CHAPTER 9
Self-Organizing Maps

Problem 9.1

Expanding the function $g(y_j)$ in a Taylor series around $y_j = 0$, we get

$$g(y_j) = g(0) + g^{(1)}(0)y_j + \frac{1}{2!}g^{(2)}(0)y_j^2 + \ldots \tag{1}$$

where

$$g^{(k)}(0) = \left.\frac{\partial^k g(y_j)}{\partial y_j^k}\right|_{y_j = 0} \qquad \text{for } k = 1, 2, \ldots.$$

Let

$$y_j = \begin{cases} 1, & \text{neuron } j \text{ is on} \\ 0, & \text{neuron } j \text{ is off} \end{cases}$$

Then, we may rewrite (1) as

$$g(y_j) = \begin{cases} g(0) + g^{(i)}(0) + \frac{1}{2!}g^{(2)}(0) + \ldots, & \text{neuron } j \text{ is on} \\ g(0) & \text{neuron } j \text{ is off} \end{cases}$$

Correspondingly, we may write

$$\frac{d\mathbf{w}_j}{dt} = \eta y_j \mathbf{x} - g(y_j)\mathbf{w}_j$$

$$= \begin{cases} \eta\mathbf{x} - \mathbf{w}_j\left[g(0) + g^{(1)}(0) + \frac{1}{2!}g^{(2)}(0) + \ldots\right] & \text{neuron } j \text{ is on} \\ -g(0)\mathbf{w}_j & \text{neuron } j \text{ is off} \end{cases}$$

Consequently, a nonzero $g(0)$ has the effect of making $d\mathbf{w}_j/dt$ assume a nonzero value when neuron j is off, which is undesirable. To alleviate this problem, we make $g(0) = 0$.

Problem 9.2

Assume that $y(\mathbf{c})$ is a minimum L_2 (least-squares) distortion vector quantizer for the code vector $\mathbf{c}$. We may then form the distortion function

$$D_2 = \frac{1}{2}\int f(\mathbf{c})\|\mathbf{c}'(y(\mathbf{c})) - \mathbf{c}\|^2 d\mathbf{c}$$

This distortion function is similar to that of Eq. (10.20) in the text, except for the use of $\mathbf{c}$ and $\mathbf{c}'$ in place of $\mathbf{x}$ and $\mathbf{x}'$, respectively. We wish to minimize D_2 with respect to $y(\mathbf{c})$ and $\mathbf{c}'(y)$.

Assuming that $\pi(\nu)$ is a smooth function of the noise vector ν, we may expand the decoder output $\mathbf{x}'$ in ν using the Taylor series. In particular, using a second-order approximation, we get (Luttrell, 1989b)

$$\int \pi(\nu)\|\mathbf{x}'(\mathbf{c}(\mathbf{x}) + \nu) - \mathbf{x}\|^2 d\nu \approx \left(1 + \frac{D_2}{2}\nabla_k^2\right)\|\mathbf{x}'(\mathbf{c}) - \mathbf{x}\|^2 \tag{1}$$

where

$$\int \pi(\nu) d\nu = 1$$
$$\int n_i \pi(\nu)(d\nu) = 0$$
$$\int n_i n_j \pi(\nu)(d\nu) = D_2 \delta_{ij}$$

where δ_{ij} is a Kronecker delta function. We now make the following observations:

- The first term on the right-hand side of (1) is the conventional distortion term.
- The second term (i.e., curvature term) arises due to the output noise model $\pi(\nu)$.

Problem 9.3

Consider the Peano curve shown in part (d) of Fig. 9.9 of the text. This particular self-organizing feature map pertains to a one-dimensional lattice fed with a two-dimensional input. We see that (counting from left to right) neuron 14, say, is quite close to neuron 97. It is therefore possible for a large enough input perturbation to make neuron 14 jump into the neighborhood of neuron 97, or vice versa. If this change were to happen, the topological preserving property of the SOM algorithm would no longer hold

For a more convincing demonstration, consider a higher-dimensional, namely, three-dimensional input structure mapped onto a two-dimensional lattice of 10-by-10 neurons. The

network is trained with an input consisting of 8 Gaussian clouds with unit variance but different centers. The centers are located at the points (0,0,0,...,0), (4,0,0,...,0), (4,4,0,...,0), (0,4,0,...,0), (0,0,4,...,0), (4,0,4, ...,0), (4,4,4, ..., 0), and (0,4,4, ...,0). The clouds occupy the 8 corners of a cube as shown in Fig. 1a. The resulting labeled feature map computed by the SOM algorithm is shown in Fig. 1b. Although each of the classes is grouped together in the map, the planar feature map fails to capture the complete topology of the input space. In particular, we observe that class 6 is adjacent to class 2 in the input space, but is *not* adjacent to it in the feature map.

The conclusion to be drawn here is that although the SOM algorithm does perform clustering on the input space, it may not always completely preserve the topology of the input space.

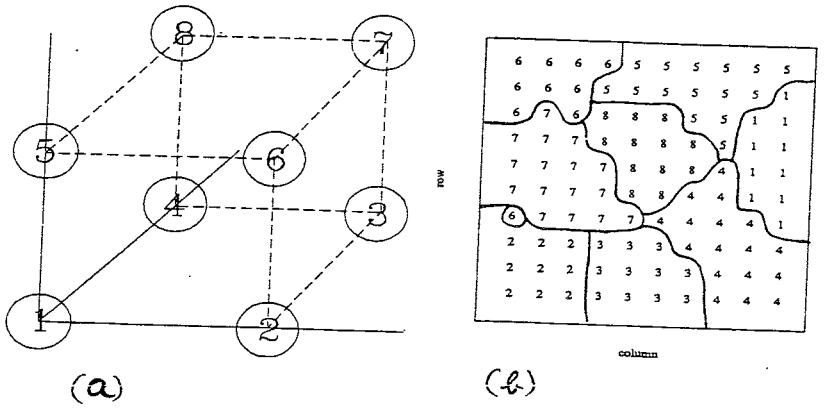

Figure 1: Problem 9.3

Problem 9.4

Consider for example a two-dimensional lattice using the SOM algorithm to learn a two-dimensional input distribution as illustrated in Fig. 9.8 in the textbook. Suppose that the neuron at the center of the lattice breaks down; this failure may have a dramatic effect on the evolution of the feature map. On the other hand, a small perturbation applied to the input space leaves the map learned by the lattice essentially unchanged.

Problem 9.5

The batch version of the SOM algorithm is defined by

$$\mathbf{w}_j = \frac{\sum_i \pi_{j,i}\mathbf{x}_i}{\sum_i \pi_{j,i}} \quad \text{for some prescribed neuron } j \tag{1}$$

where $\pi_{j,i}$ is the discretized version of the pdf $\pi(\nu)$ of noise vector ν. From Table 9.1 of the text we recall that $\pi_{j,i}$ plays a role analogous to that of the neighborhood function. Indeed, we can

substitute $h_{j,i(\mathbf{x})}$ for $\pi_{j,i}$ in (1). We are interested in rewriting (1) in a form that highlights the role of Voronoi cells. To this end we note that the dependence of the neighborhood function $h_{j,i(\mathbf{x})}$ and therefore $\pi_{j,i}$ on the input pattern $\mathbf{x}$ is indirect, with the dependence being through the Voronoi cell in which $\mathbf{x}$ lies. Hence, for all input patterns that lie in a particular Voronoi cell the same neighborhood function applies. Let each Voronoi cell be identified by an indicator function $I_{i,k}$ interpreted as follows:

$I_{i,k} = 1$ if the input pattern $\mathbf{x}_i$ lies in the Voronoi cell corresponding to winning neuron k. Then in light of these considerations we may rewrite (1) in the new form

$$\mathbf{w}_j = \frac{\sum_k \pi_{j,k} \sum_i I_{i,k} \mathbf{x}_i}{\sum_k \pi_{j,k} \sum_i I_{i,k}} \tag{2}$$

Now let $\mathbf{m}_k$ denote the centroid of the Voronoi cell of neuron k and N_k denote the number of input patterns that lie in that cell. We may then simplify (2) as

$$\begin{aligned} \mathbf{w}_j &= \frac{\sum_k \pi_{j,k} N_k \mathbf{m}_k}{\sum_k \pi_{j,k} N_k} \\ &= \sum_k W_{j,k} \mathbf{m}_k \end{aligned} \tag{3}$$

where $W_{j,k}$ is a weighting function defined by

$$W_{j,k} = \frac{\pi_{j,k} N_k}{\sum_k \pi_{j,k} N_k} \tag{4}$$

with

$$\sum_k W_{j,k} = 1 \quad \text{for all } j$$

Equation (3) bears a close resemblance to the Watson-Nadaraya regression estimator defined in Eq. (5.61) of the textbook. Indeed, in light of this analogy, we may offer the following observations:

- The SOM algorithm is similar to nonparametric regression in a statistical sense.
- Except for the normalizing factor N_k, the discretized pdf $\pi_{j,i}$ and therefore the neighborhood function $h_{j,i}$ plays the role of a kernel in the Watson-Nadaraya estimator.

- The width of the neighborhood function plays the role of the span of the kernel.

Problem 9.6

In its basic form, Hebb's postulate of learning states that the adjustment Δw_{kj} applied to the synaptic weight w_{kj} is defined by

$$\Delta w_{kj} = \eta y_k x_j$$

where y_k is the output signal produced in response to the input signal x_j.

The weight update for the maximum eigenfilter includes the term $\eta y_k x_j$ and, additionally, a stabilizing term defined by $-y_k^2 w_{kj}$. The term $\eta y_k x_j$ provides for synaptic amplification.

In contrast, in the SOM algorithm two modifications are made to Hebb's postulate of learning:

1. The stabilizing term is set equal to $-y_k w_{kj}$.
2. The output y_k of neuron k is set equal to a neighborhood function.

The net result of these two modifications is to make the weight update for the SOM algorithm assume a form similar to that in competitive learning rather than Hebbian learning.

Problem 9.7

In Fig. 1 (shown on the next page), we summarize the density matching results of computer simulation on a one-dimensional lattice consisting of 20 neurons. The network is trained with a triangular input density. Two sets of results are displayed in this figure:

1. The standard SOM (Kohonen) algorithm, shown as the solid line.
2. The conscience algorithm, shown as the dashed line; the line labeled "predict" is its straight-line approximation.

In Fig. 1, we have also included the exact result. Although it appears that both algorithms fail to match the input density exactly, we see that the conscience algorithm comes closer to the exact result than the standard SOM algorithm.

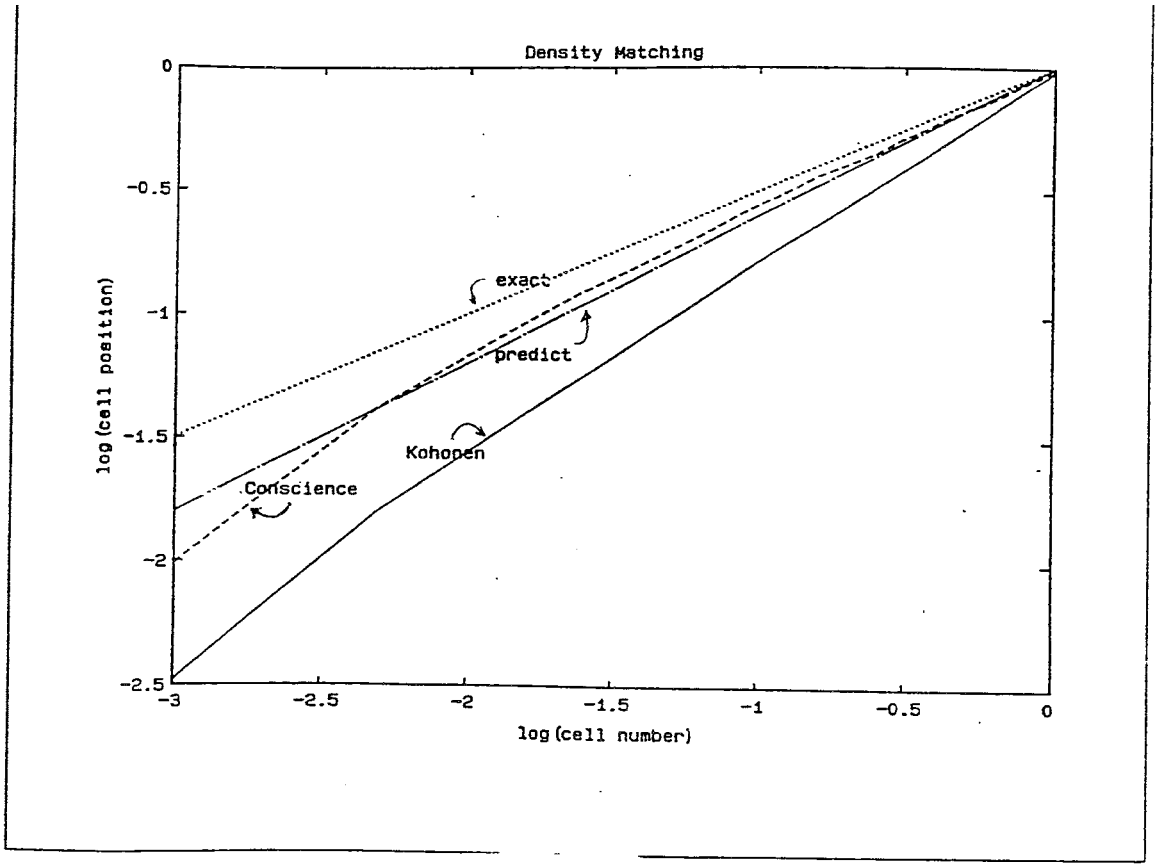

Figure 1: Problem 9.7

Problem 9.11

The results of computer simulation for a one-dimensional lattice with a two-dimensional (triangular) input are shown in Fig. 1 on the next page for an increasing number of iterations. The experiment begins with random weights at zero time, and then the neurons start spreading out.

Two distinct phases in the learning process can be recognized from this figure:

The neurons become ordered (i.e., the one-dimensional lattice becomes untangled), which happens at about 20 iterations.

The neurons spread out to match the density of the input distribution, culminating in the steady-state condition attained after 25,000 iterations.

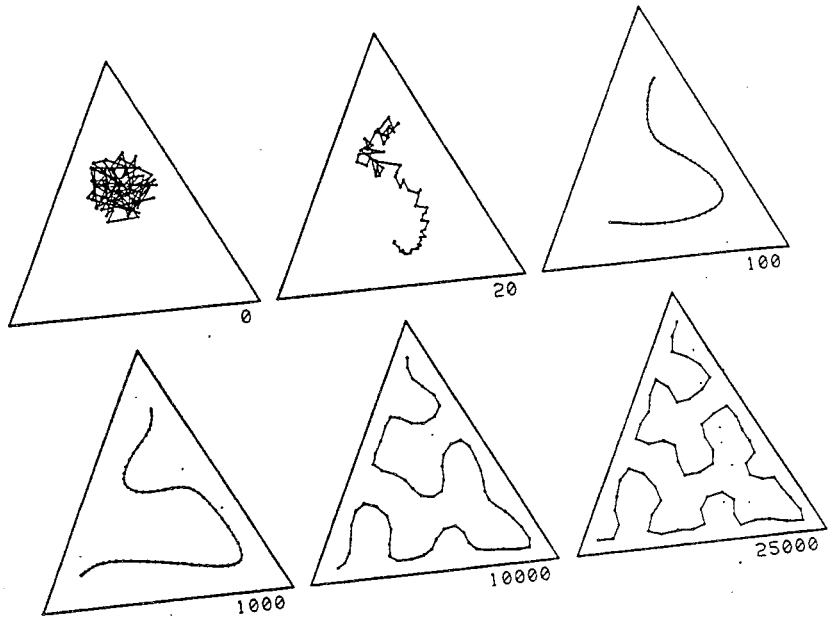

Figure 1: Problem 9.11