


Self-Organizing Map

Introduction

- ❖ We will continue our study of self-organizing systems by considering a special class of artificial neural networks known as Kohonen's *self-organizing map (SOM)*. These networks are based on *competitive learning*; the output neurons compete among themselves to be activated or fired, with the result that only one output neuron, or one neuron per group is on at any time.
- ❖ In an SOM, the neurons are placed at the nodes of a *lattice* that is usually one or two dimensional. The locations of the *winning* (and *excited*) neurons are ordered with respect to each other in such a way that a meaningful coordinate system for different input features is created over the lattice.

- 
- ❖ Therefore, an SOM is characterized by the formulation of a *topographic map* of the input patterns, in which the spatial locations (i.e., coordinates) of the neurons in the lattice are indicative of *intrinsic statistical features* contained in the input patterns—hence, the name “self-organizing map”.
 - ❖ As a neural model, an SOM is organized in such a way that different inputs are represented by topologically ordered computational maps. In other words, each incoming piece of information is kept in its proper context, and the spatial location of an output neuron corresponds to a particular domain or feature of data drawn from the input space.
 - ❖ Due to the use of neighborhood, an SOM is inherently *nonlinear*. When the neighborhood is zero, it is reduced to a simplest *vector quantization* process.

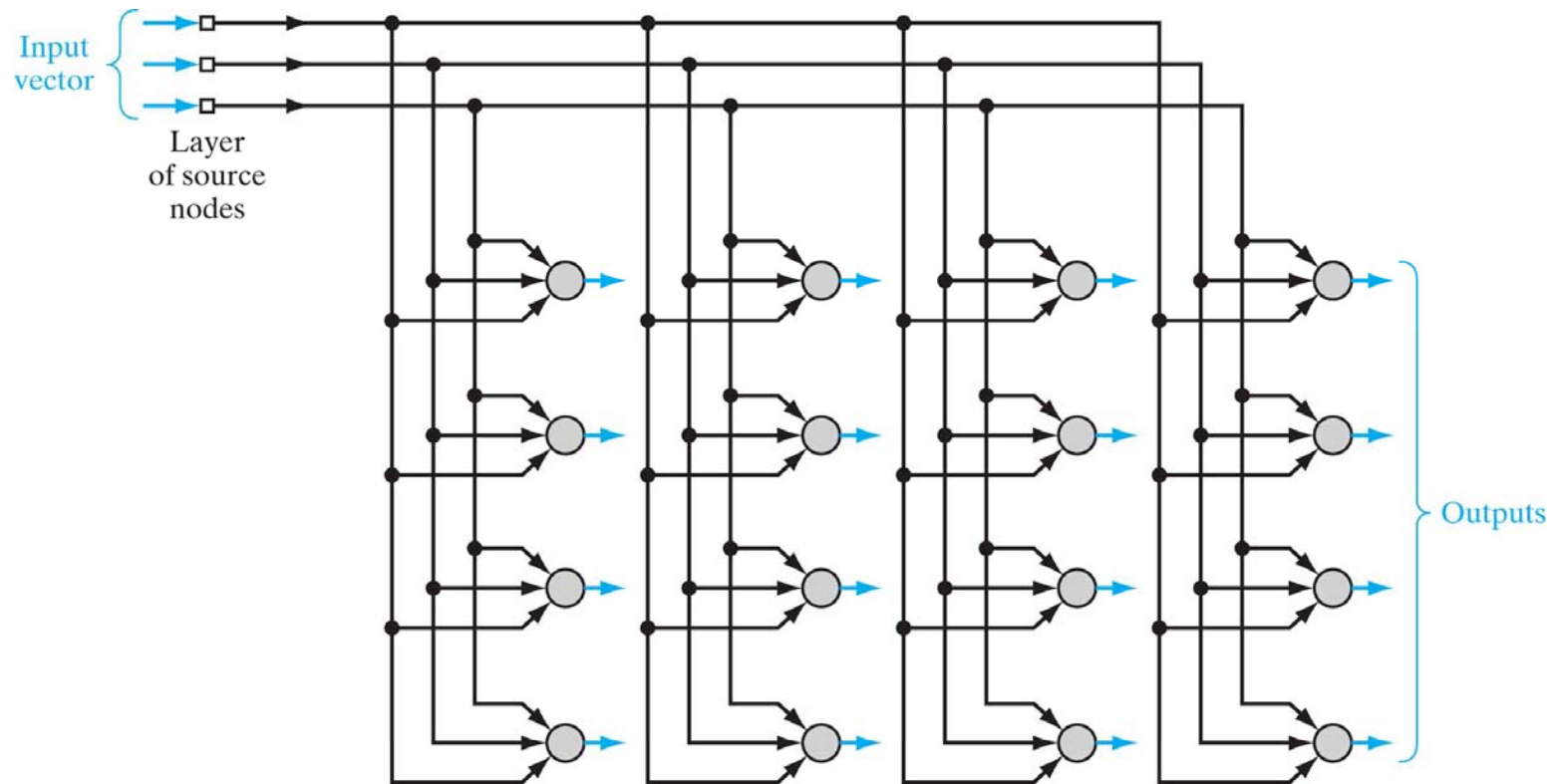



Figure 1 Two-dimensional lattice of neurons, illustrated for a three-dimensional input and four-by-four dimensional output (all shown in blue).



Self-Organizing Map

- ❖ The goal of the SOM is to transform an incoming signal pattern of arbitrary dimension into a one or two-dimensional discrete map in a topologically ordered fashion.
- ❖ The network represents a feedforward structure with a single computational (output) layer consisting of neurons arranged in column (one-dimensional) or columns and rows (two-dimensional). Each neuron in the (output) lattice is fully connected to all the source nodes in the input layer.
- ❖ The formation of the map proceeds first by initializing the synaptic weights by assigning them small values picked from a random-number generator. Once the neural network is properly initiated, there are three essential processes involved in the formation of the map:

- 
- *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. The particular neuron with the largest value of discriminant function is declared winner of the competition.
 - *Cooperation.* The winning neuron determines the spatial location of a topological neighborhood of excited neurons, thereby providing the basis for cooperation among such neighboring neurons.
 - *Adaption.* The synaptic weights of the winning neuron and its neighbors are adjusted such that their individual values of the discriminant function in relation to similar input patterns will be increased.

❖ Competitive Process

Let an m -dimensional input pattern selected at random from the input space be denoted as

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

Let the weight vector of the j th output neuron be denoted by

$$\mathbf{w}_j = [w_{j1}, w_{j2}, \dots, w_{jm}]^T, \quad j = 1, 2, \dots, l$$

The value of the j th output neuron is

$$y_j = \mathbf{w}_j^T \mathbf{x}, \quad j = 1, 2, \dots, l$$

A matching criterion is used to generate the winner:

$$i(\mathbf{x}) = \max_j (\mathbf{w}_j^T \mathbf{x})$$

or

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

❖ Cooperative Process

The winning neuron locates the center of a topological neighborhood of cooperating neurons. Let $h_{j,i}$ denote the 2D topological neighborhood centered on winning neuron i and encompassing a set of *excited* (i.e., *cooperating*) neurons, a typical one of which is denoted by j . Let $d_{j,i}$ denote the lateral distance between the winning neuron i and the excited neuron j . The basic requirements are:

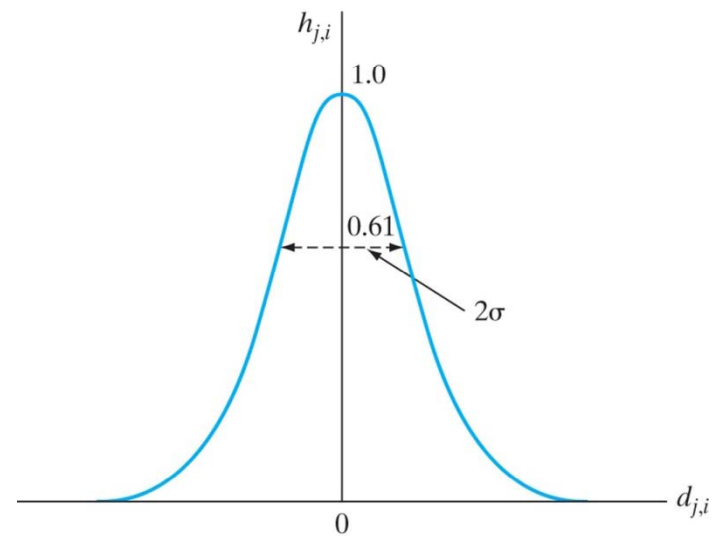
- The topological neighborhood $h_{j,i}$ is symmetric about the maximum point defined by $d_{j,i} = 0$; in other words, it attains its maximum value at the winning neuron i for which $d_{j,i} = 0$.
- The amplitude of the topological neighborhood $h_{j,i}$ decreases monotonically with increasing lateral distance $d_{j,i}$, decaying to zero as $d_{j,i} \rightarrow \infty$. This is a necessary condition for convergence.

A good choice of $h_{j,i}$ that satisfies these requirements is the Gaussian function:

$$h_{j,i}(\mathbf{x}) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right)$$

For 1D lattice, $d_{j,i}^2 = |j - i|$

For 2D lattice, $d_{j,i}^2 = \|\mathbf{r}_j - \mathbf{r}_i\|^2$



where \mathbf{r}_j and \mathbf{r}_i are the vectors representing the coordinates of the excited neuron j and the winning neuron i .

The size of the topological neighborhood should be shrunk with time. This requirement can be satisfied by making the width σ of the topological neighborhood function decrease with time.

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right)$$
$$h_{j,i(\mathbf{x})}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma(n)^2}\right)$$

where σ and τ_1 are user-defined constants.

The neighborhood eventually includes the winning neuron i itself, meaning competition dominates over cooperation.

❖ Adaptive Process

The Hebb learning rule can be used to update weights, with weight vector normalization step or a *forgetting term*.

$$\Delta \mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j) \mathbf{w}_j = \eta y_j \mathbf{x} - \eta y_j \mathbf{w}_j = \eta y_j (\mathbf{x} - \mathbf{w}_j)$$

To include the neighbors for update,


$$\Delta \mathbf{w}_j = \eta h_{j,i(\mathbf{x})} (\mathbf{x} - \mathbf{w}_j)$$

which means closer neighbors receive larger updates and the winner receives the largest update. Here, the specific value of y_j is not important, which is mainly used to determine the winner; this value is absorbed by $\eta h_{j,i(\mathbf{x})}$.

The learning rate should be decreased with the time as:

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

where η and τ_2 are user-defined parameters.



Two phases of the adaptive process:

- *Self-organizing or ordering phase*: It is during this phase that the topological ordering of the weight vectors takes place. The ordering phase may take many iterations. Careful consideration must therefore be given to the choice of the learning rate parameter and neighborhood function.
 - The learning rate should begin with some large (but not too large) value, say, 0.1; then it is gradually decreased to some small (but not too small) value, say 0.01.
 - The neighborhood function should initially include almost all neurons, and then shrink slowly with time; or eventually include a couple of neurons or the winning neuron itself.
- *Convergence phase*: This second phase is needed to fine-tune the feature map, and therefore provide an accurate statistical quantification of the input space.
 - The learning rate should remain a small value.
 - The neighborhood function should contain one or zero neighboring neurons only.



❖ Summary of the Overall Algorithm

- *Initialization*: Randomly initiate the weight vectors with small random values.
- *Sampling*: Draw a sample \mathbf{x} from the input space with a certain probability.
- *Similarity matching*: Find the winner by using a matching function.
- *Updating*: Adjust the weights for the winner and all excited neurons.
- *Continuation*: Adjust the learning rate and neighborhood size as needed.
- *Termination*: Terminate the algorithm when no noticeable changes in the feature map (represented by weight vectors).

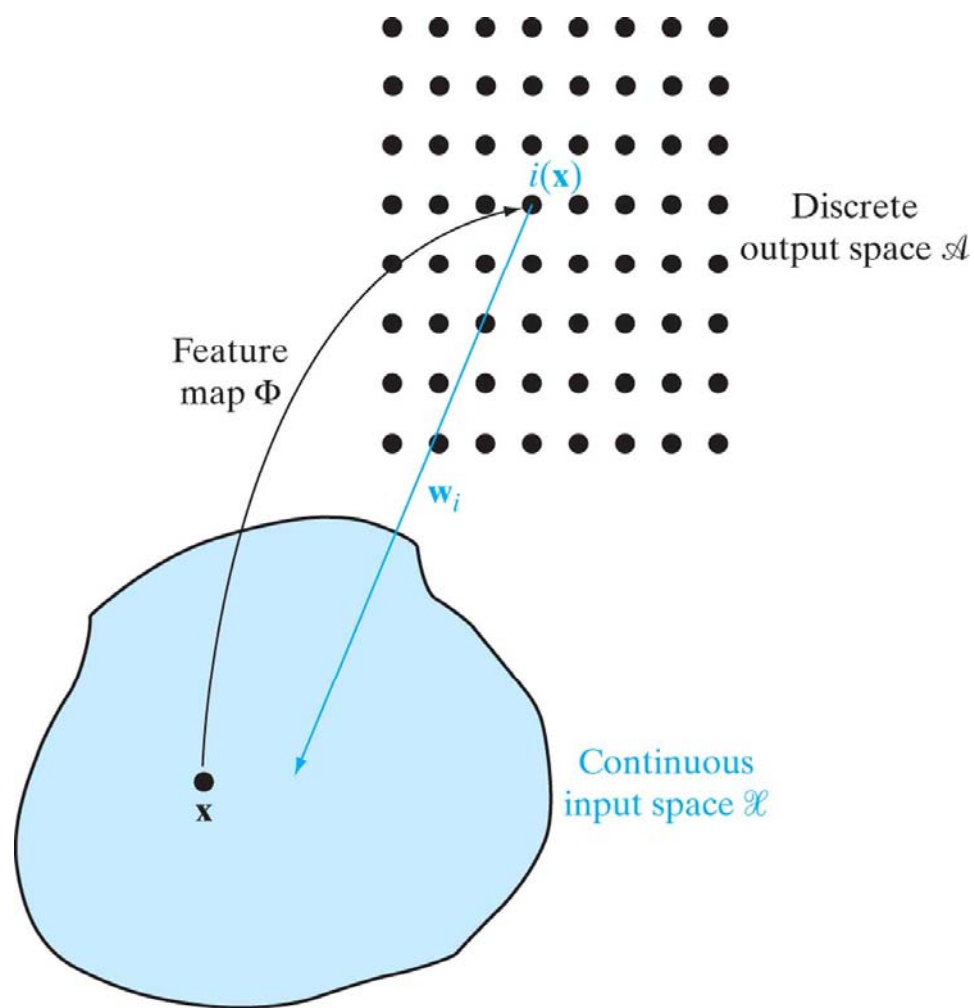


Figure 2 Illustration of the relationship between feature map Φ and weight vector \mathbf{w}_i of winning neuron i .

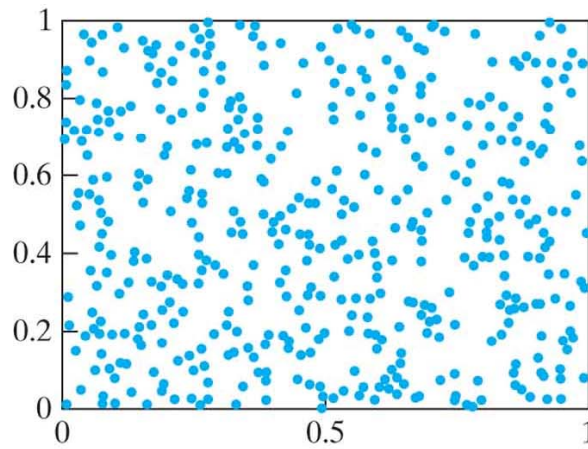
Computer Simulations

- ❖ 2D lattice driven by 2D stimulus

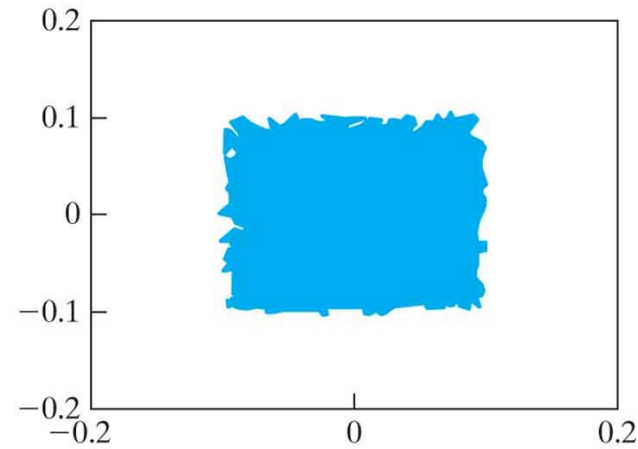
Input vectors are 2D vectors whose x_1 and x_2 elements are uniformly distributed within $(-1,1)$. The output layer consists of $24 \times 24 = 576$ neurons organized as a 2D lattice.

- ❖ 1D lattice driven by 2D stimulus

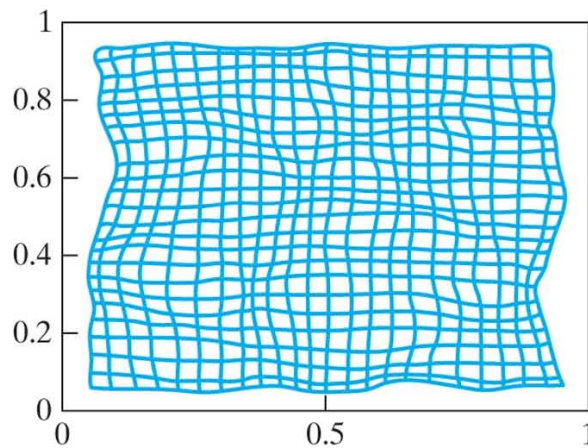
The output layer consists of 100 neurons as a 1D lattice.



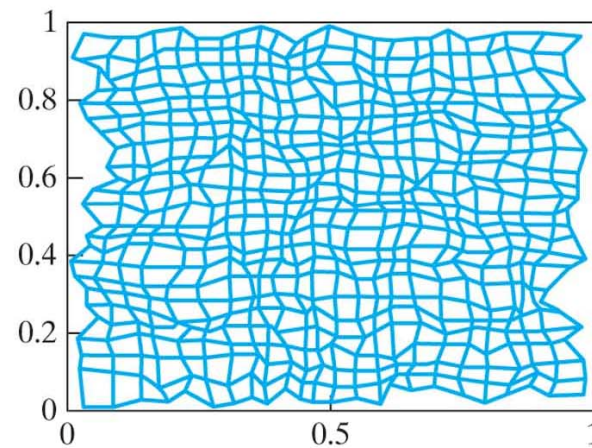
(a) Input distribution



Time = 0
(b) Initial weights

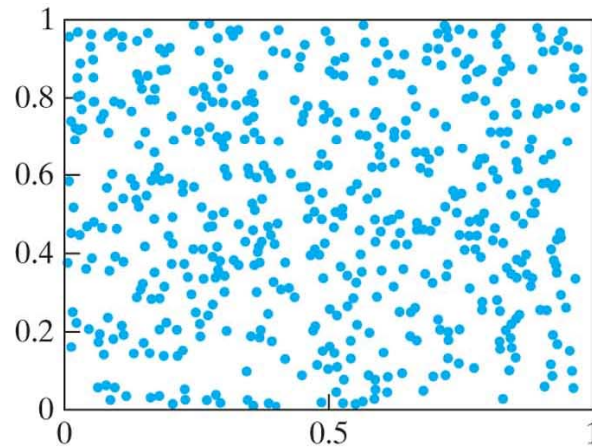


Time = 160 K
(c) Ordering phase

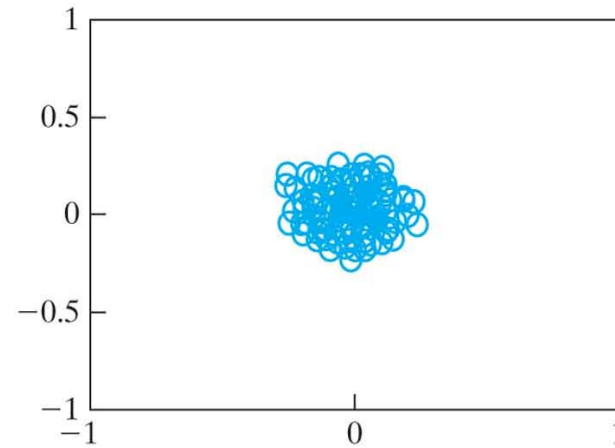


Time = 800 K
(d) Convergence phase

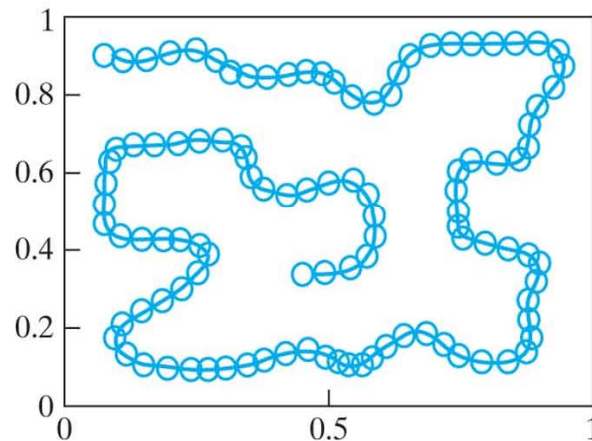
Figure 3 (a) Distribution of the input data. (b) Initial condition of the two-dimensional lattice. (c) Condition of the lattice at the end of the ordering phase. (d) Condition of the lattice at the end of the convergence phase. The times indicated under maps (b), (c), and (d) represent the numbers of iterations.



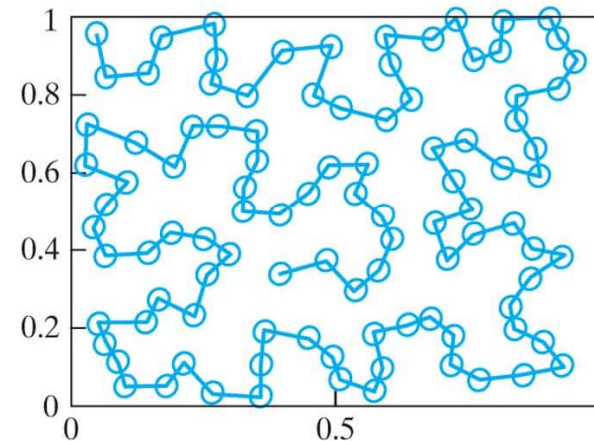
(a) Input distribution



Time = 0
(b) Initial weights



Time = 50 K
(c) Ordering phase



Time = 100 K
(d) Converging phase

Figure 4 (a) Distribution of the two-dimensional input data. (b) Initial condition of the one-dimensional lattice. (c) Condition of the one dimensional lattice at the end of the ordering phase. (d) Condition of the lattice at the end of the convergence phase. The times included under maps (b), (c), and (d) represent the numbers of iterations.



❖ Contextual Maps

There are two fundamentally different ways of visualizing a self-organizing feature map.

- In the first method, the feature map is viewed as an elastic net, with the synaptic-weight vectors treated as pointers for the respective neurons, which are directed into the input space. This method of visualization is particularly useful for displaying the topological-ordering property of the SOM algorithm.
- In the second method, class labels are assigned to neurons in a 2D lattice, depending on how an input pattern excites a particular neuron in the network. The neurons in the 2D lattice are partitioned into a number of coherent regions, coherent in the sense that each grouping of neurons represents a distinct set of contiguous symbols or labels.

■ An example with 16 animals:

- The features of 16 animals shown in Table 9.2 can be represented by 13-dimensional binary code \mathbf{x}_a .
- An animal itself is specified by a symbol code \mathbf{x}_s , the composition of which does not convey any information or known similarities between animals. It consists of a column vector whose k th element, representing animal $k = 1, 2, \dots, 16$, is given a fixed value of a ; the remaining elements are all set equal to zero. The parameter a determines the relative influence of the symbol code compared with that of the attribute code. To make sure the attribute code is the dominant one, a is chosen to be 0.2.
- The input vector \mathbf{x} is the one concatenating \mathbf{x}_s and \mathbf{x}_a : $\mathbf{x} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{x}_a \end{bmatrix}$, and the vector is normalized to unit length. The output neurons are organized in 10×10 lattice.
- After the SOM is converged, the test patterns including symbol code only, i.e., $\mathbf{x} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{0} \end{bmatrix}$. The neuron with the strongest response is identified.
- The feature map essentially captures the “family relationships” among the 16 different animals.

TABLE 9.2 Animal Names and Their Attributes

Animal		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	1	0
	fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

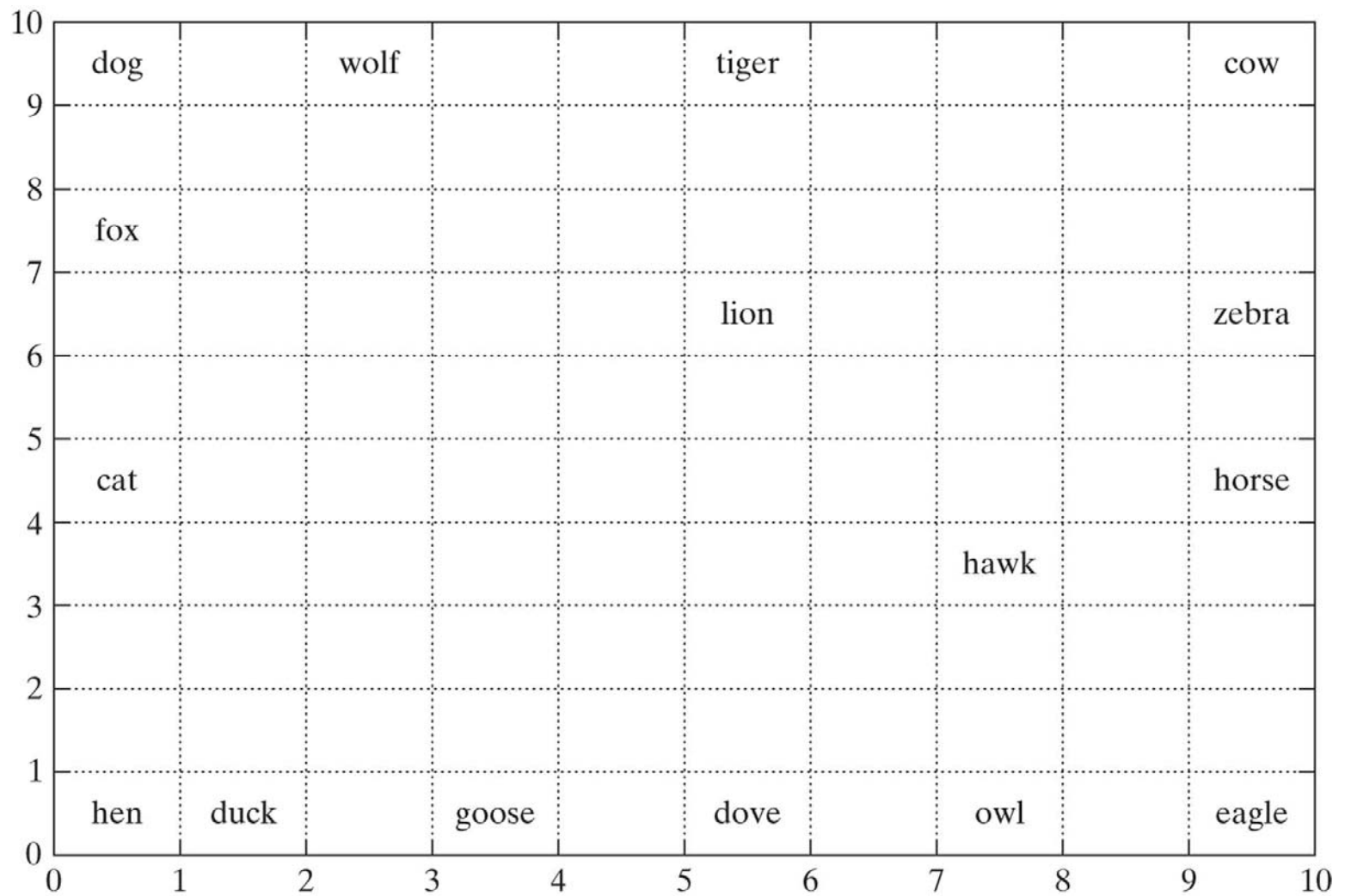


Figure 5 Feature map containing labeled neurons with strongest responses to their respective inputs.

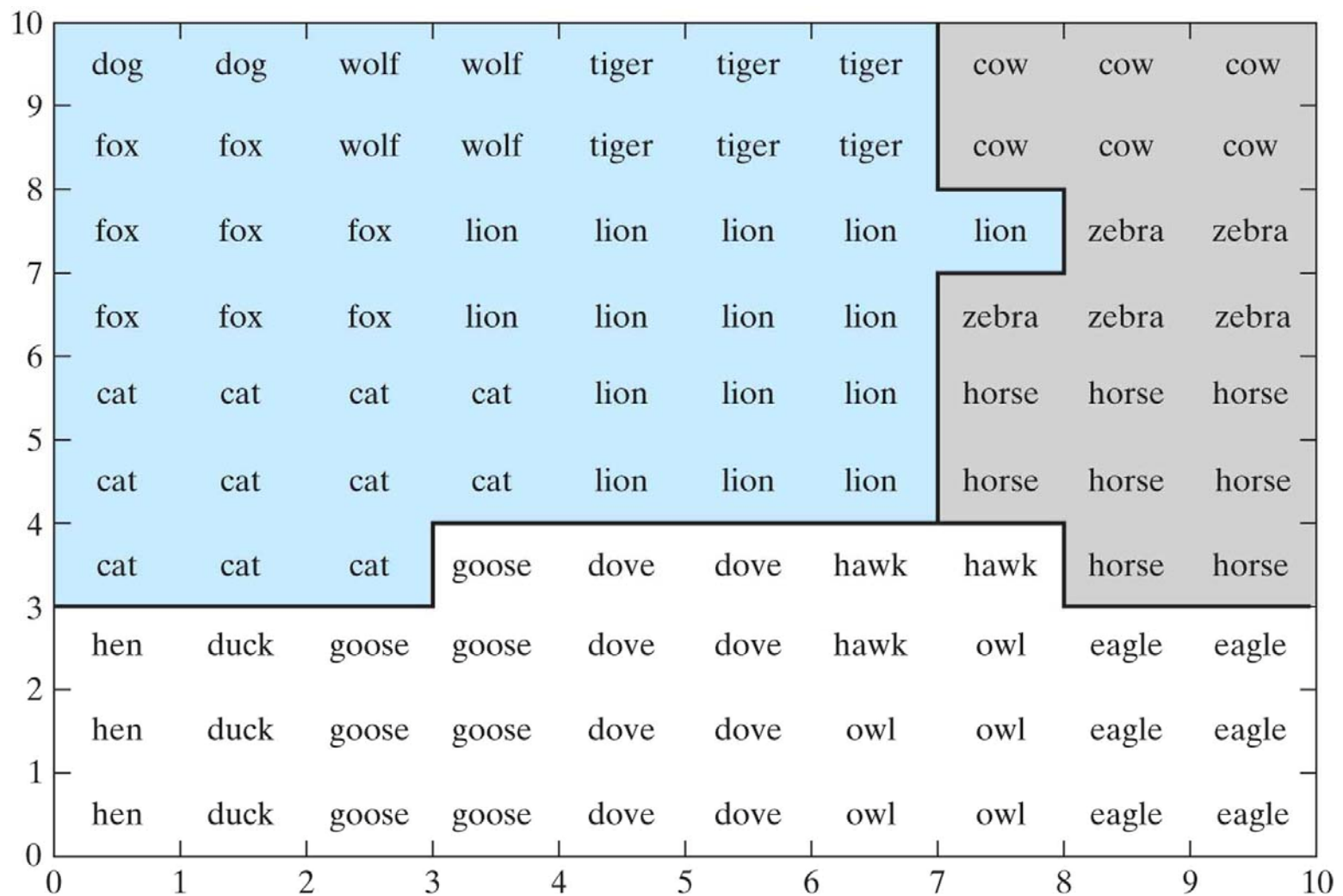


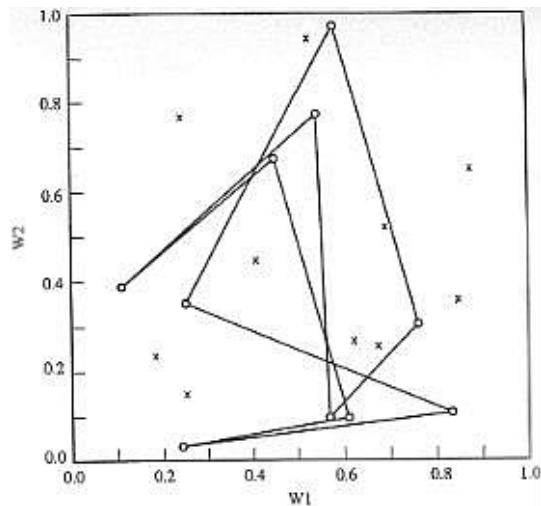
Figure 6 Semantic map obtained through the use of simulated electrode penetration mapping, where each neuron is marked by the animal for which it produced the best response. The map is divided into three regions, representing birds (white), peaceful species (grey), and hunters (blue).

The traveling Salesman Problem

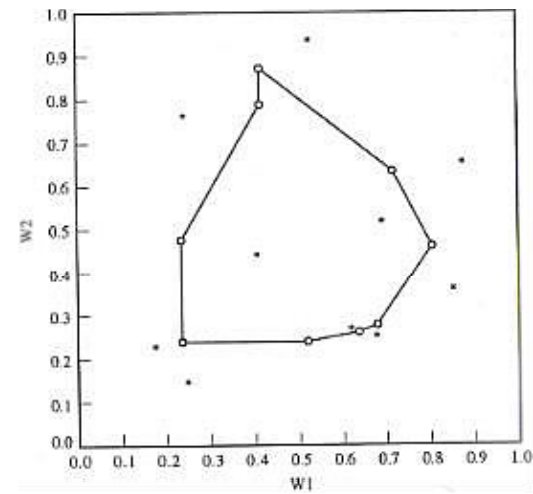
The linear topology for the cluster units in a SOM can be used to solve a classic problem in constrained optimization, the so-called *traveling salesman problem* (TSP). The aim of the TSP is to find a tour of a given set of cities that is of minimum length. A tour consists of visiting each city exactly once and returning to the starting city.

The SOM net uses the city coordinates as input; there are as many cluster units as there are cities to be visited. A linear structure is assumed with the first and last cluster units also being connected, and winning unit and its nearest neighbor unit on either side ($J, J+1, J-1$) are allowed to learn.

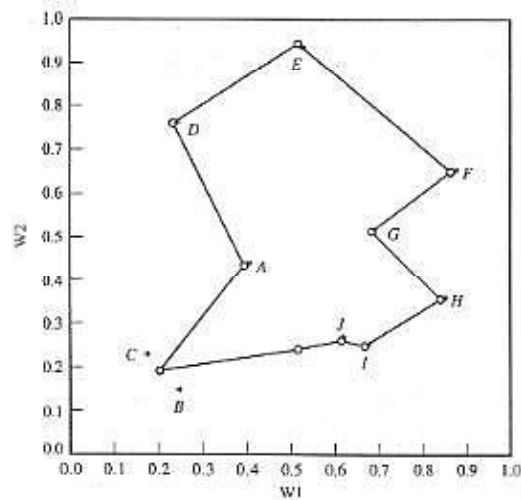
In this example, the final tour is ambiguous in terms of the order in which city B and city C are visited, because one cluster unit is positioned midway between the cities and another unit is trapped between city J and cities B and C. However, the results can easily be interpreted as representing one of the tours: A D E F G I J B C or A D E F G I J C B.



**Initial position of cluster units
and location of cities**



**Initial position of cluster units
after 100 epochs**




**Initial position of cluster units
after 200 epochs**

Kernel Self-Organizing Map

- ❖ In a kernel SOM, each neuron in the lattice structure of the map acts as a kernel. As such, the kernel parameters are adjusted individually in accordance with a prescribed objective function, which is maximized iteratively so as to facilitate the formation of a satisfactory topographic map.
- ❖ Here, we focus on the *joint entropy* of the kernel (i.e., neural) outputs as the objective function. Consider a continuous random variable Y_i , whose pdf is denoted by $p_{Y_i}(y_i)$, where the sample value y_i lies in the range $0 \leq Y_i < \infty$. The *differential entropy* of Y_i is defined by

$$H(Y_i) = - \int_{-\infty}^{+\infty} p_{Y_i}(y_i) \log p_{Y_i}(y_i) dy_i$$

For the kernel SOM, the random variable Y_i , refers to the output of the i th kernel in the lattice, and y_i refers to a sample value of Y_i .

- 
- ❖ For the kernel SOM learning, we will proceed in a bottom-up manner:
 - The differential entropy of a given kernel is first maximized.
 - Then, when this maximization is attained, the kernel parameters are adjusted so as to maximize the “mutual information” between the kernel’s output and input.
 - ❖ Let the kernel be noted by $k(\mathbf{x}, \mathbf{w}_i, \sigma_i)$, where \mathbf{x} is the input vector of dimensionality m , \mathbf{w}_i is the weight vector of the i th kernel, and σ_i is its width; the index $i = 1, 2, \dots, l$, where l is the total number of neurons constituting the lattice structure of the map. The distance between \mathbf{x} and \mathbf{w}_i is measured by the Euclidean distance, i.e., $r = \|\mathbf{x} - \mathbf{w}_i\|$.

The differential entropy is maximized when Y_i is uniformly distributed because entropy is a measure of randomness and a uniform distribution is the extreme form of randomness.

- ❖ The differential entropy is maximized when Y_i is uniformly distributed because entropy is a measure of randomness and a uniform distribution is the extreme form of randomness. The optimality condition occurs when the output distribution matches the cumulative distribution of the input space, *because the cumulative distribution function (CDF) of any random variable is another random variable with uniform pdf*. For a Gaussian distributed input vector \mathbf{x} , the CDF of the corresponding Euclidean distance $r = \|\mathbf{x} - \mathbf{w}_i\|$ is the *incomplete gamma distribution*. Thus, this distribution should be the desired kernel definition:

$$y_i = k(\mathbf{x}, \mathbf{w}_i, \sigma_i) = \frac{1}{\Gamma\left(\frac{m}{2}\right)} \Gamma\left(\frac{m}{2}, \frac{\|\mathbf{x} - \mathbf{w}_i\|^2}{2\sigma_i^2}\right), \quad i = 1, 2, \dots, l$$

This adoption ensures that the kernel's differential entropy is maximized when the input distribution is Gaussian.

- ❖ With the kernel function at hand, we can derive formulas for the gradients of the objective function (defined as differential entropy) with respect to the kernel parameters: \mathbf{w}_i and σ_i

$$\Delta \mathbf{w}_i = \eta_w \frac{(\mathbf{x} - \mathbf{w}_i)}{\sigma_i^2} \quad \Delta \sigma_i = \frac{\eta_\sigma}{\sigma_i} \left(\frac{\|\mathbf{x} - \mathbf{w}_i\|^2}{m \sigma_i^2} - 1 \right)$$

To supply the information needed for topological map formation, we introduce a neighborhood function $h_{j,i(\mathbf{x})}$, centered on the winning neuron $i(\mathbf{x})$ as

$$h_{j,i(\mathbf{x})} = \exp\left(-\frac{d_{ji}^2}{2\sigma^2}\right)$$

where σ denotes the range of the neighborhood function, and winning neuron is defined by

$$i(\mathbf{x}) = \arg \max_i y_j(\mathbf{x})$$

Summary of Kernel SOM

- *Initialization*: choose random values for the initial weight vectors $\mathbf{w}_i(0)$ and kernel widths $\sigma_i(0)$ for $i=1, 2, \dots, l$, where l is the total number of output neurons. $\mathbf{w}_i(0)$ and $\sigma_i(0)$ should be different for the different neurons.
- *Sampling*: Draw a sample \mathbf{x} from the input data points.
- *Matching*: At time-step n , identify the winning neuron

$$i(\mathbf{x}) = \arg \max_i y_j(\mathbf{x})$$

- *Adaptation*: Adjust the weight vector and width of each kernel:

$$\Delta \mathbf{w}_i = \eta_{\mathbf{w}} h_{j,i(\mathbf{x})} \frac{(\mathbf{x} - \mathbf{w}_i)}{\sigma_i^2} \quad \Delta \sigma_i = \frac{\eta_{\sigma} h_{j,i(\mathbf{x})}}{\sigma_i} \left(\frac{\|\mathbf{x} - \mathbf{w}_i\|^2}{m \sigma_i^2} - 1 \right)$$

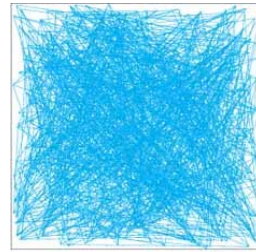
Computer Simulation

❖ 2D lattice driven by 2D stimulus

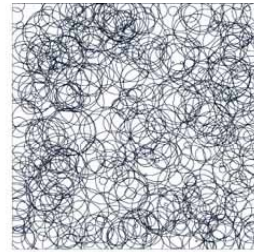
Input vectors are 2D vectors whose x_1 and x_2 elements are uniformly distributed within $(-1,1)$. The output layer consists of $24 \times 24 = 576$ neurons organized as a 2D lattice.

Comparing the final form of the topographic map on the left-hand column of Fig. 9.15 with that of Fig. 9.8, computed by the kernel SOM and the conventional SOM on 24-by-24 lattices and for roughly the same number of iterations, respectively, we may make the following significant observation:

The distribution of the topographic map computed by the kernel SOM is much closer to the uniform distribution assigned to the input data space than that of the topologic map computed by the conventional SOM.



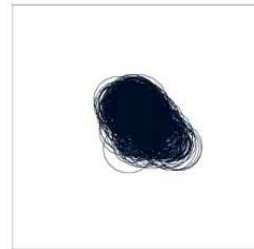
Time = 0



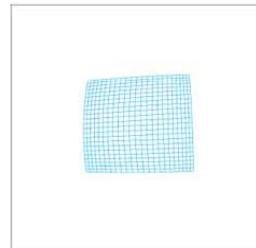
Time = 0



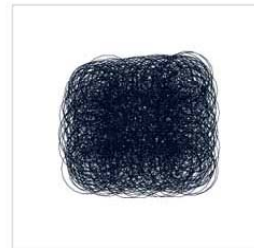
Time = 1k



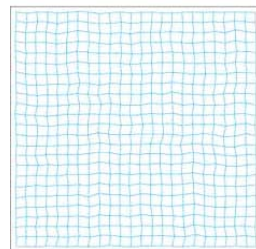
Time = 1k



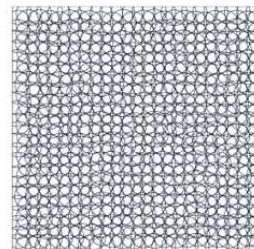
Time = 10k



Time = 10k



Time = 2M



Time = 2M

Figure 9.15 The evolution of a 24-by-24 lattice over time, the values of which (in terms of the number of iterations) are given below each picture. Left column: Evolution of the kernel weights. Right column: Evolution of the kernel widths. Each box in the figure outlines the result of a uniform input distribution. The time given below each map represents the number of iterations.