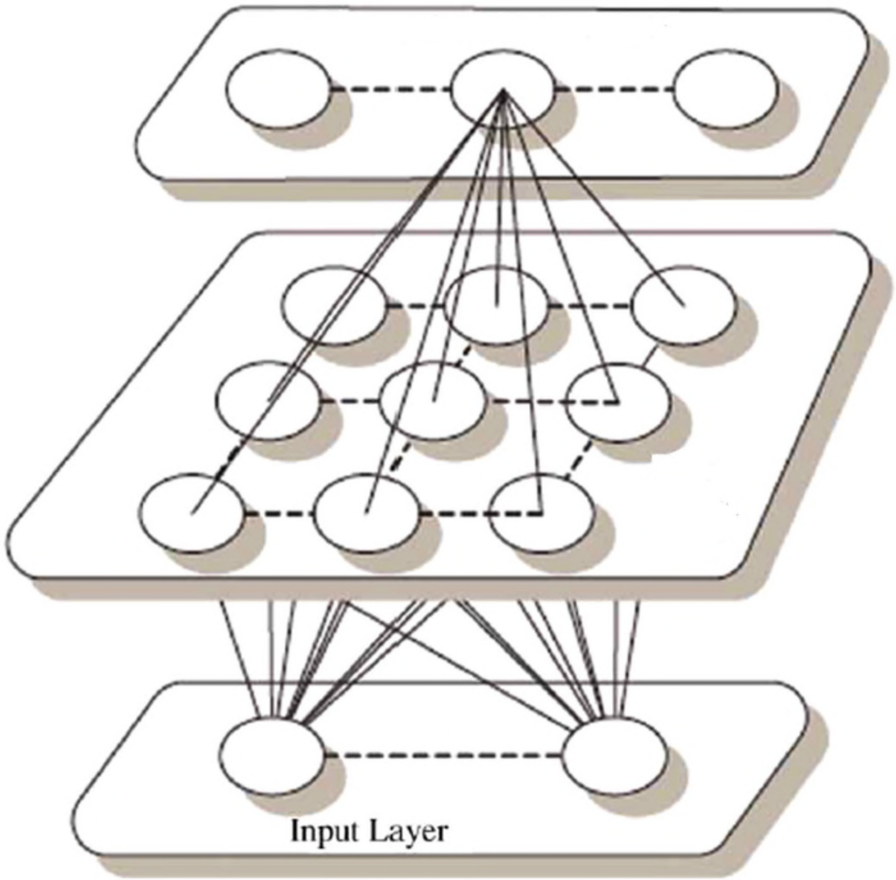
Principle of topographic map

The essential point of the discovery in the neurobiology lies in the principle of topographic map formation stated as follows:

The spatial location of an output neuron in the topographic map corresponds to a particular domain or feature of the input data.



Model 1 of the self-organizing map (SOM) neural network



The second layer

The first layer

Model 2 of the self-organizing map neural network

- **Ingredients of Self-Organizing Map (SOM)**

The essential ingredients of the SOM neural network include:

- ❑ A 1D or 2D lattice of neurons that computes simple discriminant functions of inputs received.
- ❑ A mechanism that compares these functions and selects the neuron with the largest discriminant function value.
- ❑ An adaptive process that enables the activated neurons to increase their discriminant values.
- ❑ An interactive network that activates the selected neuron and its neighbors.

The above points are named as follows:

- **Competition**. For each input pattern, the neurons in the network compute their respective values of a discriminant function. This discriminant function provides the basis for competition among the neurons;
- **Cooperation.** The winning neuron determines the spatial location of a topological neighborhood of excited neurons in the 1D or 2D lattice;
- **Weights adaptation**. This mechanism enables the excited neurons to adjust their weights.

(1) Competition process

Assuming that an input pattern selected randomly from the input space is denoted by:

$$\mathbf{x} = [x_1, x_2, \cdots, x_m]^T$$

Let the weight vector of neuron *j* be denoted by:

$$\mathbf{w}_j = [w_{j1}, w_{j2}, \cdots, w_{jm}]^T$$

The similarity between $\mathbf{x}$ and $\boldsymbol{w}_j$ can be used as the discriminant function for competition. One of the similarity measures that we can use here is the cosine similarity measure:

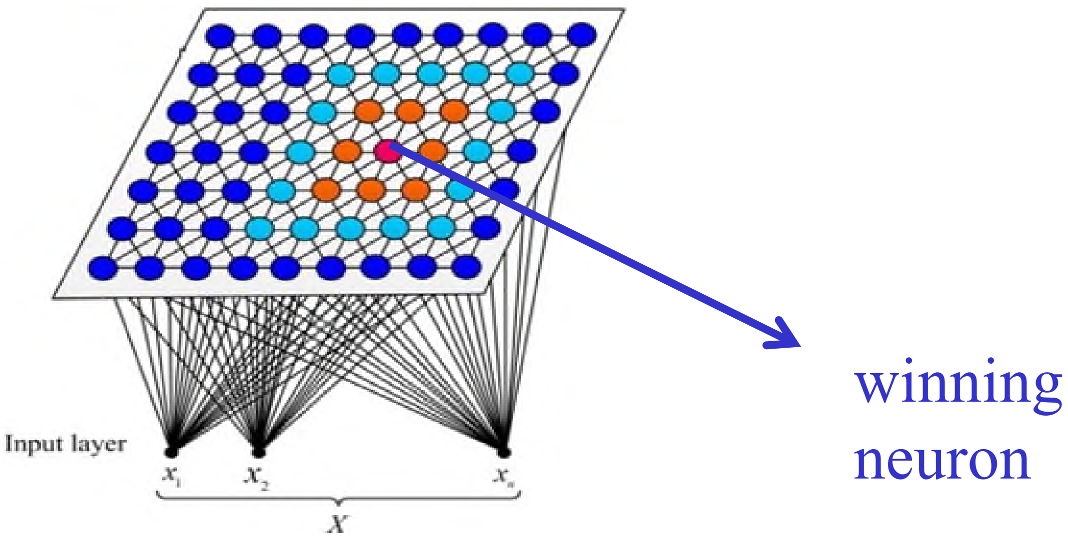
$$\frac{\mathbf{w}_j^T \mathbf{x}}{\|\mathbf{w}_j\| \|\mathbf{x}\|}$$

Where $\|\cdot\|$ denote the norm of a vector

In practice, Euclidean distance is more often used.

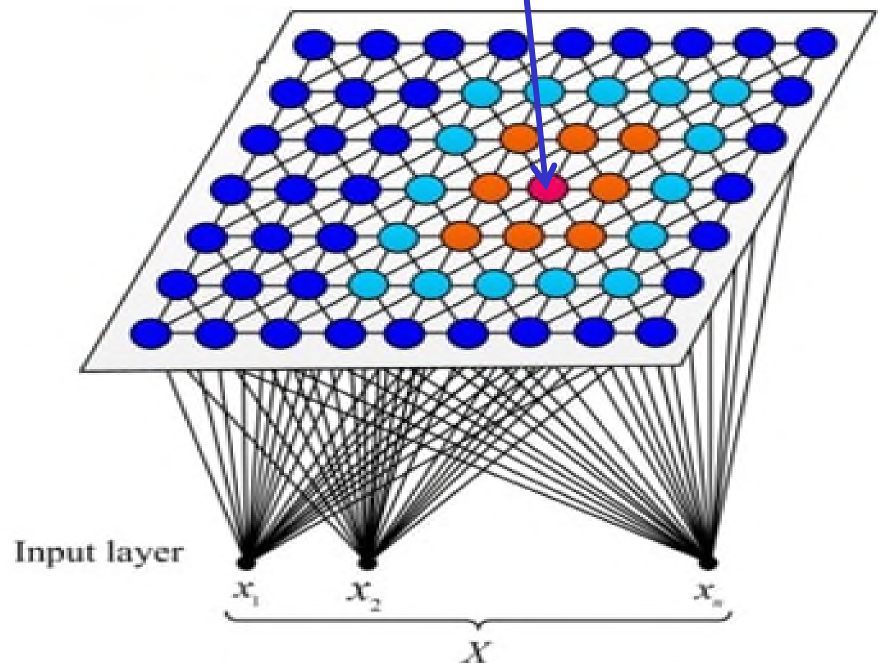
If we use $i(\mathbf{x})$ to denote the index of the neuron that best matches the input vector $\mathbf{x}$, we may then determine $i(\mathbf{x})$ by using:

$$i(\mathbf{x}) = \operatorname{argmin}\|\mathbf{x} - \mathbf{w}_j\|$$



(2) Cooperation process

winning neuron
and its neighbourhood



Let h_{ji} denote the topological neighborhood centered on winning neuron i.

d_{ji} denotes the lateral distance between winning neuron i and neuron j in the 1D or 2D lattice.

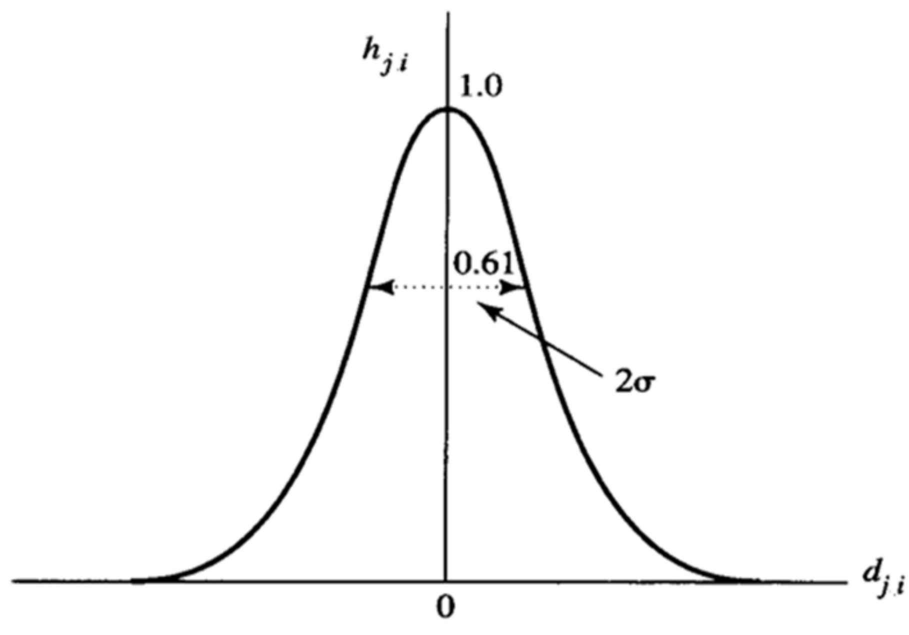
The neighborhood h_{ji} is a function of d_{ji} satisfying two distinct requirements:

(1) The topological neighborhood h_{ji} attains its maximum value at the winning neuron.

(2) The amplitude of the topological neighborhood h_{ji} decreases monotonically with increasing lateral distance d_{ji}, and decays to zero when $d_{ji} \rightarrow \infty$.

A typical choice of h_{ji} that satisfies the two requirements is the Gaussian function:

$$h_{ji} = \exp\left(-\frac{d_{ji}^2}{2\sigma^2}\right)$$

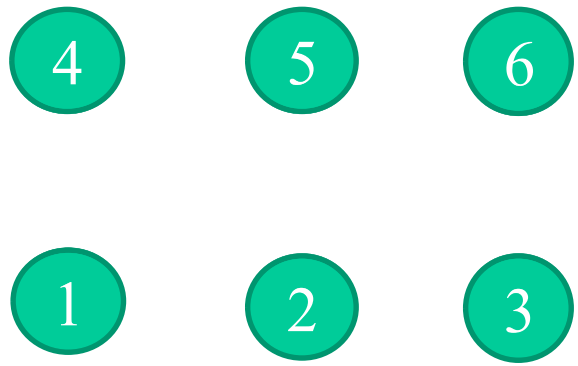


Lateral distance d_{ji} is the distance between neuron i and j in the 2D (or 1D) lattice defined as:

$$d_{ji} = \|\mathbf{r}_j - \mathbf{r}_i\|$$

Where $\mathbf{r}_i$ and $\mathbf{r}_j$ denotes the discrete position of winning neuron i and neuron j measured in the output space (2D lattice), respectively.

Example: 2×3 lattice



Neuron 1 position is $\mathbf{r}_1 = (0,0)$

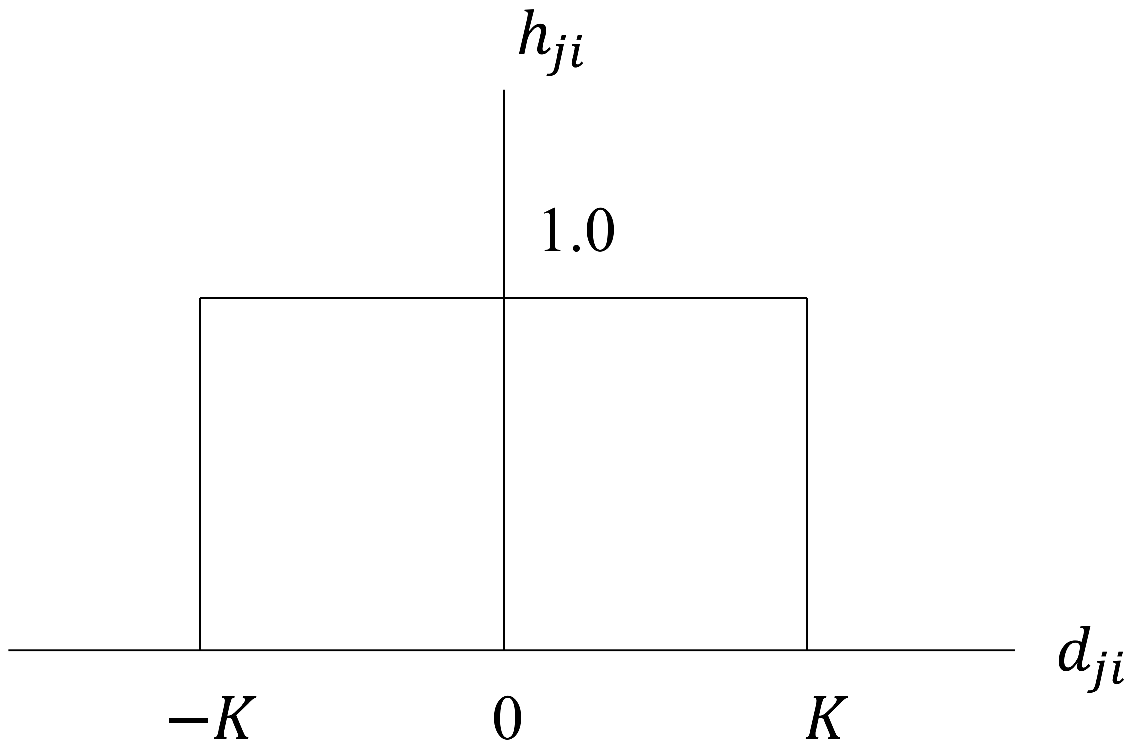
Neuron 2 position is $\mathbf{r}_2 = (1,0)$

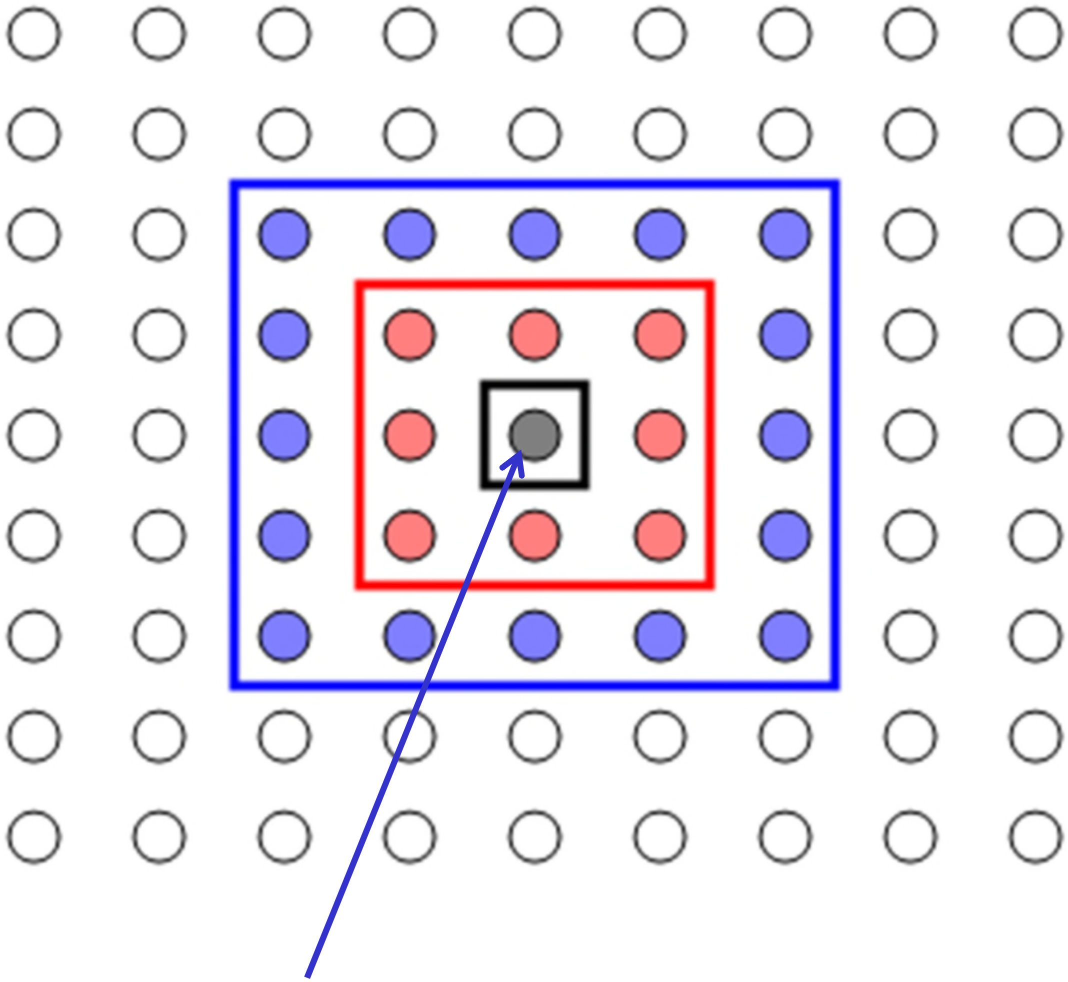
Neuron 6 position is $\mathbf{r}_6 = (2,1)$

The lateral distance between neurons 1 and 6 is:

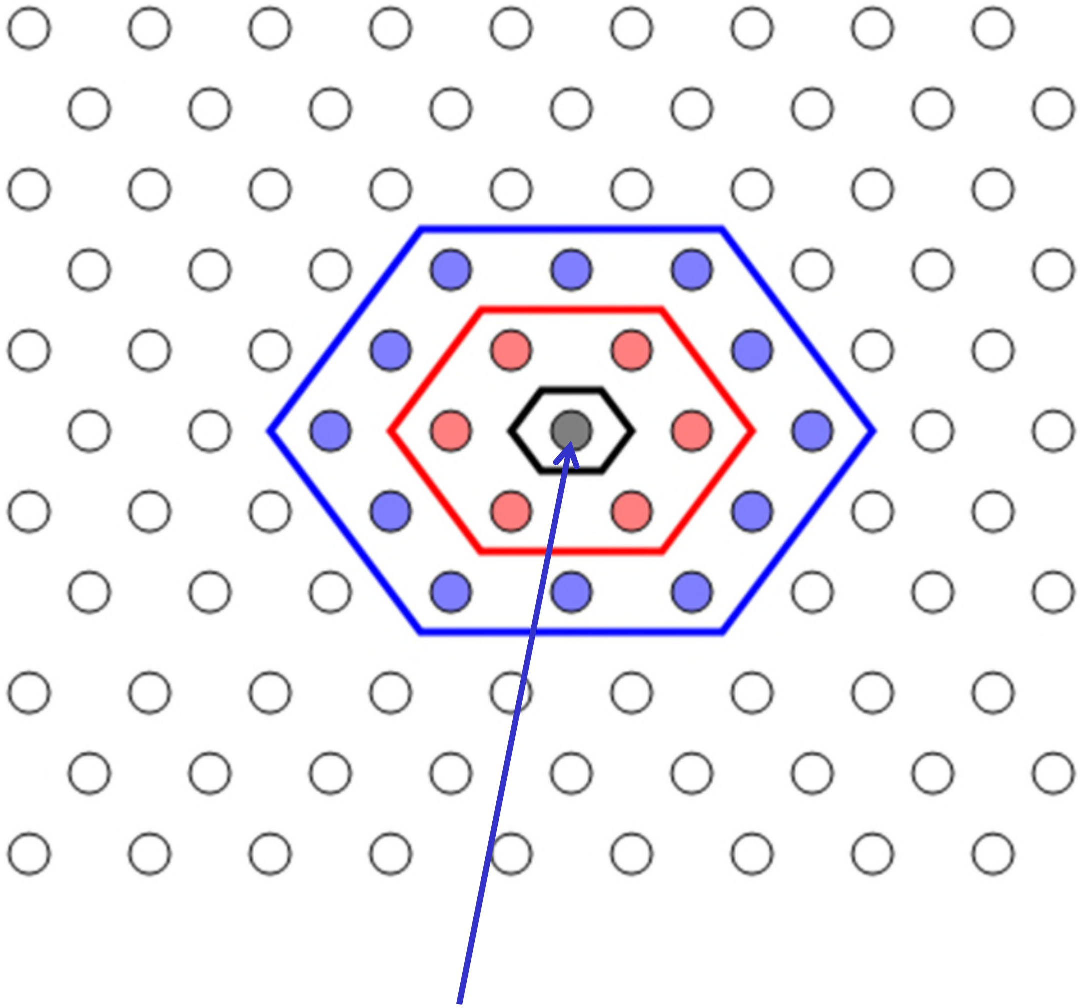
$$d_{16} = \|\mathbf{r}_1 - \mathbf{r}_6\| = \sqrt{(2-0)^2 + (1-0)^2} = 2.2361$$

Besides the Gaussian neighborhood function, hexagon and square neighborhood can also be used in SOM. Neurons inside the defined neighborhood K are excited. In such cases, the h_{ji} is in the following form:





winning neuron and its square neighborhood



winning neuron and its hexagon neighborhood

(3) Adaptive process

In the weight adaption process, the weight vector $\mathbf{w}_j$ of neuron j in the network is required to change in relation to the input vector $\mathbf{x}$:

$$\Delta \mathbf{w}_j(n) = \eta(n) h_{ji(\mathbf{x})}(n)[\mathbf{x} - \mathbf{w}_j(n)]$$

Where $\eta(n)$ is the learning rate at iteration n.

After change, the weight vector becomes:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{ji(\mathbf{x})}(n)[\mathbf{x} - \mathbf{w}_j(n)]$$

One important feature of SOM is that the size of the topological neighborhood shrinks with the learning process.

This can be implemented by making the width parameter σ of the topological neighborhood function h_{ji} decrease with the learning process as follows:

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right)$$

Where σ_0 is the value at the initiation of the SOM algorithm, and τ_1 is a time constant, n is the number of iterations.

Then the topological neighborhood is:

$$h_{ji}(n) = \exp\left(-\frac{d_{ji}^2}{2\sigma^2(n)}\right)$$

The learning-rate parameter $\eta(n)$ should also be time-varying.

It should start at an initial value and then decrease gradually with the increase of iteration number n as shown below:

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right)$$

Where η_0 is the initial value, and τ_2 is another time constant.

- **Two phases of the adaptive process**

(1) Self-organizing

It is in this first phase of the adaptive process that topological ordering of the weight vectors takes place.

❑ Learning Rate

It is found experimentally that the learning rate parameter $\eta(n)$ should begin with a value close to 0.1; and the value decreases gradually but remains above 0.01. These desirable values are satisfied by the setting $\eta_0 = 0.1$ and $\tau_2 = 1000$.

Thus, the varying learning rate formula becomes:

$$\eta(n) = 0.1 \exp\left(-\frac{n}{1000}\right)$$

❑ Neighbourhood Function

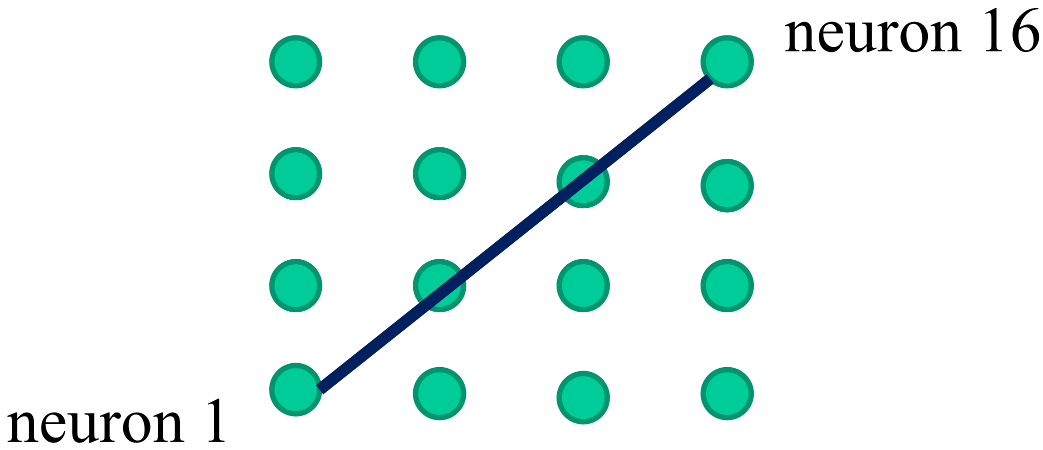
The neighbourhood function $h_{ji}(n)$ should initially include almost all neurons in the network centered on the winning neuron i, and then shrink slowly with iterations.

We may set the size of σ_0 to the radius of the two-dimensional lattice, and set the time constant τ_1 as follow:

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right)$$

$$\tau_1 = \frac{1000}{\ln \sigma_0}$$

Example: 4×4 two-dimensional lattice



Assuming the position of neuron 1 is (0,0), and the position of neuron 16 is (3,3), then the lateral distance between the two neurons is:

$$d = \sqrt{(3-0)^2 + (3-0)^2} = 4.242$$

The initial value of the width can be set as:

$$\sigma_0 = \frac{d}{2} = 2.121$$

(2) Convergence phase

This second phase of the adaptive process is needed to fine tune the map and therefore provide an accurate statistical quantification of the input space.

As a general rule, the number of iterations of the convergence phase must be at least 500 times the number of neurons in the network. Thus, the convergence phase may have to go on for thousands and possibly tens of thousands of iterations.

For good performance, the learning rate parameter $\eta(n)$ should be maintained during the convergence phase at a small value on the order of 0.01 but should not be zero.

We may, for example, simply set the learning rate to a small constant:

$$\eta(n) = 0.01$$

The neighborhood function $h_{ji}(n)$ should contain only the nearest neighbors of a winning neuron, which may eventually reduce to one or zero neighboring neurons:

$$h_{ji}(n) = \begin{cases} 1 & \text{if neuron } j \text{ is the winning neuron } i \\ 0 & \text{otherwise} \end{cases}$$

- **Summary of the SOM algorithm**

(1) Initialization. Choose random values for the initial weight vectors $\mathbf{w}_j(0)$. The only restriction on the initialization is that the initial weight vectors must be different from each other, and it is desirable to keep the magnitude of the weights small.

(2) Sampling. Draw a sample $\mathbf{x}$ from the input distribution with a certain probability. Usually, the $\mathbf{x}$ is drawn from the given training samples set.

(3) Competition. Find the winning neuron $i(\mathbf{x})$ at step n using the minimum-Euclidean distance criterion:

$$i(\mathbf{x}) = \arg\min\left\|\mathbf{x} - \mathbf{w}_j(n)\right\|$$

where $j = 1,2, \dots, N$.

(4) Updating (cooperation and adaption). Adjust the weight vectors of all neurons using the updating formula:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{ji(\mathbf{x})}(n)[\mathbf{x} - \mathbf{w}_j(n)]$$

(5) Continuation. Repeat steps (2)-(4) until the stopping criterion is satisfied. The stopping criterion can be:

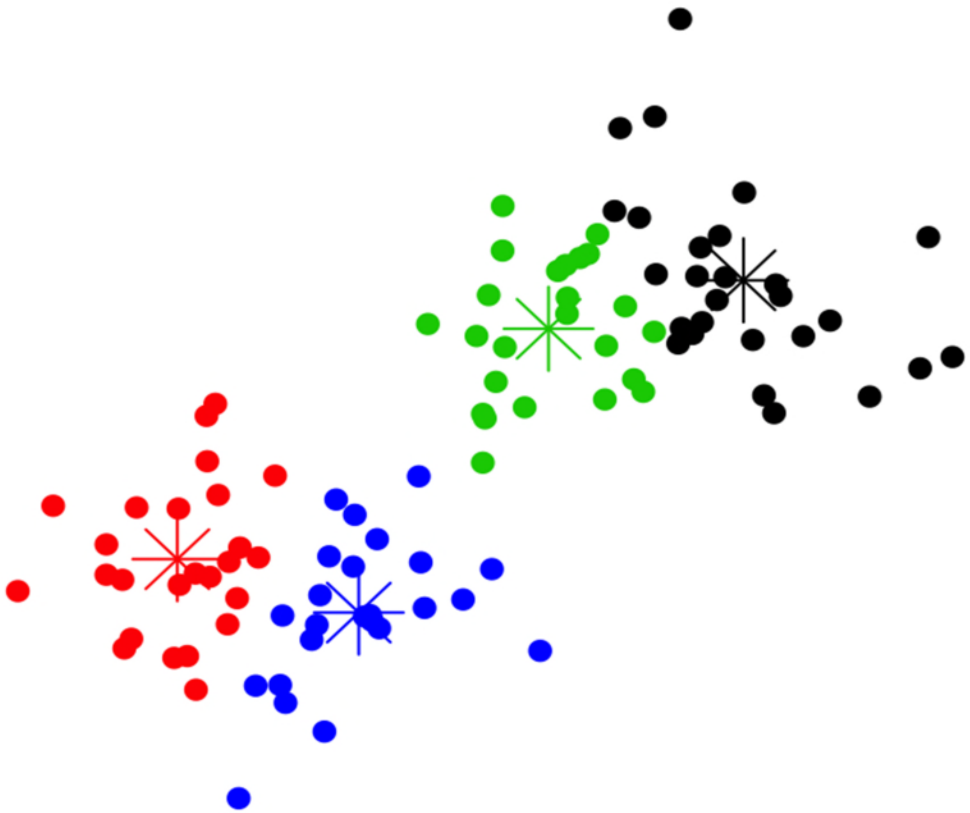
(i) Pre-defined iteration number;
(ii) No noticeable changes in the map

Remarks:

- In the self-organizing phase, random values are assigned to the initial weights.
- In the convergence phase, the weights learned in the self-organizing phase will be used as initial weights.

	Self-organizing Phase	**Convergence Phase**
Initial weights	Random values	Weights learned in the self-organizing phase are used as initial values
Learning rate	Decay with increasing iterations	Set to a small constant, say 0.01
Neighbourhood	All neurons are considered as the neighbourhood of the winning neuron, and all neurons update weights	Only the winning neuron updates weights

Upon repeated presentation of the training data, the weight vectors (denoted by ✳) tend to follow the distribution of the input vectors.



Next a few slides show the Matlab implementation of the SOM training algorithm to enhance your understanding.

```
function w=SOM(data,n1,n2);
% Input
% data----data matrix, each row is one sample
% n1,n2 ---the no. of rows and columns of the 2D lattice
% Output:
% w ---- weight vectors of the neurons
%
% No. of samples, dimensionality of input space, and total number of
neurons
[nSample,nDim]=size(data);
nNeuron=n1*n2;

% Generate the initial weight vectors
w=randn(nNeuron,nDim);

% Define initial values for the time constants and the learning rate
eta0=0.1;
sigma0=sqrt((n1-1)^2+(n2-1)^2)/2;
tau1=1000/log(sigma0);

% Generate the lateral distance matrix
Dist=distance_matrix(n1,n2);
```

```
% The self-organizing phase
for k=1:1000
  % calculate the learning rate and width of the neighbourhood function at current iteration
  eta=eta0*exp(-k/1000);
  sigma=sigma0*exp(-k/tau1);

  % randomly select a training sample
  j=randperm(nSample,1);
  x=data(j,:);

  % compete and find the winning neuron
  for i=1:nNeuron
      d(i,1)=(w(i,:)-x)*(w(i,:)-x)';
  end
  [xx,index_win]=min(d);

  % update weight vectors of all neurons
  for i=1:nNeuron
      h=exp(-Dist(i,index_win)^2/2/sigma^2);
      w(i,:)=w(i,:)+eta*h*(x-w(i,:));
  end
end
```

```
% The convergence phase
% set the learning rate to a small constant
eta=0.01;

% repeat 500*nNeuron times
for k=1:500*nNeuron
    % randomly select a sample
     j=randperm(nSample,1);
     x=data(j,:);

   % compete and find the winning neuron
    for i=1:nNeuron
        d(i,1)=(w(i,:)-x)*(w(i,:)-x)';
    end
    [xx,index_win]=min(d);

   % update the weight vector of the winning neuron only.
    h=1;
    w(index_win,:)=w(index_win,:)+eta*h*(x-w(index_win,:));

end
```

```
function Dist=distance_matrix(n1,n2);
% This function calculates the lateral distance between neurons in
the 2D lattice
% Dist(i,j) denotes the distance between neurons i and j
%
% Define the position or coordinates of all neurons in the 2D lattice
for i=1:n1
    posit((i-1)*n2+1:i*n2,1)=i-1;
    posit((i-1)*n2+1:i*n2,2)=0:n2-1;
end
%
% calculate the lateral distance in the 2D lattice between neurons
Dist=pdist2(posit,posit);
```

Example 1 (for demonstration purpose only)

Given some training samples:

$$\mathbf{x}_1 = [1,1,0,0]^T \qquad \mathbf{x}_2 = [0,0,0,1]^T$$

$$\mathbf{x}_3 = [1,0,0,0]^T \qquad \mathbf{x}_4 = [0,0,1,1]^T$$

We wish to find 2 clusters of the training samples.

Suppose the learning rate is set as:

$$\eta(0) = 0.1$$
$$\eta(n+1) = \eta(0)\exp(-n/1000)$$

The neighborhood function is so set that only the winning neuron is updated with its weights at each step.

Step 1: initialization. Initialize two weight vectors:

$$\mathbf{w}_1(0) = [0.2,0.6,0.5,0.9]^T$$

$$\mathbf{w}_2(0) = [0.8,0.4,0.7,0.3]^T$$

Step 2: randomly select a sample, say $\mathbf{x}_1$, we have:

$$d_1 = \|\mathbf{w}_1(0) - \mathbf{x}_1\|$$
$$= \sqrt{(0.2-1)^2 + (0.6-1)^2 + (0.5-0)^2 + (0.9-0)^2} = 1.3638$$
$$d_2 = \|\mathbf{w}_2(0) - \mathbf{x}_1\| = 0.99$$

Neuron 2 is the winning neuron, and its weights are updated:

$$\mathbf{w}_2(1) = \mathbf{w}_2(0) + \eta(0)[\mathbf{x}_1 - \mathbf{w}_2(0)]$$
$$= [0.82,0.46,0.63,0.27]^T$$

The weights of neuron 1 remain unchanged:

$$\mathbf{w}_1(1) = \mathbf{w}_1(0) = [0.2,0.6,0.5,0.9]^T$$

Step 3: randomly select another sample, say $\mathbf{x}_2$, we have:

$$d_1 = \|\mathbf{w}_1(1) - \mathbf{x}_2\| = 0.8124$$
$$d_2 = \|\mathbf{w}_2(1) - \mathbf{x}_2\| = 1.3468$$

Neuron 1 wins, and its weights are updated:

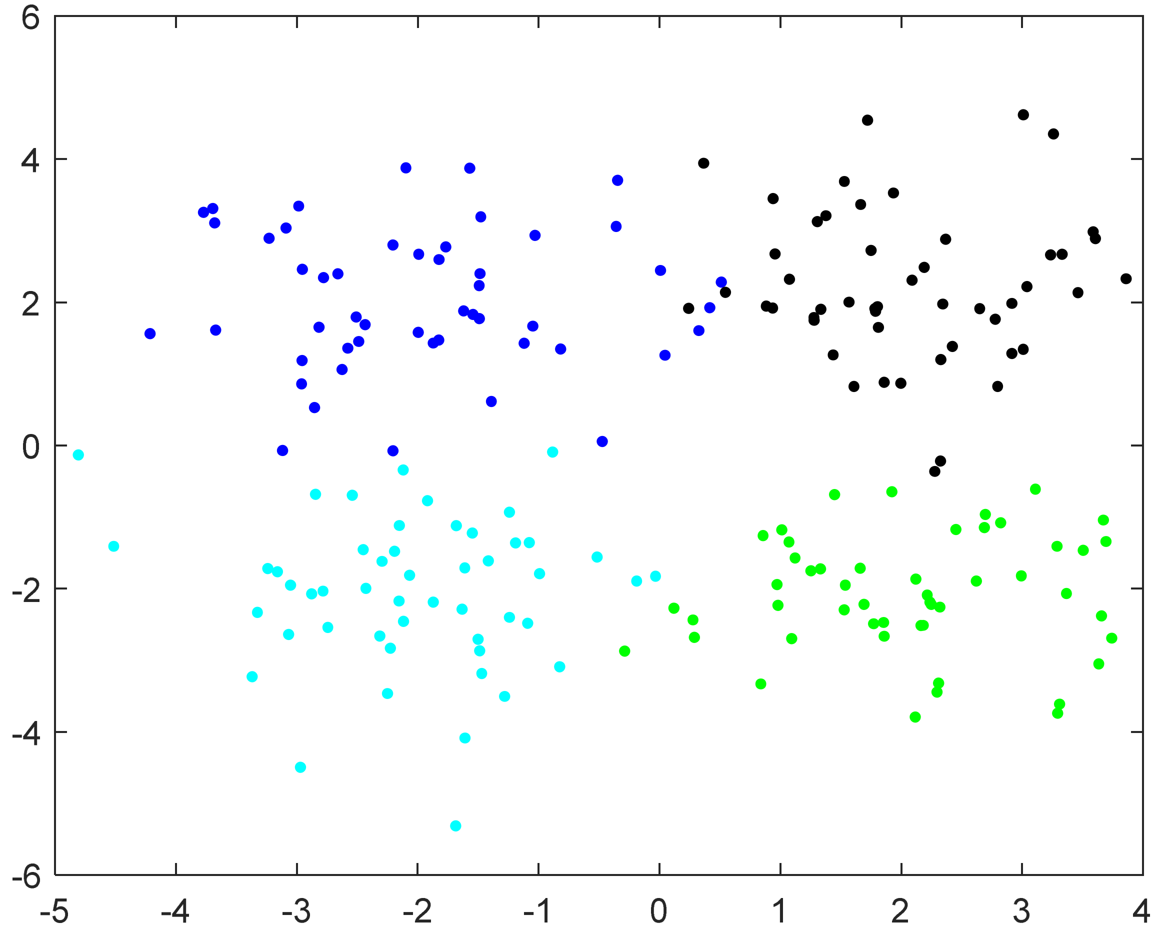
$$\eta(1) = \eta(0) \times \exp(-1/1000) = 0.0999$$
$$\mathbf{w}_1(2) = \mathbf{w}_1(1) + \eta(1)[\mathbf{x}_2 - \mathbf{w}_1(1)]$$
$$= [0.18, 0.5401, 0.4501, 0.91]^T$$

The weights of Neuron 2 remain unchanged:

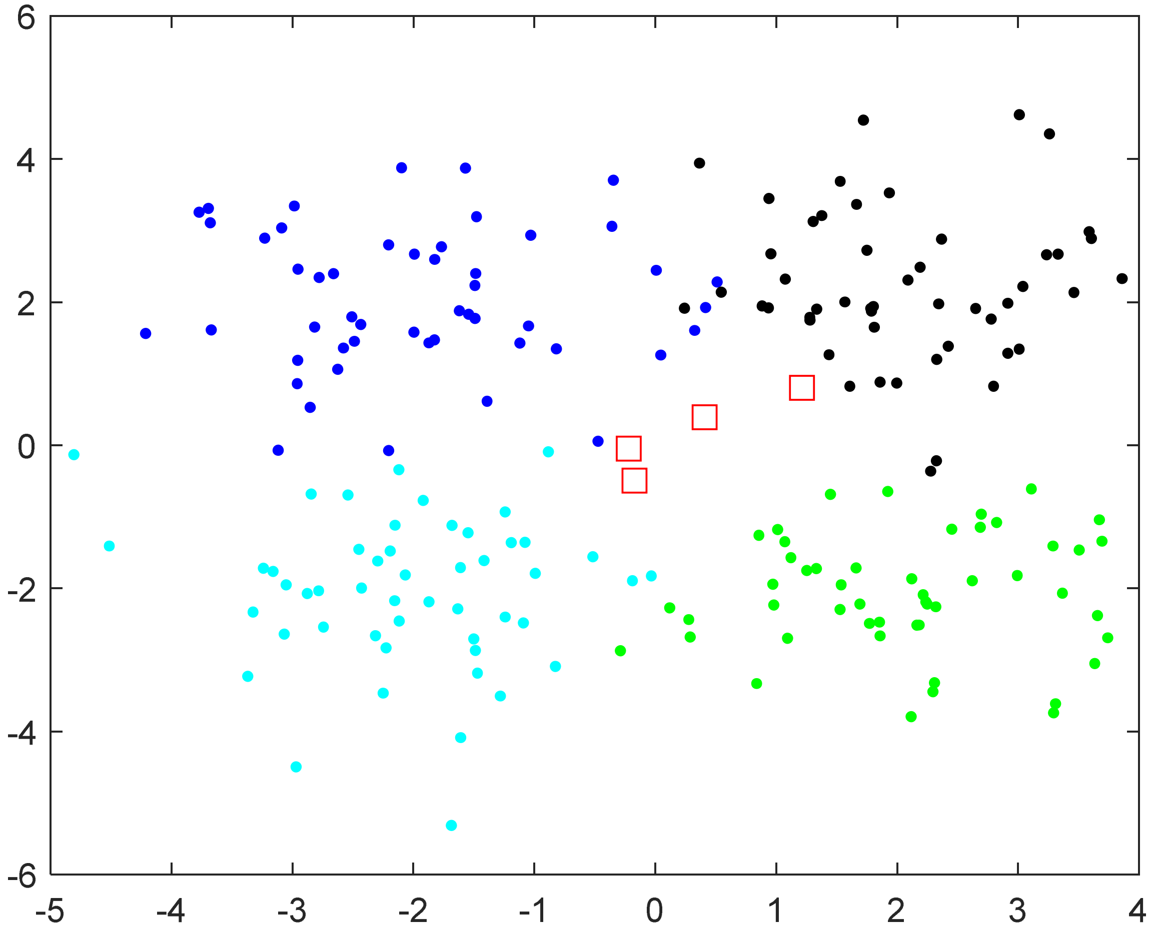
$$\mathbf{w}_2(2) = \mathbf{w}_2(1) = [0.82, 0.46, 0.63, 0.27]^T$$

The above process is continued until the stopping criterion is satisfied.

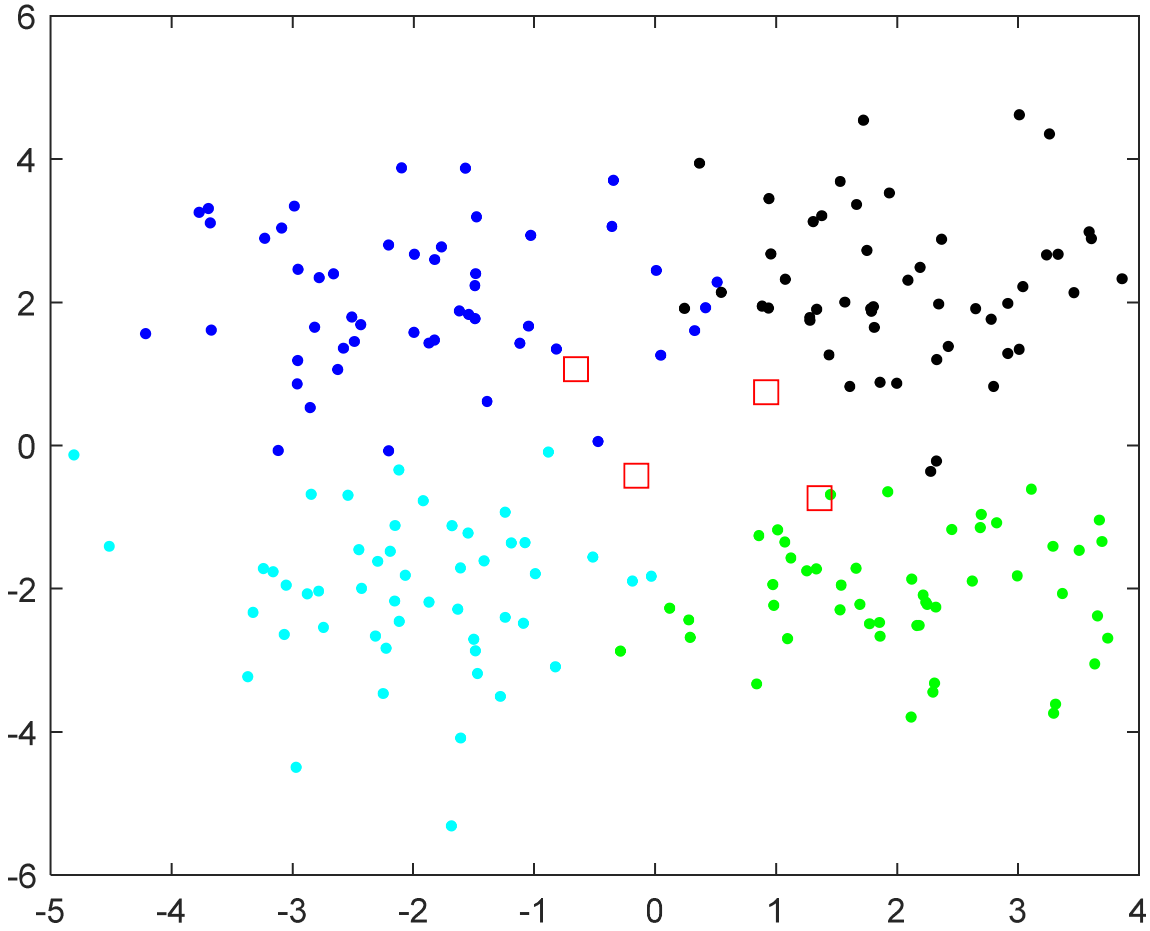
Example 2



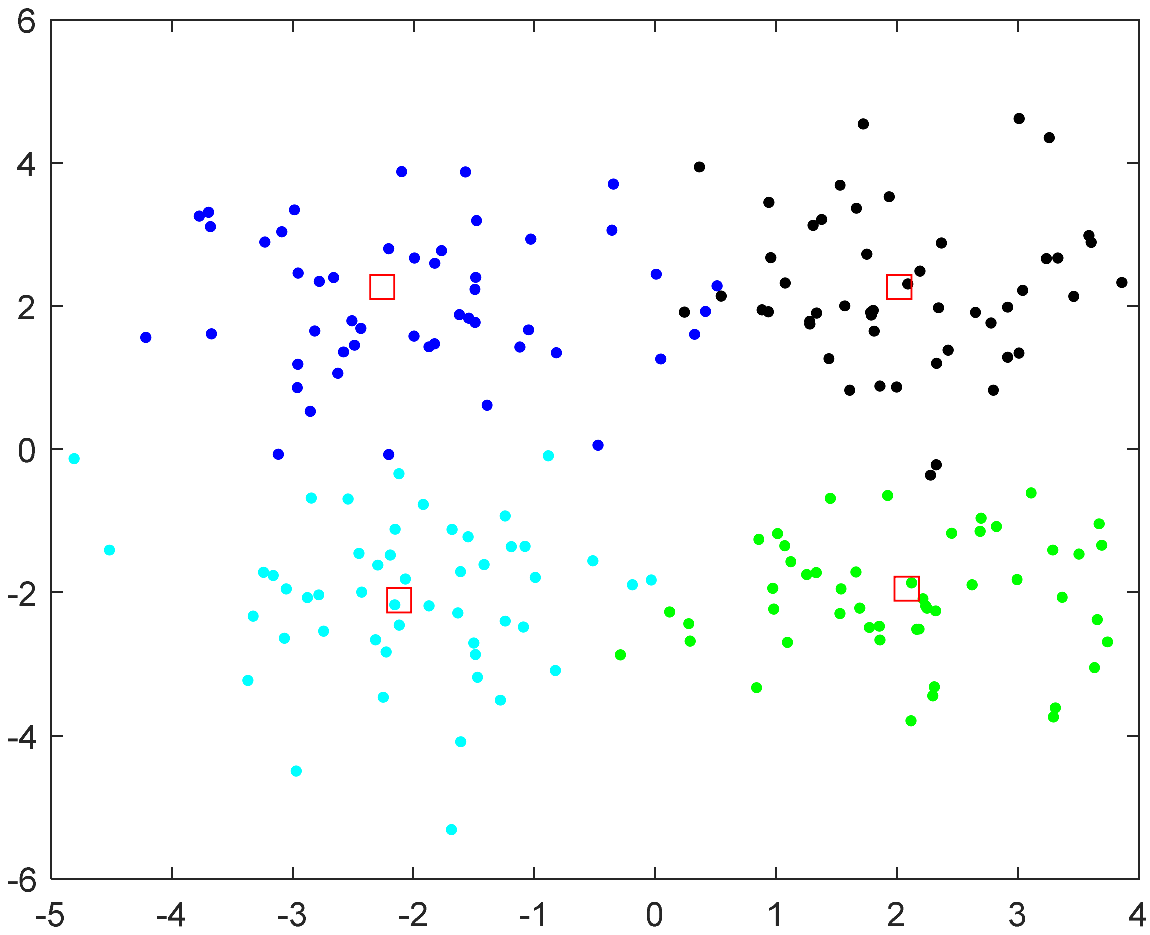
Initial weight vectors:



Weight vectors at the end of the self-organizing phase



Weight vectors at the end of the convergence phase

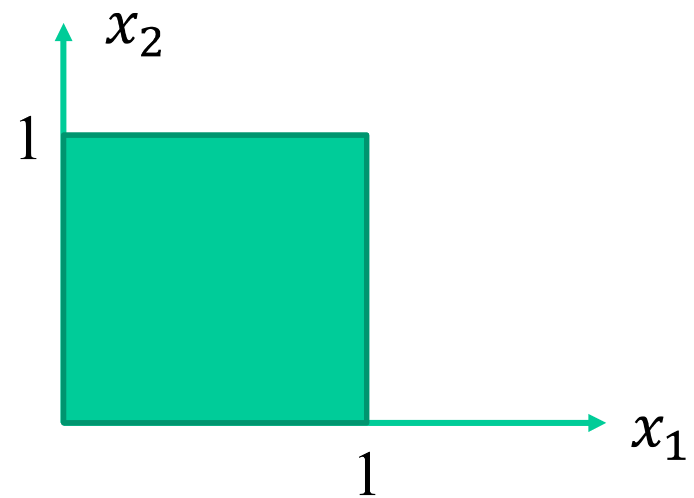


Example 3

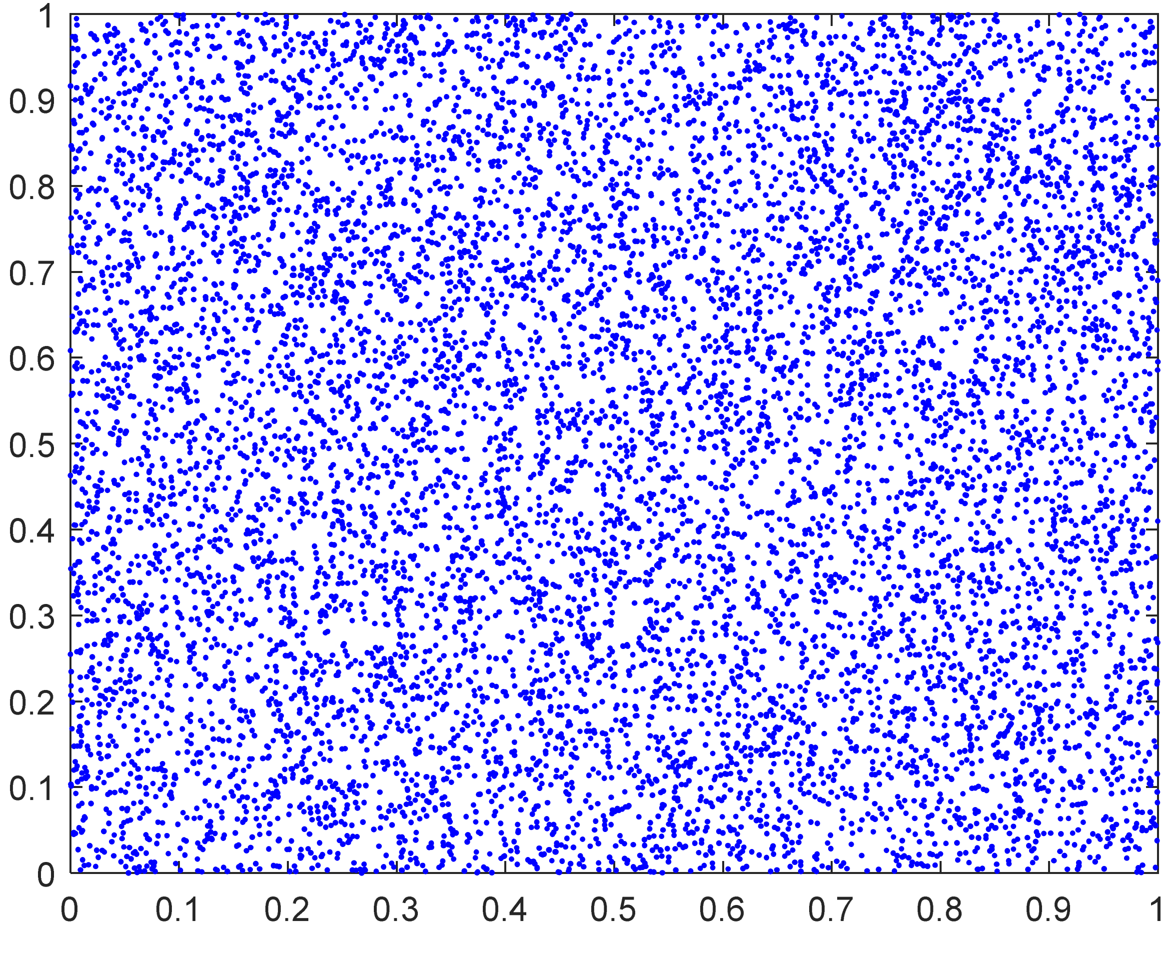
Consider a SOM neural network with 100 neurons arranged in a 2D lattice with 10 rows and 10 columns. The network is trained with a two-dimensional input vector **x**, whose elements x_1 and x_2 are uniformly distributed in a region of square:

$$0 < x_1 < 1$$

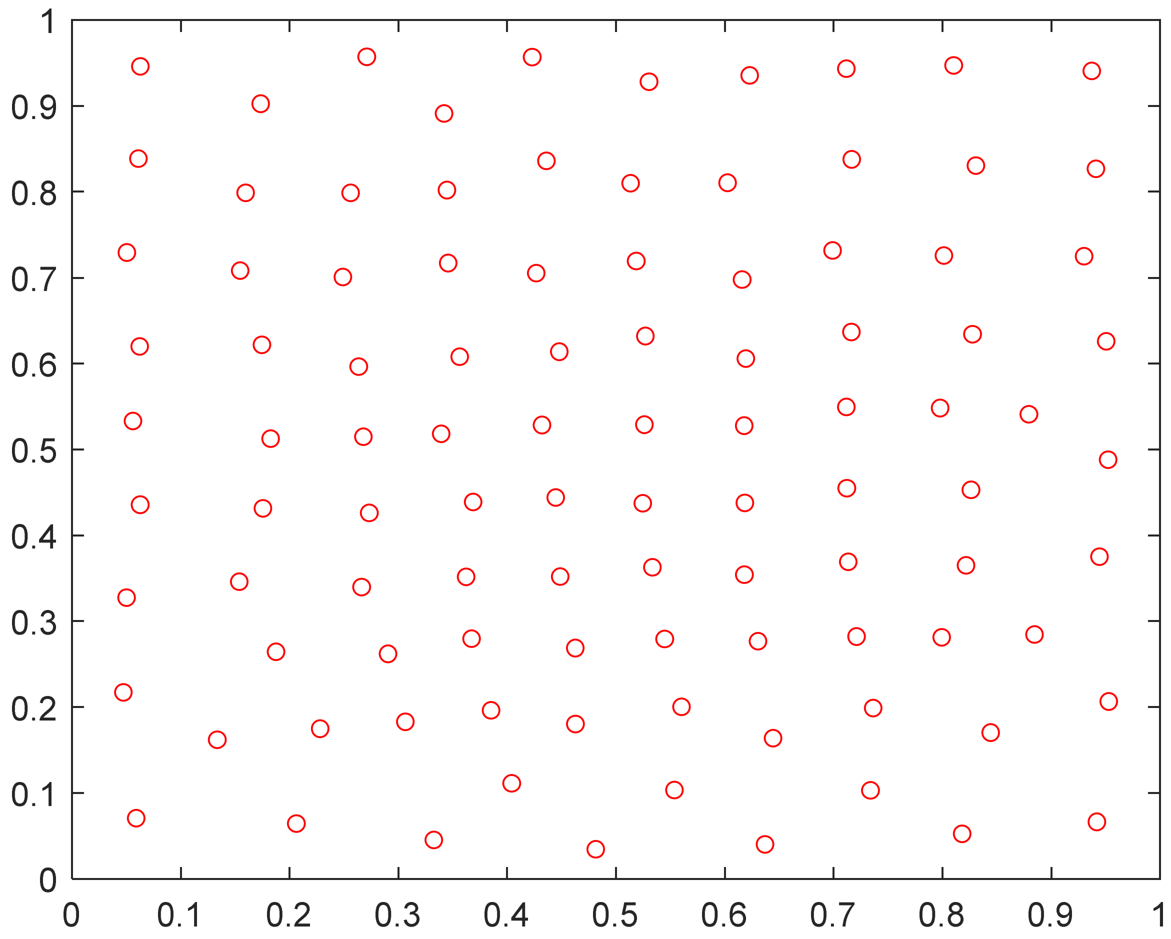
$$0 < x_2 < 1$$



10000 input vectors (data points) uniformly distributed in the region



After training, the 100 weight vectors divide the region uniformly

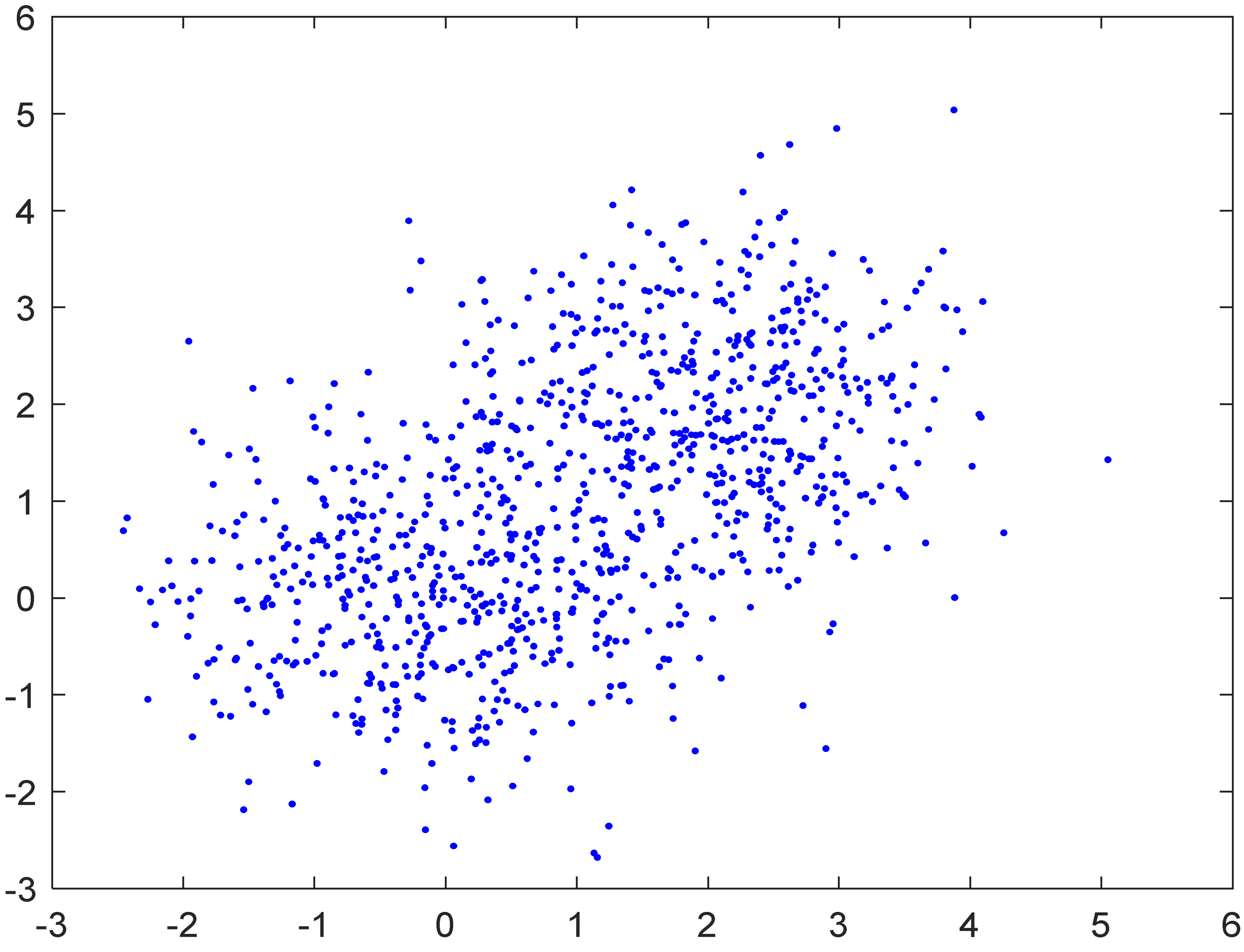


- **Properties of the Feature Map**

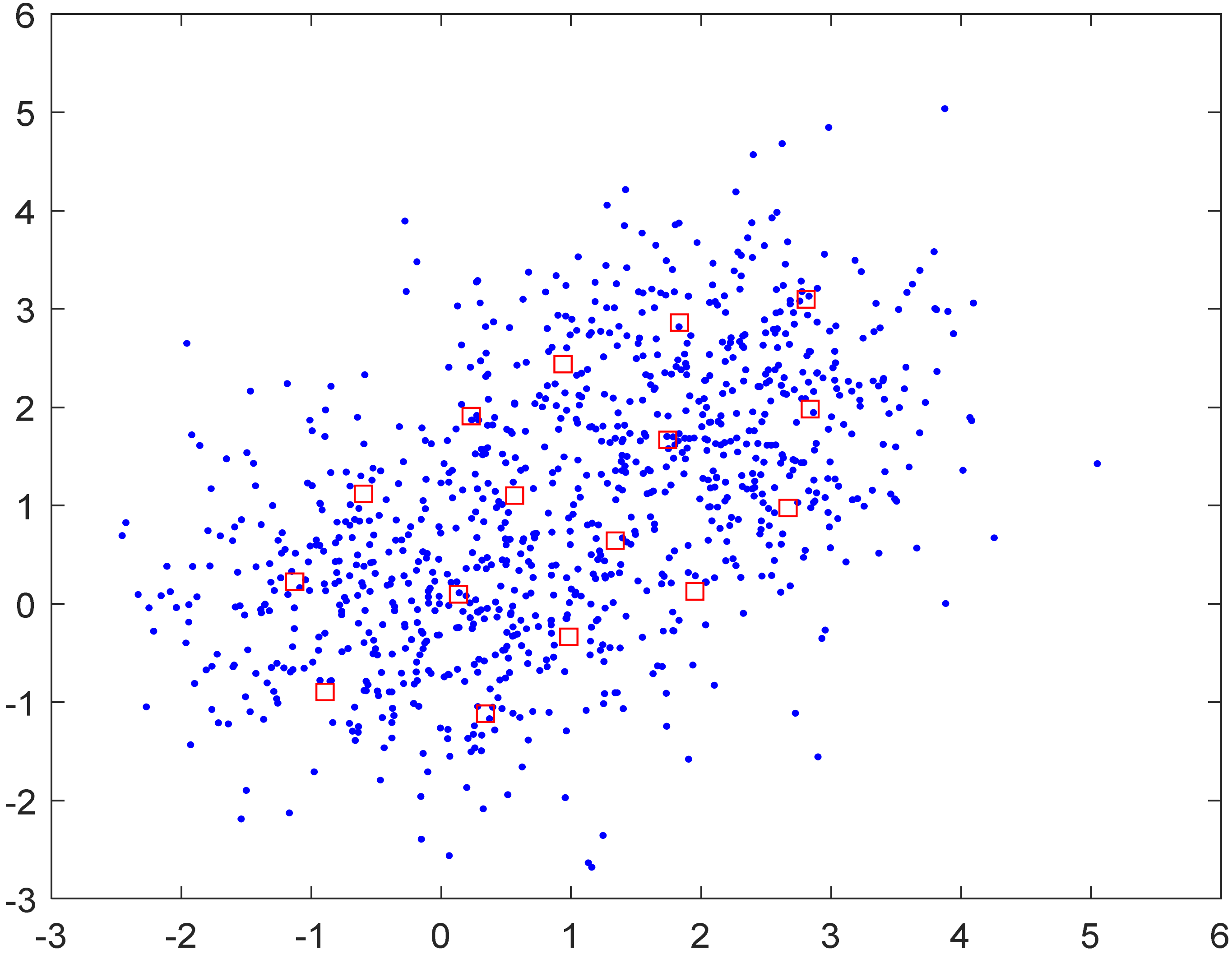
(1) **Approximation of the input space**. The feature map represented by a set of weight vectors in the output space provide a good approximation to the input space. Thus, a large set of input vectors can be represented by a smaller set of prototypes.

In the following example, we have 1000 data points. If we want to use a small set of prototypes, say 16, 25, or 100, to represent the distribution of the 1000 data points, we may use the SOM algorithm to find the prototypes as shown in the following figures.

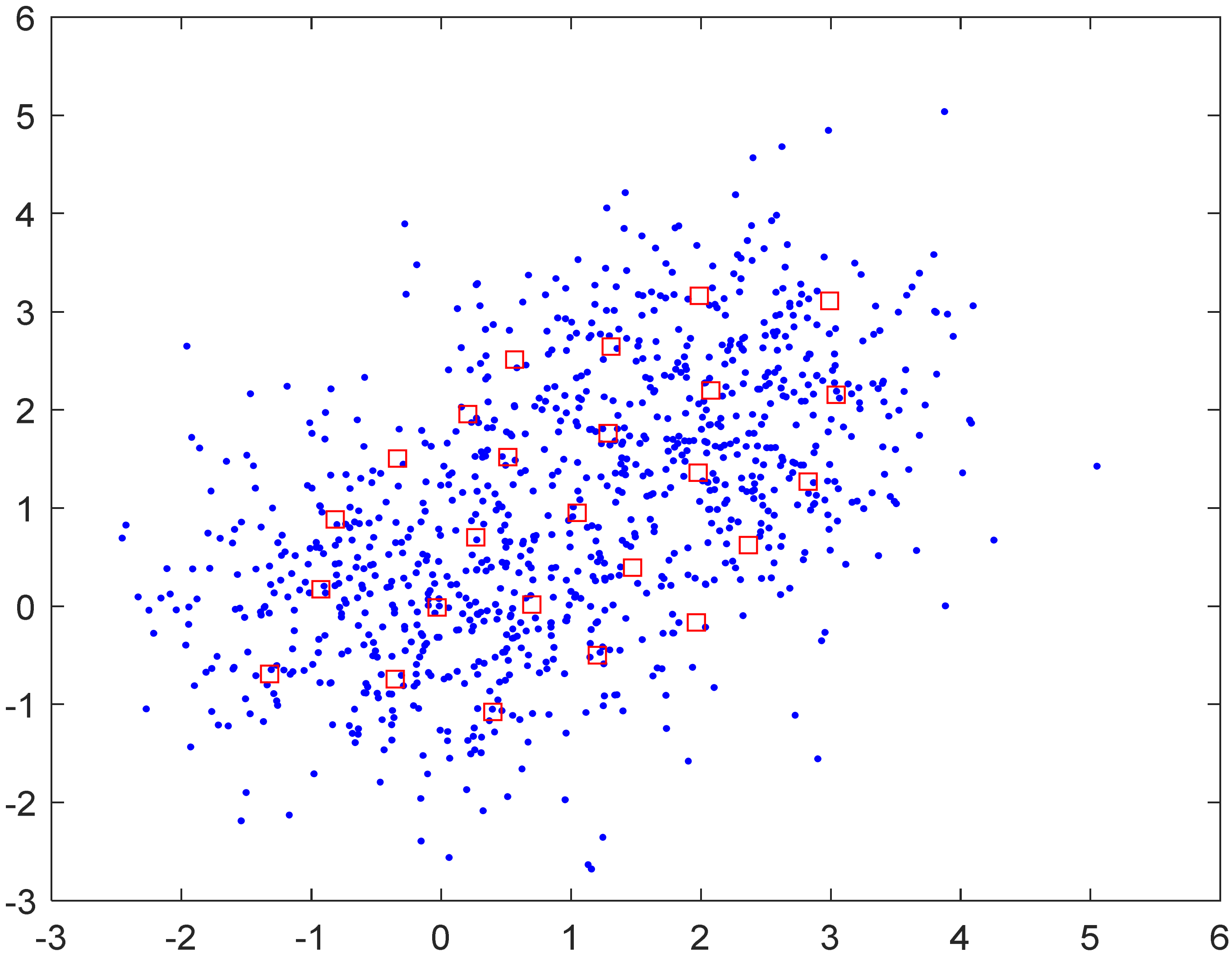
Example 4



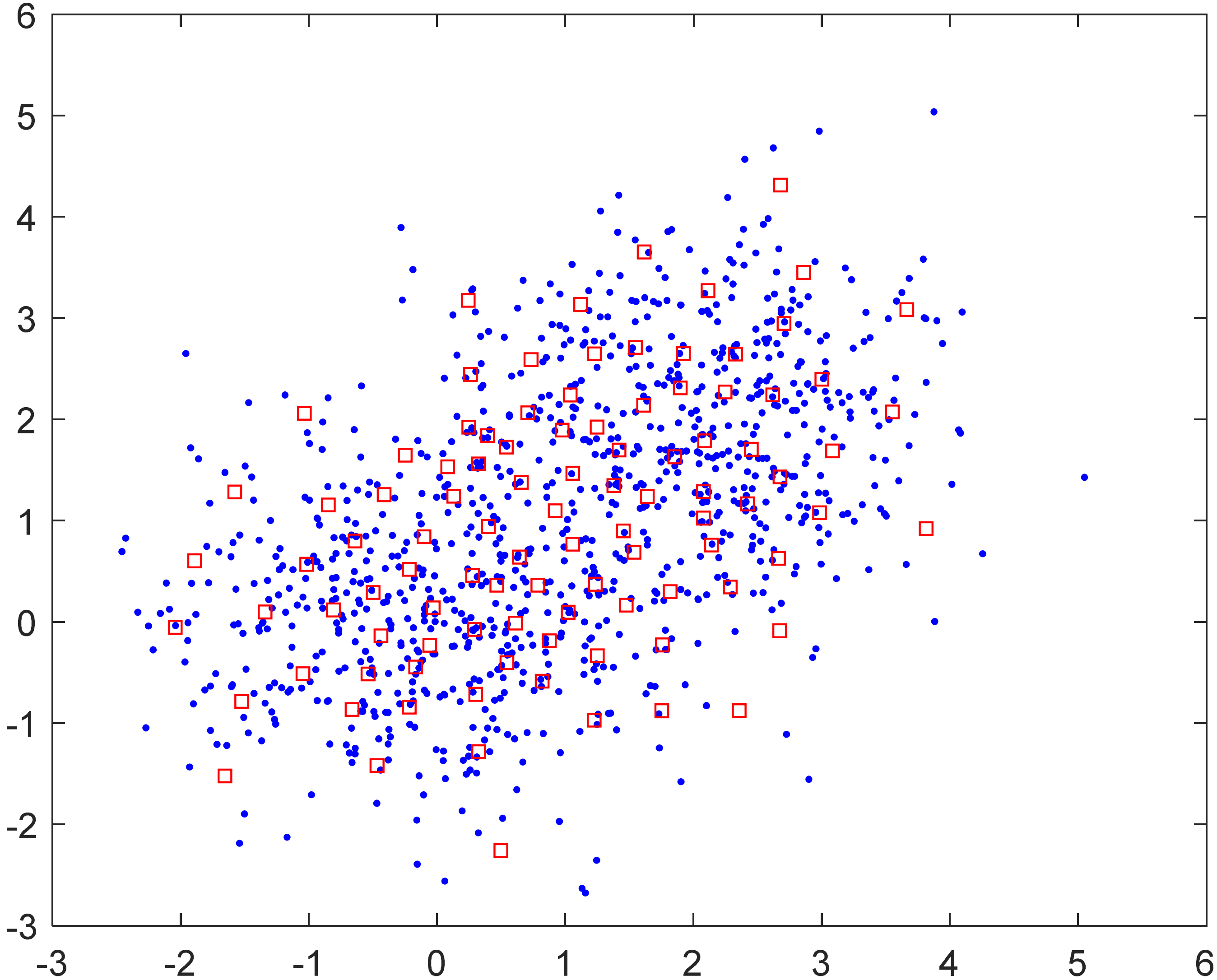
1000 data points



16 prototypes

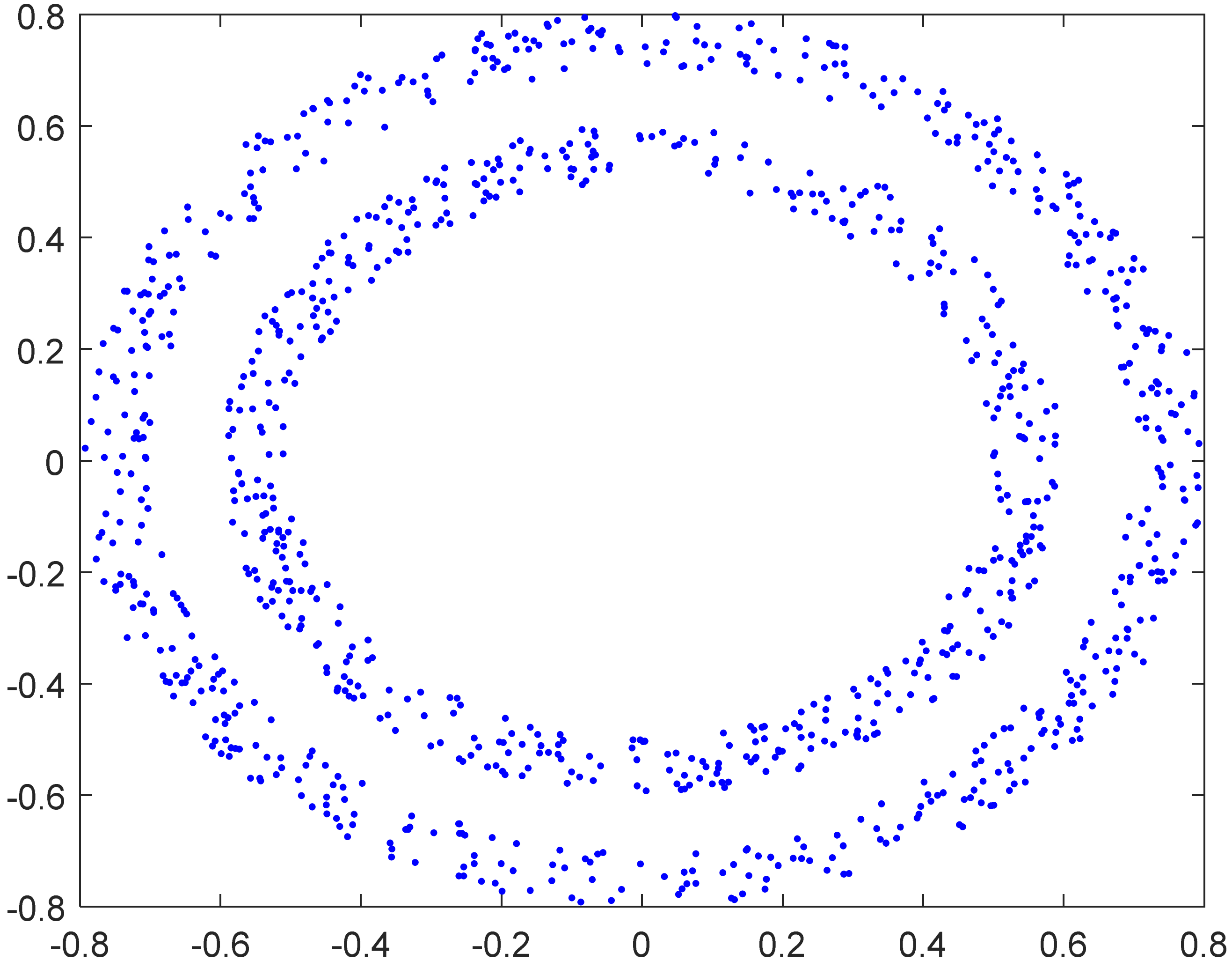


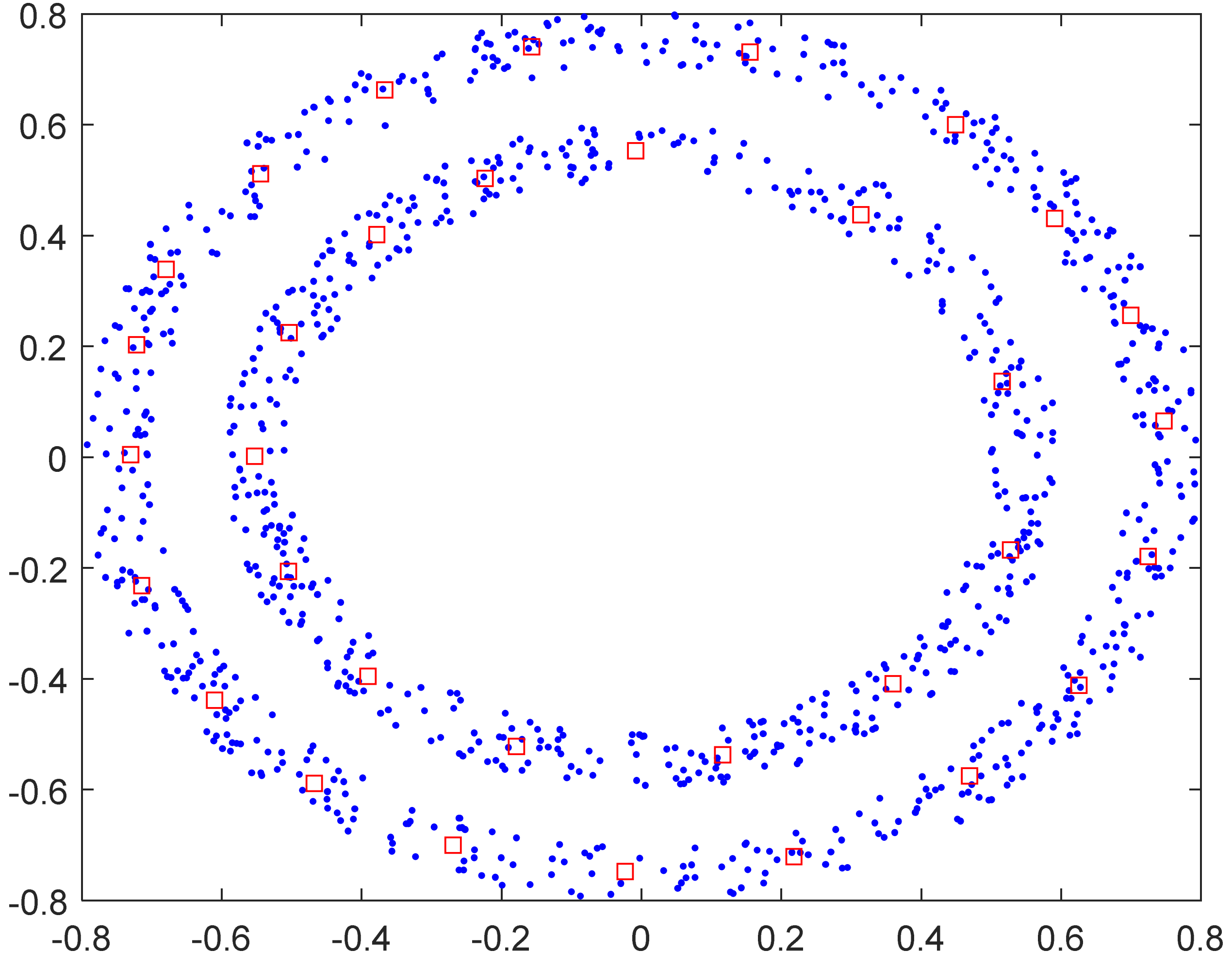
25 prototypes

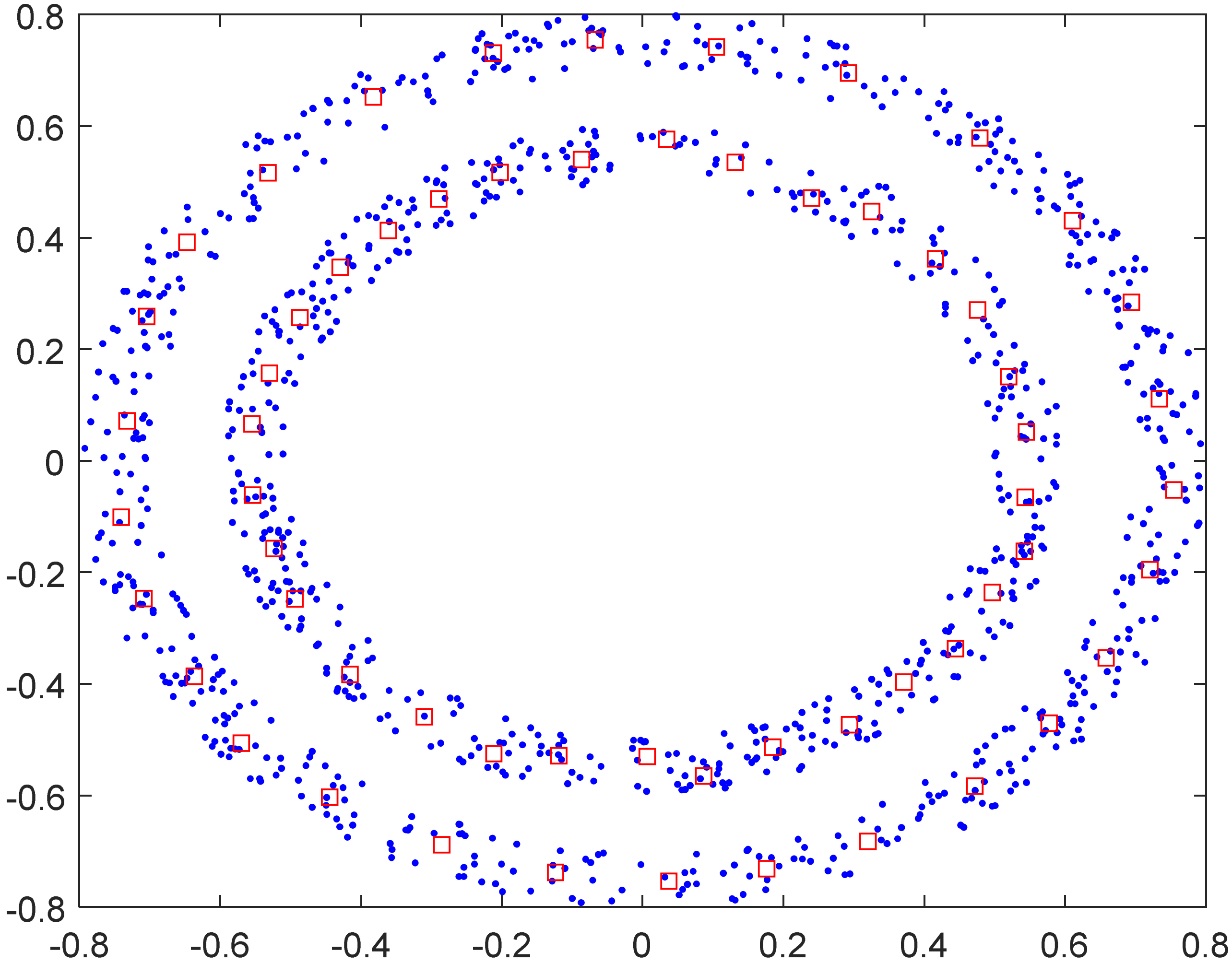


100 prototypes

Example 5

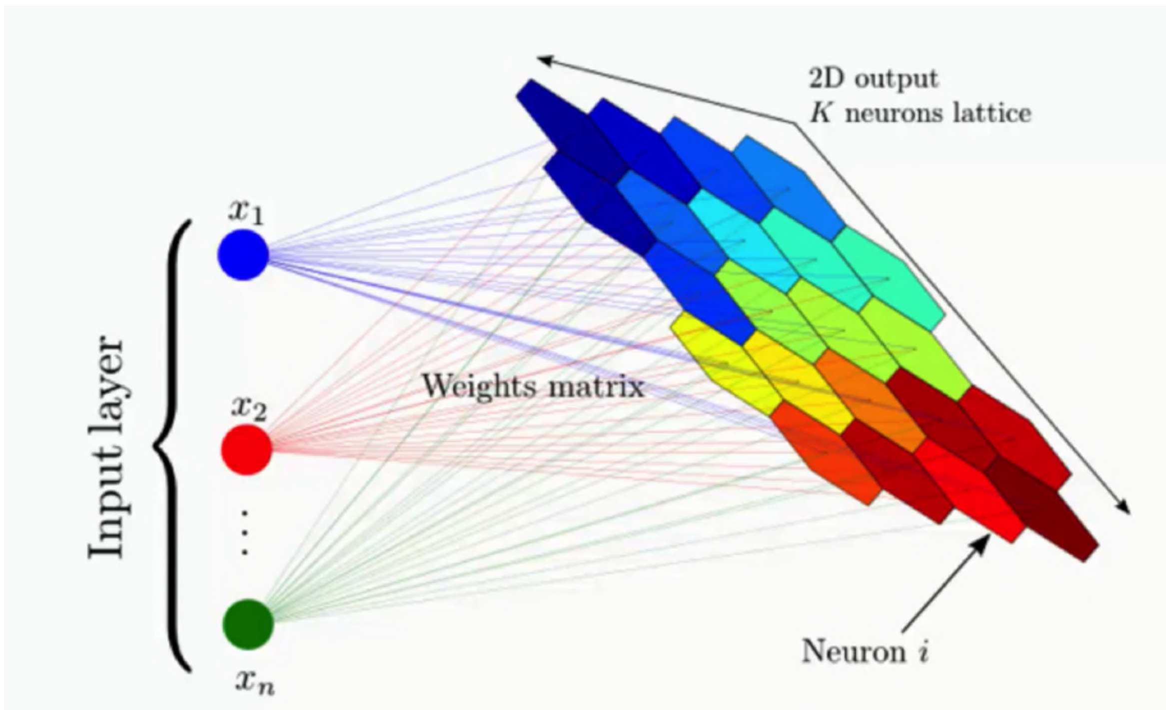



0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8



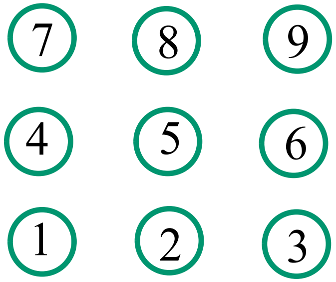
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8

(2) **Topological ordering.** The feature map computed by the SOM algorithm is topologically ordered in the sense that the spatial location of a neuron in the lattice corresponds to a particular domain of the input patterns.

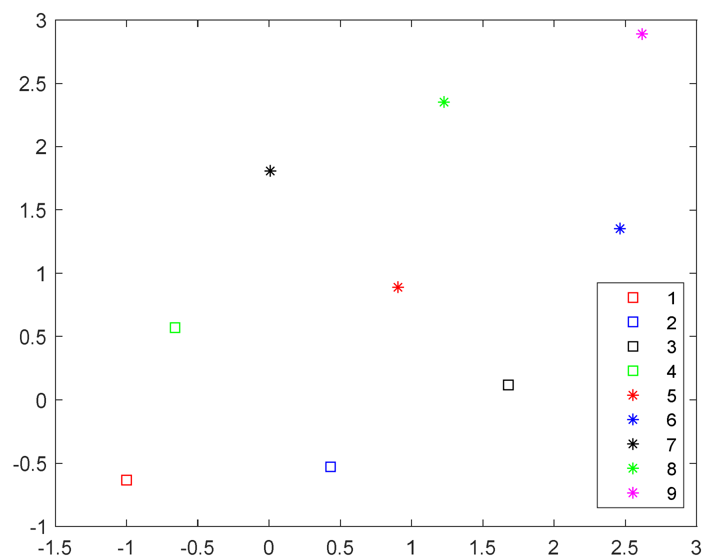


Example 6

The location of 9 neurons in 2D lattice:

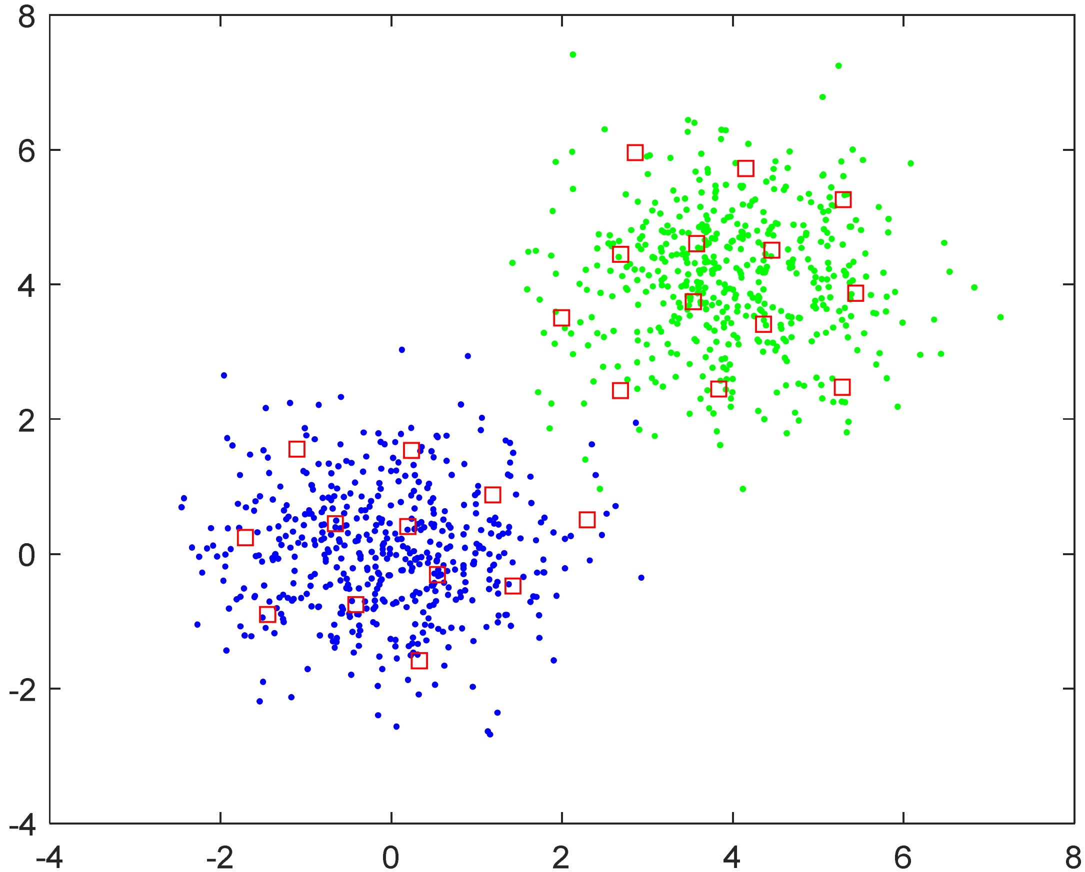


The location of the weight vectors of the 9 neurons in input space (after training with the 1000 data points)



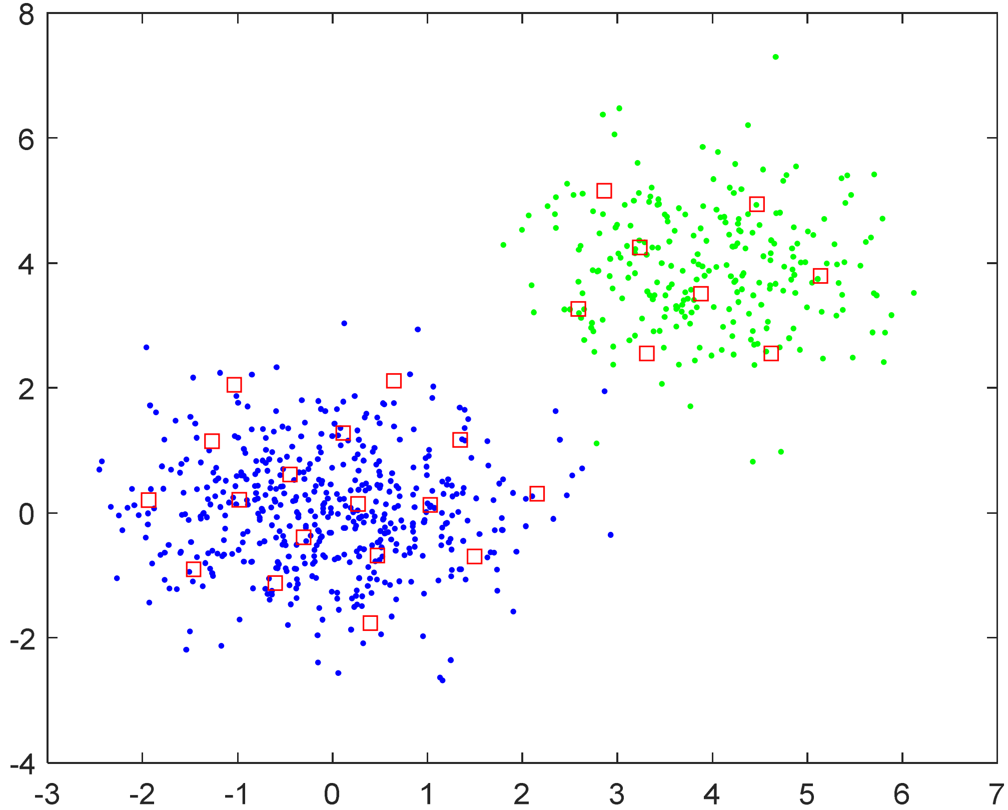
(3) **Density Matching**. The map reflects variations in the statistics of the input distribution: regions in the input space from which sample are drown with a high probability of occurrence are mapped onto larger domains of the output space, and therefore with better resolution than regions in the input space from which samples are drawn with a low probability of occurrence.

Case 1: 500 samples from each of the two clusters (cluster 1 to cluster 2 sample ratio is 1:1)



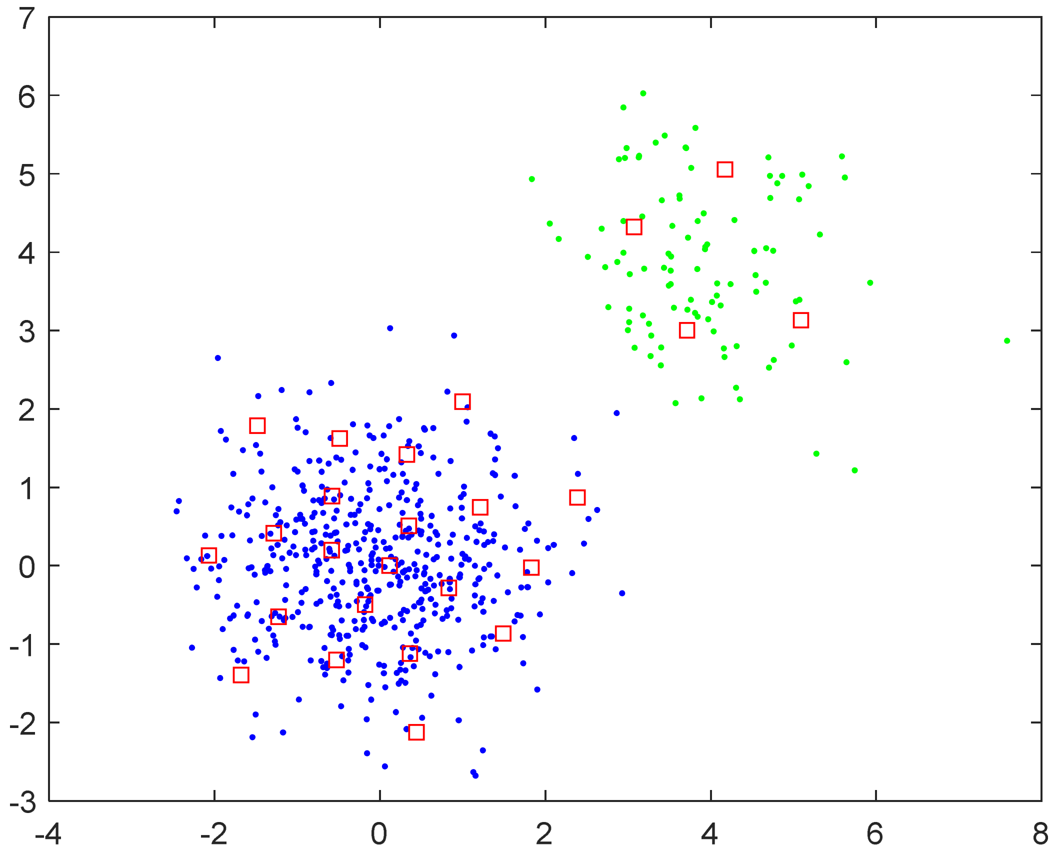
Sample ratio is 1:1, prototype ratio is 11:13

Case 2: 500 samples from cluster 1, 250 samples from cluster 2 (cluster 1 to cluster 2 sample ratio is 2:1)



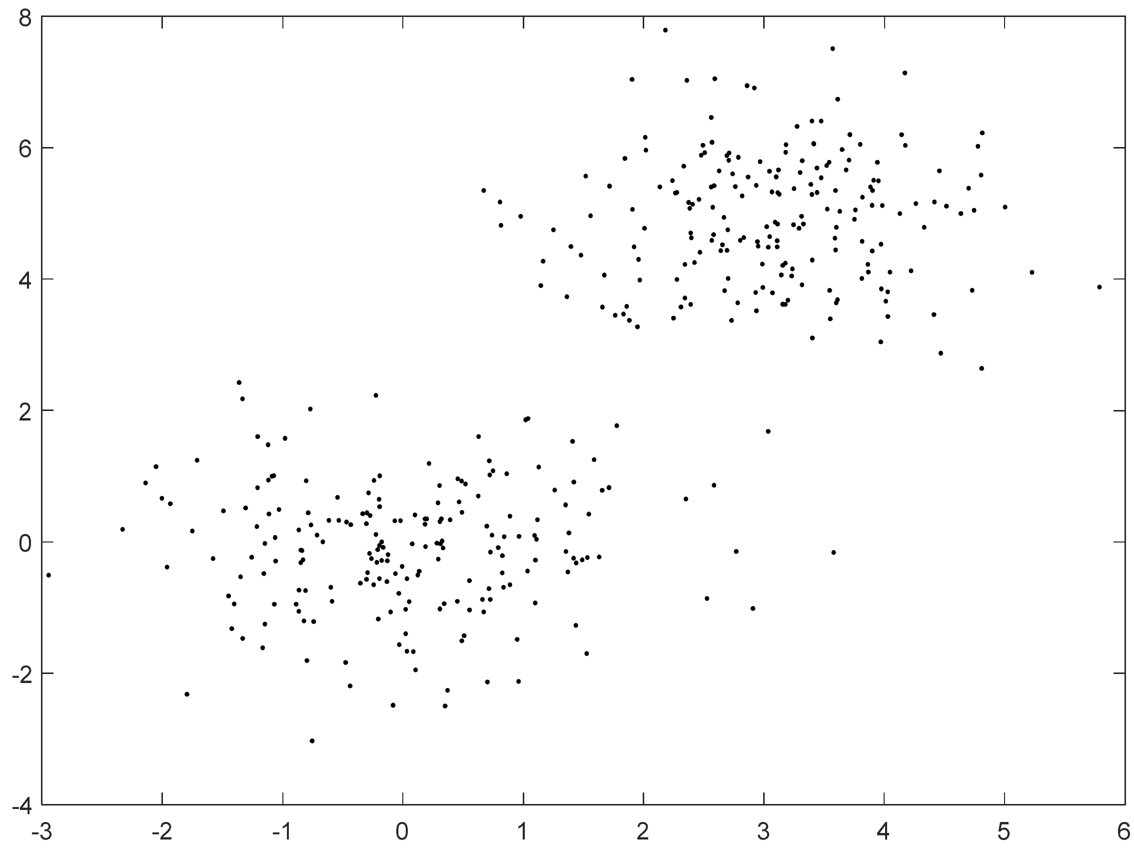
Sample ratio is 2:1, prototype ratio is 17:8

Case 3: 500 samples from cluster 1, 100 samples from cluster 2 (cluster 1 to cluster 2 sample ratio is 5:1)

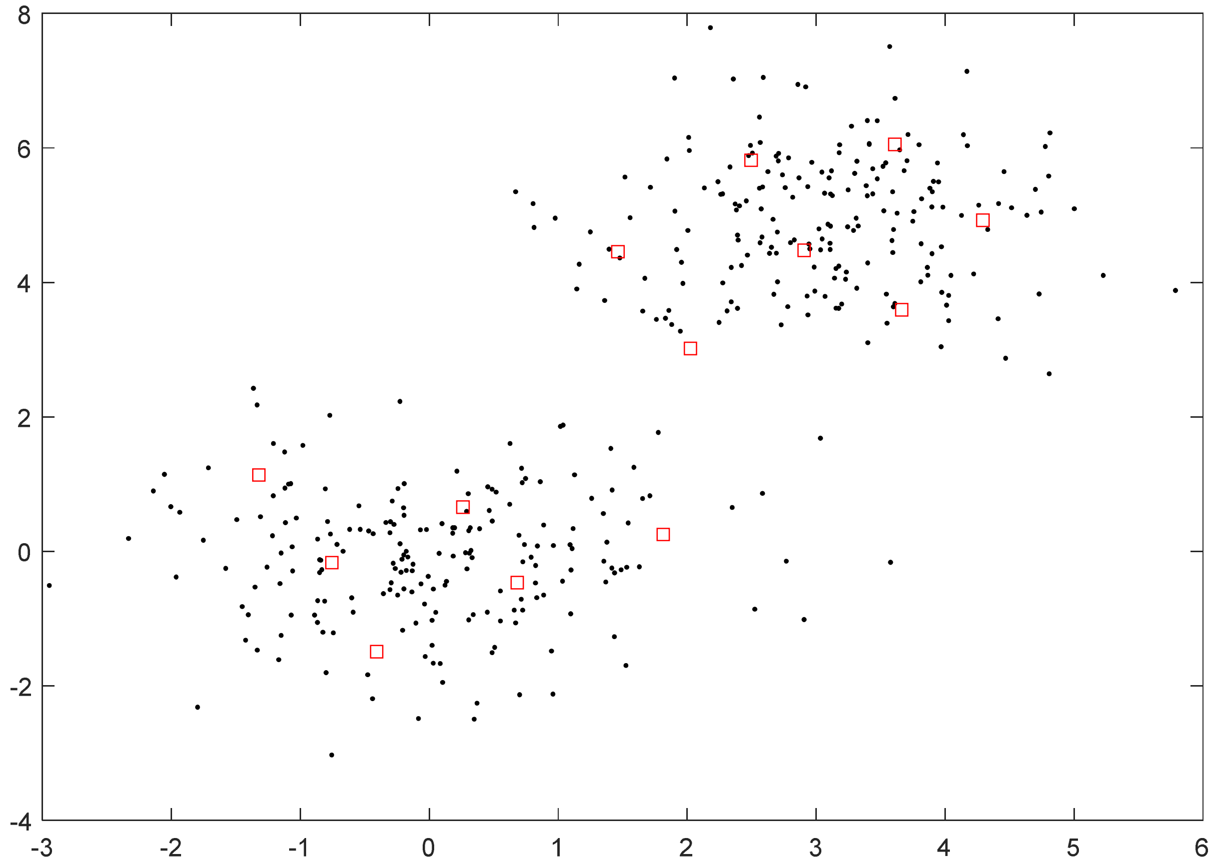


Sample ratio is 5:1, prototype ratio is 21:4

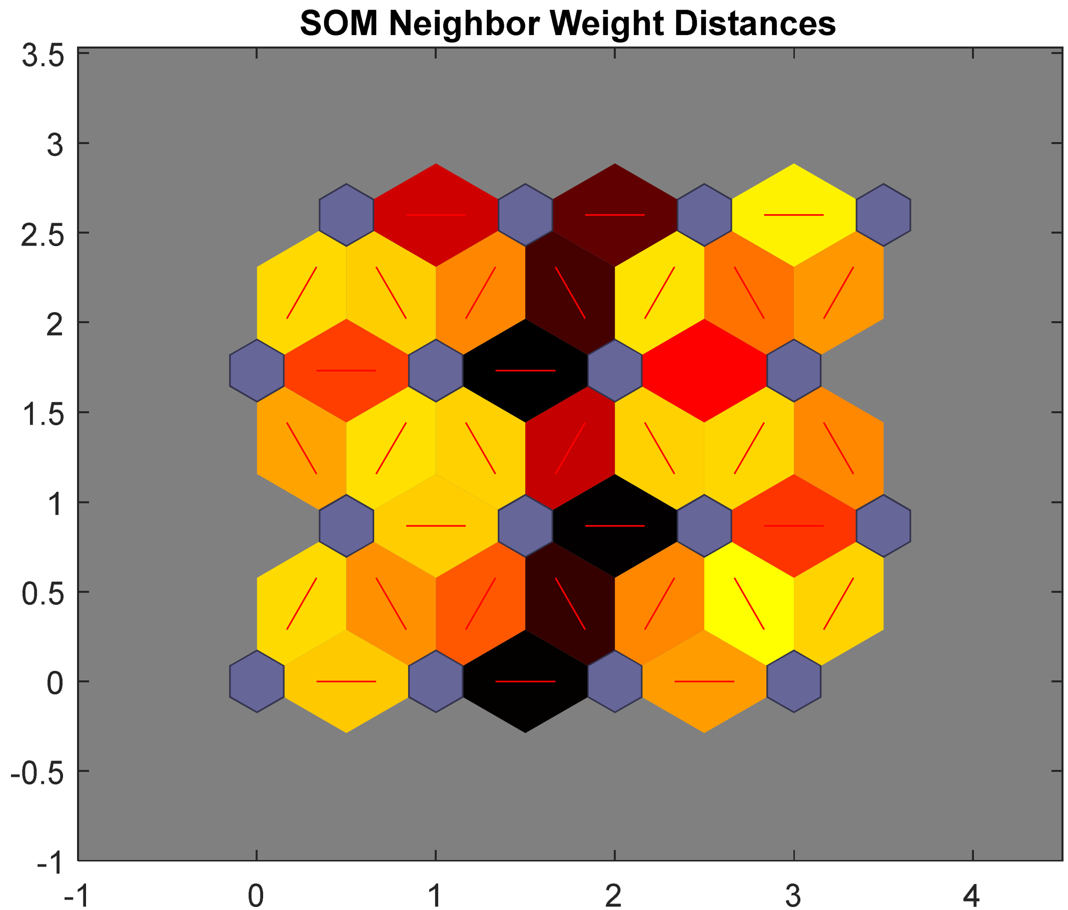
(4) **Visualization of distance between weight vectors of neighboring neurons** can help identify the suitable number of clusters underlying the data.

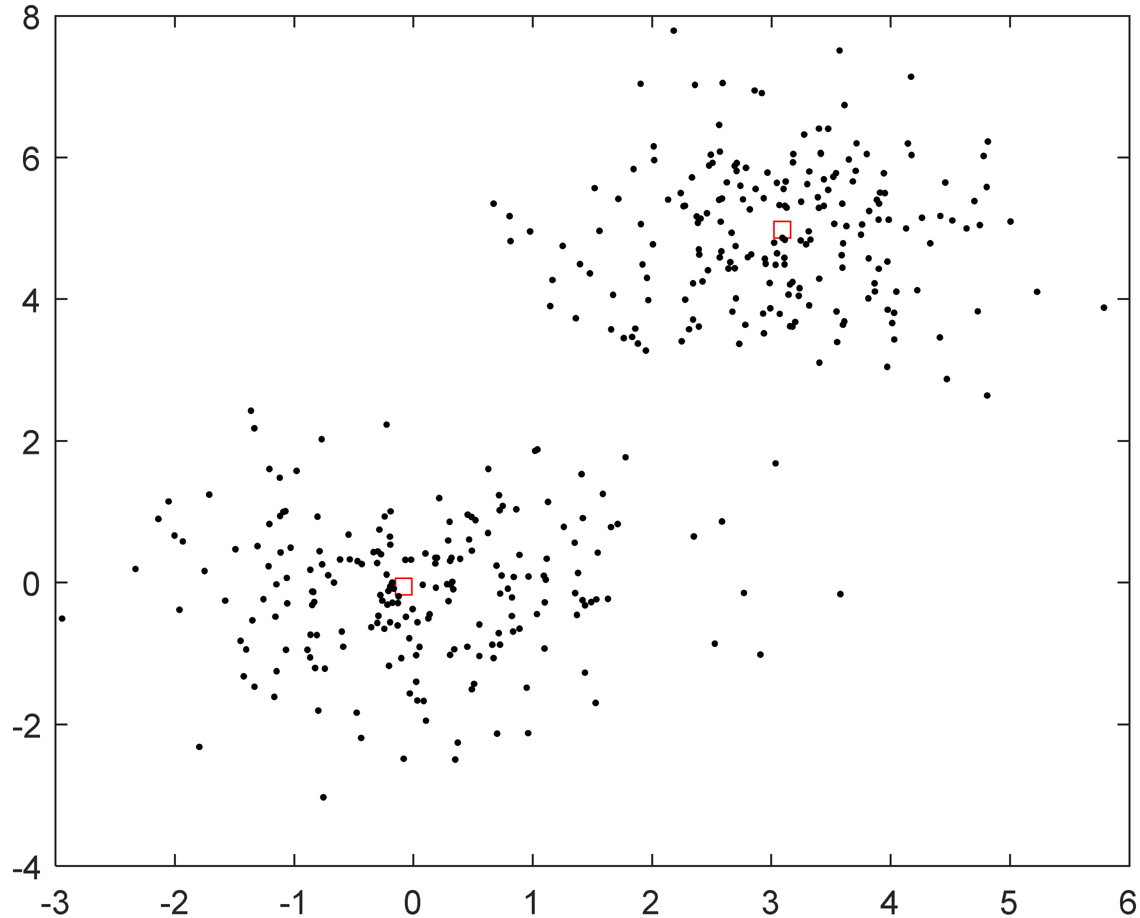


Assume we do not know the number of clusters underlying the data, and therefore use 16 neurons in the SOM neural network. After training, the obtained clusters are as follows



Visualization of distance between neighboring neurons help us identify two clusters underlying the data (darker colors means larger distance.)





Clustering results if 2 neurons are used.

(5) Dimensionality reduction. SOM neural network maps data from the original high-dimensional input space to output space, i.e. 2D lattice. The projection of input $\mathbf{x}$ to each neuron in the SOM network constitute a new representation of the input $\mathbf{x}$.

The projection of $\mathbf{x}$ onto neuron i can be represented as:

$$p_i = \varphi(\mathbf{x}, \mathbf{w}_i) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{w}_i\|^2}{2\sigma^2}\right)$$

The new representation of $\mathbf{x}$ is as:

$$\mathbf{p} = [p_1 \quad p_2 \quad \cdots \quad p_m]$$

Where m is the number of neurons of the SOM neural network.

Discussion:

How to determine the number of neurons in SOM neural networks?

Scenario 1:

SOM neural network is used for clustering (grouping of data), and the number of clusters is known

- Set the number of neurons to the number of clusters

Scenario 2:

SOM neural network is used for clustering (grouping of data), and the number of clusters is unknown

- Set the number of neurons to multiple numbers
- Train multiple SOM neural networks
- Evaluate and compare the clustering performance
- Set the neuron number to the one with the best performance

Scenario 3:

SOM neural network is used for other applications such as representative sample selection, dimensionality reduction etc

- Set to the number as you need