SOLUTIONS MANUAL

THIRD EDITION

Neural Networks and Learning Machines

Simon Haykin
and
Yanbo Xue
McMaster University
Canada

CHAPTER 1
Rosenblatt's Perceptron

Problem 1.1

(1) If $\mathbf{w}^T(n)\mathbf{x}(n) > 0$, then $y(n) = +1$.
If also $\mathbf{x}(n)$ belongs to C_1, then $d(n) = +1$.
Under these conditions, the error signal is

$$e(n) = d(n) - y(n) = 0$$

and from Eq. (1.22) of the text:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta e(n)\mathbf{x}(n) = \mathbf{w}(n)$$

This result is the same as line 1 of Eq. (1.5) of the text.

(2) If $\mathbf{w}^T(n)\mathbf{x}(n) < 0$, then $y(n) = -1$.
If also $\mathbf{x}(n)$ belongs to C_2, then $d(n) = -1$.
Under these conditions, the error signal $e(n)$ remains zero, and so from Eq. (1.22) we have

$$\mathbf{w}(n+1) = \mathbf{w}(n)$$

This result is the same as line 2 of Eq. (1.5).

(3) If $\mathbf{w}^T(n)\mathbf{x}(n) > 0$ and $\mathbf{x}(n)$ belongs to C_2 we have

$$y(n) = +1$$
$$d(n) = -1$$

The error signal $e(n)$ is -2, and so Eq. (1.22) yields

$$\mathbf{w}(n+1) = \mathbf{w}(n) - 2\eta\mathbf{x}(n)$$

which has the same form as the first line of Eq. (1.6), except for the scaling factor 2.

(4) Finally if $\mathbf{w}^T(n)\mathbf{x}(n) < 0$ and $\mathbf{x}(n)$ belongs to C_1, then

$$y(n) = -1$$
$$d(n) = +1$$

In this case, the use of Eq. (1.22) yields

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\eta\mathbf{x}(n)$$

which has the same mathematical form as line 2 of Eq. (1.6), except for the scaling factor 2.

Problem 1.2

The output signal is defined by

$$\begin{aligned} y &= \tanh\left(\frac{v}{2}\right) \\ &= \tanh\left(\frac{b}{2} + \frac{1}{2}\sum_i w_i x_i\right) \end{aligned}$$

Equivalently, we may write

$$b + \sum_i w_i x_i = y' \qquad (1)$$

where

$$y' = 2\tanh^{-1}(y)$$

Equation (1) is the equation of a hyperplane.

Problem 1.3

(a) AND operation: Truth Table 1

Inputs		Output
x_1	x_2	y
1	1	1
0	1	0
1	0	0
0	0	0

This operation may be realized using the perceptron of Fig. 1

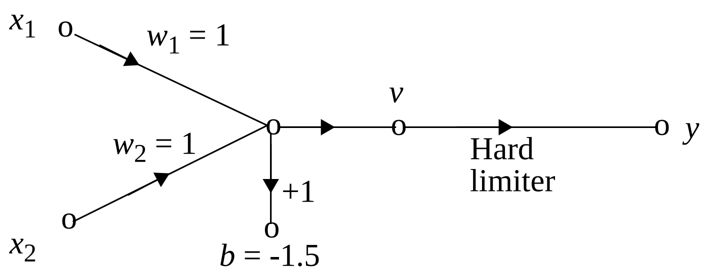


Figure 1: Problem 1.3

The hard limiter input is

$$\begin{aligned} v &= w_1 x_1 + w_2 x_2 + b \\ &= x_1 + x_2 - 1.5 \end{aligned}$$

If $x_1 = x_2 = 1$, then $v = 0.5$, and $y = 1$
If $x_1 = 0$, and $x_2 = 1$, then $v = -0.5$, and $y = 0$
If $x_1 = 1$, and $x_2 = 0$, then $v = -0.5$, and $y = 0$
If $x_1 = x_2 = 0$, then $v = -1.5$, and $y = 0$

These conditions agree with truth table 1.

OR operation: Truth Table 2

Inputs		Output
x_1	x_2	y
1	1	1
0	1	1
1	0	1
0	0	0

The OR operation may be realized using the perceptron of Fig. 2:

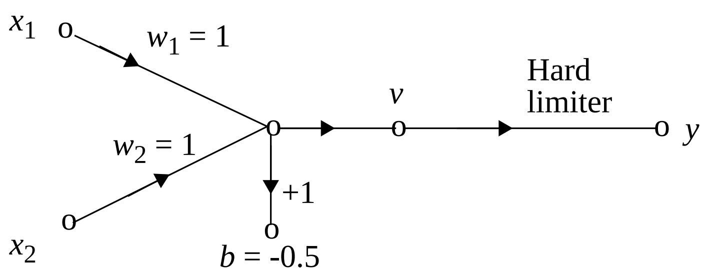


Figure 2: Problem 1.3

In this case, the hard limiter input is

$$v = x_1 + x_2 - 0.5$$

If $x_1 = x_2 = 1$, then $v = 1.5$, and $y = 1$
If $x_1 = 0$, and $x_2 = 1$, then $v = 0.5$, and $y = 1$
If $x_1 = 1$, and $x_2 = 0$, then $v = 0.5$, and $y = 1$
If $x_1 = x_2 = 0$, then $v = -0.5$, and $y = -1$

These conditions agree with truth table 2.

COMPLEMENT operation: Truth Table 3

Input x,	Output, y
1	0
0	1

The COMPLEMENT operation may be realized as in Figure 3::

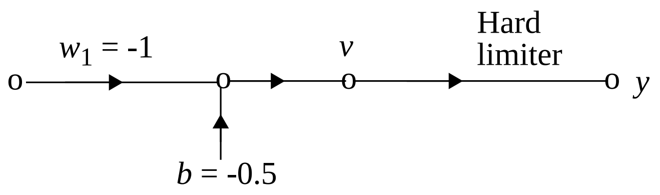


Figure 3: Problem 1.3

The hard limiter input is

$$v = wx + b = -x + 0.5$$

If $x = 1$, then $v = -0.5$, and $y = 0$
If $x = 0$, then $v = 0.5$, and $y = 1$

These conditions agree with truth table 3.

(b) EXCLUSIVE OR operation: Truth table 4

Inputs		Output
x_1	x_2	y
1	1	0
0	1	1
1	0	1
0	0	0

This operation is nonlinearly separable, which cannot be solved by the perceptron.

Problem 1.4

The Gaussian classifier consists of a single unit with a single weight and zero bias, determined in accordance with Eqs. (1.37) and (1.38) of the textbook, respectively, as follows:

$$\begin{aligned} w &= \frac{1}{\sigma^2}(\mu_1 - \mu_2) \\ &= -20 \end{aligned}$$

$$\begin{aligned} b &= \frac{1}{2\sigma^2}(\mu_2^2 - \mu_1^2) \\ &= 0 \end{aligned}$$

Problem 1.5

Using the condition

$$\mathbf{C} = \sigma^2\mathbf{I}$$

in Eqs. (1.37) and (1.38) of the textbook, we get the following formulas for the weight vector and bias of the Bayes classifier:

$$\mathbf{w} = \frac{1}{\sigma^2}(\mu_1 - \mu_2)$$

$$\mathbf{b} = \frac{1}{2\sigma^2}(\|\mu_1\|^2 - \|\mu_2\|^2)$$

CHAPTER 4
Multilayer Perceptrons

Problem 4.1

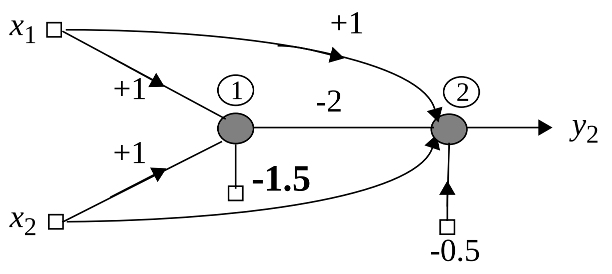


Figure 4: Problem 4.1

Assume that each neuron is represented by a McCulloch-Pitts model. Also assume that

$$x_i = \begin{cases} 1 & \text{if the input bit is 1} \\ 0 & \text{if the input bit is 0} \end{cases}$$

The induced local field of neuron 1 is

$$v_1 = x_1 + x_2 - 1.5$$

We may thus construct the following table:

x_1	0	0	1	1
x_2	0	1	0	1
v_1	-1.5	-0.5	-0.5	0.5
y_2	0	0	0	1

The induced local field of neuron 2 is

$$v_2 = x_1 + x_2 - 2y_1 - 0.5$$

Accordingly, we may construct the following table:

x_1	0	0	1	1
x_2	0	1	0	1
y_1	0	0	0	1
v_2	-0.5	0.5	-0.5	-0.5
y_2	0	1	1	1

From this table we observe that the overall output y_2 is 0 if x_1 and x_2 are both 0 or both 1, and it is 1 if x_1 is 0 and x_2 is 1 or vice versa. In other words, the network of Fig. P4.1 operates as an EXCLUSIVE OR gate.

Problem 4.2

Figure 1 shows the evolutions of the free parameters (synaptic weights and biases) of the neural network as the back-propagation learning process progresses. Each epoch corresponds to 100 iterations. From the figure, we see that the network reaches a steady state after about 25 epochs. Each neuron uses a logistic function for its sigmoid nonlinearity. Also the desired response is defined as

$$d = \begin{cases} 0.9 & \text{for symbol (bit) 1} \\ 0.1 & \text{for symbol (bit) 0} \end{cases}$$

Figure 2 shows the final form of the neural network. Note that we have used biases (the negative of thresholds) for the individual neurons.

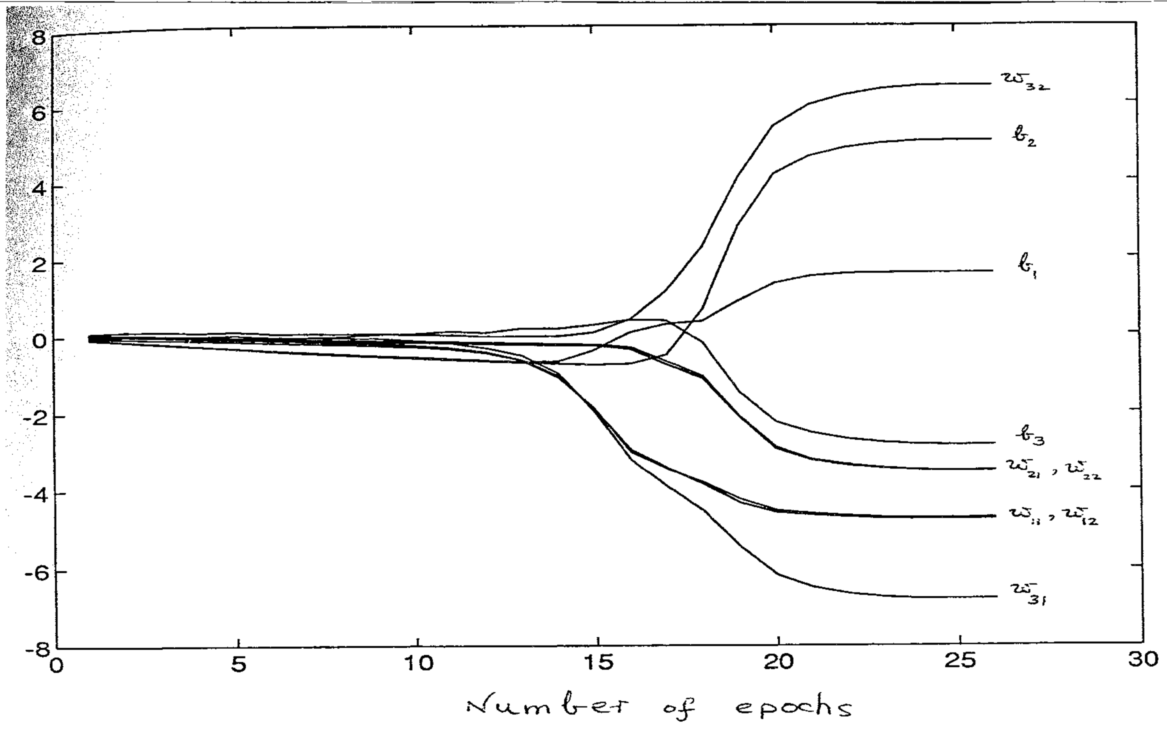


Figure 1: Problem 4.2, where one epoch = 100 iterations

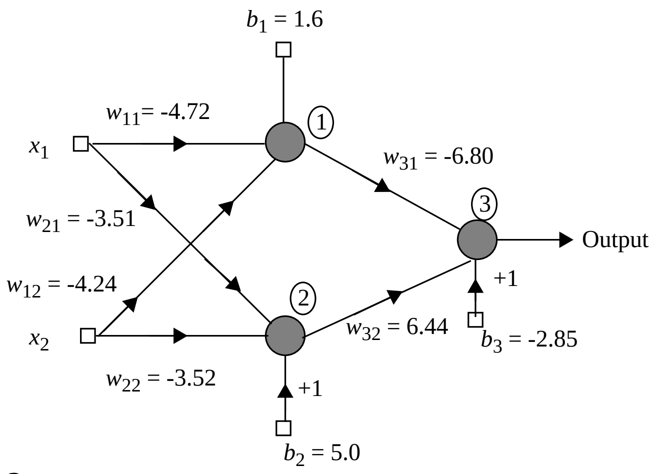


Figure 2: Problem 4.2

Problem 4.3

If the momentum constant α is negative, Equation (4.43) of the text becomes

$$\begin{aligned}\Delta w_{ji}(n) &= -\eta\sum_{t=0}^{n}\alpha^{n-t}\frac{\partial E(t)}{\partial w_{ji}(t)}\\ &= -\eta\sum_{t=0}^{n}(-1)^{n-t}|\alpha|^{n-t}\frac{\partial E(t)}{\partial w_{ji}(t)}\end{aligned}$$

Now we find that if the derivative $\partial E/\partial w_{ji}$ has the same algebraic sign on consecutive iterations of the algorithm, the magnitude of the exponentially weighted sum is reduced. The opposite is true when $\partial E/\partial w_{ji}$ alternates its algebraic sign on consecutive iterations. Thus, the effect of the momentum constant α is the same as before, except that the effects are reversed, compared to the case when α is positive.

Problem 4.4

From Eq. (4.43) of the text we have

$$\Delta w_{ji}(n) = -\eta\sum_{t=1}^{n}\alpha^{n-t}\frac{\partial E(t)}{\partial w_{ji}(t)} \qquad (1)$$

For the case of a single weight, the cost function is defined by

$$E = k_1(w-w_0)^2 + k_2$$

Hence, the application of (1) to this case yields

$$\Delta w(n) = -2k_1 \eta \sum_{t=1}^{n} \alpha^{n-t}(w(t) - w_0)$$

In this case, the partial derivative $\partial E(t)/\partial w(t)$ has the same algebraic sign on consecutive iterations. Hence, with $0 \leq \alpha < 1$ the exponentially weighted adjustment $\Delta w(n)$ to the weight w at time n grows in magnitude. That is, the weight w is adjusted by a large amount. The inclusion of the momentum constant α in the algorithm for computing the optimum weight $w^* = w_0$ tends to *accelerate* the downhill descent toward this optimum point.

Problem 4.5

Consider Fig. 4.14 of the text, which has an input layer, two hidden layers, and a single output neuron. We note the following:

$$y_1^{(3)} = F(A_1^{(3)}) = F(\mathbf{w}, \mathbf{x})$$

Hence, the derivative of $F(A_1^{(3)})$ with respect to the synaptic weight $w_{1k}^{(3)}$ connecting neuron k in the second hidden layer to the single output neuron is

$$\frac{\partial F(A_1^{(3)})}{\partial w_{1k}^{(3)}} = \frac{\partial F(A_1^{(3)})}{\partial y_1^{(3)}} \frac{\partial y_1^{(3)}}{\partial v_1^{(3)}} \frac{\partial v_1^{(3)}}{\partial w_{1k}^{(3)}} \tag{1}$$

where $v_1^{(3)}$ is the activation potential of the output neuron. Next, we note that

$$\frac{\partial F(A_1^{(3)})}{\partial y_1^{(3)}} = 1$$

$$y_1^{(3)} = \varphi(v_1^{(3)})$$

$$v_1^{(3)} = \sum_k w_{1k}^{(3)} y_k^{(2)} \tag{2}$$

where $y_k^{(2)}$ is the output of neuron k in layer 2. We may thus proceed further and write

$$\frac{\partial y_1^{(3)}}{\partial v_1^{(3)}} = \varphi'(v_1^{(3)}) = \varphi' A_1^{(3)} \tag{3}$$

$$\frac{\partial v_1^{(3)}}{\partial w_{1k}^{(3)}} = y_k^{(2)}$$

$$= \varphi(A_k^{(2)}) \tag{4}$$

Thus, combining (1) to (4):

$$\frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial w_{1k}^{(3)}} = \frac{\partial F(A_1^{(3)})}{\partial w_{1k}^{(3)}}$$

$$= \varphi'(A_1^{(3)})\varphi(A_k^{(3)})$$

Consider next the derivative of F(**w**,**x**) with respect to $w_{kj}^{(2)}$, the synaptic weight connecting neuron *j* in layer 1 (i.e., first hidden layer) to neuron k in layer 2 (i.e., second hidden layer):

$$\frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial w_{kj}^{(2)}} = \frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial y_1^{(3)}} \frac{\partial y_1^{(3)}}{\partial v_1^{(3)}} \frac{\partial v_1^{(3)}}{\partial y_k^{(2)}} \frac{\partial y_k^{(2)}}{\partial v_k^{(2)}} \frac{\partial v_k^{(2)}}{\partial w_{kj}^{(2)}} \tag{5}$$

where $y_k^{(2)}$ is the output of neuron in layer 2, and $v_k^{(1)}$ is the activation potential of that neuron. Next we note that

$$\frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial y_1^{(3)}} = 1 \tag{6}$$

$$\frac{\partial y_1^{(3)}}{\partial v_1^{(3)}} = \varphi'(A_1^{(3)}) \tag{7}$$

$$v_1^{(3)} = \sum_k w_{1k}^{(3)} y_k^{(2)}$$

$$\frac{\partial v_1^{(3)}}{\partial y_k^{(2)}} = w_{1k}^{(3)} \tag{8}$$

$$y_k^{(2)} = \varphi(v_k^{(2)})$$

$$\frac{\partial y_k^{(2)}}{\partial v_k^{(2)}} = \varphi'(v_k^{(2)}) = \varphi'(A_k^{(2)}) \tag{9}$$

$$v_k^{(2)} = \sum_j w_{kj}^{(1)} y_j^{(1)}$$

$$\frac{\partial v_k^{(2)}}{\partial w_{kj}^{(1)}} = y_j^{(1)} = \varphi(v_j^{(1)}) = \varphi(A_j^{(1)}) \tag{10}$$

Substituting (6) and (10) into (5), we get

$$\frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial w_{kj}^{(2)}} = \varphi'(A_1^{(3)}) w_{1k}^{(3)} \varphi'(A_k^{(2)}) \varphi(A_j^{(1)})$$

Finally, we consider the derivative of $F(\mathbf{w,x})$ with respect to $w_{ji}^{(1)}$, the synaptic weight connecting source node *i* in the input layer to neuron *j* in layer 1. We may thus write

$$\frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial w_{ji}^{(1)}} = \frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial y_1^{(3)}} \frac{\partial y_1^{(3)}}{\partial v_1^{(3)}} \frac{\partial v_1^{(3)}}{\partial y_j^{(1)}} \frac{\partial y_j^{(1)}}{\partial v_j^{(1)}} \frac{\partial v_j^{(1)}}{\partial w_{ji}^{(1)}} \tag{11}$$

where $y_j^{(1)}$ is the output of neuron *j* in layer 1, and $v_i^{(1)}$ is the activation potential of that neuron. Next we note that

$$\frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial y_1^{(3)}} = 1 \tag{12}$$

$$\frac{\partial y_1^{(3)}}{\partial v_1^{(3)}} = \varphi'(A^{(3)}) \tag{13}$$

$$v_1^{(3)} = \sum_k w_{1k}^{(3)} y_k^{(2)}$$

$$\begin{aligned}
\frac{\partial v_1^{(3)}}{\partial y_j^{(1)}} &= \sum_k w_{1k}^{(3)} \frac{\partial y_k^{(2)}}{\partial y_j^{(1)}} \\
&= \sum_k w_{1k}^{(3)} \frac{\partial y_k^{(2)}}{\partial v_k^{(2)}} \frac{\partial v_k^{(2)}}{\partial y_j^{(1)}} \\
&= \sum_k w_{1k}^{(3)} \varphi'(A_k^{(2)}) \frac{\partial v_k^{(2)}}{\partial y_j^{(1)}} \qquad (14)
\end{aligned}$$

$$\frac{\partial v_k^{(2)}}{\partial y_j^{(1)}} = w_{kj}^{(2)} \tag{15}$$

$$y_j^{(1)} = \varphi(v_j^{(1)})$$

$$\frac{\partial y_j^{(1)}}{\partial v_j^{(1)}} = \varphi'(v_j^{(1)}) = \varphi'(A_j^{(1)}) \tag{16}$$

$$v_j^{(1)} = \sum_i w_{ji}^{(1)} x_i$$

$$\frac{\partial v_j^{(1)}}{\partial w_{ji}^{(1)}} = x_i \tag{17}$$

Substituting (12) to (17) into (11) yields

$$\frac{\partial F(\mathbf{w}, \mathbf{x})}{\partial w_{ji}^{(1)}} = \varphi'(A_1^{(3)})\left(\sum_k w_{1k}^{(3)} \varphi'(A_k^{(2)}) w_{kj}^{(2)}\right)\varphi'(A_j^{(1)}) x_i$$

Problem 4.12

According to the conjugate-gradient method, we have

$$\begin{aligned}\Delta\mathbf{w}(n) &= \eta(n)\mathbf{p}(n)\\ &= \eta(n)[-\mathbf{g}(n) + \beta(n-1)\mathbf{p}(n-1)]\\ &\approx -\eta(n)\mathbf{g}(n) + \beta(n-1)\eta(n-1)\mathbf{p}(n-1)\end{aligned} \tag{1}$$

where, in the second term of the last line in (1), we have used $\eta(n - 1)$ in place of $\eta(\eta)$. Define

$$\Delta\mathbf{w}(n-1) = \eta(n-1)\mathbf{p}(n-1)$$

We may then rewrite (1) as

$$\Delta\mathbf{w}(n) \approx -\eta(n)\mathbf{g}(n) + \beta(n-1)\Delta\mathbf{w}(n-1) \tag{2}$$

On the other hand, according to the generalized delta rule, we have for neuron *j*:

$$\Delta\mathbf{w}_j(n) = \alpha\Delta\mathbf{w}_j(n-1) + \eta\delta_j(n)\mathbf{y}(n) \tag{3}$$

Comparing (2) and (3), we observe that they have a similar mathematical form:

- The vector **-g**(n) in the conjugate gradient method plays the role of $\delta_j(n)\mathbf{y}(n)$, where $\delta_j(n)$ is the local gradient of neuron j and **y**(n) is the vector of inputs for neuron j.
- The time-varying parameter $\beta(n - 1)$ in the conjugate-gradient method plays the role of momentum α in the generalized delta rule.

Problem 4.13

We start with (4.127) in the text:

$$\beta(n) = -\frac{\mathbf{s}^T(n-1)\mathbf{A}\mathbf{r}(n)}{\mathbf{s}^T(n-1)\mathbf{A}\mathbf{s}(n-1)} \tag{1}$$

The residual **r**(n) is governed by the recursion:

$$\mathbf{r}(n) = \mathbf{r}(n-1) - \eta(n-1)\mathbf{A}\mathbf{s}(n-1)$$

Equivalently, we may write

$$-\eta(n-1)\mathbf{A}\mathbf{s}(n-1) = \mathbf{r}(n) - \mathbf{r}(n-1) \tag{2}$$

Hence multiplying both sides of (2) by $\mathbf{s}^T(n - 1)$, we obtain

$$\begin{aligned}\eta(n-1)\mathbf{s}^T(n-1)\mathbf{A}\mathbf{s}(n-1) &= -\mathbf{s}^T(n-1)(\mathbf{r}(n) - \mathbf{r}(n-1)) \\ &= \mathbf{s}^T(n-1)\mathbf{r}(n-1)\end{aligned} \tag{3}$$

where it is noted that (by definition)

$$\mathbf{s}^T(n-1)\mathbf{r}(n) = 0$$

Moreover, multiplying both sides of (2) by $\mathbf{r}^T(n)$, we obtain

$$\begin{aligned}-\eta(n-1)\mathbf{r}^T(n)\mathbf{A}\mathbf{s}(n-1) &= -\eta(n-1)\mathbf{s}^T(n-1)\mathbf{A}\mathbf{r}(n-1) \\ &= \mathbf{r}^T(n)(\mathbf{r}(n) - \mathbf{r}(n-1))\end{aligned} \tag{4}$$

where it is noted that $\mathbf{A}^T = \mathbf{A}$. Dividing (4) by (3) and invoking the use of (1):

$$\beta(n) = \frac{\mathbf{r}^T(n)(\mathbf{r}(n) - \mathbf{r}(n-1))}{\mathbf{s}^T(n-1)\mathbf{r}(n-1)} \tag{5}$$

which is the Hesteness-Stiefel formula.

In the linear form of conjugate gradient method, we have

$$\mathbf{s}^T(n-1)\mathbf{r}(n-1) = \mathbf{r}^T(n-1)\mathbf{r}(n-1)$$

in which case (5) is modified to

$$\beta(n) = \frac{\mathbf{r}^T(n)(\mathbf{r}(n)-\mathbf{r}(n-1))}{\mathbf{r}^T(n-1)\mathbf{r}(n-1)} \tag{6}$$

which is the Polak-Ribiére formula. Moreover, in the linear case we have

$$\mathbf{r}^T(n)\mathbf{r}(n-1) = 0$$

in which case (6) reduces to the Fletcher-Reeves formula:

$$\beta(n) = \frac{\mathbf{r}^T(n)\mathbf{r}(n)}{\mathbf{r}^T(n-1)\mathbf{r}(n-1)}$$

Problem 4.15

In this problem, we explore the operation of a fully connected multilayer perceptron trained with the back-propagation algorithm. The network has a single hidden layer. It is trained to realize the following one-to-one mappings:

(a) Inversion:

$$f(x) = \frac{1}{x}, \qquad 1 \le x \le 100$$

(b) Logarithmic computation

$$f(x) = \log_{10} x, \qquad 1 \le x \le 10$$

(c) Exponentiation

$$f(x) = e^{-x}, \qquad 1 \le x \le 10$$

(d) Sinusoidal computation

$$f(x) = \sin x, \qquad 0 \le x \le \frac{\pi}{2}$$

(a) $f(x) = 1/x$ for $1 \le x \le 100$
The network is trained with:

learning-rate parameter $\eta = 0.3$, and
momentum constant $\alpha = 0.7$.

Ten different network configurations were trained to learn this mapping. Each network was trained identically, that is, with the same η and α, with bias terms, and with 10,000 passes of the training vectors (with one exception noted below). Once each network was trained, the test dataset was applied to compare the performance and accuracy of each configuration. Table 1 summarizes the results obtained:

Table 1

Number of hidden neurons	Average percentage error at the network output
3	4.73%
4	4.43
5	3.59
7	1.49
10	1.12
15	0.93
20	0.85
30	0.94
100	0.9
30 (trained with 100,000 passes)	0.19

The results of Table 1 indicate that even with a small number of hidden neurons, and with a relatively small number of training passes, the network is able to learn the mapping described in (a) quite well.

(b) $f(x) = \log_{10}x$ for $1 \leq x \leq 10$
The results of this second experiment are presented in Table 2:

Table 2

Number of hidden neurons	Average percentage error at the network output
2	2.55%
3	2.09
4	0.46
5	0.48
7	0.85
10	0.42
15	0.85
20	0.96
30	1.26
100	1.18
30 (trained with 100,000 passes)	0.41

Here again, we see that the network performs well even with a small number of hidden neurons. Interestingly, in this second experiment the network peaked in accuracy with 10 hidden neurons, after which the accuracy of the network to produce the correct output started to decrease.

(c) $f(x) = e^{-x}$ for $1 \leq x \leq 10$
The results of this third experiment (using the logistic function as with experiments (a)

and (b)), are summarized in Table 3:

Table 3

Number of hidden neurons	Average percentage error at the network output
2	244.0‘%
3	185.17
4	134.85
5	133.67
7	141.65
10	158.77
15	151.91
20	144.79
30	137.35
100	98.09
30 (trained with 100,000 passes)	103.99

These results are unacceptable since the network is unable to generalize when each neuron is driven to its limits.

The experiment with 30 hidden neurons and 100,000 training passes was repeated, but this time the hyperbolic tangent function was used as the nonlinearity. The result obtained this time was an average percentage error of 3.87% at the network output. This last result shows that the hyperbolic tangent function is a better choice than the logistic function as the sigmoid function for realizing the mapping $f(x) = e^{-x}$.

(d) $f(x) = \sin x$ for $0 \leq x \leq \pi/2$

Finally, the following results were obtained using the logistic function with 10,000 training passes, except for the last configuration:

Table 4

Number of hidden neurons	Average percentage error at the network output
2	1.63‘%
3	1.25
4	1.18
5	1.11
7	1.07
10	1.01
15	1.01
20	0.72
30	1.21
100	3.19
30 (trained with 100,000 passes)	0.4

The results of Table 4 show that the accuracy of the network peaks around 20 neurons, where after the accuracy decreases.

CHAPTER 6
Support Vector Machines

Problem 6.1

From Eqs. (6.2) in the text we recall that the optimum weight vector $\mathbf{w}_o$ and optimum bias b_o satisfy the following pair of conditions:

$$\mathbf{w}_o^T\mathbf{x}_i + b_o \geq +1 \qquad \text{for } d_i = +1$$
$$\mathbf{w}_o^T\mathbf{x}_i + b_o < -1 \qquad \text{for } d_i = -1$$

where $i = 1, 2, ...,N$. Equivalently, we may write

$$\min_{i = 1, 2, \dots, N} \left|\mathbf{w}^T\mathbf{x}_i + b\right| = 1$$

as the defining condition for the pair ($\mathbf{w}_o$, b_o).

Problem 6.2

In the context of a support vector machine, we note the following:

1. Misclassification of patterns can only arise if the patterns are nonseparable.
2. If the patterns are nonseparable, it is possible for a pattern to lie inside the margin of separation and yet be on the correct side of the decision boundary. Hence, nonseparability does not necessarily mean misclassification.

Problem 6.3

We start with the primel problem formulated as follows (see Eq. (6.15)) of the text

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha_i d_i \mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{N}\alpha_i d_i + \sum_{i=1}^{N}\alpha_i \qquad (1)$$

Recall from (6.12) in the text that

$$\mathbf{w} = \sum_{i=1}^{N}\alpha_i d_i \mathbf{x}$$

Premultiplying $\mathbf{w}$ by $\mathbf{w}^T$:

$$\mathbf{w}^T\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \mathbf{w}^T \mathbf{x}_i \tag{2}$$

We may also write

$$\mathbf{w}^T = \sum_{i=1}^{N} \alpha_i d_i \mathbf{x}_i^T$$

Accordingly, we may redefine the inner product $\mathbf{w}^T\mathbf{w}$ as the double summation:

$$\mathbf{w}^T\mathbf{w} = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i d_i \alpha_j d_j \mathbf{x}_j^T \mathbf{x}_i \tag{3}$$

Thus substituting (2) and (3) into (1) yields

$$Q(\alpha) = -\frac{1}{2}\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i d_i \alpha_j d_j \mathbf{x}_j^T \mathbf{x}_i + \sum_{i=1}^{N} \alpha_i \tag{4}$$

subject to the constraint

$$\sum_{i=1}^{N} \alpha_i d_i = 0$$

Recognizing that $\alpha_i > 0$ for all i, we see that (4) is the formulation of the dual problem.

Problem 6.4

Consider a support vector machine designed for nonseparable patterns. Assuming the use of the "leave-one-out-method" for training the machine, the following situations may arise when the example left out is used as a test example:

1. The example is a support vector.
   Result: Correct classification.
2. The example lies inside the margin of separation but on the correct side of the decision boundary.
   Result: Correct classification.
3. The example lies inside the margin of separation but on the wrong side of the decision boundary.
   Result: Incorrect classification.

Problem 6.5

By definition, a support vector machine is designed to maximize the margin of separation between the examples drawn from different classes. This definition applies to all sources of data, be they noisy or otherwise. It follows therefore that by the very nature of it, the support vector machine is robust to the presence of additive noise in the data used for training and testing, provided that all the data are drawn from the same population.

Problem 6.6

Since theGram $\mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}$ is a square matrix, it can be diagonalized using the similarity transformation:

$$\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T$$

where Λ is a diagonal matrix consisting of the eigenvalues of **K** and **Q** is an orthogonal matrix whose columns are the associated eigenvectors. With **K** being a positive matrix, Λ has nonnegative entries. The inner-product (i.e., Mercer) kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ is the *ij*th element of matrix **K**. Hence,

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{Q}\Lambda\mathbf{Q}^T)_{ij} \\ &= \sum_{l=1}^{m_1}(\mathbf{Q})_{il}(\Lambda)_{ll}(\mathbf{Q}^T)_{lj} \\ &= \sum_{l=1}^{m_1}(\mathbf{Q})_{il}(\Lambda)_{ll}(\mathbf{Q})_{lj} \qquad (1) \end{aligned}$$

Let $\mathbf{u}_i$ denote the *i*th row of matrix **Q**. (Note that $\mathbf{u}_i$ is *not* an eigenvector.) We may then rewrite (1) as the inner product

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &= \mathbf{u}_i^T\Lambda\mathbf{u}_j \\ &= (\Lambda^{1/2}\mathbf{u}_i)^T(\Lambda^{1/2}\mathbf{u}_j) \qquad (2) \end{aligned}$$

where $\Lambda^{1/2}$ is the square root of Λ.

By definition, we have

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi^T(\mathbf{x}_i)\varphi(\mathbf{x}_j) \qquad (3)$$

Comparing (2) and (3), we deduce that the mapping from the input space to the hidden (feature) space of a support vector machine is described by

$$\varphi: \ \mathbf{x}_i \rightarrow \Lambda^{1/2}\mathbf{u}_i$$

Problem 6.7

(a) From the solution to Problem 6.6, we have

$$\phi: \ \mathbf{x}_i \rightarrow \Lambda^{1/2}\mathbf{u}_i$$

Suppose the input vector $\mathbf{x}_i$ is multiplied by the orthogonal (unitary) matrix $\mathbf{Q}$. We then have a new mapping ϕ' described by

$$\phi': \ \mathbf{Q}\mathbf{x}_i \rightarrow \mathbf{Q}\Lambda^{1/2}\mathbf{u}_i$$

Correspondingly, we may write

$$\begin{aligned} k(\mathbf{Q}\mathbf{x}_i, \mathbf{Q}\mathbf{x}_j) &= (\mathbf{Q}\Lambda^{1/2}\mathbf{u}_i)^T(\mathbf{Q}\Lambda^{1/2}\mathbf{u}_j) \\ &= (\Lambda^{1/2}\mathbf{u}_i)^T\mathbf{Q}^T\mathbf{Q}(\Lambda^{1/2}\mathbf{u}_j) \end{aligned} \qquad (1)$$

where $\mathbf{u}_i$ is the *i*th row of $\mathbf{Q}$. From the definition of an orthogonal (unitary) matrix:

$$\mathbf{Q}^{-1} = \mathbf{Q}^T$$

or equivalently

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix. Hence, (1) reduces to

$$\begin{aligned} k(\mathbf{Q}\mathbf{x}_i, \mathbf{Q}\mathbf{x}_j) &= (\Lambda^{1/2}\mathbf{u}_i)^T(\Lambda^{1/2}\mathbf{u}_j) \\ &= k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned}$$

In words, the Mercer kernel exhibits the *unitary* invariance property.

(b) Consider first the polynomial machine described by

$$
\begin{aligned}
k(\mathbf{Qx}_i, \mathbf{Qx}_j) &= ((\mathbf{Qx}_i)^T(\mathbf{Qx}_j) + 1)^p \\
&= (\mathbf{x}_i^T\mathbf{Q}^T\mathbf{Qx}_j + 1)^p \\
&= (\mathbf{x}_i^T\mathbf{x}_j + 1)^p \\
&= k(\mathbf{x}_i, \mathbf{x}_J)
\end{aligned}
$$

Consider next the RBF network described by the Mercer kernel:

$$
\begin{aligned}
k(\mathbf{Qx}_i, \mathbf{Qx}_j) &= \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{Qx}_i - \mathbf{Qx}_j\|^2\right) \\
&= \exp\left(-\frac{1}{2\sigma^2}(\mathbf{Qx}_i - \mathbf{Qx}_j)^T(\mathbf{Qx}_i - \mathbf{Qx}_j)\right) \\
&= \exp\left(-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^T\mathbf{Q}^T\mathbf{Q}(\mathbf{x}_i - \mathbf{x}_j)\right) \\
&= \exp\left(-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)\right), \qquad \mathbf{Q}^T\mathbf{Q} = \mathbf{I} \\
&= k(\mathbf{x}_i, \mathbf{x}_J)
\end{aligned}
$$

Finally, consider the multilayer perceptron described by

$$
\begin{aligned}
k(\mathbf{Qx}_i, \mathbf{Qx}_j) &= \tanh(\beta_0(\mathbf{Qx}_i)^T(\mathbf{Qx}_j) + \beta_1) \\
&= \tanh(\beta_0\mathbf{x}_i^T\mathbf{Q}^T\mathbf{Qx}_j + \beta_1) \\
&= \tanh(\beta_0\mathbf{x}_i^T\mathbf{x}_j + \beta_1) \\
&= k(\mathbf{x}_i, \mathbf{x}_J)
\end{aligned}
$$

Thus all three types of the support vector machine, namely, the polynomial machine, RBF network, and MLP, satisfy the unitary invariance property in their own individual ways.

Problem 6.17

The truth table for the XOR function, operating on a three-dimensional pattern x, is as follows:

Table 1

Inputs			Desired response
x_1	x_2	x_3	y
+1	+1	+1	+1
+1	-1	+1	-1
-1	+1	+1	-1
+1	+1	-1	-1
+1	-1	-1	+1
-1	+1	-1	+1
-1	-1	-1	-1
-1	-1	+1	+1

To proceed with the support vector machine for solving this multidimensional XOR problem, let the Mercer kernel

$$k(\mathbf{x}, \mathbf{x}_j) = (1 + \mathbf{x}^T\mathbf{x}_i)^p$$

The minimum value of power p (denoting a positive integer) needed for this problem is $p = 3$. For $p = 2$, we end up with a zero weight vector, which is clearly unacceptable.

Setting $p = 3$, we thus have

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}_i) &= (1 + \mathbf{x}^T\mathbf{x}_i)^3 \\ &= 1 + 3\mathbf{x}^T\mathbf{x}_i + 3(\mathbf{x}^T\mathbf{x}_i)^2 + (\mathbf{x}^T\mathbf{x}_i)^3 \end{aligned}$$

where

$$\mathbf{x} = [x_1, x_2, x_3]^T$$

and likewise for $\mathbf{x}_i$. Then, proceeding in a manner similar but much more cumbersome than that described for the two-dimensional XOR problem in Section 6.6, we end up with a polynomial machine defined by

$$y = x_1, x_2, x_3$$

This machine satisfies the entries of Table 1.

CHAPTER 8
Principal-Components Analysis

Problem 8.5

From Example 8.2 in the text:

$$\lambda_0 = 1 + \sigma^2 \tag{1}$$

$$\mathbf{q}_0 = \mathbf{s} \tag{2}$$

The correlation matrix of the input is

$$\mathbf{R} = \mathbf{s}\mathbf{s}^T + \sigma^2\mathbf{I} \tag{3}$$

where **s** is the signal vector and σ^2 is the variance of an element of the additive noise vector. Hence, using (2) and (3):

$$\begin{aligned}\lambda_0 &= \frac{\mathbf{q}_0^T\mathbf{R}\mathbf{q}_0}{\mathbf{q}_0^T\mathbf{q}_0} \\ &= \frac{\mathbf{s}^T(\mathbf{s}\mathbf{s}^T + \sigma^2\mathbf{I})\mathbf{s}}{\mathbf{s}^T\mathbf{s}} \\ &= \frac{(\mathbf{s}^T\mathbf{s})(\mathbf{s}^T\mathbf{s}) + \sigma^2(\mathbf{s}^T\mathbf{s})}{\mathbf{s}^T\mathbf{s}} \\ &= \mathbf{s}^T\mathbf{s} + \sigma^2 \\ &= \|\mathbf{s}\|^2 + \sigma^2 \end{aligned} \tag{4}$$

The vector **s** is a signal vector of unit length:

$$\|\mathbf{s}\| = 1$$

Hence, (4) simplifies to

$$\lambda_0 = 1 + \sigma^2$$

which is the desired result given in (1).

Problem 8.6

From (8.46) in the text we have

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta y(n)[\mathbf{x}(n) - y(n)\mathbf{w}(n)] \tag{1}$$

As $n \to \infty$, $\mathbf{w}(n) \to \mathbf{q}_1$, and so we deduce from (1) that

$$\mathbf{x}(n) = y(n)\mathbf{q}_1 \qquad \text{for } n \to \infty \tag{2}$$

where $\mathbf{q}_1$ is the eigenvector associated with the largest eigenvalue λ_1 of the correlation matrix $\mathbf{R} = \mathbf{E}[\mathbf{x}(n)\mathbf{x}^T(n)]$, where $\mathbf{E}$ is the expectation operator. Multiplying (2) by its own transpose and then taking expectations, we get

$$\mathbf{E}[\mathbf{x}(n)\mathbf{x}^T(n)] = \mathbf{E}[y^2(n)]\mathbf{q}_1\mathbf{q}_1^T$$

Equivalently, we may write

$$\mathbf{R} = \sigma_Y^2\mathbf{q}_1\mathbf{q}_1^T \tag{3}$$

where σ_Y^2 is the variance of the output $y(n)$. Post-multiplying (3) by $\mathbf{q}_1$:

$$\mathbf{R}\mathbf{q}_1 = \sigma_Y^2\mathbf{q}_1\mathbf{q}_1^T\mathbf{q}_1 = \sigma_Y^2\mathbf{q}_1 \tag{4}$$

where it is noted that $\|\mathbf{q}_1\| = 1$ by definition. From (4) we readily see that $\sigma_Y^2 = \lambda_1$, which is the desired result.

Problem 8.7

Writing the learning algorithm for minor components analysis in matrix form:

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

Proceeding in a manner similar to that described in Section (8.5) of the textbook, we have the nonlinear differential equation:

$$\frac{d}{dt}\mathbf{w}(t) = [\mathbf{w}^T(t)\mathbf{R}\mathbf{w}(t)]\mathbf{w}(t) - \mathbf{R}\mathbf{w}(t)$$

Define

$$\mathbf{w}(t) = \sum_{k=1}^{M} \theta_k(t)\mathbf{q}_k \tag{1}$$

where $\mathbf{q}_k$ is the kth eigenvector of correlation matrix $\mathbf{R} = \mathbf{E}[\mathbf{x}(n)\mathbf{x}^T(n)]$ and the coefficient $\theta_k(t)$ is the projection of $\mathbf{w}(t)$ onto $\mathbf{q}_k$. We may then identify two cases as summarized here:

***Case* I:** $1 \le k < m$

For this first case, we define

$$\alpha_k(t) = \frac{\theta_k(t)}{\theta_m(t)} \qquad \text{for some fixed } m \tag{2}$$

Accordingly, we find that

$$\frac{d\alpha_k(t)}{dt} = -(\lambda_m - \lambda_k)\alpha_k(t) \tag{3}$$

With the eigenvalues of $\mathbf{R}$ arranged in decreasing order:

$\lambda_1 > \lambda_2 > \dots > \lambda_k > \dots > \lambda_m > 0$

it follows that $\alpha_k(t) \to 0$ as $t \to \infty$.

***Case* II:** $k = m$

For this second case, we find that

$$\frac{d\theta_m(t)}{dt} = \lambda_m\theta_m(t)(\theta_m^2(t) - 1) \text{ for } t \to \infty \tag{4}$$

Hence, $\theta_m(t) = \pm 1$ as $t \to \infty$.

Thus, in light of the results derived for cases I and II, we deduce from (1) that:

$\mathbf{w}(t) \to \mathbf{q}_m$ = eigenvector associated with the smallest eigenvalue λ_m as $t \to \infty$, and $\sigma_Y^2 = \mathbf{E}[y^2(n)] \to \lambda_m$.

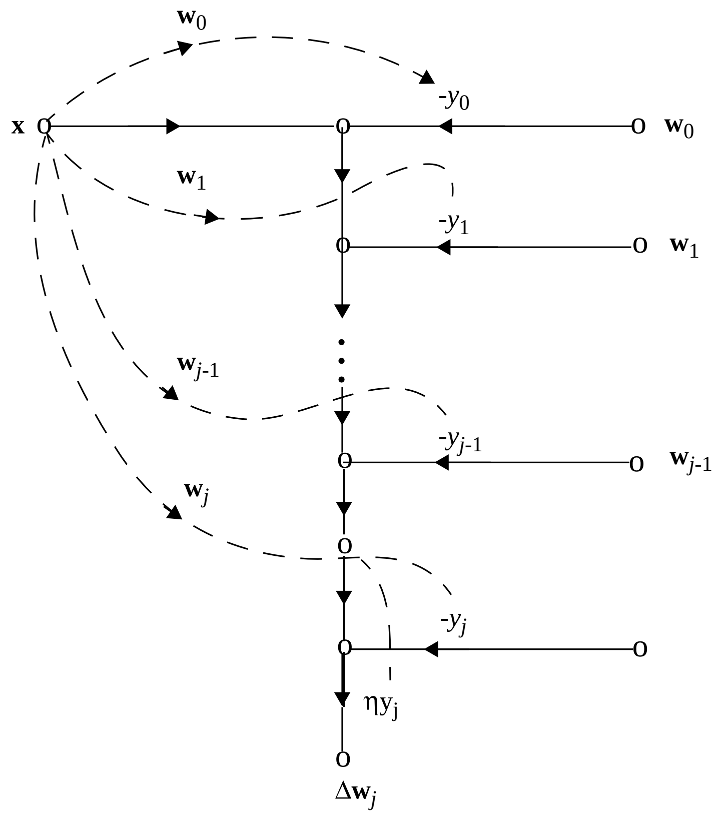
Problem 8.8

From (8.87) and (8.88) of the text:

$$\Delta \mathbf{w}_j = \eta y_j \mathbf{x}' - \eta y_j^2 \mathbf{w}_j \qquad (1)$$

$$\mathbf{x}' = \mathbf{x} - \sum_{k=0}^{j-1} \mathbf{w}_k y_k \qquad (2)$$

where, for convenience of presentation, we have omitted the dependence on time *n*. Equations (1) and (2) may be represented by the following vector-valued signal flow graph:



Note: The dashed lines indicate inner (dot) products formed by the input vector **x** and the pertinent synaptic weight vectors $\mathbf{w}_0, \mathbf{w}_1, ..., \mathbf{w}_j$ to produce $y_0, y_1, ..., y_j$, respectively.

Problem 8.9

Consider a network consisting of a single layer of neurons with feedforward connections. The algorithm for adjusting the matrix of synaptic weights $\mathbf{W}(n)$ of the network is described by the recursive equation (see Eq. (8.91) of the text):

$$\mathbf{W}(n) = \mathbf{W}(n) + \eta(n)\{\mathbf{y}(n)\mathbf{x}^T(n) - LT[\mathbf{y}(n)\mathbf{y}^T(n)]\mathbf{W}(n)\} \qquad (1)$$

where $\mathbf{x}(n)$ is the input vector, $\mathbf{y}(n)$ is the output vector; and $LT[.]$ is a matrix operator that sets all the elements above the diagonal of the matrix argument to zero, thereby making it lower triangular.

First, we note that the asymptotic stability theorem discussed in the text does not apply directly to the convergence analysis of stochastic approximation algorithms involving matrices; it is formulated to apply to vectors. However, we may write the elements of the parameter (synaptic weight) matrix $\mathbf{W}(n)$ in (1) as a vector, that is, one column vector stacked up on top of another. We may then interpret the resulting nonlinear update equation in a corresponding way and so proceed to apply the asymptotic stability theorem directly.

To prove the convergence of the learning algorithm described in (1), we may use the *method of induction* to show that if the first j columns of matrix $\mathbf{W}(n)$ converge to the first j eigenvectors of the correlation matrix $\mathbf{R} = \boldsymbol{E}[\mathbf{x}(n)\mathbf{x}^T(n)]$, then the $(j + 1)$th column will converge to the $(j + 1)$th eigenvector of $\mathbf{R}$. Here we use the fact that in light of the convergence of the maximum eigenfilter involving a single neuron, the first column of the matrix $\mathbf{W}(n)$ converges with probability 1 to the first eigenvector of $\mathbf{R}$, and so on.

Problem 8.10

The results of a computer experiment on the training of a single-layer feedforward network using the generalized Hebbian algorithm are described by Sanger (1990). The network has 16 output neurons, and 4096 inputs arranged as a 64 x 64 grid of pixels. The training involved presentation of 2000 samples, which are produced by low-pass filtering a white Gaussian noise image and then multiplying wi6th a Gaussian window function. The low-pass filter was a Gaussian function with standard deviation of 2 pixels, and the window had a standard deviation of 8 pixels.

Figure 1, presented on the next page, shows the first 16 receptive field masks learned by the network (Sanger, 1990). In this figure, positive weights are indicated by “white” and negative weights are indicated by “black”; the ordering is left-to-right and top-to-bottom.

The results displayed in Fig. 1 are rationalized as follows (Sanger, 1990):

- The first mask is a low-pass filter since the input has most of its energy near dc (zero frequency).
- The second mask cannot be a low-pass filter, so it must be a band-pass filter with a mid-band frequency as small as possible since the input power decreases with increasing frequency.
- Continuing the analysis in the manner described above, the frequency response of successive masks approaches dc as closely as possible, subject (of course) to being orthogonal to previous masks.

The end result is a sequence of orthogonal masks that respond to progressively higher frequencies.

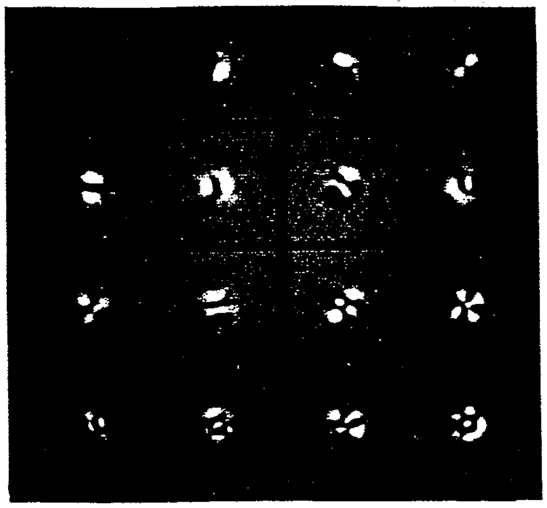


Figure 1: Problem 8.10 (Reproduced with permission of Biological Cybernetics)

CHAPTER 9
Self-Organizing Maps

Problem 9.1

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

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

where

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

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

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

Correspondingly, we may write

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

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)

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

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.

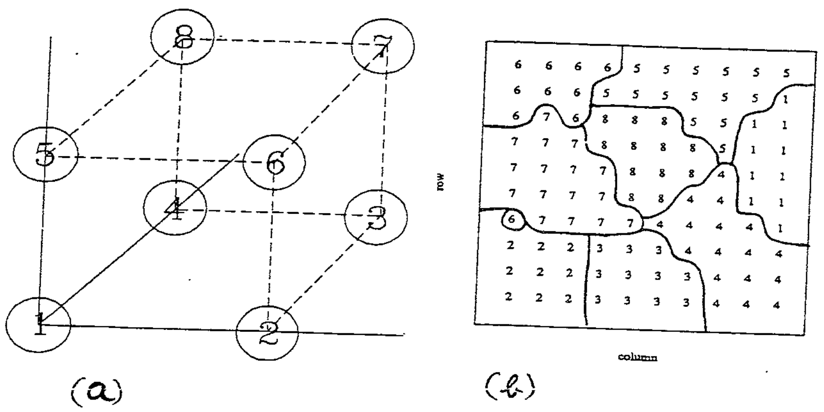


Figure 1: Problem 9.3

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.

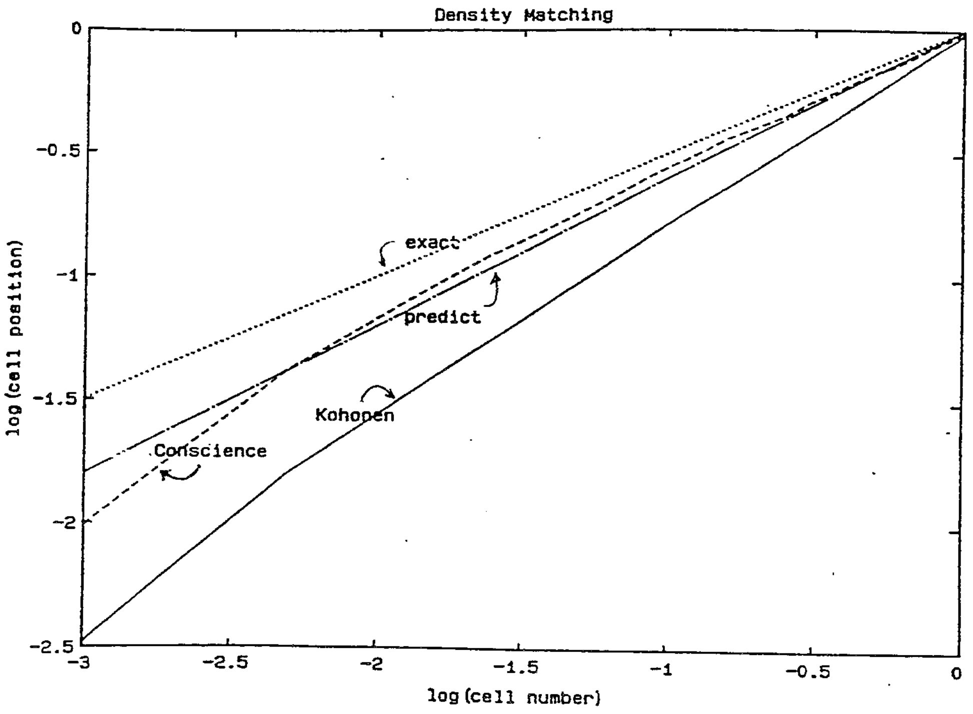


Figure 1: Problem 9.7

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.

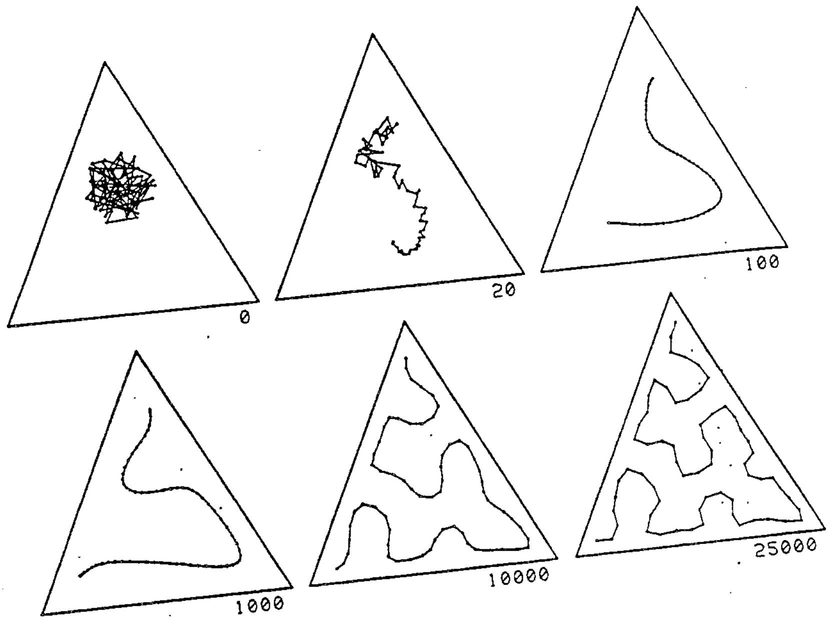


Figure 1: Problem 9.11