

## 10 Self-Organizing Feature Maps

Self-Organizing Feature Maps (SOFM) also known as Kohonen maps or topographic maps were first introduced by von der Malsburg (1973) and in its present form by Kohonen (1982).

According to Kohonen the idea of feature map formation can be 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.

More specifically:

Self-Organizing Feature maps are competitive neural networks in which neurons are organized in an  $l$ -dimensional lattice (grid) representing the **feature space**

The output lattice characterizes a relative position of neurons with regards to its neighbours, that is their topological properties rather than exact geometric locations.

In practice, dimensionality of the feature space is often restricted by its visualisation aspect and typically is  $l = 1, 2$  or  $3$ .

Consider an example of a self-organizing feature map in which the **input space** is 3-dimensional ( $p = 3$ ) and **feature space** is 2-dimensional ( $l = 2$ ). The structure of such SOFM is illustrated in Figure 10–1.

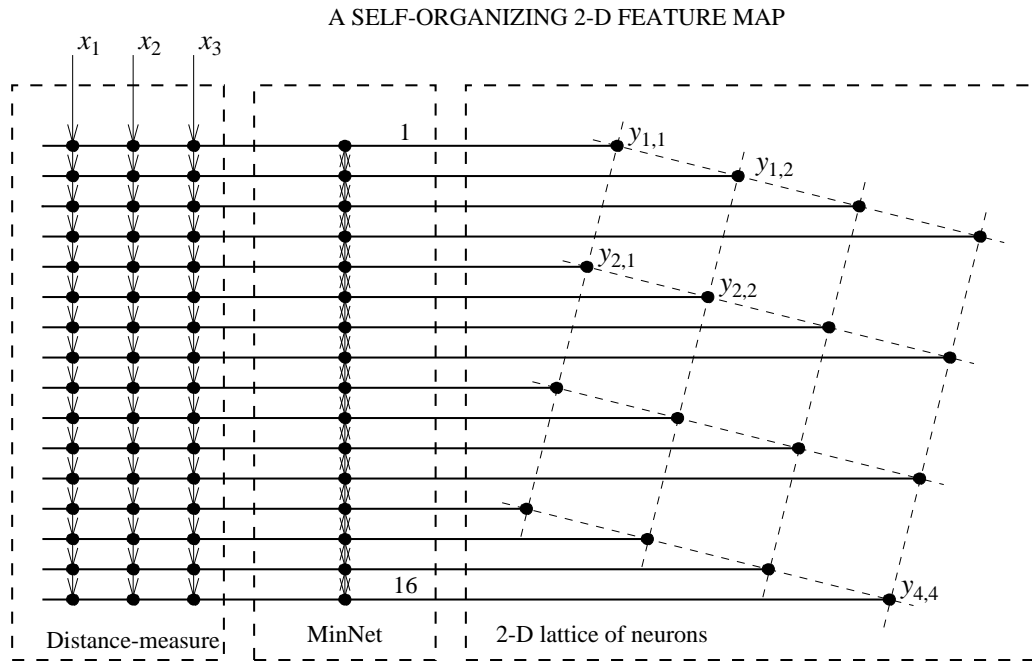


Figure 10–1: A 2-D SOFM with  $p = 3$ ;  $m = [4 \ 4]$ ;  $l = 2$ .

Each neuron,  $y_v$  in the above SOFM is characterized by its position in the lattice specified by a 2-D vector  $\mathbf{v} = [v_1 \ v_2]$ , and by a 3-D weight vector  $\mathbf{w}_v = [w_{1v} \ w_{2v} \ w_{3v}]$ .

SOFM, as a competitive neural network, consists of a distance-measure layer and a competitive layer which implements the MinNet algorithm through the lateral inhibitive and local self-excitatory connections.

During the competition phase (the MinNet), the winner is selected from all neurons in the lattice.

A general structure of a Self-Organizing Feature Map is presented in Figure 10–2

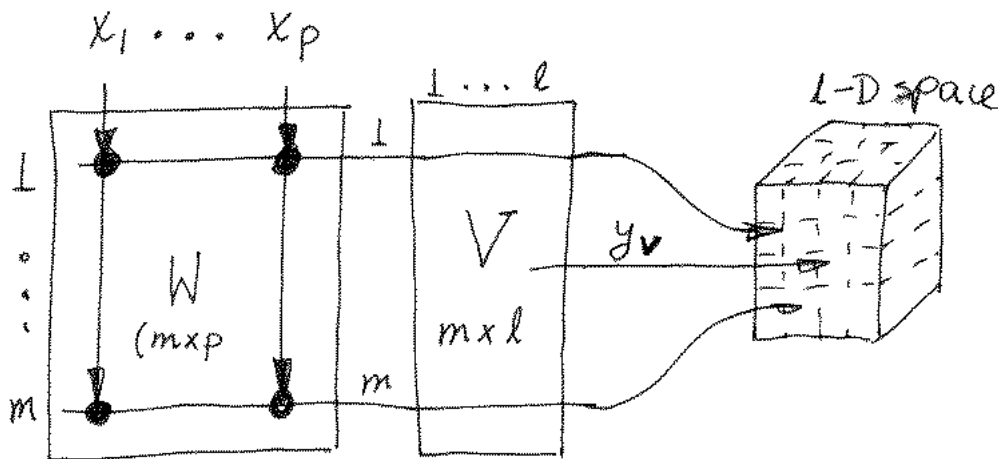


Figure 10–2: A general structure of a SOFM

and can be characterized by the following parameters:

- $p$  — dimensionality of the **input space**
- $l$  — dimensionality of the **feature space**
- $m$  — the total number of neurons
- $W$  —  $m \times p$  matrix of synaptic weights
- $V$  —  $m \times l$  matrix of topological positions of neurons

In subsequent considerations neurons will be identified either by their index  $k = 1, \dots, m$ , or by their position vector  $V(k, :)$  in the feature space.

## 10.1 Feature Maps

A **Feature Map** is a plot of synaptic **weights** in the **input space** in which weights of the neighbouring neurons are joined by lines or plane segments (patches).

### Example: 2-D input space, 1-D feature space

Consider a SOFM neural network with two inputs ( $p = 2$ ) and  $m$  outputs organized in a 1-D feature space:

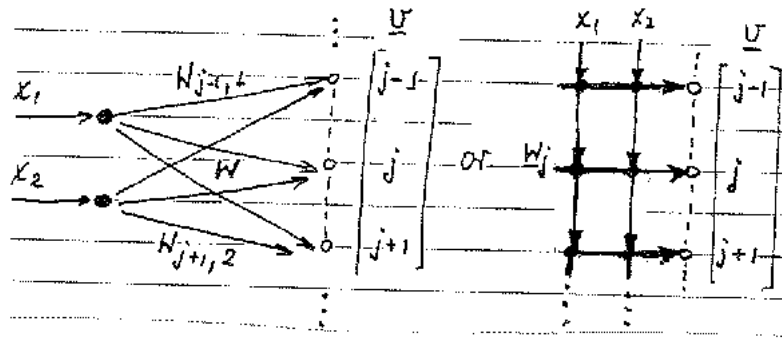


Figure 10-3: A general structure of a (2-D,1-D) SOFM

The feature map has the following structure:

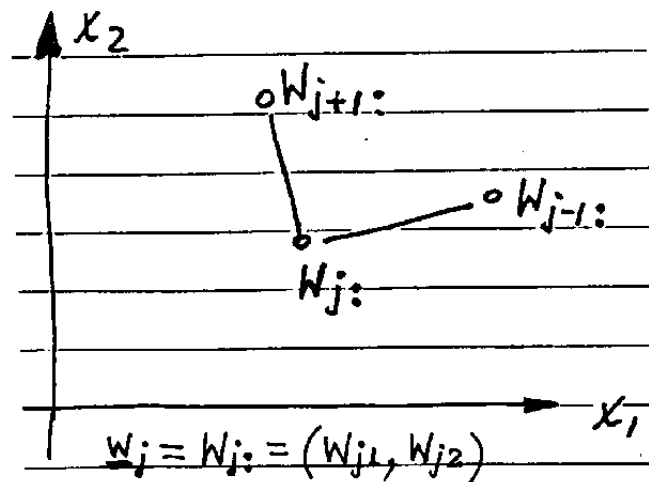


Figure 10-4: A feature map for a (2-D,1-D) SOFM

Note that in the feature map the point representing the weight vector,  $w_j$ , is joined by line segments with points representing weights  $w_{j-1}$  and  $w_{j+1}$  because neurons  $j - 1$ ,  $j$ , and  $j + 1$  are located in the adjacent positions of the 1-D lattice.

### Example: 2-D input space, 2-D feature space

Let us consider a SOFM with two inputs ( $p = 2$ ) and  $m$  neurons arranged in a 2-D lattice as in Figure 10–5.

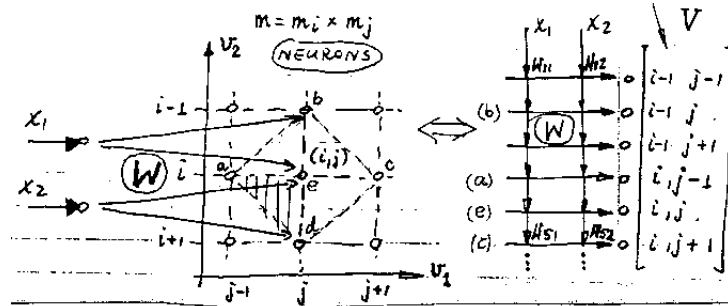


Figure 10–5: A general structure of a (2-D,2-D) SOFM

The feature map which maps a 2-D input space into a 2-D feature space has the following structure:

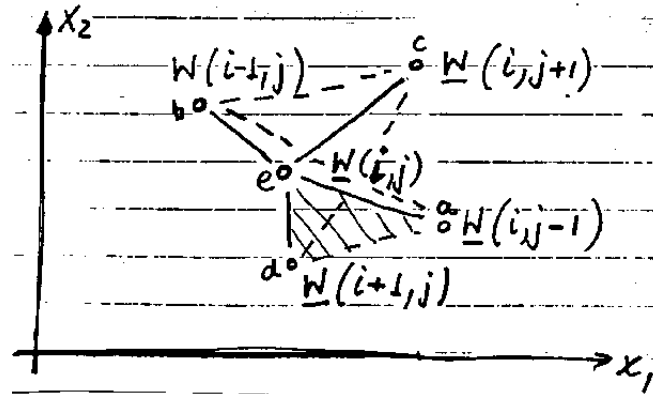


Figure 10–6: A feature map for a (2-D,2-D) SOFM

Consider a neuron located at the vertex  $e = (i, j)$  of the output lattice and its two neighbours,  $a = (i, j - 1)$  and  $d = (i + 1, j)$ . The related three weights vectors,  $\mathbf{w}_e = \mathbf{w}_{i,j}$ ,  $\mathbf{w}_a = \mathbf{w}_{i,j-1}$ , and  $\mathbf{w}_d = \mathbf{w}_{i+1,j}$  are represented by the respective points in the feature map in Figure 10–6. The points can be joined by three line segments, and form a triangular patch.

### Plotting (2-D,2-D) feature maps: FM22.m script

Consider a SOFM with  $p = 2$  inputs and  $m = 12$  organized on a  $3 \times 4$  lattice

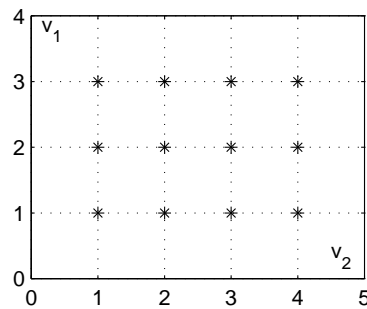


Figure 10–7: The output lattice of a (2-D,2-D) SOFM

The following MATLAB code can be used to generate an example of the weight and position matrices:

```
m = [4 3]; mm = prod(m) ; % p = 2 ;
[V1, V2] = meshgrid(1:m(1),1:m(2)); V = [V2(:), V1(:)];
W = V-1.4*rand(mm, 2) ;
```

The resulting  $W$  and  $V$  matrices can be as follows:

k	$W$		$V$	
1	0.83	0.91	1	1
2	0.72	2.01	2	1
3	0.18	2.39	3	1
4	2.37	0.06	1	2
5	1.38	2.18	2	2
6	1.41	2.82	3	2
7	2.38	1.27	1	3
8	2.06	1.77	2	3
9	2.51	2.61	3	3
10	3.36	0.85	1	4
11	3.92	2.05	2	4
12	3.16	2.90	3	4

Two versions of the feature map, which illustrates the weight vectors in the input space together with the neighbourhood information, can be plotted with the following MATLAB code

### A colour patches version

```
FM1 = full(sparse(V(:,1), V(:,2), W(:,1))) ;
FM2 = full(sparse(V(:,1), V(:,2), W(:,2))) ;
pcolor(FM1, FM2, 32*(FM1+FM2)) ;
```

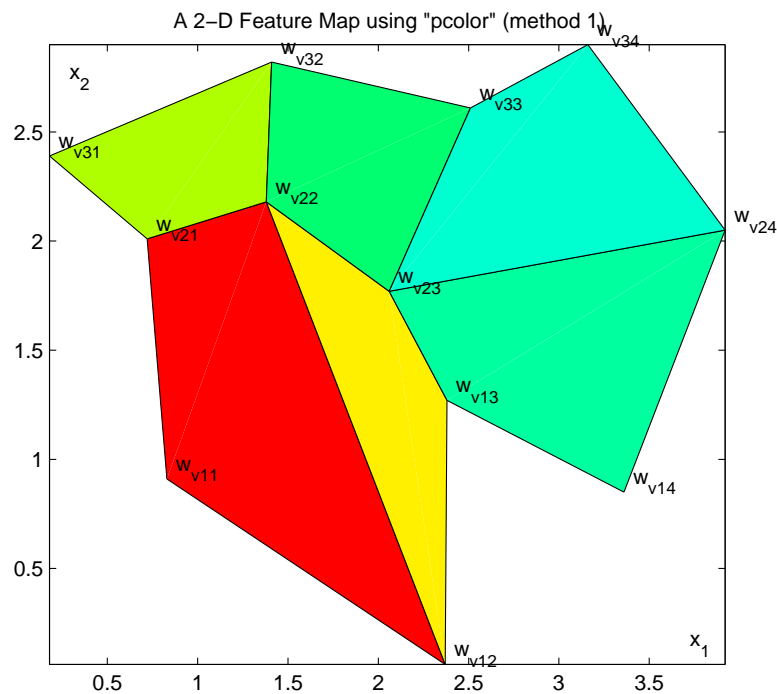


Figure 10–8: A method-1 plot of a feature map for a (2-D,2-D) SOFM

**A grid lines version**

```

FM = FM1+j*FM2;
plot(FM), hold on, plot(FM.', plot(FM, 'o')), hold off
title('A 2-D Feature Map using "grid lines" (method 2)')

```

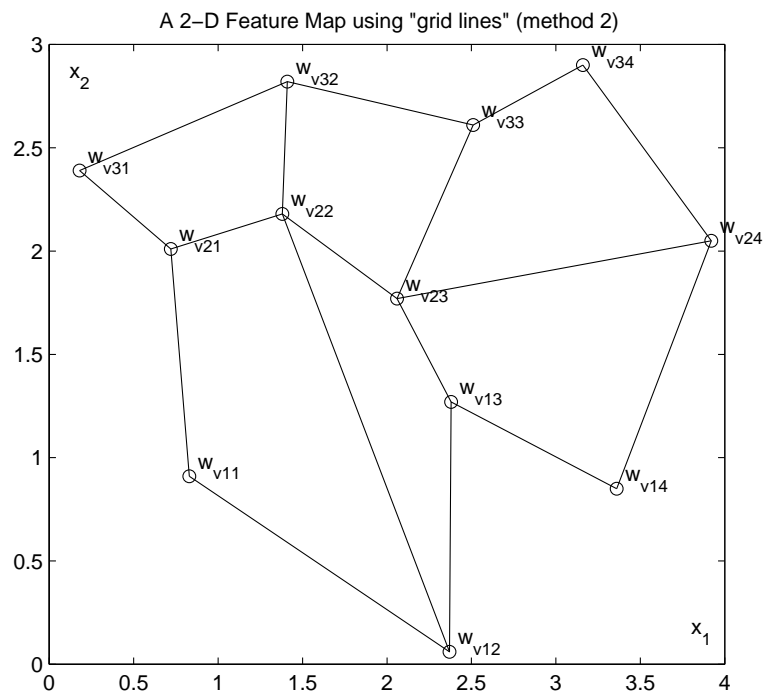


Figure 10–9: A method-2 plot of a feature map for a (2-D,2-D) SOFM



## 10.2 Learning Algorithm for Self-Organizing Feature Maps

The objective of the learning algorithm for the SOFM neural networks is formation of the feature map which captures of the essential characteristics of the  $p$ -dimensional input data and maps them on the typically 1-D or 2-D feature space.

The learning algorithm captures two essential aspects of the map formation, namely, **competition** and **cooperation** between neurons of the output lattice.

**Competition** is implemented as in competitive learning: each input vector  $\mathbf{x}(n)$  is compared with each weight vector from the weight matrix  $W$  and the position  $V(k(n), :)$  of the winning neuron  $k(n)$  is established. For the winning neuron the distance

$$|\mathbf{x}^T(n) - W(k(n), :)|$$

attains minimum.

**Cooperation** All neurons located in a topological neighbourhood of the winning neurons  $k(n)$  will have their weights updated usually with a strength  $\Lambda(j)$  related to their distance  $\rho(j)$  from the winning neuron,

$$\rho(j) = |V(j, :) - V(k(n), :)| \quad \text{for } j = 1, \dots, m.$$

The **neighbourhood function**,  $\Lambda(j)$ , is usually an  $l$ -dimensional Gausssian function:

$$\Lambda(j) = \exp\left(-\frac{\rho^2(j)}{2\sigma^2}\right)$$

where  $\sigma^2$  is the variance parameter specifying the spread of the Gaussian function.

Example of a 2-D Gaussian neighbourhood function for a  $40 \times 30$  neuronal lattice is given in Figure 10–10.

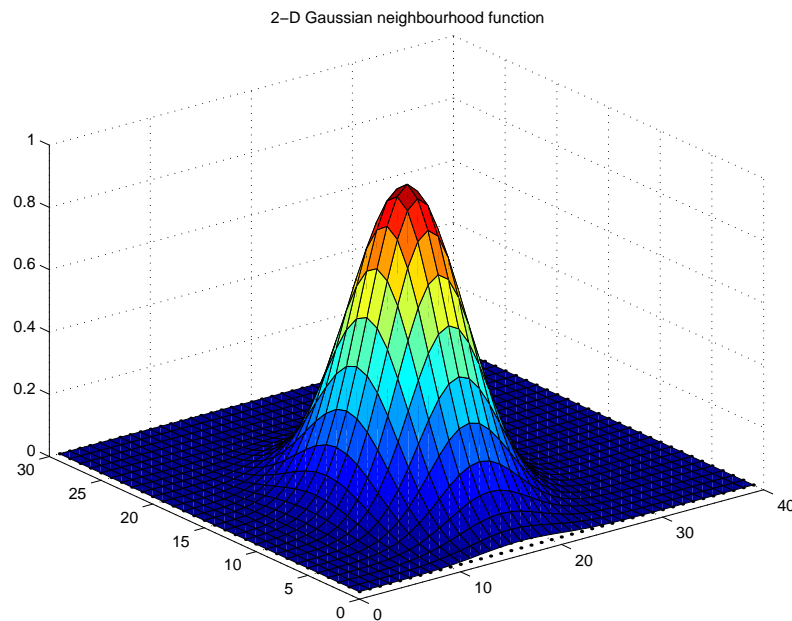


Figure 10–10: 2-D Gaussian neighbourhood function

Feature map formation is critically dependent on the learning parameters, namely, the learning gain,  $\eta$ , and the spread of the neighbourhood function specified for the Gaussian case by the variance,  $\sigma^2$ . In general, both parameters should be time-varying, but their values are selected experimentally.

Usually, the **learning gain** should stay close to unity during the **ordering phase** of the algorithm which can last for, say, 1000 iteration. After that, during the **convergence phase**, should be reduced to reach the value of, say, 0.1.

The **spread** of the neighbourhood function should initially include all neurons for any winning neuron and during the ordering phase should be slowly reduced to eventually include only a few neurons in the winner's neighbourhood. During the convergence phase, the neighbourhood function should include only the winning neuron.

## The complete SOFM learning algorithm

The complete algorithm can be described as consisting of the following steps

1. Initialise:

- (a) the weight matrix  $W$  with a random sample of  $m$  input vectors.
- (b) the learning gain and the spread of the neighbourhood function.

2. for every input vector,  $\mathbf{x}(n)$ ,  $n = 1, \dots, N$ :

- (a) Determine the winning neuron,  $k(n)$ , and its position  $V(k, :)$  as

$$k(n) = \arg \min_j |\mathbf{x}^T(n) - W(j, :)|$$

- (b) Calculate the neighbourhood function

$$\Lambda(n, j) = \exp\left(-\frac{\rho^2(j)}{2\sigma^2}\right)$$

where

$$\rho(j) = |V(j, :) - V(k(n), :)| \quad \text{for } j = 1, \dots, m.$$

- (c) Update the weight matrix as

$$\Delta W = \eta(n) \cdot \Lambda(n) \cdot (\mathbf{x}^T(n) - W(j, :))$$

**All** neurons (unlike in the simple competitive learning) have their weights modified with a strength proportional to the neighbourhood function and to the distance of their weight vector from the current input vector (as in competitive learning).

- (d) During the ordering phase, shrink the neighbourhood until it includes only one neuron:

$$\sigma(n+1) = \sigma(n) \cdot \delta\sigma$$

- (e) During the convergence phase, “cool down” the learning process by reducing the learning gain:

$$\eta(n+1) = \eta(n) \cdot \delta\eta$$

### 10.3 A demo script `sofm.m`

A MATLAB script, `sofm.m`, can be used to study the behaviour of the Kohonen learning algorithm which creates self-organizing feature maps. A process of generation an example of 1-D and 2-D feature maps using the `sofm.m` script is illustrated in Figures 10–11 and 10–12, respectively.

The first plot in Figure 10–11 represents a 2-D input space in which a uniformly distributed points form a letter ‘A’. Subsequent plots illustrate the feature space from its initial to final form which is attained after one pass through the training data. Neurons are organized in a 1-D lattice, their 2-D weight vectors forming an elastic string which approximates two dimensional object ‘A’.

Similarly, the plots in Figure 10–12 represent formation of a 2-D feature map approximating a 2-D triangle from the input space.

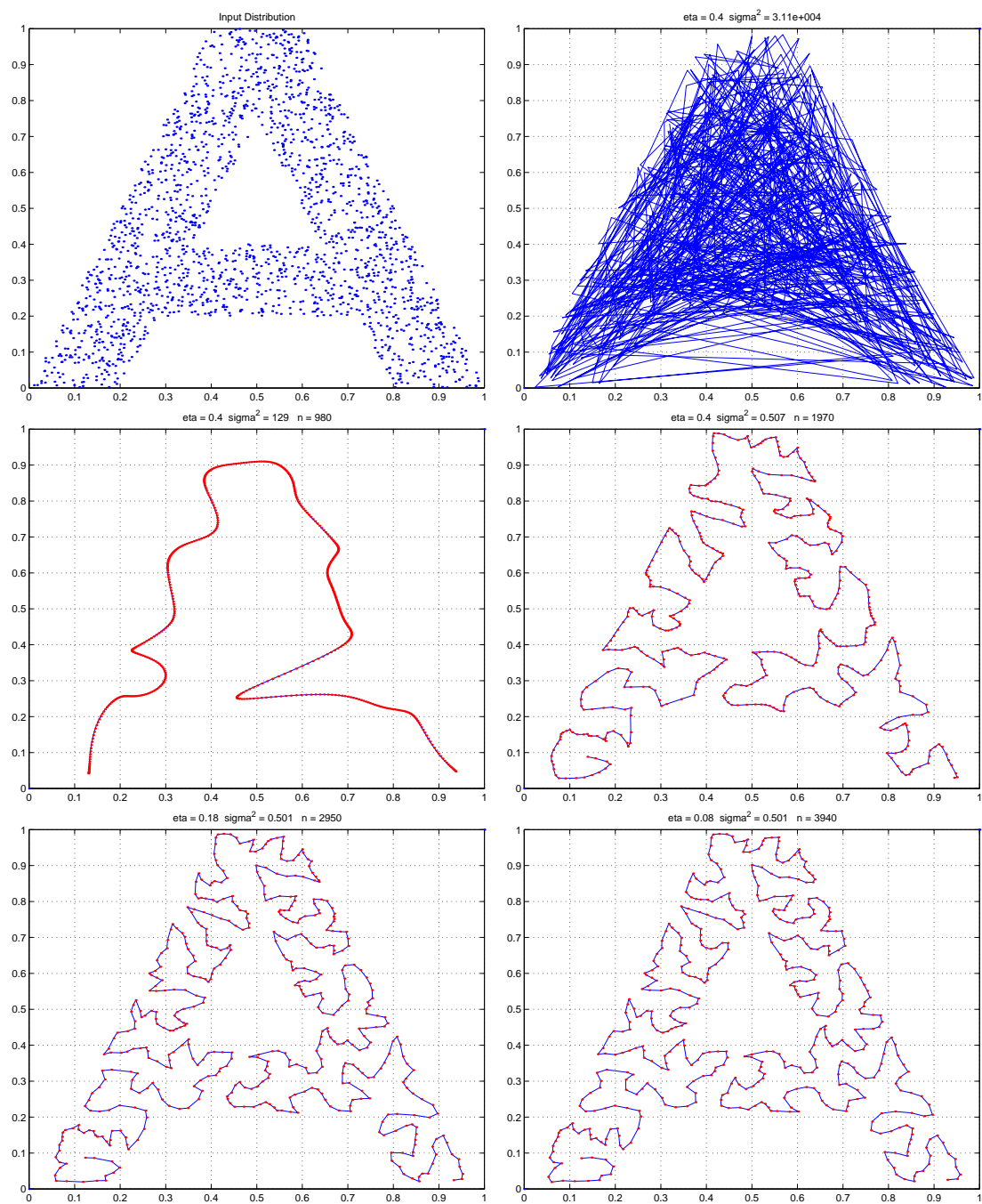


Figure 10–11: A 1-D Self-Organizing Feature Map

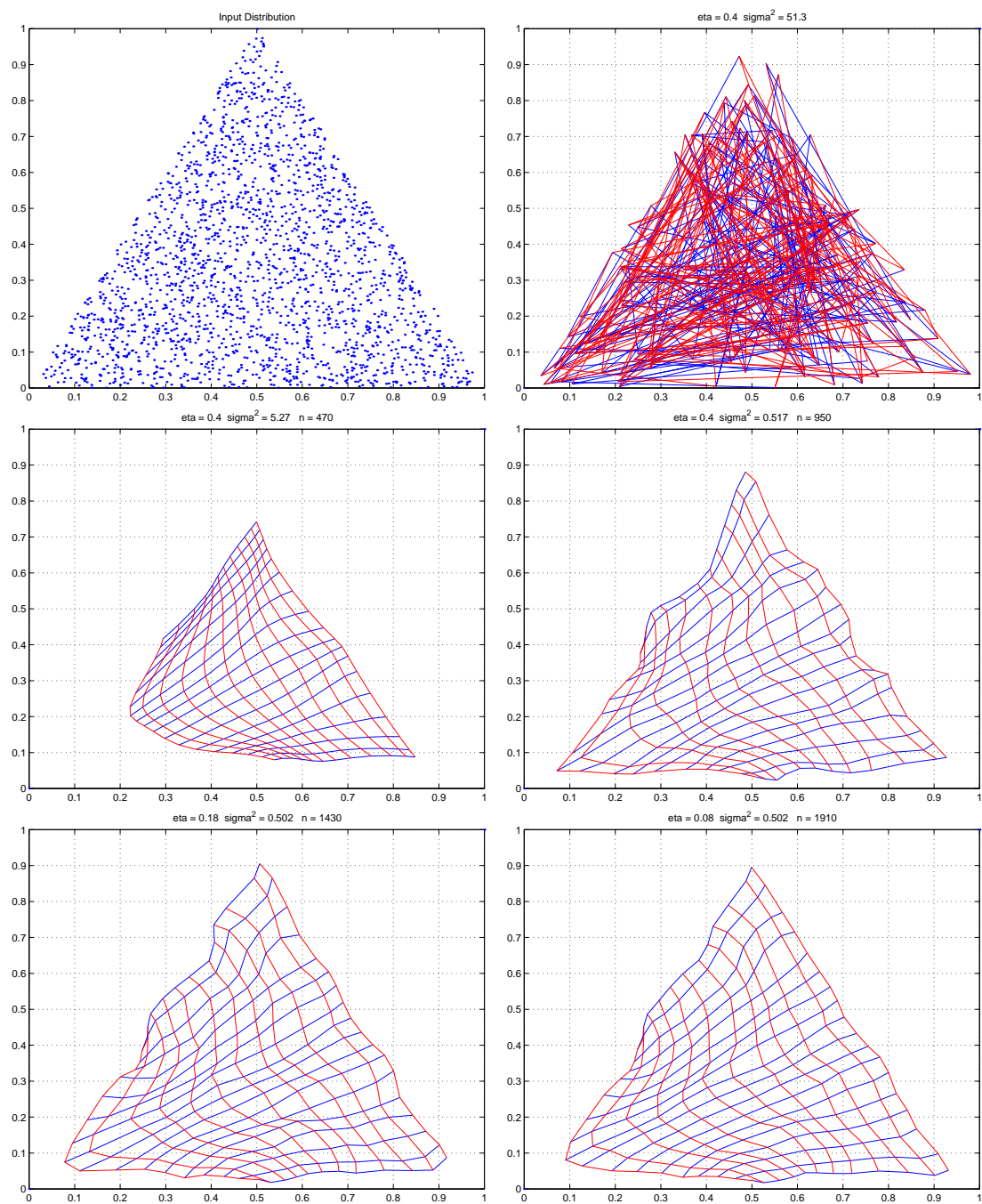


Figure 10–12: A 2-D Self-Organizing Feature Map

## References

- [DB98] H. Demuth and M. Beale. *Neural Network TOOLBOX User's Guide. For use with MATLAB*. The MathWorks Inc., 1998.
- [FS91] J.A. Freeman and D.M. Skapura. *Neural Networks. Algorithms, Applications, and Programming Techniques*. Addison-Wesley, 1991. ISBN 0-201-51376-5.
- [Has95] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. The MIT Press, 1995. ISBN 0-262-08239-X.
- [Hay99] Simon Haykin. *Neural Networks – a Comprehensive Foundation*. Prentice Hall, New Jersey, 2nd edition, 1999. ISBN 0-13-273350-1.
- [HDB96] Martin T. Hagan, H Demuth, and M. Beale. *Neural Network Design*. PWS Publishing, 1996.
- [HKP91] Hertz, Krogh, and Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991. ISBN 0-201-51560-1.
- [Kos92] Bart Kosko. *Neural Networks for Signal Processing*. Prentice-Hall, 1992. ISBN 0-13-617390-X.
- [Sar98] W.S. Sarle, editor. *Neural Network FAQ*. Newsgroup: comp.ai.neural-nets, 1998. URL: <ftp://ftp.sas.com/pub/neural/FAQ.html>.