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

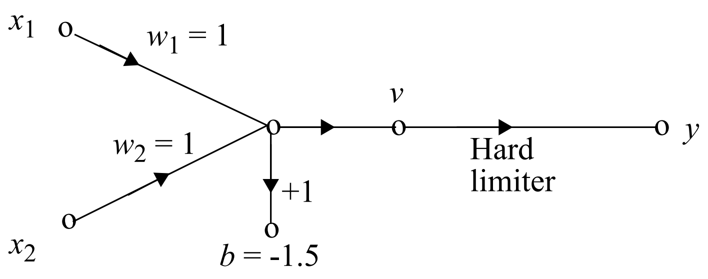


Figure 1: Problem 1.3

The hard limiter input is

$$\begin{aligned} v &= w_1x_1 + w_2x_2 + b \\ &= x_1 + x_2 - 1.5 \end{aligned}$$

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:

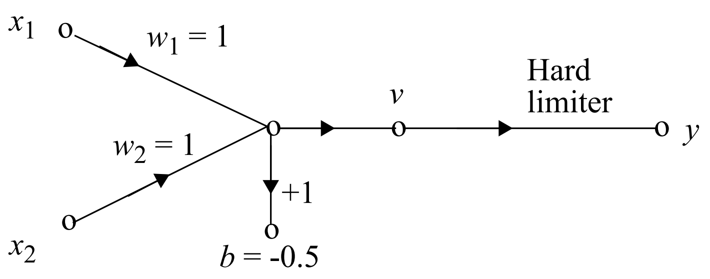


Figure 2: Problem 1.3

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::

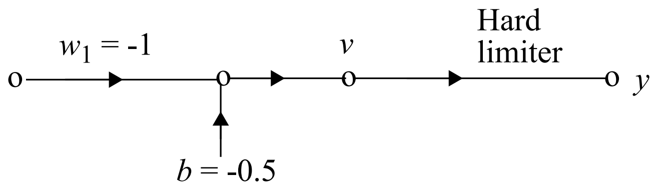


Figure 3: Problem 1.3

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:

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

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

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

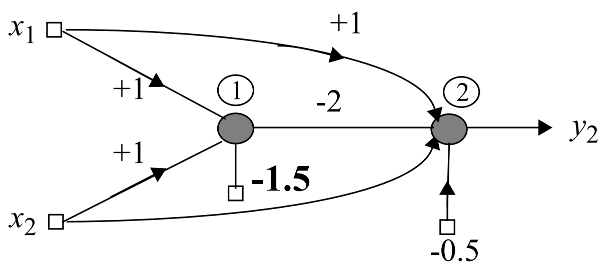


Figure 4: Problem 4.1

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.

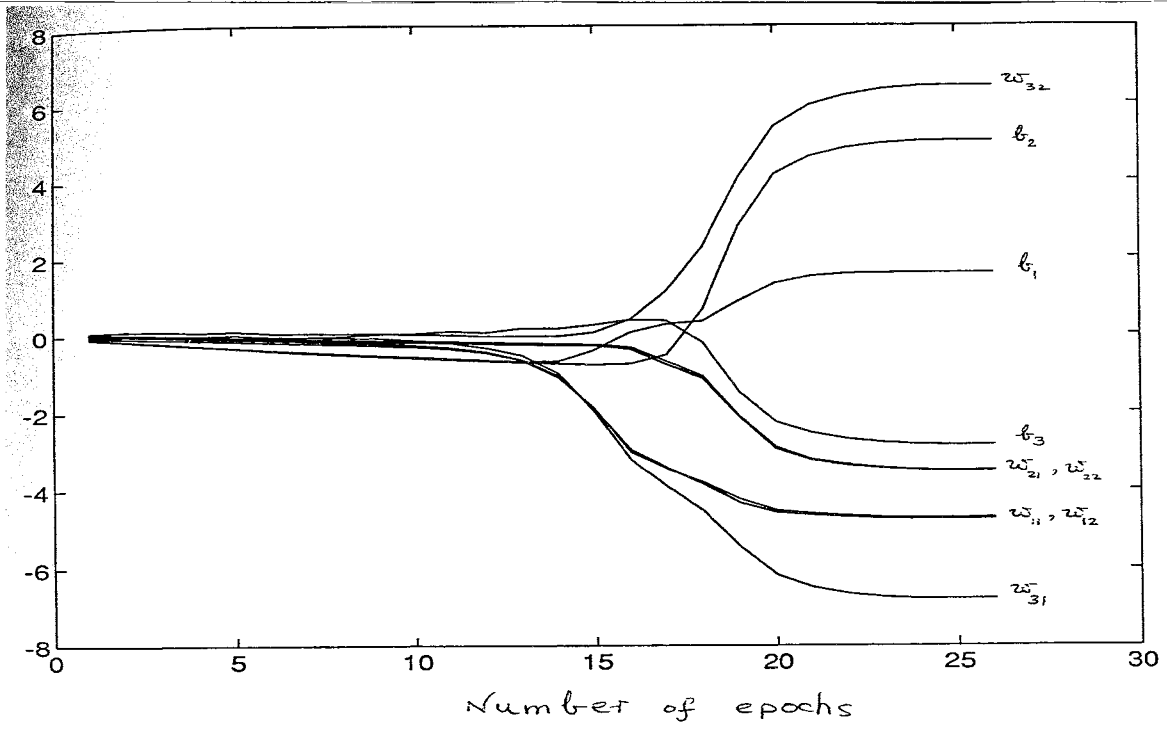


Figure 1: Problem 4.2, where one epoch = 100 iterations

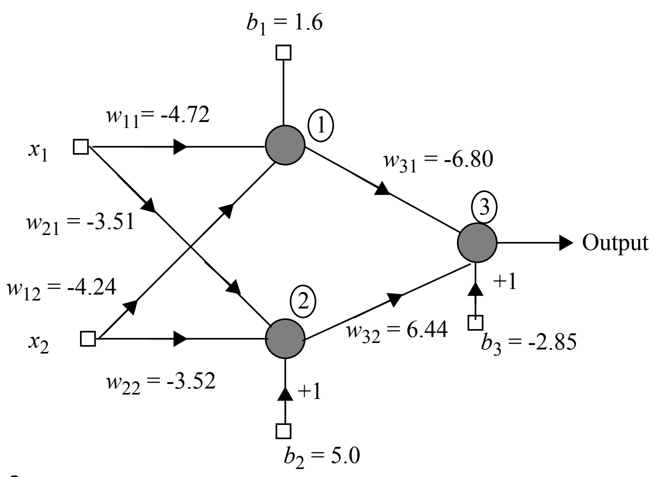


Figure 2: Problem 4.2

Problem 4.3

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

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

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

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

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 $0 \le \alpha < 1$ 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

$$\begin{aligned}\frac{\partial F(A_1^{(3)})}{\partial y_1^{(3)}} &= 1\\ y_1^{(3)} &= \varphi(v_1^{(3)})\\ v_1^{(3)} &= \sum_k w_{1k}^{(3)} y_k^{(2)}\end{aligned} \tag{2}$$

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 -$\mathbf{g}(n)$ 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 $\mathbf{y}(n)$ 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 $\mathbf{r}(n)$ 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 5
Kernel Methods and Radial-Basis Function Networks

Problem 5.9

The expected square error is given by

$$J(F) = \frac{1}{2}\sum_{i=1}^{N} \int_{R^{m_0}} (f(\mathbf{x}_i) - F(\mathbf{x}_i, \xi))^2 f_\xi(\xi) d\xi$$

where $f_\xi(\xi)$ is the probability density function of a noise distribution in the input space R^{m_0}. It is reasonable to assume that the noise vector ξ is additive to the input data vector **x**. Hence, we may define the cost function $J(F)$ as

$$J(F) = \frac{1}{2}\sum_{i=1}^{N} \int_{R^{m_0}} (f(\mathbf{x}_i) - F(\mathbf{x}_i + \xi))^2 f_\xi(\xi) d\xi \qquad (1)$$

where (for convenience of presentation) we have interchanged the order of summation and integration, which is permissible because both operations are linear. Let

$\mathbf{z} = \mathbf{x}_i + \xi \quad \text{or} \quad \xi = \mathbf{z} - \mathbf{x}_i$

Hence, we may rewrite (1) in the equivalent form:

$$J(F) = \frac{1}{2}\int_{R^{m_0}} \sum_{i=1}^{N} (f(\mathbf{x}_i) - F(\mathbf{z}))^2 f_\xi(\mathbf{z} - \mathbf{x}_i) d\mathbf{z} \qquad (2)$$

Note that the subscript ξ in $f_\xi(\cdot)$ merely refers to the "name" of the noise distribution and is therefore untouched by the change of variables. Differentiating (2) with respect to *F*, setting the result equal to zero, and finally solving for $F(\mathbf{z})$, we get the optimal estimator

$$\hat{F}(\mathbf{z}) = \frac{\sum_{i=1}^{N} f(\mathbf{x}_i) f_\xi(\mathbf{z} - \mathbf{x}_i)}{\sum_{i=1}^{N} f_\xi(\mathbf{z} - \mathbf{x}_i)}$$

This result bears a close resemblance to the Watson-Nadaraya estimator.

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

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

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 $\mathbf{w}^T$:

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

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:

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

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

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

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

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

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 **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):

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

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 $\sigma_Y^2 = \mathbf{E}[y^2(n)] \to \lambda_m$.

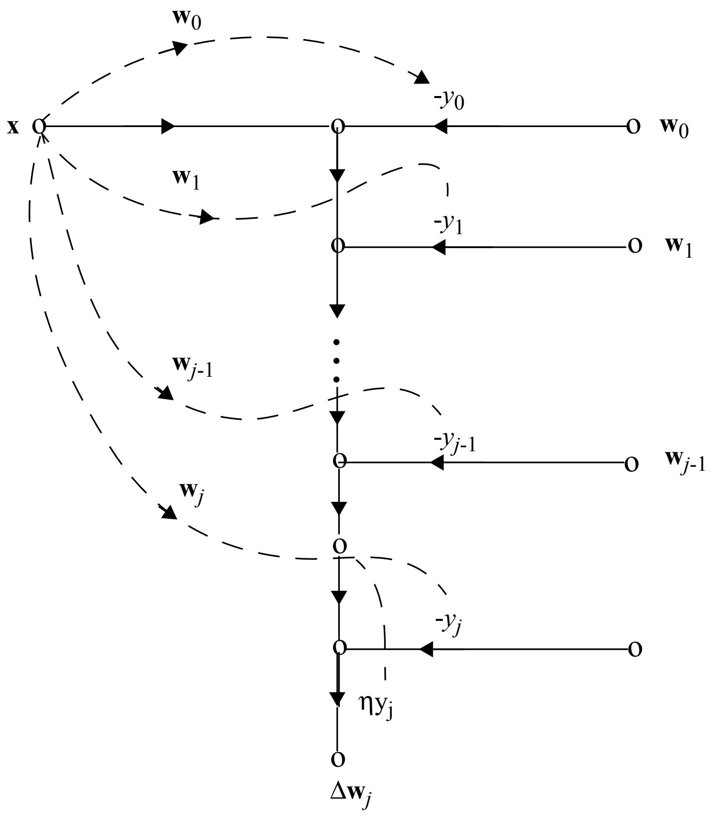
Problem 8.8

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

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

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

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 $\mathbf{x}$ 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):

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

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.

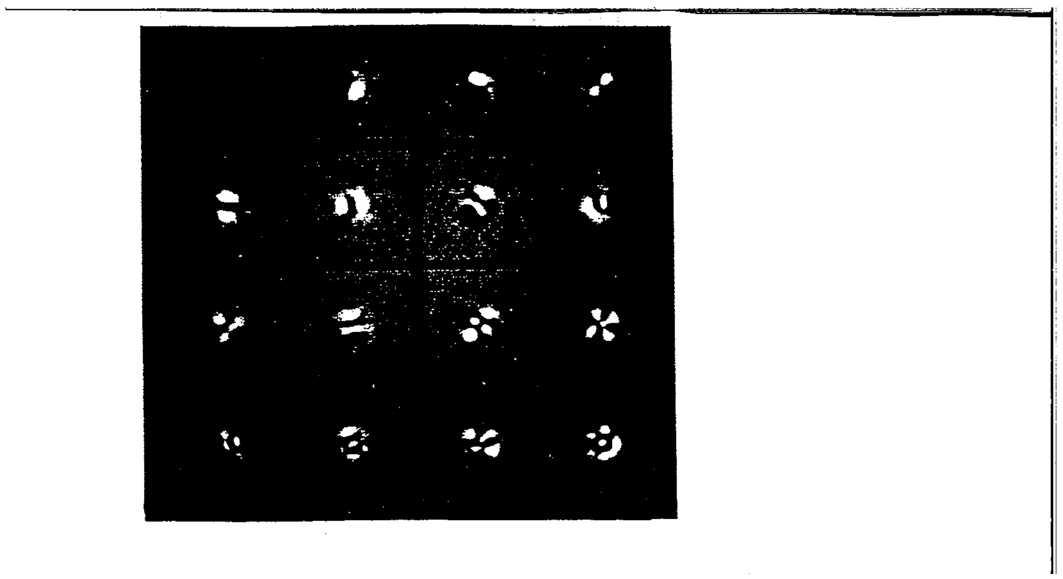


Figure 1: Problem 8.10 (Reproduced with permission of Biological Cybernetics)

CHAPTER 9
Self-Organizing Maps

Problem 9.1

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

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

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

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

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.

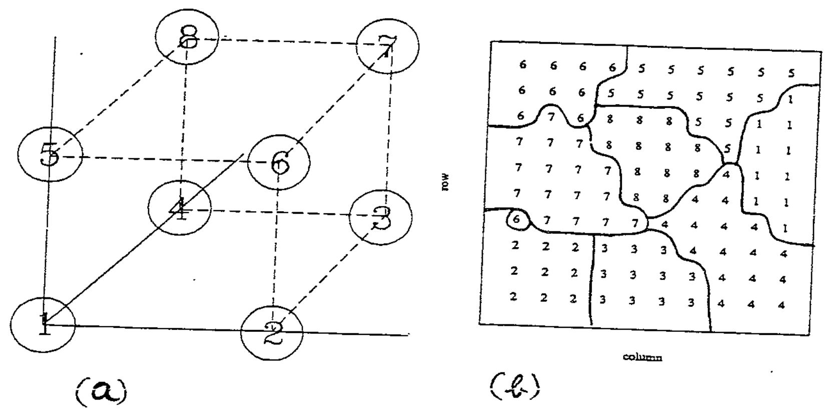


Figure 1: Problem 9.3

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.

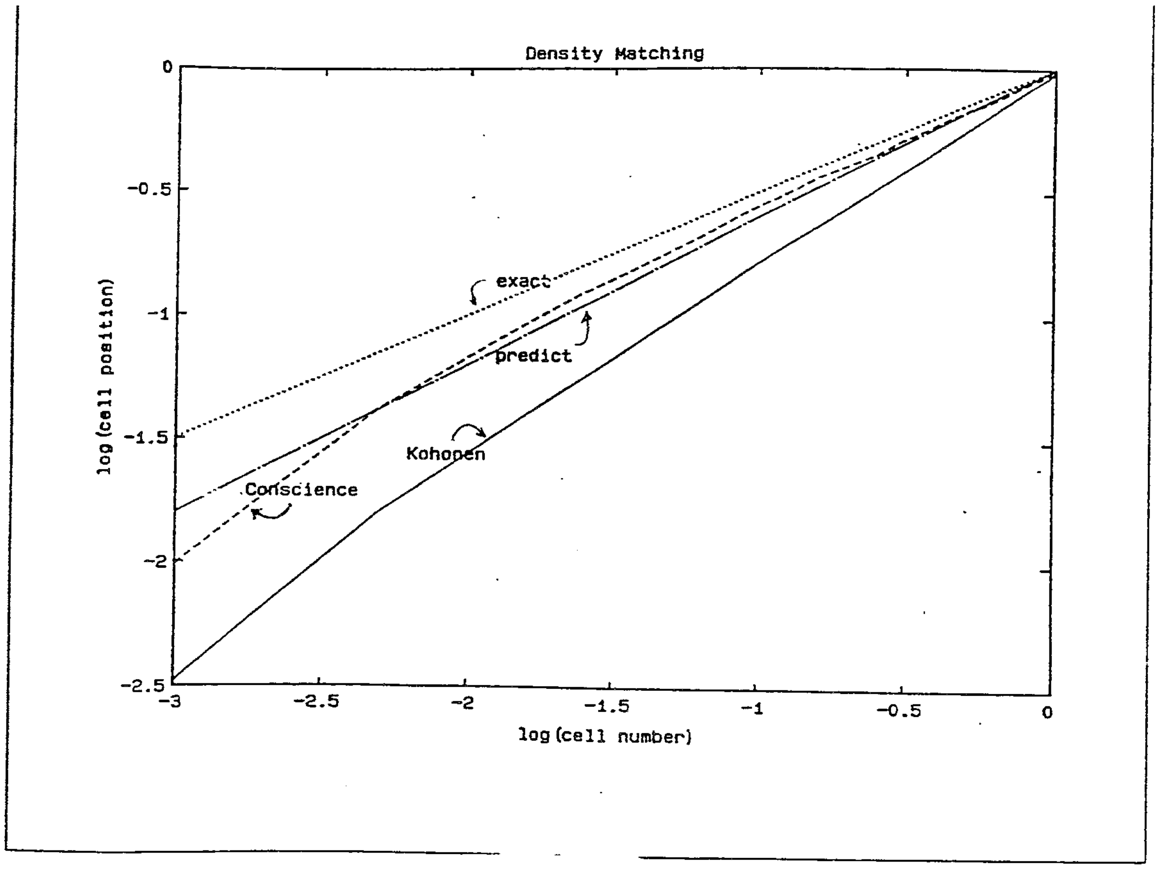


Figure 1: Problem 9.7

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.

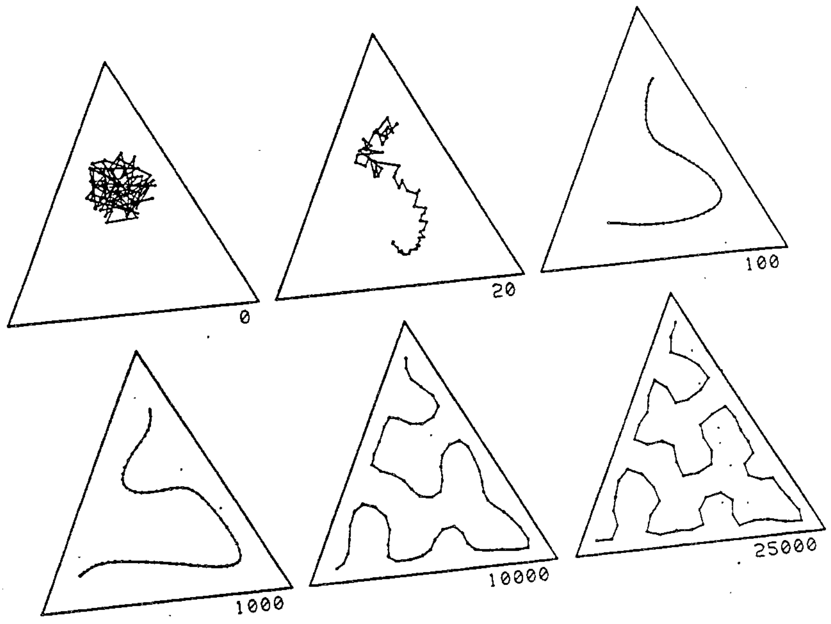


Figure 1: Problem 9.11

CHAPTER 10
Information-Theoretic Learning Models

Problem 10.1

The maximum entropy distribution of the random variable X is a uniform distribution over the range, [a, b], as shown by

$$f_X(x) = \begin{cases} \dfrac{1}{a-b}, & a \le x \le b \\ 0, & \text{otherwise} \end{cases}$$

Hence,

$$\begin{aligned} h(X) &= -\int_{-\infty}^{\infty} f_X(x) \log f_X(x)\, dx \\ &= \int_b^a \frac{1}{a-b} \log(a-b)\, dx \\ &= \log(a-b) \end{aligned}$$

Problem 10.3

Let

$$\begin{aligned} Y_i &= \mathbf{a}_i^T \mathbf{X}_1 \\ Z_i &= \mathbf{b}_i^T \mathbf{X}_2 \end{aligned}$$

where the vectors $\mathbf{X}_1$ and $\mathbf{X}_2$ have multivariate Gaussian distributions. The correlation coefficient between Y_i and Z_i is defined by

$$\begin{aligned} \rho_i &= \frac{\mathbf{E}[Y_i Z_i]}{\sqrt{\mathbf{E}[Y_i^2]\mathbf{E}[Z_i^2]}} \\ &= \frac{\mathbf{a}_i^T \mathbf{E}[\mathbf{X}_1 \mathbf{X}_2^T] \mathbf{b}_i}{\{(\mathbf{a}_i^T \mathbf{E}[\mathbf{X}_1 \mathbf{X}_1^T] \mathbf{a}_i)(\mathbf{b}_i^T \mathbf{E}[\mathbf{X}_1 \mathbf{X}_2^T] \mathbf{b}_i)\}^{1/2}} \\ &= \frac{\mathbf{a}_i^T \Sigma_{12} \mathbf{b}_i}{\{(\mathbf{a}_i^T \Sigma_{11} \mathbf{a}_i)(\mathbf{b}_i^T \Sigma_{22} \mathbf{b}_i)\}^{1/2}} \qquad (1) \end{aligned}$$

where

$$\Sigma_{11} = \mathbf{E}[\mathbf{X}_1\mathbf{X}_1^T]$$

$$\Sigma_{12} = \mathbf{E}[\mathbf{X}_1\mathbf{X}_2^T] = \Sigma_{21}$$

$$\Sigma_{22} = \mathbf{E}[\mathbf{X}_2\mathbf{X}_2^T]$$

The mutual information between Y_i and Z_i is defined by

$$I(Y_i;Z_i) = -\log(1-\rho_i^2)$$

Let r denote the rank of the cross-covariance matrix Σ_{12}. Given the vectors $\mathbf{X}_1$ and $\mathbf{X}_2$, we may invoke the idea of canonical correlations as summarized here:

- Find the pair of random variables $Y_1 = \mathbf{a}_1^T\mathbf{X}_1$ and $Z_1 = \mathbf{b}_1^T\mathbf{X}_2$ that are most highly correlated.
- Extract the pair of random variables $Y_2 = \mathbf{a}_2^T\mathbf{X}_1$ and $Z_2 = \mathbf{b}_2^T\mathbf{X}_2$ in such a way that Y_1 and Y_2 are uncorrelated and so are Z_1 and Z_2.
- Continue these two steps until at most r pairs of variables $\{(Y_1, Z_i), (Y_2, Z_i), ..., (Z_r, Z_r)\}$ have been extracted.

The essence of the canonical correlation described above is to encapsulate the dependence between random vectors $\mathbf{X}_1$ and $\mathbf{X}_2$ in the sequence $\{(Y_1, Z_i), (Y_2, Z_i), ..., (Z_r, Z_r)\}$. The uncorrelatedness of the pairs in this serquence, that is,

$$\mathbf{E}[Y_iY_j] = \mathbf{E}[Z_iZ_j] = 0 \qquad \text{for all } j \neq i$$

means that the mutual information between the vectors $\mathbf{X}_1$ and $\mathbf{X}_2$ is the sum of the mutual information measures between the individual elements of the pairs $\{(Y_i, Z_i)\}_{i=1}^r$. That is, we may write

$$I(\mathbf{X}_1, \mathbf{X}_2) = \sum_{i=1}^{r} I(Y_{ij}, Z_i) + \text{constant}$$
$$= -\sum_{i=1}^{r} \log(1-\rho_i^2) + \text{constant}$$

where ρ_i is defined by (1).

Problem 10.4

Consider a multilayer perceptron with a single hidden layer. Let w_{ji} denote the synaptic weight of hidden neuron j connected to source node i in the input layer. Let $x_{i|\alpha}$ denote the ith component of the input vector **x**, given example α. Then the induced local field of neuron j is

$$v_{j|\alpha} = \sum_i w_{ji} x_{i|\alpha} \tag{1}$$

Correspondingly, the output of hidden neuron j for example α is given by

$$y_{j|\alpha} = \varphi(v_{j|\alpha}) \tag{2}$$

where $\varphi(\cdot)$ is the logistic function

$$\varphi(v) = \frac{1}{1+e^{-v}}$$

Consider next the output layer of the network. Let w_{kj} denote the synaptic weight of output neuron k connected to hidden neuron j. The induced local field of output neuron k is

$$v_{k|\alpha} = \sum_i w_{kj} y_{j|\alpha} \tag{3}$$

The kth output of the network is therefore

$$y_{k|\alpha} = \varphi(v_{k|\alpha}) \tag{4}$$

The output $y_{k|\alpha}$ is assigned a probabilistic interpretation by writing

$$p_{k|\alpha} = y_{k|\alpha} \tag{5}$$

Accordingly, we may view $y_{k|\alpha}$ as an estimate of the conditional probability that the proposition k is true, given the example α at the input. On this basis, we may interpret

$$1 - y_{k|\alpha} = 1 - p_{k|\alpha}$$

as the estimate of the conditional probability that the proposition k is false, given the input example α. Correspondingly, let $q_{k|\alpha}$ denote the actual (true) value of the conditional probability that the proposition k is true, given the input example α. This means that 1 - $q_{k|\alpha}$ is the actual value of the conditional probability that the proposition k is false, given the input example α. Thus, we may define the Kullback-Leibler divergence for the multilayer perceptron as

$$D_{p||q} = \sum_{\alpha} p_{\alpha} \sum_{k} \left[q_{k|\alpha} \log\left(\frac{q_{k|\alpha}}{p_{k|\alpha}}\right) + (1 - q_{k|\alpha}) \log\left(\frac{1 - q_{k|\alpha}}{1 - p_{k|\alpha}}\right) \right]$$

where p_{α} is the a priori probability of occurrence of example α at the input.

To perform supervised training of the multilayer perceptron, we use gradient descent on $D_{p||q}$ in weight space. First, we use the chain rule to express the partial derivative of $D_{p||q}$ with respect to the synaptic weight w_{kj} of output neuron k as follows:

$$\begin{aligned}\frac{\partial D_{p||q}}{\partial w_{kj}} &= \frac{\partial D_{p||q}}{\partial p_{k|\alpha}} \frac{\partial p_{k|\alpha}}{\partial y_{k|\alpha}} \frac{\partial y_{k|\alpha}}{\partial v_{k|\alpha}} \frac{\partial v_{k|\alpha}}{\partial w_{kj}} \\ &= -\sum_{\alpha} p_{\alpha}(q_{k|\alpha} - p_{k|\alpha}) y_{j|\alpha} \qquad (6)\end{aligned}$$

Next, we express the partial derivative of $D_{p||q}$ with respect to the synaptic weight w_{ji} of hidden neuron j by writing

$$\frac{\partial D_{p||q}}{\partial w_{ji}} = -\sum_{\alpha} p_{\alpha} \sum_{\alpha} \left(\frac{q_{k|\alpha}}{p_{k|\alpha}} - \frac{1 - q_{k|\alpha}}{1 - p_{k|\alpha}} \right) \frac{\partial p_{k|\alpha}}{\partial w_{ji}} \qquad (7)$$

Via the chain rule, we write

$$\begin{aligned}\frac{\partial p_{k|\alpha}}{\partial w_{ji}} &= \frac{\partial p_{k|\alpha}}{\partial y_{k|\alpha}} \frac{\partial y_{k|\alpha}}{\partial v_{k|\alpha}} \frac{\partial v_{k|\alpha}}{\partial y_{j|\alpha}} \frac{\partial y_{j|\alpha}}{\partial v_{j|\alpha}} \frac{\partial v_{j|\alpha}}{\partial w_{ji}} \\ &= \varphi'(v_{k|\alpha}) w_{kj} \varphi'(v_{j|\alpha}) x_{i|\alpha} \qquad (8)\end{aligned}$$

But

$$\begin{aligned}\varphi'(v_{k|\alpha}) &= y_{k|\alpha}(1 - y_{k|\alpha}) \\ &= p_{k|\alpha}(1 - p_{k|\alpha}) \qquad (9)\end{aligned}$$

Hence, using (8) and (9) we may simplify (7) as

$$\frac{\partial D_{p||q}}{\partial w_{ji}} = -\sum_{\alpha} p_{\alpha} x_{i|\alpha} \varphi'\left(\sum_{i} w_{ji} x_{i|\alpha} \right) \sum_{k} (p_{k|\alpha} - q_{k|\alpha}) w_{kj}$$

where $\varphi'(\cdot)$ is the derivative of the logistic function $\varphi(\cdot)$ with respect to its argument.

Assuming the use of the learning-rate parameter η for all weight changes applied to the network, we may use the method of steepest descent to write the following two-step probabilistic algorithm:

1. For output neuron k, compute

$$\Delta w_{kj} = -\eta \frac{\partial D_{p||q}}{\partial w_{kj}}$$
$$= \eta \sum_{\alpha} p_{\alpha}(q_{k|\alpha} - P_{k|\alpha}) y_{j|\alpha}$$

2. For hidden neuron j, compute

$$\Delta w_{ji} = -\eta \frac{\partial D_{p||q}}{\partial w_{ji}}$$
$$= \eta \sum_{\alpha} p_{\alpha} x_{i|\alpha} \varphi'\left(\sum_{i} w_{ji} x_{i|\alpha}\right) \sum_{k} (p_{k|\alpha} - q_{k|\alpha}) w_{kj}$$

Problem 10.9

We first note that the mutual information between the random variables X and Y is defined by

$$I(X;Y) = h(X) + h(Y) - h(X, Y)$$

To maximize the mutual information $I(X;Y)$ we need to maximize the sum of the differential entropy $h(X)$ and the differential entropy $h(\hat{Y})$ and also minimize the joint differential entropy $h(X,Y)$. From the definition of differential entropy, both $h(X)$ and $h(Y)$ attain their maximum value of 0.5 when X and Y occur with probability 1/2. Moreover $h(X,Y)$ is minimized when the joint probability of X and Y occupies the smallest possible region in the probability space.

Problem 10.10

The outputs Y_1 and Y_2 of the two neurons in Fig. P10.6 in the text are respectively defined by

$$Y_1 = \left(\sum_{i=1}^{m} w_{1i} x_i\right) + N_1$$

$$Y_2 = \left(\sum_{i=1}^{L} w_{2i} x_i\right) + N_2$$

where are w_{1i} the synaptic weights of output neuron 1, and the w_{2i} are synaptic weights of output neuron 2. The mutual information between the output vector $\mathbf{Y} = [Y_1, Y_2]^T$ and the input vector $\mathbf{X} = [X_1, X_2, .., X_m]^T$ is

$$\begin{aligned} I(\mathbf{X};\mathbf{Y}) &= h(\mathbf{Y}) - h(\mathbf{Y}|\mathbf{X}) \\ &= h(\mathbf{Y}) - h(\mathbf{N}) \end{aligned} \tag{1}$$

where $h(\mathbf{Y})$ is the differential entropy of the output vector $\mathbf{Y}$ and $h(\mathbf{N})$ is the differential entropy of the noise vector $\mathbf{N} = [\mathbf{N}_1, \mathbf{N}_2]^T$.

Since the noise terms $\mathbf{N}_1$ and $\mathbf{N}_2$ are Gaussian and uncorrelated, it follows that they are statistically independent. Hence,

$$\begin{aligned} h(\mathbf{N}) &= h(\mathbf{N}_1, \mathbf{N}_2) \\ &= h(\mathbf{N}_1) + h(\mathbf{N}_2) \\ &= 1 + \log(2\pi\sigma_N^2) \end{aligned} \tag{2}$$

The differential entropy of the output vector $\mathbf{Y}$ is

$$\begin{aligned} h(\mathbf{Y}) &= h(Y_1, Y_2) \\ &= -\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{Y_1,Y_2}(y_1, y_2)\log f_{Y_1,Y_2}(y_1, y_2)dy_1 dy_2 \end{aligned}$$

where $f_{Y_1,Y_2}(y_1, y_2)$ is the joint pdf of Y_1 and Y_2. Both Y_1 and Y_2 are dependent on the same set of input signals, and so they are correlated with each other. Let

$$\begin{aligned} \mathbf{R} &= \mathbf{E}[\mathbf{Y}\mathbf{Y}^T] \\ &= \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \end{aligned}$$

where

$$r_{ij} = \mathbf{E}[Y_i Y_j], \qquad i, j = 1, 2$$

The individual element of the correlation matrix $\mathbf{R}$ are given by

$$\begin{aligned} r_{11} &= \sigma_1^2 + \sigma_N^2 \\ r_{12} &= r_{21} = \sigma_1\sigma_1\rho_{12} \end{aligned}$$

$$r_{22} = \sigma_2^2 + \sigma_N^2$$

where σ_1^2 and σ_2^2 are the respective variances of Y_1 and Y_2 in the absence of noise, and ρ_{12} is their correlation coefficient also in the absence of noise. For the general case of an N-dimensional Gaussian distribution, we have

$$f_{\mathbf{Y}}(\mathbf{y}) = \frac{1}{(2\pi)^{N/2}(\det\mathbf{R})^{1/2}}\exp\left(\frac{1}{2}\mathbf{y}^T\mathbf{R}^{-1}\mathbf{y}\right)$$

Correspondingly, the differential entropy of the N-dimensional vector $\mathbf{Y}$ is described as

$$h(\mathbf{Y}) = \log((2\pi e)^{N/2}\det(\mathbf{R}))$$

where e is the base of the natural logarithm. For the problem at hand, we have $N = 2$ and so

$$\begin{aligned} h(\mathbf{Y}) &= \log(2\pi e \det(\mathbf{R})) \\ &= 1 + \log(2\pi \det(\mathbf{R})) \end{aligned}$$

Hence, the use of (2) and (3) in (1) yields

$$I(\mathbf{X};\mathbf{Y}) = \log\left(\frac{\det(\mathbf{R})}{\sigma_N^2}\right) \tag{4}$$

For a fixed noise variance σ_N^2, the mutual information I(**X**;**Y**) is maximized by maximizing the determinant det(**R**). By definition,

$$\det(\mathbf{R}) = r_{11}r_{22} - r_{12}r_{21}$$

That is,

$$\det(\mathbf{R}) = \sigma_N^4 + \sigma_N^2(\sigma_1^2 + \sigma_2^2) + \sigma_1^2\sigma_2^2(1 - \rho_{12}^2) \tag{5}$$

Depending on the value of noise variance σ_N^2, we may identify two distinct situations:

1. **Large noise variance**. When σ_N^2 is large, the third term in (5) may be neglected, obtaining

   $$\det(\mathbf{R}) \approx \sigma_N^4 + \sigma_N^2(\sigma_1^2 + \sigma_2^2)$$

In this case, maximizing det(**R**) requires that we maximize $(\sigma_1^2 + \sigma_2^2)$. This requirement may be satisfied simply by maximizing the variance σ_1^2 of output Y_1 or the variance σ_2^2 of output Y_2, separately. Since the variance of output $Y_i : i = 1, 2$, is equal to σ_i^2 in the absence of noise and $\sigma_1^2 + \sigma_N^2$ in the presence of noise, it follows from the Infomax principle that the optimum solution for a fixed noise variance is to maximize the variance of either output, Y_1 or Y_2.

2. **Low noise variance**. When the noise variance σ_N^2 is small, the third term $\sigma_1^2\sigma_2^2(1-\rho_{12}^2)$ in (5) becomes important relative to the other two terms. The mutual information $I(\mathbf{X};\mathbf{Y})$ is then maximized by making an optimal tradeoff between two options: keeping the output variances σ_1^2 and σ_2^2 large, and making the outputs Y_1 and Y_2 of the two neurons uncorrelated.

Based on these observations, we may now make the following two statements:

- A high-noise level favors redundancy of response, in which case the two output neurons compute the same linear combination of inputs. Only one such combination yields a response with maximum variance.
- A low-noise level favors diversity of response, in which case the two output neurons compute different linear combinations of inputs even though such a choice may result in a reduced output variance.

Problem 10.11

(a) We are given

$$Y_a = S + N_a$$
$$Y_b = S + N_b$$

Hence,

$$\frac{Y_a + Y_b}{2} = S + \frac{1}{2}(N_a + N_b)$$

The mutual information between $\frac{1}{2}(Y_a + Y_b)$ and the signal component S is

$$I\left(\frac{Y_a + Y_b}{2};S\right) = h\left(\frac{Y_a + Y_b}{2}\right) - h\left(\frac{Y_a + Y_b}{2}\middle|S\right) \quad (1)$$

The differential entropy of $\frac{Y_a + Y_b}{2}$ is

$$h\left(\frac{Y_a+Y_b}{2}\right) = \frac{1}{2}\left[1+\log\left(\frac{\pi}{2}\mathrm{var}[Y_a+Y_b]\right)\right] \tag{2}$$

The conditional differential entropy of $\frac{Y_a+Y_b}{2}$ given S is

$$h\left(\frac{Y_a+Y_b}{2}\middle|S\right) = h\left(\frac{N_a+N_b}{2}\right)$$
$$= \frac{1}{2}\left[\log\left(\frac{\pi}{2}\mathrm{var}[N_a+N_b]\right)\right] \tag{3}$$

Hence, the use of (2) and (3) in (1) yields (after the simplification of terms)

$$I\left(\frac{Y_a+Y_b}{2};S\right) = \log\left(\frac{\mathrm{var}[Y_a+Y_b]}{\mathrm{var}[N_a+N_b]}\right)$$

(b) The signal component S is ordinarily independent of the noise components N_a and N_b. Hence with

$$Y_a+Y_b = 2S+N_a+N_b$$

it follows that

$$\mathrm{var}[Y_a+Y_b] = 4\mathrm{var}[S]+\mathrm{var}[N_a+N_b]$$

The ratio $(\mathrm{var}[Y_a+Y_b])/(\mathrm{var}[N_a+N_b])$ in the expression for the mutual information $I\left(\frac{Y_a+Y_b}{2};S\right)$ may therefore be interpreted as a signal-plus-noise to noise ratio.

Problem 10.12

Principal components analysis (PCA) and independent-components analysis (ICA) share a common feature: They both linearly transform an input signal into a fixed set of components.

However, they differ from each other in two important respects:

1. PCA performs decorrelation by minimizing second-order moments; higher-order moments are not involved in this computation. On the other hand, ICA performs statistical independence by using higher-order moments.

2. The output signal vector resulting from PCA has a diagonal covariance matrix. The first principal component defines a direction in the original signal space that captures the maximum possible variance; the second principal component defines another direction in the remaining orthogonal subspace that captures the next maximum possible variance, and so on. On the other hand, ICA does not find the directions of maximum variances but rather interesting directions where the term “interesting” refers to “deviation from Gaussianity”.

Problem 10.13

Independent components analysis may be used as a preprocessing tool before signal detection and pattern classification. In particular, through a change of coordinates resulting from the use of ICA, the probability density function of multichannel data may be expressed as a product of marginal densities. This change, in turn, permits density estimation with shorter observations.

Problem 10.14

Consider m random variables $X_1, X_2, ..., X_m$ that are defined by

$$X_i = \sum_{j=1}^{N} a_{ij} U_j, \qquad i = 1, 2, ..., N$$

where the U_j are independent random variables. The Darmois’ theorem states that if the X_i are independent, then the variables U_j for which $a_{ij} \neq 0$ are all Gaussian.

For independent-components analysis to work, at most a single X_i can be Gaussian. If all the X_i are independent to begin with, there is no need for the application of independent-components analysis. This, in turn, means that all the X_i must be Gaussian. For a finite N, this condition can only be satisfied if all the U_j are not only independent but also Gaussian.

Problem 10.15

The use of independent-components analysis results in a set of components that are as statistically independent of each other as possible. In contrast, the use of decorrelation only addresses second-order statistics and there is therefore no guarantee of statistical independence.

Problem 10.16

The Kullback-Leibler divergence between the joint pdf $f_{\mathbf{Y}}(\mathbf{y}, \mathbf{w})$ and the factorial pdf $\tilde{f}_{\mathbf{Y}}(\mathbf{y}, \mathbf{w})$ is the multifold integral

$$D_{f_Y || \tilde{f}_Y} = \int_{-\infty}^{\infty} f_{\mathbf{Y}}(\mathbf{y}, \mathbf{w}) \left(\log \frac{f_{\mathbf{Y}}(\mathbf{y}, \mathbf{w})}{\tilde{f}_{\mathbf{Y}}(\mathbf{y}, \mathbf{w})} \right) d\mathbf{y} \tag{1}$$

Let

$$d\mathbf{y} = dy_i dy_j d\mathbf{y}'$$

where $\mathbf{y}'$ excludes y_i and y_j. We may then rewrite (1) as

$$\begin{aligned}
D_{f_Y||\tilde{f}_Y} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dy_i dy_j \int_{-\infty}^{\infty} f_{\mathbf{Y}}(\mathbf{y},\mathbf{w})\log f_{\mathbf{Y}}(\mathbf{y},\mathbf{w})d\mathbf{y}' \\
&\quad -\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dy_i dy_j \int_{-\infty}^{\infty} f_{\mathbf{Y}}(\mathbf{y},\mathbf{w})\log \tilde{f}_{\mathbf{Y}}(\mathbf{y},\mathbf{w})d\mathbf{y}' \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{Y_i,Y_j}(y_i, y_j, \mathbf{w})\log f_{Y_i,Y_j}(y_i, y_j, \mathbf{w})dy_i dy_j \\
&\quad -\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{Y_i,Y_j}(y_i, y_j, \mathbf{w})\log(f_{Y_i}(y_i, \mathbf{w})f_{Y_j}(y_j, \mathbf{w}))dy_i dy_j \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{Y_i,Y_j}(y_i, y_j)\log\left(\frac{f_{Y_i,Y_j}(y_i, y_j, \mathbf{w})}{f_{Y_i}(y_i, \mathbf{w})f_{Y_j}(y_j, \mathbf{w})}\right)dy_i dy_j \\
&= I(Y_i;Y_j)
\end{aligned}$$

That is, the Kullback-Leibler divergence between the joint pdf $f_{\mathbf{Y}}(\mathbf{y}, \mathbf{w})$ and the factorial pdf distribution $\tilde{f}_{\mathbf{Y}}(\mathbf{y}, \mathbf{w})$ is equal to the mutual information between the components Y_i and Y_j of the output vector $\mathbf{Y}$ for any pair (i, j).

Problem 10.18

Define the output matrix

$$\mathbf{Y} = \begin{bmatrix} y_1(0) & y_1(1) & \dots & y_1(N-1) \\ y_2(0) & y_2(1) & \dots & y_2(N-1) \\ \vdots & \vdots & & \vdots \\ y_m(0) & y_m(1) & \dots & y_m(N-1) \end{bmatrix} \tag{1}$$

where m is the dimension of the output vector $\mathbf{y}(n)$ and N is the number of samples used in computing the matrix $\mathbf{Y}$. Correspondingly, define the m-by-N matrix of activation functions

$$\Phi(\mathbf{Y}) = \begin{bmatrix} \varphi(y_1(0)) & \varphi(y_1(1)) & \dots & \varphi(y_1(N-1)) \\ \varphi(y_2(0)) & \varphi(y_2(1)) & \dots & \varphi(y_2(N-1)) \\ \vdots & \vdots & & \vdots \\ \varphi(y_m(0)) & \varphi(y_m(1)) & \dots & \varphi(y_m(N-1)) \end{bmatrix}$$

In the batch mode, we define the average weight adjustment (see Eq. (10.100) of the text)

$$\begin{aligned}\Delta\mathbf{W} &= \frac{1}{N}\sum_{n=0}^{N\text{-}1}\Delta\mathbf{W}(n) \\ &= \eta\left(\mathbf{I} - \frac{1}{N}\left(\sum_{n=0}^{N\text{-}1}\phi(\mathbf{y}(n))\mathbf{y}^T(n)\right)\right)\mathbf{W}\end{aligned}$$

Equivalently, using the matrix definitions introduced in (2), we may write

$$\Delta\mathbf{W} = \eta\left(\mathbf{I} - \frac{1}{N}\Phi(\mathbf{Y})\mathbf{Y}^T\right)\mathbf{W}$$

which is the desired formula.

Problem 10.19

(a) Let $q(\mathbf{y})$ denote a pdf equal to the determinant det($\mathbf{J}$) with the elements of the Jacobian $\mathbf{J}$ being as defined in Eq. (10.115). Then using Eq. (10.116) we may express the entropy of the random vector $\mathbf{Z}$ at the output of the nonlinearity in Fig. 10.16 of the text as

$$h(\mathbf{Z}) = -D_{f||q}$$

Invoking the pythogorean decomposition of the Kullback-Leibler divergence, we write

$$D_{f||q} = D_{f||\tilde{f}} + D_{\tilde{f}||q}$$

Hence, the differential entropy

$$h(\mathbf{Z}) = D_{f||\tilde{f}} - D_{\tilde{f}||q} \tag{1}$$

(b) If $q(\mathbf{y}_i)$ happens to equal the source pdf $f_U(\mathbf{y}_i)$ for all i, we then find that $D_{\tilde{f}||q} = 0$. In such a case, (1) reduces to

$$h(\mathbf{Z}) = -D_{f||\tilde{f}}$$

That is, the entropy $h(\mathbf{Z})$ is equal to the negative of the Kullback-Leibler divergence between the pdf $f_{\mathbf{Y}}(\mathbf{y})$ and the corresponding factorial distribution $\tilde{f}_{\mathbf{Y}}(\mathbf{y})$.

Problem 10.20

(a) From Eq. (10.124) in the text,

$$\Phi = \log|\det(\mathbf{A})| + \log|\det(\mathbf{W})| + \sum_i \log\left(\frac{\partial z_i}{\partial y_i}\right)$$

The matrix $\mathbf{A}$ of the linear mixer is fixed. Hence differentiating Φ with respect to $\mathbf{W}$:

$$\frac{\partial \Phi}{\partial \mathbf{W}} = \mathbf{W}^{-T} + \sum_i \frac{\partial}{\partial \mathbf{W}} \log\left(\frac{\partial z_i}{\partial y_i}\right) \tag{1}$$

(b) From Eq. (10.126) pf the text,

$$z_i = \frac{1}{1 + e^{-y_i}}$$

Differentiating z_i with respect to y_i:

$$\begin{aligned} \frac{\partial z_i}{\partial y_i} &= \frac{e^{-y_i}}{(1 + e^{-y_i})^2} \\ &= z_i - z_i^2 \end{aligned} \tag{2}$$

Hence, differentiating $\log\left(\frac{\partial z_i}{\partial y_i}\right)$ with respect to the demixing matrix $\mathbf{W}$, we get

$$\frac{\partial}{\partial \mathbf{W}} \log\left(\frac{\partial z_i}{\partial y_i}\right) = \frac{\partial}{\partial \mathbf{W}} \log(z_i - z_i^2)$$

$$= \frac{\partial z_i}{\partial \mathbf{W}} \frac{\partial}{\partial z_i} \log(z_i - z_i^2)$$

$$= \frac{\partial z_i}{\partial \mathbf{W}} \frac{1}{(z_i - z_i^2)}(1 - 2z_i$$

$$= \frac{\partial z_i}{\partial y_i} \frac{\partial y_i}{\partial \mathbf{W}} \frac{1}{(z_i - z_i^2)}(1 - 2z_i \qquad (3)$$

But from (2) we have

$$\frac{\partial z_i}{\partial y_i} \left(\frac{1}{z_i - z_i^2} \right) = 1$$

Hence, we may simplify (3) to

$$\frac{\partial}{\partial \mathbf{W}} \log\left(\frac{\partial z_i}{\partial y_i}\right) = \frac{\partial y_i}{\partial \mathbf{W}}(1 - 2z_i)$$

We may thus rewrite (1) as

$$\frac{\partial \Phi}{\partial \mathbf{W}} = \mathbf{W}^{-T} + \sum_i \frac{\partial y_i}{\partial \mathbf{W}}(1 - 2z_i)$$

Putting this relation in matrix form and recognizing that the demixer output **y** is equal to **Wx** where **x** is the observation vector, we find that the adjustment applied to **W** is defined by

$$\Delta \mathbf{W} = \eta \frac{\partial \Phi}{\partial \mathbf{W}}$$
$$= \eta(\mathbf{W}^{-T} + (\mathbf{1} - 2\mathbf{z})\mathbf{x}^T)$$

where η is the learning-rate parameter and **1** is a vector of ones.

CHAPTER 11
Stochastic Methodfs Rooted in Statistical Mechanics

Problem 11.1

By definition, we have

$$p_{ij}^{(n)} = P(X_t = j|X_{t-n} = i)$$

where *t* denotes time and *n* denotes the number of discrete steps. For $n = 1$, we have the one-step transition probability

$$p_{ij}^{(1)} = p_{ij} = P(X_t = j|X_{t-1} = i)$$

For $n = 2$ we have the two-step transition probability

$$p_{ij}^{(2)} = \sum_k p_{ik}p_{kj}$$

where the sum is taken over all intermediate steps *k* taken by the system. By induction, it thus follows that

$$p_{ij}^{(n-1)} = \sum_k p_{ik}p_{kj}^{(n)}$$

Problem 11.2

For $p > 0$, the state transition diagram for the random walk process shown in Fig., P11.2 of the test is irreducible. The reason for saying so is that the system has only one class, namely,
$\{0, \pm 1, \pm 2, ...\}$.

Problem 11.3

The state transition diagram of Fig. P11.3 in the text pertains to a Markov chain with two classes: $\{x_1\}$ and $\{x_1, x_2\}$.

Problem 11.4

The stochastic matrix of the Markov chain in Fig. P11.4 of the text is given by

$$\mathbf{P} = \begin{bmatrix} \frac{3}{4} & \frac{1}{4} & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{4} & \frac{3}{4} & 0 \end{bmatrix}$$

Let π_1, π_2, and π_3 denote the steady-state probabilities of this chain. We may then write (see Eq. (11.27) of the text)

$$\pi_1 = \pi_1\left(\frac{3}{4}\right) + \pi_2(0) + \pi_3\left(\frac{1}{4}\right)$$

$$\pi_2 = \pi_1\left(\frac{1}{4}\right) + \pi_2\left(\frac{2}{3}\right) + \pi_3\left(\frac{3}{4}\right)$$

$$\pi_3 = \pi_1(0) + \pi_2\left(\frac{1}{3}\right) + \pi_3(0)$$

That is,

$$\pi_1 = \pi_3$$
$$\pi_2 = 3\pi_3$$

We also have, by definition,

$$\pi_1 + \pi_2 + \pi_3 = 1$$

Hence,

$$\pi_3 + 3\pi_3 + \pi_3 = 1$$

or equivalently

$$\pi_3 = \frac{1}{5}$$

and so

$$\pi_1 = \frac{1}{5}$$

$$\pi_2 = \frac{3}{5}$$

Problem 11.6

The Metropolis algorithm and the Gibbs sampler are similar in that they both generate a Markov chain with the Gibbs distribution as the equilibrium distribution.

They differ from each other in the following respect: In the Metropolis algorithm, the transition probabilities of the Markov chain are stationary. In contrast, in the Gibbs sampler, they are nonstationary.

Problem 11.7

Simulated annealing algorithm for solving the travelling salesman problem:

1. Set up an annealing schedule for the algorithm.
2. Initialize the algorithm by picking a tour at random.
3. Choose a pair of cities in the tour and then reverse the order that the cities in-between the selected pairs are visited. This procedure, illustrated in Figure 1 below, generates new tours in a local manner:

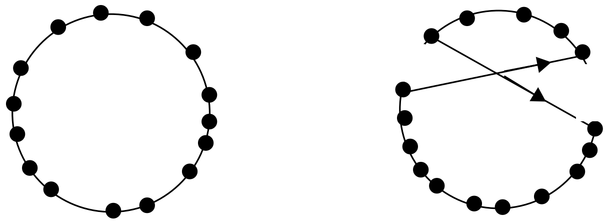


Figure 1: Problem 11.7

4. Calculate the energy difference due to the reversal of paths applied in step 3.
5. If the energy difference so calculated is negative or zero, accept the new tour. If, on the other hand, it is positive, accept the change in the tour with probability defined in accordance with the Metropolis algorithm.
6. Select another pair of cities, and repeat steps 3 to 5 until the required number of iterations is accepted.
7. Lower the temperature in the annealing schedule, and repeat steps 3 to 6.

Problem 11.8

(a) We start with the notion that a neuron j flips from state x_j to $-x_j$ at temperature T with probability

$$P(x_j \rightarrow -x_j) = \frac{1}{1 + \exp(-\Delta E_j / T)} \qquad (1)$$

where ΔE_j is the energy difference resulting from such a flip. The energy function of the Boltzman machine is defined by

$$E = -\frac{1}{2}\sum_{i}\sum_{\substack{j \\ i \neq j}} w_{ji} x_i x_j$$

Hence, the energy change produced by neuron j flipping from state x_j to $-x_j$ is

$$\begin{aligned}
\Delta E_j &= \begin{pmatrix}\text{energy with neuron } j \\ \text{in state } x_j\end{pmatrix} - \begin{pmatrix}\text{energy with neuron } j \\ \text{in state } -x_j\end{pmatrix} \\
&= -(x_j)\sum_j w_{ji} x_i + (-x_j)\sum_j w_{ji} x_i \\
&= -2x_j \sum_j w_{ji} x_i \\
&= -2x_j v_j
\end{aligned} \qquad (2)$$

where v_i is the induced local field of neuron j.

(b) In light of the result in (2), we may rewrite (1) as

$$P(x_j \rightarrow -x_j) = \frac{1}{1 + \exp(2x_j v_j / T)}$$

This means that for an initial state $x_j = -1$, the probability that neuron j is flipped into state +1 is

$$\frac{1}{1 + \exp(-2v_j / T)} \qquad (3)$$

(c) For an initial state of $x_j = +1$, the probability that neuron j is flipped into state -1 is

$$\frac{1}{1 + \exp(+2v_j / T)} = 1 - \frac{1}{1 + \exp(-2v_j / T)} \qquad (4)$$

The flipping probability in (4) and the one in (3) are in perfect agreement with the following probabilistic rule

$$x_j = \begin{cases} +1 & \text{with probability } P(v_j) \\ -1 & \text{with probability } 1\text{-}P(v_j) \end{cases}$$

where $P(v_j)$ is itself defined by

$$P(v_j) = \frac{1}{1 + \exp(-2v_j / T)}$$

Problem 11.9

The log-likelihood function $L(\mathbf{w})$ is (see (11.48) of the text)

$$L(\mathbf{w}) = \sum_{\mathbf{x}_\alpha \in T} \log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \log \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right)$$

Differentiating $L(\mathbf{w})$ with respect to weight w_{ji}:

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T} \sum_{\mathbf{x}_\alpha \in T} \frac{\partial E(\mathbf{x})}{\partial w_{ji}} \left(-\frac{1}{\sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right)} + \frac{1}{\sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right)} \right)$$

The energy function $E(\mathbf{x})$ is defined by (see (11.39) of the text)

$$E(\mathbf{x}) = -\frac{1}{2} \sum_{i} \sum_{\substack{j \\ i \neq j}} w_{ji} x_i x_j$$

Hence,

$$\frac{\partial E(\mathbf{x})}{\partial w_{ji}} = -x_i x_j, \qquad i \neq j \tag{1}$$

We also note the following:

$$P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) = \frac{1}{\sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right)} \tag{2}$$

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{\sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right)} \tag{3}$$

Accordingly, using the formulas of (1) to (3), we may redefine the derivative $\partial L(\mathbf{w})/\partial w_{ji}$ as follows:

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T}\sum_{\mathbf{x}_\alpha \in T}\left(\sum_{\mathbf{x}_\beta} P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) x_j x_i - \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) x_j x_i\right)$$

which is the desired result.

Problem 11.10

(a) Factoring the transition process from state *i* to state *j* into a two-step process, we may express the transition probability p_{ji} as

$$p_{ji} = \tau_{ji} q_{ji} \qquad \text{for } j \neq i \tag{1}$$

where τ_{ji} is the probability that a transition from state *j* to state *i* is attempted, and q_{ji} is the conditional probability that the attempt is successful given that it was attempted. When $j = i$, the property that each row of the stochastic matrix must add to unity implies that

$$\begin{aligned} p_{ii} &= 1 - \sum_{j \neq i} p_{ij} \\ &= 1 - \sum_{j \neq i} \tau_{ij} q_{ij} \end{aligned}$$

(b) We require that the attempt-rate matrix be symmetric:

$$\tau_{ji} = \tau_{ij} \qquad \text{for all } i \neq j \tag{2}$$

and that it satisfies the normalization condition

$$\sum_j \tau_{ji} = 1 \qquad \text{for all } i \neq j$$

We also require the property of complementary conditional transition probability

$$q_{ji} = 1 - q_{ij} \tag{3}$$

For a stationary distribution, we have

$$\pi_i = \sum_j \pi_j p_{ji} \qquad \text{for all } i \tag{4}$$

Hence, using (1) to (3) in (4):

$$\begin{aligned}\pi_i &= \sum_j \pi_j \tau_{ji} p_{ji} \\ &= \sum_j \pi_j \tau_{ji}(1 - q_{ij}) \qquad (5)\end{aligned}$$

Next, recognizing that

$$\sum_j p_{ij} = 1 \qquad \text{for all } i$$

we may go on to write

$$\begin{aligned}\pi_i &= \pi_i \sum_j p_{ij} \\ &= \sum_j \pi_i p_{ij} \\ &= \sum_j \pi_i \tau_{ij} q_{ij} \qquad (6)\end{aligned}$$

Hence, combining (5) and (6), using the symmetry property of (2), and then rearranging terms:

$$\sum_j \tau_{ji}(\pi_i q_{ij} + \pi_j q_{ij} - \pi_j) = 0 \qquad (7)$$

(c) For $\tau_{ji} \neq 0$ the condition of (7) can only be satisfied if

$$\pi_i q_{ij} + \pi_j q_{ij} - \pi_j = 0$$

which, in turn, means that q_{ij} is defined by

$$q_{ij} = \frac{1}{1 + (\pi_i / \pi_j)} \qquad (8)$$

(d) Make a change of variables:

$$E_i = -T \log \pi_i + T^*$$

where T and T^* are arbitrary constants. We may then express π_i in terms of E_i as

$$\pi_i = \frac{1}{Z}\exp\left(-\frac{E_i}{T}\right)$$

where

$$Z = \exp\left(-\frac{T^*}{T}\right)$$

Accordingly, we may reformulate (8) in the new form

$$\begin{aligned} q_{ij} &= \frac{1}{1 + \exp\left(-\frac{1}{T}(E_i - E_j)\right)} \\ &= \frac{1}{1 + \exp(-\Delta E / T)} \end{aligned} \tag{9}$$

where $\Delta E = E_i - E_j$. To evaluate the constant Z, we note that

$$\sum_i \pi_i = 1$$

and therefore

$$Z = \sum_i \exp(-E_i / T)$$

(e) The formula of (9) is the only possible distribution for state transitions in the Boltzmann machine; it is recognized as the Gibbs distribution.

Problem 11.11

We start with the Kullback-Leibler divergence

$$D_{p^+||p^-} = \sum_\alpha p_\alpha^+ \log\left(\frac{p_\alpha^+}{p_\alpha^-}\right) \tag{1}$$

The probability distribution p_α^+ in the clamped condition is naturally independent of the synaptic weights w_{ji} in the Boltzman machine, whereas the probability distribution p_α^- is dependent on w_{ji}. Hence differentiating (1) with respect to w_{ji}:

$$\frac{\partial D_{p^+||p^-}}{\partial w_{ji}} = -\sum_{\alpha} \frac{p_{\alpha}^{+}}{p_{\alpha}^{-}} \frac{p_{\alpha}^{-}}{\partial w_{ji}} \tag{2}$$

To minimize $D_{p^+||p^-}$, we use the method of gradient descent:

$$\begin{aligned}\Delta w_{ji} &= -\varepsilon \frac{D_{p^+||p^-}}{\partial w_{ji}} \\ &= \varepsilon \sum_{\alpha} \frac{p_{\alpha}^{+}}{p_{\alpha}^{-}} \frac{p_{\alpha}^{-}}{\partial w_{ji}} \end{aligned} \tag{3}$$

where ε is a positive constant.

Let $p_{\alpha\beta}^{-}$ denote the joint probability that the visible neurons are in state α and the hidden neurons are in state β, given that the network is in its clamped condition. We may then write

$$p_{\alpha}^{-} = \sum_{\beta} p_{\alpha\beta}^{-}$$

Assuming that the network is in thermal equilibrium, we may use the Gibbs distribution

$$p_{\alpha\beta}^{-} = \frac{1}{Z} \exp\left(-\frac{E_{\alpha\beta}}{T}\right)$$

to write

$$p_{\alpha}^{-} = \frac{1}{Z} \sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right) \tag{4}$$

where $E_{\alpha\beta}$ is the energy of the network when the visible neurons are in state α and the hidden neurons are in state β. The partition function Z is itself defined by

$$Z = \sum_{\alpha} \sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right)$$

The energy $E_{\alpha\beta}$ is defined in terms of the synaptic weights w_{ji} by

$$E_{\alpha\beta} = -\frac{1}{2}\sum_{\substack{i \\ i \neq j}} \sum_{j} w_{ji} x_{j|\alpha\beta} x_{i|\alpha\beta} \tag{5}$$

where $x_{i|\alpha\beta}$ is the state of neuron i when the visible neurons are in state α and the hidden neurons are in state β. Therefore, using (4):

$$\frac{p_{\alpha}^{-}}{\partial w_{ji}} = -\frac{1}{ZT}\sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right)\frac{\partial E_{\alpha\beta}}{\partial w_{ji}} - \frac{1}{Z^2}\ \frac{\partial Z}{\partial w_{ji}}\sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right) \tag{6}$$

From (5) we have (remembering that in a Boltzmann machine $w_{ji} = w_{ij}$)

$$\frac{\partial E_{\alpha\beta}}{\partial w_{ji}} = -x_{j|\alpha\beta} x_{i|\alpha\beta} \tag{7}$$

The first term on the right-hand side of (6) is therefore

$$\begin{aligned} -\frac{1}{ZT}\sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right)\frac{\partial E_{\alpha\beta}}{\partial w_{ji}} &= +\frac{1}{ZT}\sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right) x_{j|\alpha\beta} x_{i|\alpha\beta} \\ &= \frac{1}{T}\sum_{\beta} p_{\alpha\beta}^{-} x_{j|\alpha\beta} x_{i|\alpha\beta} \end{aligned}$$

where we have made use of the Gibbs distribution

$$p_{\alpha\beta}^{-} = \frac{1}{Z}\exp\left(-\frac{E_{\alpha\beta}}{T}\right)$$

as the probability that the visible neurons are in state α and the hidden neurons are in state β in the free-running condition. Consider next the second term on the right-hand side of (6). Except for the minus sign, we may express this term as the product of two factors:

$$\frac{1}{Z^2}\ \frac{\partial Z}{\partial w_{ji}}\sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right) = \left[\frac{1}{Z}\sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right)\right]\left[\frac{1}{Z}\frac{\partial Z}{\partial w_{ji}}\right] \tag{8}$$

The first factor in (8) is recognized as the Gibbs distribution p_{α}^{-} defined by

$$p_{\alpha}^{-} = \frac{1}{Z}\sum_{\beta} \exp\left(-\frac{E_{\alpha\beta}}{T}\right) \tag{9}$$

To evaluate the second factor in (8), we write

$$\begin{aligned}\frac{1}{Z}\frac{\partial Z}{\partial w_{ji}} &= \frac{1}{Z}\frac{\partial}{\partial w_{ji}}\sum_{\alpha}\sum_{\beta}\exp\left(-\frac{E_{\alpha\beta}}{T}\right) \\ &= -\frac{1}{TZ}\sum_{\alpha}\sum_{\beta}\exp\left(-\frac{E_{\alpha\beta}}{T}\right)\frac{\partial E_{\alpha\beta}}{\partial w_{ji}} \\ &= \frac{1}{TZ}\sum_{\alpha}\sum_{\beta}\exp\left(-\frac{E_{\alpha\beta}}{T}\right)x_{j|\alpha\beta}x_{i|\alpha\beta} \\ &= \frac{1}{T}\sum_{\alpha}\sum_{\beta}p^{-}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta} \qquad (10)\end{aligned}$$

Using (9) and (10) in (8):

$$\frac{1}{Z^2}\frac{\partial Z}{\partial w_{ji}}\sum_{\beta}\exp\left(-\frac{E_{\alpha\beta}}{T}\right) = \frac{p^{-}_{\alpha}}{T}\sum_{\alpha}\sum_{\beta}p^{-}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta} \qquad (11)$$

We are now ready to revisit (6) and thus write

$$\frac{\partial p^{-}_{\alpha}}{\partial w_{ji}} = \frac{1}{T}\sum_{\beta}p^{-}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta} - \frac{p^{-}_{\alpha}}{T}\sum_{\alpha}\sum_{\beta}p^{-}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta}$$

We now make the following observations:

1. The sum of probability p^{+}_{α} over the states α is unity, that is,

   $$\sum_{\alpha}p^{+}_{\alpha} = 1 \qquad (12)$$

2. The joint probability

   $$p^{-}_{\alpha\beta} = p^{-}_{\beta|\alpha}p^{-}_{\alpha} \qquad (13)$$

   Similarly

   $$p^{+}_{\alpha\beta} = p^{+}_{\beta|\alpha}p^{+}_{\alpha} \qquad (14)$$

3. The probability of a hidden state, given some visible state, is naturally the same whether the visible neurons of the network in thermal equilibrium are clamped in that state by the external environment or arrive at that state by free running of the network, as shown by

   $$p^{-}_{\beta|\alpha} = p^{+}_{\beta|\alpha} \qquad (15)$$

   In light of this relation we may rewrite Eq. (13) as

$$\bar{p}_{\alpha\beta} = p^{+}_{\beta|\alpha}\bar{p}_{\alpha} \qquad (16)$$

Moreover, we may write

$$\left(\frac{p^{+}_{\alpha}}{\bar{p}_{\alpha}}\right)\bar{p}_{\alpha\beta} = p^{+}_{\alpha}p^{+}_{\beta|\alpha}$$

$$= p^{+}_{\alpha\beta} \qquad (17)$$

Accordingly, we may rewrite (3) as follows:

$$\Delta w_{ji} = \frac{\varepsilon}{T}\left(\sum_{\alpha}\frac{p^{+}_{\alpha}}{\bar{p}_{\alpha}}\sum_{\beta}\bar{p}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta} - \sum_{\alpha}p^{+}_{\alpha}\sum_{\alpha}\sum_{\beta}\bar{p}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta}\right)$$

$$= \frac{\varepsilon}{T}\left(\sum_{\alpha}\sum_{\beta}p^{+}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta} - \sum_{\alpha}\sum_{\beta}\bar{p}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta}\right)$$

Define the following terms:

η = learning rate parameter

$= \frac{\varepsilon}{T}$

$\rho^{+}_{ji} = \langle x_{j|\alpha\beta}x_{i|\alpha\beta}\rangle^{+}$

$= \sum_{\alpha}\sum_{\beta}p^{+}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta}$

$\rho^{-}_{ji} = \langle x_{j|\alpha\beta}x_{i|\alpha\beta}\rangle^{-}$

$= \sum_{\alpha}\sum_{\beta}\bar{p}_{\alpha\beta}x_{j|\alpha\beta}x_{i|\alpha\beta}$

We may then finally formulate the Boltzmann learning rule as

$$\Delta w_{ji} = \eta(\rho^{+}_{ji} - \rho^{-}_{ji})$$

Problem 11.12

(a) We start with the relative entropy:

$$D_{p^+||p^-} = \sum_\alpha \sum_\gamma p^+_{\alpha\gamma} \log\left(\frac{p^+_{\alpha\gamma}}{p^-_{\alpha\gamma}}\right) \qquad (1)$$

From probability theory, we have

$$p^+_{\alpha\gamma} = p^+_{\gamma|\alpha} p^+_\alpha \qquad (2)$$

$$p^-_{\alpha\gamma} = p^-_{\gamma|\alpha} p^-_\alpha = p^-_{\gamma|\alpha} p^+_\alpha \qquad (3)$$

where, in the last line, we have made use of the fact that the input neurons are always clamped to the environment, which means that

$$p^-_\alpha = p^+_\alpha$$

Substituting (2) and (3) into (1):

$$D_{p^+||p^-} = \sum_\alpha p^+_\alpha \sum_\gamma p^+_{\gamma|\alpha} \log\left(\frac{p^+_{\gamma|\alpha}}{p^-_{\gamma|\alpha}}\right) \qquad (4)$$

where the state α refers to the input neurons and γ refers to the output neurons.

(b) With $p_{\gamma|\alpha}$ denoting the conditional probability of finding the output neurons in state γ, given that the input neurons are in state α, we may express the probability distribution of the output states as

$$p^-_\gamma = \sum_\alpha p^-_{\gamma|\alpha} p^-_\alpha$$

The conditional $p^-_{\gamma|\alpha}$ is determined by the synaptic weights of the network in accordance with the formula

$$p^-_{\gamma|\alpha} = \frac{1}{Z_{1\alpha}} \sum_\beta \exp\left(-\frac{E_{\gamma\beta\alpha}}{T}\right) \qquad (5)$$

where

$$E_{\gamma\beta\alpha} = \frac{1}{2} \sum_j \sum_i w_{ji} [s_j s_i]_{\gamma\beta\alpha} \qquad (6)$$

The parameter $Z_{1\alpha}$ is the partition function:

$$\frac{1}{Z_{1\alpha}} = \sum_{\beta}\sum_{\gamma}\exp\left(-\frac{E_{\gamma\beta\alpha}}{T}\right) \tag{7}$$

The function of the Boltzmann machine is to find the synaptic weights for which the conditional probability $\overset{-}{p}_{\gamma|\alpha}$ approaches the desired value $\overset{+}{p}_{\gamma|\alpha}$.

Applying the gradient method to the relative entropy of (1):

$$\Delta w_{ji} = -\varepsilon \frac{\partial D_{p^{+}||p^{-}}}{\partial w_{ji}} \tag{8}$$

Using (4) in (8) and recognizing that $\overset{+}{p}_{\gamma|\alpha}$ is determined by the environment (i.e., it is independent of the network), we get

$$\Delta w_{ji} = \varepsilon \sum_{\alpha} \overset{+}{p}_{\alpha} \sum_{\gamma} \frac{\overset{+}{p}_{\gamma|\alpha}}{\overset{-}{p}_{\gamma|\alpha}} \frac{\overset{-}{p}_{\gamma|\alpha}}{\partial w_{ji}} \tag{9}$$

To evaluate the partial derivative $\partial \overset{-}{p}_{\gamma|\alpha}/\partial w_{ji}$ we use (5) to (7):

$$\begin{aligned}\frac{\overset{-}{p}_{\gamma|\alpha}}{\partial w_{ji}} &= \frac{1}{Z_{1\alpha}}\sum_{\beta}\left(-\frac{1}{T}\right)\exp\left(-\frac{E_{\gamma\beta\alpha}}{T}\right)\frac{E_{\gamma\beta\alpha}}{\partial w_{ji}} \\ &\quad -\frac{1}{Z_{1\alpha}^{2}} \frac{\partial Z_{1\alpha}}{\partial w_{ji}}\sum_{\beta}\exp\left(-\frac{E_{\gamma\beta\alpha}}{T}\right) \\ &= \frac{1}{Z_{1\alpha}} \sum_{\beta}\frac{1}{T}[s_j s_i]_{\gamma\beta\alpha}\exp\left(-\frac{E_{\gamma\beta\alpha}}{T}\right) \\ &\quad +\frac{1}{Z_{1\alpha}^{2}} \sum_{\beta}\sum_{\gamma}[s_j s_i]_{\gamma\beta\alpha}\exp\left(-\frac{E_{\gamma\beta\alpha}}{T}\right)\end{aligned} \tag{10}$$

Next, we recognize the following pair of relations:

$$\frac{1}{Z_{1\alpha}}\exp\left(-\frac{E_{\gamma\beta\alpha}}{T}\right) = \overset{-}{p}_{\gamma\beta 1\alpha} \tag{11}$$

$$\frac{1}{Z_{1\alpha}} \sum_{\beta}\sum_{\gamma}[s_j s_i]_{\gamma\beta\alpha}\exp\left(-\frac{E_{\gamma\beta\alpha}}{T}\right) = \langle s_j s_i\rangle_{\alpha}^{-}\; p_{\gamma|\alpha}^{-} \tag{12}$$

where the term $\langle s_j s_i\rangle_{\alpha}^{-}$ is the averaged correlation of the states s_j and s_i with the input neurons clamped to state α and the network in a free-running condition. Substituting (11) and (12) in (10):

$$\frac{\partial p_{\gamma|\alpha}^{-}}{\partial w_{ji}} = \frac{1}{T}\left(\sum_{\beta}[s_j s_i]_{\gamma\beta\alpha}p_{\gamma\beta\alpha}^{-} - \langle s_j s_i\rangle_{\alpha}^{-}\; p_{\gamma|\alpha}^{-}\right) \tag{13}$$

Next, substituting (13) into (9):

$$\Delta w_{ji} = \frac{\varepsilon}{T}\left\{\sum_{\alpha}p_{\alpha}^{+}\sum_{\alpha}\sum_{\beta}[s_j s_i]_{\gamma\beta\alpha}p_{\gamma\beta 1\alpha}^{-}\left(\frac{p_{\gamma|\alpha}^{+}}{p_{\gamma|\alpha}^{-}}\right)\right.$$
$$\left. -\sum_{\alpha}p_{\alpha}^{+}\langle s_j s_i\rangle_{\alpha}^{-}\;\sum_{\gamma}p_{\gamma|\alpha}^{+}\right\} \tag{14}$$

We now recognize that

$$\sum_{\alpha}p_{\alpha}^{+} = 1 \qquad \text{for all } \alpha \tag{15}$$

$$\begin{aligned}\sum_{\gamma}\sum_{\beta}[s_j s_i]_{\gamma\beta\alpha}p_{\gamma\beta|\alpha}^{-}\left(\frac{p_{\gamma|\alpha}^{+}}{p_{\gamma|\alpha}^{-}}\right) &= \sum_{\gamma}p_{\gamma|\alpha}^{+}\sum_{\beta}[s_j s_i]_{\gamma\beta\alpha}\left(\frac{p_{\gamma\beta|\alpha}^{+}}{p_{\gamma|\alpha}^{-}}\right)\\ &= \sum_{\gamma}p_{\gamma|\alpha}^{+}\langle s_j s_i\rangle_{\gamma\alpha}\\ &= \langle s_j s_i\rangle_{\alpha}^{+}\end{aligned} \tag{16}$$

Accordingly, substituting (15) and (16) into (14):

$$\begin{aligned}\Delta w_{ji} &= \frac{\varepsilon}{T}\sum_{\alpha}p_{\alpha}^{+}(\langle s_j s_i\rangle_{\alpha}^{+} - \langle s_j s_i\rangle_{\alpha}^{-})\\ &= \eta\sum_{\alpha}p_{\alpha}^{+}(\rho_{ji|\alpha}^{+} - \rho_{ji|\alpha}^{-})\end{aligned}$$

where $\eta = \varepsilon/T$; and $\rho^{+}_{ji|\alpha}$ and $\rho^{-}_{ji|\alpha}$ are the averaged correlations in the clamped and free-running conditions, given that the input neurons are in state α.

Problem 11.15

Consider the expected distortion (energy)

$$E = \sum_{\mathbf{x}} \sum_{j} P(\mathbf{x} \in C_j) d(\mathbf{x}, \mathbf{y}_j) \tag{1}$$

where $d(\mathbf{x}, \mathbf{y}_j)$ is the distortion measure for representing the data point $\mathbf{x}$ by the vector $\mathbf{y}_j$, and $P(\mathbf{x} \in C_j)$ is the probability that $\mathbf{x}$ belongs to the cluster of points represented by $\mathbf{y}_j$. To determine the association probabilities at a given expected distortion, we maximize the entropy subject to the constraint of (1). For a fixed $Y = \{\mathbf{y}_j\}$, we assume that the association probabilities of different data points are independent. We may thus express the entropy as

$$H = -\sum_{\mathbf{x}} \sum_{j} P(\mathbf{x} \in C_j) \log P(\mathbf{x} \in C_j) \tag{2}$$

The probability distribution that maximizes the entropy under the expectation constraint is the Gibbs distribution:

$$P(\mathbf{x} \in C_j) = \frac{1}{Z_{\mathbf{x}}} \exp\left(-\frac{1}{T} d(\mathbf{x}, \mathbf{y}_j)\right) \tag{3}$$

where

$$Z_{\mathbf{x}} = \sum_{j} \exp\left(-\frac{1}{T} d(\mathbf{x}, \mathbf{y}_j)\right)$$

is the partition function. The inverse temperature $B = 1/T$ is the Lagrange multiplier defined by the value of E in (1).

Problem 11.6

(a) The free energy is

$$F = D - TH \tag{1}$$

where D is the expected distortion, T is the temperature, and H is the conditional entropy. The expected distortion is defined by

$$D = \sum_{\mathbf{x}}\sum_{\mathbf{y}} P(\mathbf{X} = \mathbf{x})P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})d(\mathbf{x}, \mathbf{y}) \tag{2}$$

The conditional entropy if defined by

$$H(\mathbf{Y}|\mathbf{X}) = -\sum_{\mathbf{x}}\sum_{\mathbf{y}} P(\mathbf{X} = \mathbf{x})P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})\log P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}) \tag{3}$$

The minimizing $P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})$ is itself defined by the Gibbs distribution:

$$P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}) = \frac{1}{Z_{\mathbf{x}}}\exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right) \tag{4}$$

where

$$Z_{\mathbf{x}} = \sum_{y}\exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right) \tag{5}$$

is the partition function. Substituting (2) to (5) into (1), we get

$$\begin{aligned} F^* &= \sum_{\mathbf{x}}\sum_{\mathbf{y}} P(\mathbf{X} = \mathbf{x})\frac{1}{Z_{\mathbf{x}}}\exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right)d(\mathbf{x}, \mathbf{y}) \\ &\quad + T\sum_{\mathbf{x}}\sum_{\mathbf{y}} P(\mathbf{X} = \mathbf{x})\frac{1}{Z_{\mathbf{x}}}\exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right)\left(-\log Z_{\mathbf{x}} - \frac{1}{T}d(\mathbf{x}, \mathbf{y})\right) \\ &= T\sum_{\mathbf{x}}\sum_{\mathbf{y}} P(\mathbf{X} = \mathbf{x})\frac{1}{Z_{\mathbf{x}}}\exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right)(-\ \log Z_{\mathbf{x}}) \end{aligned}$$

This result simplifies as follows by virtue of the definition given in (5) for the partition function:

$$F^* = -T\sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x})\log Z_{\mathbf{x}} \tag{6}$$

(b) Differentiating the minimum free energy F^* of (6) with respect to $\mathbf{y}$:

$$\frac{\partial F^*}{\partial \mathbf{y}} = -T\sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x})\frac{1}{Z_{\mathbf{x}}}\ \frac{\partial Z_{\mathbf{x}}}{\partial \mathbf{y}} \tag{7}$$

Using the definition of $Z_{\mathbf{x}}$ given in (5), we write:

$$\frac{\partial Z_{\mathbf{x}}}{\partial \mathbf{y}} = -\frac{1}{T}\sum_{\mathbf{y}}\exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right)\frac{\partial d(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} \tag{8}$$

Hence, we may rewrite (7) as

$$\frac{\partial F^*}{\partial \mathbf{y}} = \sum_{\mathbf{x}} \sum_{\mathbf{y}} P(\mathbf{X} = \mathbf{x}) \frac{1}{Z_{\mathbf{x}}} \exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right) \frac{\partial d(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}}$$

$$= \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = x) \frac{\partial d(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} \tag{9}$$

where use has been made of (4). Noting that

$$P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) = P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x}) P(\mathbf{X} = \mathbf{x})$$

we may then state that the condition for minimizing the Lagrangian with respect to **y** is

$$\sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) \frac{\partial d(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} = \mathbf{0} \qquad \text{for all } \mathbf{y} \tag{10}$$

Normalizing this result with respect to $P(\mathbf{X} = \mathbf{x})$ we get the minimizing condition:

$$\sum_{\mathbf{x}} P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x}) \frac{\partial d(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} = \mathbf{0} \qquad \text{for all } \mathbf{y} \tag{11}$$

(c) Consider the squared Euclidean distortion

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2 = (\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})$$

for which we have

$$\frac{\partial d(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} = \frac{\partial}{\partial \mathbf{y}} (\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})$$

$$= -2((\mathbf{x} - \mathbf{y})) \tag{12}$$

For this particular measure we find it more convenient to normalize (10) with respect to the probability

$$P(\mathbf{Y} = \mathbf{y}) = \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y})$$

We may then write the minimizing condition with respect to **y** as

$$\sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}) \frac{\partial d(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} = 0 \tag{13}$$

Using (12) in (13) and solving for **y**, we get the desired minimizing solution

$$\mathbf{y} = \frac{\sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y})\mathbf{x}}{\sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y})} \tag{14}$$

which is recognized as the formula for a centroid.

Problem 11.17

The advantage of deterministic annealing over maximum likelihood is that it does not make any assumption on the underlying probability distribution of the data.

Problem 11.18

(a) Let

$$\varphi_k(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} = \mathbf{t}_k\|^2\right), \qquad k = 1, 2, ..., K$$

where $\mathbf{t}_k$ is the center or prototype vector of the *k*th radial basis function and *K* is the number of such functions (i.e., hidden units). Define the normalized radial basis function

$$P_k(\mathbf{x}) = \frac{\varphi_k(\mathbf{x})}{\sum_k \varphi_k(\mathbf{x})}$$

The average squared cost over the training set is

$$d = \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{y}_i - \mathbf{F}(\mathbf{x}_i)\|^2 \tag{1}$$

where $\mathbf{F}(\mathbf{x}_i)$ is the output vector of the RBF network in response to the input $\mathbf{x}_i$. The Gibbs distribution for $P(\mathbf{x} \in R)$ is

$$P(\mathbf{x} \in R) = \frac{1}{Z_{\mathbf{x}}}\exp\left(-\frac{d}{T}\right) \tag{2}$$

where *d* is defined in (1) and

$$Z_{\mathbf{x}} = \sum_{\mathbf{y}_i} \exp\left(-\frac{d}{T}\right) \tag{3}$$

(b) The Lagrangian for minimizing the average misclassification cost is

$F = d - TH$

where the average squared cost d is defined in (1), and the entropy H is defined by

$$H = -\sum_{\mathbf{x}} \sum_{j} p(j|\mathbf{x}) \log(j|\mathbf{x})$$

where $p(j|\mathbf{x})$ is the probability of associating class j at the output of the RBF network with the input **x**.

CHAPTER 12
Dynamic Programming

Problem 12.1

As the discount factor γ approaches 1, the computation of the cost-to-go function $J^{\pi}(i)$ becomes longer because of the corresponding increase in the number of time steps involved in the computation.

Problem 12.2

(a) Let π be an arbitrary policy, and suppose that this policy chooses action $a \in A_i$ at time step 0. We may then write

$$J^{\pi}(i) = \sum_{a \in A_i} p_a \left(c(i, a) + \sum_{j=1}^{N} p_{ij}(a) W^{\pi}(j) \right)$$

where p_a is the probability of choosing action a, $c(i, a)$ is the expected cost, $p_{ij}(a)$ is the probability of transition from state i to state j under action a, $W^{\pi}(j)$ is the expected cost-to-go function from time step $n = 1$ onward, and j is the state at that time step. We now note that

$$W^{\pi}(j) \le \gamma \ J(j)$$

which follows from the observation that if the state at time step $n = 1$ is j, then the situation at that time step is the same as if the process had started in state j with the exception that all the returns are multiplied by the discount factor γ. Hence, we have

$$\begin{aligned} J^{\pi}(i) &\ge p_a \left(c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \\ &\ge p_a \min_{a \in A_i} \left(c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \\ &= \min_a \left(c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \end{aligned}$$

which implies that

$$J^{\pi}(i) \min_{a \in A_i} \ge \left(c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \tag{1}$$

(b) Suppose we next go the other way by choosing a_0 with

$$c(i, a_0) + \gamma \sum_j p_{ij}(a_0) = \min_a \left(c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \qquad \mathbf{(2)}$$

Let π be the policy that chooses a_0 at time step 0 and, if the next state is j, the process is viewed as originating in state j following a policy π_j such that

$$J^{\pi_j} \le J(j) + \varepsilon$$

Where ε is a small positive number. Hence

$$\begin{aligned} J^{\pi_j} &= c(i, a_0) + \sum_{j=1}^{N} p_{ij}(a_0) J^{\pi_j}(j) \\ &\le c(i, a_0) + \sum_{j=1}^{N} p_{ij}(a_0) J(j) + \gamma\varepsilon \end{aligned} \qquad (3)$$

Since $J(i) \le J^{\pi}(i)$, (3) implies that

$$J(i) \le c(i, a_0) + \gamma \sum_{j=1}^{N} p_{ij}(a_0) J(j) + \gamma\varepsilon$$

Hence, from (2) it follows that

$$J(i) \le \min_{a \in A_i} \left(c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) + \gamma\varepsilon \qquad (4)$$

(c) Finally, since ε is arbitrary, we immediately deduce from (1) and (4) that the optimum cost-to-go function

$$J^*(i) = \min_a \left(c(i, a) + \gamma \sum_j p_{ij}(a) J^*(j) \right) \qquad (5)$$

which is the desired result

Problem 12.3

Writing the system of N simultaneous equations (12.22) of the text in matrix form:

$$\mathbf{J}^{\propto} = \mathbf{c}(\propto) + \gamma\mathbf{P}(\propto)\mathbf{J}^{\propto} \tag{1}$$

where

$$\mathbf{J}^{(\propto)} = \left[J^{(\propto)}(1),\ J^{(\propto)}(2),\ \dots\ J^{(\propto)}(N)\right]^T$$

$$\mathbf{c}(\propto) = \left[\mathbf{C}(1,\propto),\ \mathbf{C}(2,\propto),\ \dots\ \mathbf{C}(N,\propto)\right]^T$$

$$\mathbf{P}(\propto) = \begin{bmatrix} p_{11}(\propto) & p_{12}(\propto) & \dots & p_{1N}(\propto) \\ p_{21}(\propto) & p_{22}(\propto) & \dots & p_{2N}(\propto) \\ \vdots & \vdots & & \vdots \\ p_{N1}(\propto) & p_{N2}(\propto) & \dots & p_{NN}(\propto) \end{bmatrix}$$

Rearranging terms in 1), we may write

$$\mathbf{I} - \gamma\mathbf{P}(\propto)\mathbf{J}^{(\propto)} = \mathbf{c}(\propto)$$

where **I** is the *N*-by-*N* identity matrix. For the solution $\mathbf{J}^{\propto}$ to be unique we require that the *N*-by-*N* matrix (**I** - γ**P**(∝)) have an inverse matrix for all possible values of the discount factor γ.

Problem 12.4

Consider an admissible policy $\{\propto_0, \propto_1, \dots\}$, a positive integer *K*, and cost-to-go function *J*. Let the costs of the first *K* stages be accumulated, and add the terminal cost $\gamma^K J(X_K)$, thereby obtaining the total expected cost

$$\mathbf{E}\left[\gamma^K J(X_K) + \sum_{n=0}^{K-1} \gamma^n g(X_n, \propto_n(X_n), X_{n-1})\right]$$

where **E** is the expectational operator, To minimize the total expected cost, we start with $\gamma^K J(X_K)$ and perform *K* iterations of the dynamic programming algorithm, as shown by

$$J_n(X_n) = \min_{\propto_n} \mathbf{E}[g_n(X_n, \propto_n(X_n), X_{n-1}) + J_{n+1}(X_{n+1})] \tag{1}$$

with the initial condition

$$J_K(X) = \gamma^K J(X)$$

Now consider the function V_n defined by

$$V_n(X) = \frac{J_{K-n}(X)}{\gamma^{K-n}} \qquad \text{for all } n \text{ and } X \tag{2}$$

The function $V_n(X)$ is the optimal K-stage cost $J_0(X)$. Hence, the dynamic programming algorithm of (1) can be rewritten in terms of the function $V_n(X)$ as follows:

$$V_{n+1}(X_0) = \min_{\propto} \mathbf{E}[g(X_0, \propto(X_0), X_1) + \gamma V_n(X_1)]$$

with the initial condition

$V_0(X) = J(X)$

which has the same mathematical form as that specified in the problem.

Problem 12.5

An important property of dynamic programming is the *monotonicity* property described by

$$J^{\propto_{n+1}} \le J^{\propto_n}$$

This property follows from the fact that if the terminal cost g_K for K stages is changed to a uniformly larger cost $\bar{g}_K$, that is,

$$\bar{g}_K(X_K) \ge g_K(X_K) \qquad \text{for all } X_K,$$

then the last stage cost-to-go function $J_{K-1}(X_{K-1})$ will be uniformly increased. In more general terms, we may state the following.

Given two cost-to-go functions $J_{K'+1}$ and $\bar{J}_{K+1}$ with

$$\bar{J}_{K+1}(X_{K+1}) \ge J_{K+1}(X_{K+1}) \qquad \text{for all } X_{K+1},$$

we find that for all X_K and $\propto_K$ the following relation holds

$$\mathbf{E}[g_K(X_K, \propto_K, X_{K+1}) + J_{K+1}(X_{K+1})] \le \mathbf{E}[g_K(X_K, \propto_K, X_{K+1}) + \bar{J}_{K+1}(X_{K+1})]$$

This relation merely restates the monotonicity property of the dynamic programming algorithm.

Problem 12.6

According to (12.24) of the text the Q-factor for state-action pair (i, a) and stationary policy $\propto$ satisfies the condition

$$Q^{\propto}(i, \propto(i)) = \min_a Q^{\propto}(i, a) \qquad \text{for all } i$$

This equation emphasizes the fact that the policy $\propto$ is *greedy* with respect to the cost-to-go function $J^{\propto}(i)$.

Problem 12.7

Figure 1, shown below, presents an interesting interpretation of the policy iteration algorithm. In this interpretation, the policy evaluation step is viewed as the work of a *critic* that evaluates the performance of the current policy; that is, it calculates an estimate of the cost-to-go function $J^{\propto_n}$. The policy improvement step is viewed as the work of a *controller* or *actor* that accounts for the latest evaluation made by her critic and acts out the improved policy $\propto_{n+1}$. In short, the critic looks after policy evaluation and the controller (actor) looks after policy improvement and the iteration between them goes on.

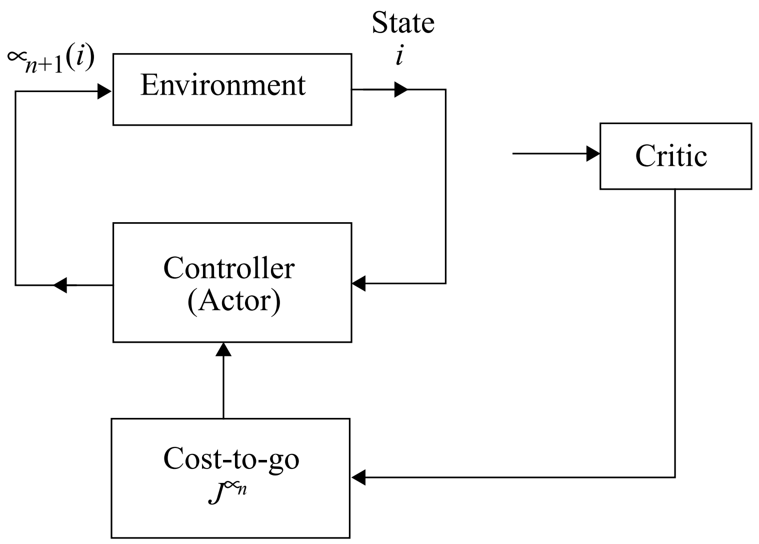


Figure 1: Problem 12.7

Problem 12.8

From (12.29) in the text, we find that for each possible state, the value iteration algorithm requires NM iterations, where N is the number of states and M is the number of admissible actions. Hence, the total number of iterations for all N states in N^2M.

Problem 12.9

To reformulate the value-iteration algorithm in terms of *Q*-factors, the only change we need to make is in step 2 of Table 12.2 in the text. Specifically, we rewrite this step as follows:

For $n = 0, 1, 2, ...,$ compute

$$Q(i, a) = c(i, a) + \gamma \sum_{j=1}^{N} p_{ij}(a) J_n(j)$$

$$J_{n+1}(i) = \min_a Q(i, a)$$

Problem 12.10

The policy-iteration algorithm alternates between two steps: policy evaluation, and policy improvement. In other words, an optimal policy is computed directly in the policy iteration algorithm. In contrast, no such thing happens in the value iteration algorithm.

Another point of difference is that in policy iteration the cost-to-go function is recomputed on each iteration of the algorithm. This burdensome computational difficulty is avoided in the value-iteration algorithm.

Problem 12.14

From the definition of *Q*-factor given in (12.24) in the text and Bellman's optimality equation (12.11), we immediately see that

$$J^*(i) = \min_a Q(i, a)$$

where the minimization is performed over all possible actions *a*.

Problem 12.15

The value-iteration algorithm requires knowledge of the state transition probabilities. In contrast, *Q*-learning operates without this knowledge. But through an interactive process, *Q*-learning learns estimates of the transition probabilities in an implicit manner. Recognizing the intimate relationship between value iteration and *Q*-learning, we may therefore view *Q*-learning as an adaptive version of the value-iteration algorithm.

Problem 12.16

Using Table 12.4 in the text, we may construct the signal-flow graph in Figure 1 for the Q-learning algorithm:

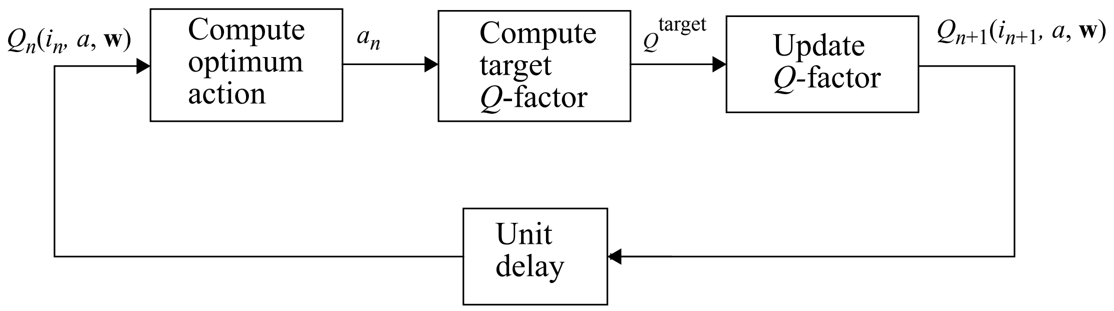

$Q_n(i_n, a, \mathbf{w})$
Compute optimum action
a_n
Compute target Q-factor
Q^{target}
Update Q-factor
$Q_{n+1}(i_{n+1}, a, \mathbf{w})$
Unit delay

Figure 1: Problem 12.16

Problem 12.17

The whole point of the Q-learning algorithm is that it eliminates the need for knowing the state transition probabilities. If knowledge of the state transition probabilities is available, then the Q-learning algorithm assumes the same form as the value-iteration algorithm.

CHAPTER 13
Neurodynamics

Problem 13.1

The equilibrium state $\mathbf{x}(0)$ is (asymptotically) stable if in a small neighborhood around $\mathbf{x}(0)$, there exists a positive definite function $V(\mathrm{x})$ such that its derivative with respect to time is negative definite in that region.

Problem 13.3

Consider the symem of coupled nonlinear differential equations:

$$\frac{dx_j}{dt} = \varphi_j(\mathbf{W}, \mathbf{i}, \mathbf{x}), \qquad j = 1, 2, ..., N$$

where $\mathbf{W}$ is the weight matrix, $\mathbf{i}$ is the bias vector, and $\mathbf{x}$ is the state vector with its jth element denoted by x_j.

(a) With the bias vector $\mathbf{i}$ treated as input and with fixed initial condition $\mathbf{x}(0)$, let $\mathbf{x}(\infty)$ denote the final state vector of the system. Then,

$$0 = \varphi_j(\mathbf{W}, \mathbf{i}, \mathbf{x}(\infty)), \qquad j = 1, 2, ..., N$$

For a given matrix $\mathbf{W}$ and input vector $\mathbf{i}$, the set of initial points $\mathbf{x}(0)$ evolves to a fixed point. The fixed points are functions of $\mathbf{W}$ and $\mathbf{i}$. Thus, the system acts as a "mapper" with $\mathbf{i}$ as input and $\mathbf{x}(\infty)$ as output, as shown in Fig. 1(a):

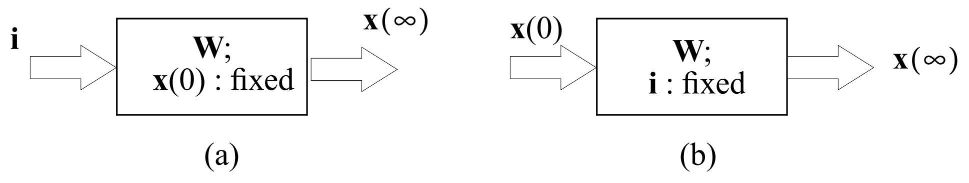


Figure 1: Problem 13.3

(b) With the initial state vector $\mathbf{x}(0)$ treated as input, and the bias vector $\mathbf{i}$ being fixed, let $\mathbf{x}(\infty)$ denote the final state vector of the system. We may then write

$$0 = \varphi_j(\mathbf{W}, \mathbf{i}\text{:fixed}, \ \mathbf{x}(\infty)), \qquad j = 1, 2, .., N$$

Thus with $\mathbf{x}(0)$ acting as input and $\mathbf{x}(\infty)$ acting as output, the dynamic system behaves like a pattern associator, as shown in Fig. 1b.

Problem 13.4

(a) We are given the fundamental memories:

$$\xi_1 = [+1,\ +1,\ +1,\ +1,\ +1]^T$$

$$\xi_2 = [+1,\ -1,\ -1,\ +1,\ -1]^T$$

$$\xi_3 = [1-,\ +1,\ -1,\ +1,\ +1]^T$$

The weight matrix of the Hopfield network (with $N = 25$ and $p = 3$) is therefore

$$\mathbf{W} = \frac{1}{N}\sum_{i=1}^{p}\xi_i\xi_i^T - \frac{P}{N}\mathbf{I}$$

$$= \frac{1}{5}\begin{bmatrix} 0 & -1 & +1 & +1 & -1 \\ -1 & 0 & +1 & +1 & +3 \\ +1 & +1 & 0 & -1 & +1 \\ +1 & +1 & -1 & 0 & +1 \\ -1 & +3 & +1 & +1 & 0 \end{bmatrix}$$

(b) According to the alignment condition, we write

$$\xi_i = \operatorname{sgn}(\mathbf{W}\xi_i), \quad i = 1, 2, 3$$

Consider first ξ_1, for which we have

$$\operatorname{sgn}(\mathbf{W}\xi_1) = \operatorname{sgn}\left(\frac{1}{5}\begin{bmatrix} 0 & -1 & +1 & +1 & -1 \\ -1 & 0 & +1 & +1 & +3 \\ +1 & +1 & 0 & -1 & +1 \\ +1 & +1 & -1 & 0 & +1 \\ -1 & +3 & +1 & +1 & 0 \end{bmatrix}\begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \end{bmatrix}\right)$$

$$= \text{sgn}\left(\frac{1}{5}\begin{bmatrix} 0 \\ +4 \\ +2 \\ +2 \\ +4 \end{bmatrix}\right) = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \end{bmatrix} = \xi_1$$

$$\text{sgn}(\mathbf{W}\xi_2) = \text{sgn}\left(\frac{1}{5}\begin{bmatrix} 0 & -1 & +1 & +1 & -1 \\ -1 & 0 & +1 & +1 & +3 \\ +1 & +1 & 0 & -1 & +1 \\ +1 & +1 & -1 & 0 & +1 \\ -1 & +3 & +1 & +1 & 0 \end{bmatrix}\begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \\ -1 \end{bmatrix}\right)$$

$$= \text{sgn}\left(\frac{1}{5}\begin{bmatrix} +2 \\ -4 \\ -2 \\ 0 \\ -4 \end{bmatrix}\right) = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \\ -1 \end{bmatrix} = \xi_2$$

$$\text{sgn}(\mathbf{W}\xi_3) = \text{sgn}\left(\frac{1}{5}\begin{bmatrix} 0 & -1 & +1 & +1 & -1 \\ -1 & 0 & +1 & +1 & +3 \\ +1 & +1 & 0 & -1 & +1 \\ +1 & +1 & -1 & 0 & +1 \\ -1 & +3 & +1 & +1 & 0 \end{bmatrix}\begin{bmatrix} -1 \\ +1 \\ -1 \\ +1 \\ +1 \end{bmatrix}\right)$$

$$= \text{sgn}\left(\frac{1}{5}\begin{bmatrix} -2 \\ +4 \\ 0 \\ +2 \\ +4 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ +1 \\ -1 \\ +1 \\ +1 \end{bmatrix} = \xi_2$$

Thus all three fundamental memories satisfy the alignment condition.

Note: Wherever a particular element of the product $\mathbf{W}\xi_i$ is zero, the neuron in question is left in its previous state.

(c) Consider the noisy probe:

$$\mathbf{x} = \left[+1,\ -1,\ +1,\ +1,\ +1\right]^T$$

which is the fundamental memory with its second element reversed in polarity. We write

$$\mathbf{Wx} = \frac{1}{5}\begin{bmatrix} 0 & -1 & +1 & +1 & -1 \\ -1 & 0 & +1 & +1 & +3 \\ +1 & +1 & 0 & -1 & +1 \\ +1 & +1 & -1 & 0 & +1 \\ -1 & +3 & +1 & +1 & 0 \end{bmatrix}\begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

$$= \frac{1}{5}\begin{bmatrix} +2 \\ +4 \\ 0 \\ 0 \\ -2 \end{bmatrix} \qquad (1)$$

Therefore,

$$\operatorname{sgn}(\mathbf{Wx}) = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$

Thus, neurons 2 and 5 want to change their states. We therefore have 2 options:

- Neuron 5 is chosen for a state change, which yields the result
  $$\mathbf{x} = \left[+1,\ +1,\ +1,\ +1,\ +1\right]^T$$
  This vector is recognized as the fundamental memory ξ_1, and the computation is thereby terminated.

- Neuron 2 is chosen to change its state, yielding the vector
  $$\mathbf{x} = \left[+1,\ -1,\ +1,\ +1,\ -1\right]^T$$
  Next, we go on to compute

$$\mathbf{Wx} = \frac{1}{5}\begin{bmatrix} 0 & -1 & +1 & +1 & -1 \\ -1 & 0 & +1 & +1 & +3 \\ +1 & +1 & 0 & -1 & +1 \\ +1 & +1 & -1 & 0 & +1 \\ -1 & +3 & +1 & +1 & 0 \end{bmatrix}\begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$

$$= \frac{1}{5}\begin{bmatrix} +4 \\ -2 \\ -2 \\ -2 \\ -2 \end{bmatrix}$$

$$\text{sgn}(\mathbf{Wx}) = \begin{bmatrix} +1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

Hence, neurons 3 and 4 want to change their states:

- If we permit neuron 3 to change its state from +1 to -1, we get
  $\mathbf{x} = [+1, -1, -1, +1, -1]^T$
  which is recognized as the fundamental memory ξ_2.

- If we permit neuron 4 to change its state from +1 to -1, we get
  $\mathbf{x} = [+1, -1, +1, -1, -1]$
  which is recognized as the negative of the third fundamental memory ξ_3.

In both cases, the new state would satisfy the alignment condition and the computation is then terminated.

Thus, when the noisy version of ξ_1 is applied to the network, with its second element changed in polarity, one of 2 things can happen with equal likelihood:

1. The original ξ_1 is recovered after 1 iteration.
2. The second fundamental memory ξ_2 or the negative of the third fundamental memory ξ_3 is recovered after 2 iterations, which, of course, is in error.

Problem 13.5

Given the probe vector

$$\mathbf{x} = [+1, -1, +1, +1, +1]^T$$

and the weight matrix of (1) Problem 13.4, we find that

$$\mathbf{Wx} = \frac{1}{5}\begin{bmatrix} 2 \\ 4 \\ 0 \\ 0 \\ -2 \end{bmatrix}$$

and

$$\text{sgn}(\mathbf{Wx}) = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$

According to this result, neurons 2 and 5 have changed their states. In synchronous updating, this is permitted. Thus, with the new state vector

$$\mathbf{x} = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$

on the next iteration,we compute

$$\mathbf{Wx} = \frac{1}{5}\begin{bmatrix} 0 & -1 & +1 & +1 & -1 \\ -1 & 0 & +1 & +1 & +3 \\ +1 & +1 & 0 & -1 & +1 \\ +1 & +1 & -1 & 0 & +1 \\ -1 & +3 & +1 & +1 & 0 \end{bmatrix}\begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$

$$= \frac{1}{5}\begin{bmatrix} +2 \\ -2 \\ 0 \\ 0 \\ +4 \end{bmatrix}$$

Hence,

$$\mathrm{sgn}(\mathbf{W}\mathbf{x}) = \begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

The new state vector is therefore

$$\mathbf{x} = \begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

which is recognized as the original probe. In this problem, we thus find that the network experiences a limit cycle of duration 2.

Problem 13.6

(a) The vectors

$$\xi_1 = [-1, -1, -1, +1, -1]^T$$
$$\xi_2 = [+1, +1, +1, -1, +1]^T$$
$$\xi_3 = [+1, -1, +1, -1, -1]^T$$

are simply the negatives of the three fundamental memories considered in Problem 13.4, respectively. These 3 vectors are therefore also fundamental memories of the Hopfield network.

(b) Consider the vector

$$\mathbf{x} = [0, +1, +1, +1, +1]^T$$

which is the result of masking the first element of the fundamental memory ξ_1 of Problem 13.4. According to our notation, a neuron of the Hopfield network is in either state +1 or -1. We therefore have the choice of setting the zero element of **x** to +1 or -1. The first option restores the vector **x** to its original form: fundamental memory ξ_1, which satisfies the alignment condition. Alternatively, we may set the zero element equal to -1, obtaining

$$\mathbf{x} = [-1, +1, +1, +1, +1]^T$$

In this latter case, the alignment condition is not satisfied. The obvious choice is therefore the former one.

Problem 13.7

We are given

$$\mathbf{W} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$

(a) For state $\mathbf{s}_2$ we have

$$\mathbf{W}\mathbf{s}_2 = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} -1 \\ +1 \end{bmatrix}$$
$$= \begin{bmatrix} -1 \\ +1 \end{bmatrix}$$

which yields

$$\text{sgn}(\mathbf{W}\mathbf{s}_2) = \begin{bmatrix} -1 \\ +1 \end{bmatrix} = \mathbf{s}_2$$

Next for state $\mathbf{s}_4$, we have

$$\mathbf{Ws}_4 = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} +1 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} +1 \\ -1 \end{bmatrix}$$

which yields

$$\text{sgn}(\mathbf{Ws}_4) = \begin{bmatrix} +1 \\ -1 \end{bmatrix} = \mathbf{s}_4$$

Thus, both states $\mathbf{s}_2$ and $\mathbf{s}_4$ satisfy the alignment condition and are therefore stable.

Consider next the state $\mathbf{s}_1$, for which we write

$$\mathbf{Ws}_1 = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

which yields

$$\text{sgn}(\mathbf{Ws}_1) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \mathbf{s}_1$$

Thus, both neurons want to change; suppose we pick neuron 1 to change its state, yielding the new state vector $[-1, +1]^T$. This is a stable vector as it satisfies the alignment condition. If, however, we permit neuron 2 to change its state, we get a state vector equal to $\mathbf{s}_4$. Similarly, we may show that the state vector $\mathbf{s}_3 = [-1, -1]^T$ is also unstable. The resulting state-transition diagram of the network is thus as depicted in Fig. 1.

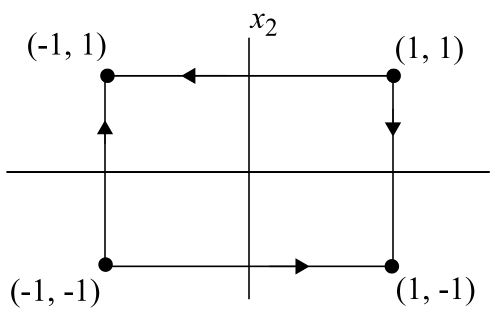


Figure 1: Problem 13.7

The results depicted in Fig. 1 assume the use of asynchronous updating. If, however, we use synchronous updating, we find that in the case of $\mathbf{s}_1$:

$$\text{sgn}(\mathbf{W}\mathbf{s}_1) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

Permitting both neurons to change state, we get the new state vector $[-1, -1]^T$. This is recognized to be stable state $\mathbf{s}_3$. Now, we find that

$$\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} +1 \\ +1 \end{bmatrix}$$

which takes back to state $\mathbf{s}_1$.

Thus, in the synchronous updating case, the states $\mathbf{s}_1$ and $\mathbf{s}_3$ represent a limit cycle with length 2.

Returning to the normal operation of the Hopfield network, we note that the energy function of the network is

$$\begin{aligned} E &= -\frac{1}{2}\sum_{i}\sum_{\substack{j \\ i \neq j}} w_{ji} s_i s_j \\ &= -\frac{1}{2} w_{12} s_1 s_2 - \frac{1}{2} w_{21} s_2 s_1 \\ &= -w_{12} s_1 s_2 \quad \text{since } w_{12} = w_{21} \\ &= s_1 s_2 \qquad (1) \end{aligned}$$

Evaluating (1) for all possible states of the network, we get the following table:

State	Energy
[+1, +1]	+1
[-1, +1]	-1
[-1. -1]	+1
[+1. -1]	-1

Thus, states $\mathbf{s}_1$ and $\mathbf{s}_3$ represent global minima and are therefore stable.

Problem 13.8

The energy function of the Hopfield network is

$$E = -\frac{1}{2}\sum_i \sum_j w_{ji} s_j s_i \tag{1}$$

The overlap m_v is defined by

$$m_v = \frac{1}{N}\sum_j s_j \xi_{v,\,j} \tag{2}$$

and the weight w_{ji} is itself defined by

$$w_{ji} = \frac{1}{N}\sum_v \xi_{v,\,j}\xi_{v,\,i} \tag{3}$$

Substituting (3) into (1) yields

$$\begin{aligned} E &= -\frac{1}{2N}\sum_i \sum_j \sum_v \xi_{v,\,j}\xi_{v,\,i} s_j s_i \\ &= -\frac{1}{2N}\sum_v \left(\sum_i s_i \xi_{v,\,i}\right)\left(\sum_j s_j \xi_{v,\,j}\right) \\ &= -\frac{1}{2N}\sum_v (m_v N)(m_v N) \\ &= -\frac{N}{2}\sum_v m_v^2 \end{aligned}$$

where, in the third line, we made use of (2).

Problem 13.11

We start with the function (see (13.48) of the text)

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} c_{ji}\varphi_i(u_i)\varphi_j(u_j) - \sum_{j=1}^{N}\int_0^{u_j} b_j(\lambda)\varphi'_j(\lambda)d\lambda \tag{1}$$

where $\varphi'_j(\cdot)$ is the derivative of the function $\varphi_j(\cdot)$ with respect to its argument. We now differentiate the function E with respect to time t and note the following relations:

1. $C_{ji} = C_{ij}$

2. $$\frac{\partial}{\partial t}\varphi_j(u_j) = \frac{\partial u_j}{\partial t}\frac{\partial}{u_j}\varphi_j(u_j)$$
$$= \frac{\partial u_j}{\partial t}\varphi'_j(u_j)$$

3. $$\frac{\partial}{\partial t}\int_0^{u_j} b_j(\lambda)\varphi'_j(\lambda)d\lambda = \frac{\partial u_j}{\partial t}\frac{\partial}{u_j}\int_0^{u_j} b_j(\lambda)\varphi'_j(\lambda)d\lambda$$
$$= \frac{\partial u_j}{\partial t}b_j(u_j)\varphi'_j(u_j)$$

Accordingly, we may use (1) to express the derivative $\partial E/\partial t$ as follows:

$$\frac{\partial E}{\partial t} = \frac{\partial u_j}{\partial t}\left(\sum_{i=1}^{N}\sum_{j=1}^{N} c_{ji}\varphi'_j(u_j) - \sum_{j=1}^{N} b_j(u_j)\varphi'_j(u_j)\right) \tag{2}$$

From Eq. (13.47) in the text, we have

$$\frac{\partial u_j}{\partial t} = a_j(u_j)\left(b_j(u_j) - \sum_{j=1}^{N} c_{ji}\varphi_i(u_i)\right), \qquad j = 1, 2, ..., N \tag{3}$$

Hence using (3) in (2) and collecting terms, we get the final result

$$\frac{\partial E}{\partial t} = -\sum_{j=1}^{N} a_j(u_j)\varphi'_j(u_j)\left(b_j(u_j) - \sum_{i=1}^{N} c_{ji}\varphi_i(u_i)\right)^2 \tag{4}$$

Provided that the coefficient $a_j(u_j)$ satisfies the nonnegativity condition

$a_j(u_j) > 0$ for all u_j

and the function $\varphi'_j(u_j)$ satisfies the monotonicity condition

$\varphi'_j(u_j) \geq 0$ for all u_j,

we then immediately see from (4) that

$\frac{\partial E}{\partial t} \leq 0$ for all t

In words, the function E defined in(1) is the Lyapunov function for the coupled system of nonlinear differential equations (3).

Problem 13.12

From (13.61) of the text:

$$\frac{d}{dt}v_j(t) = -v_j(t) + \sum_{i=1}^{N} c_{ji}\varphi_i(v_i), \qquad j = 1, 2, ..., N \tag{1}$$

where

$$c_{ji} = \delta_{ji} + \beta w_{ji}$$

where δ_{ji} is a Kronecker delta. According to the Cohen-Grossberg theorem of (13.47) in the text, we have

$$\frac{d}{dt}u_j(t) = -a_j(u_j)\left[b_j(u_j) - \sum_{i=1}^{N} c_{ji}\varphi_i(u_i)\right] \tag{2}$$

Comparison of (1) and (2) yields the following correspondences between the Cohen-Grossberg theorem and the brain-in-state-box (BSB) model:

Cohen-Grossberg Theorem	**BSB Model**
u_j	v_j
$a_j(u_j)$	1
$b_j(u_j)$	$-v_j$
c_{ji}	$-c_{ji}$
$\varphi_i(u_i)$	$\varphi(v_i)$

Therefore, using these correspondences in (13.48) of thetext:

$$E = \frac{1}{2}\sum_i \sum_j c_{ji}\varphi_i(u_i)\varphi_j(u_j) - \sum_j \int^{u_j} b_j(\lambda)\varphi'_j(\lambda)d\lambda,$$

we get the following Liapunov function for the BSB model:

$$E = -\frac{1}{2}\sum_i \sum_j c_{ji}\varphi(v_i)\varphi(v_j) + \sum_j \int_0^{v_j} \lambda\varphi'(\lambda)d\lambda \tag{3}$$

From (13.55) in the text, we note that

$$\varphi(y_j) = \begin{cases} +1 \text{ if } y_j > 1 \\ y_j \text{ if } -1 \le y_j \le 1 \\ -1 \text{ if } y_j \le -1 \end{cases}$$

We therefore have

$$\varphi'(y_j) = \begin{cases} 0, & |y_j| > 1 \\ 1, & |y_j| \le 1 \end{cases}$$

Hence, the second term of (3) is given by

$$\begin{aligned} \sum_j \int_0^{v_j} \lambda\varphi'(\lambda)d\lambda &= \sum_j \int_0^{v_j} \lambda d\lambda = \frac{1}{2}\sum_j v_j^2 \\ &= \frac{1}{2}\sum_j x_j^2 \quad \text{inside the linear region} \end{aligned} \tag{4}$$

The first term of (3) is given by

$$\begin{aligned} -\frac{1}{2}\sum_j \sum_i c_{ji}\varphi(v_i)\varphi(v_j) &= -\frac{1}{2}\sum_j \sum_i (\delta_{ji} + \beta w_{ji})\varphi(v_i)\varphi(v_j) \\ &= -\frac{\beta}{2}\sum_j \sum_i w_{ji}x_j x_i - \frac{1}{2}\sum_j \varphi^2(v_j) \\ &= -\frac{\beta}{2}\sum_j \sum_i w_{ji}x_j x_i - \frac{1}{2}\sum_j x_j^2 \end{aligned} \tag{5}$$

Finally, substituting (4) and (5) into (3), we obtain

$$E = -\frac{\beta}{2}\sum_j \sum_i w_{ji} x_j x_i - \frac{\beta}{2}\mathbf{x}^T \mathbf{W} \mathbf{x}$$

which is the desired result

Problem 13.13

The activation function $\varphi(v)$ of Fig. P13.13 is a nonmonotonic function of the argument *v*; that is, $\partial\varphi/\partial v$ assumes both positive and negative values. It therefore violates the monotonicity condition required by the Cohen-Grossberg theorem; see Eq. (4) of Problem 13.11. This means that the cohen-Grossberg theorem is not applicable to an associative memory like a Hopfield network that uses the activation function of Fig. P14.15.

CHAPTER 15
Dynamically Driven Recurrent Networks

Problem 15.1

Referring to the simple recurrent neural network of Fig. 15.3, let the vector **u**(n) denote the input signal, the vector **x**(n) denotes the signal produced at the output of the hidden layer, and the vector **y**(n) denotes the output signal of the whole network. Then, treating **x**(n) as the state of the network, we may describe the state-space model of the network as follows:

$$\begin{aligned}\mathbf{x}(n+1) &= \mathbf{f}(\mathbf{x}(n), \mathbf{u}(n)) \\ \mathbf{y}(n) &= \mathbf{g}(\mathbf{x}(n))\end{aligned}$$

where $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are vector-valued functions of their respective arguments.

Problem 15.2

Referring to the recurrent MLP of Fig. 15.4, we note the following:

$$\mathbf{x}_\mathrm{I}(n+1) = \mathbf{f}_1(\mathbf{x}_\mathrm{I}(n), \mathbf{u}(n)) \tag{1}$$

$$\mathbf{x}_\mathrm{II}(n+1) = \mathbf{f}_2(\mathbf{x}_\mathrm{II}(n), \mathbf{x}_\mathrm{I}(n+1)) \tag{2}$$

$$\mathbf{x}_0(n+1) = \mathbf{f}_3(\mathbf{x}_0(n)\mathbf{x}_\mathrm{II}(n+1)) \tag{3}$$

where $\mathbf{f}_1(\cdot)$, $\mathbf{f}_2(\cdot)$, and $\mathbf{f}_3(\cdot)$ are vector-valued functions of their respective arguments. Substituting (1) into (2), we write

$$\mathbf{x}_\mathrm{II} = \mathbf{f}_2(\mathbf{x}_\mathrm{II}, \mathbf{f}_1(\mathbf{x}_\mathrm{I}, \mathbf{u}(n))) \tag{4}$$

Define the state of the system at time n as

$$\mathbf{x}_\mathrm{II}(n+1) = \begin{bmatrix}\mathbf{x}_\mathrm{II}(n) \\ \mathbf{x}_0(n-1)\end{bmatrix} \tag{5}$$

Then, from (4) and (5) we immediately see that

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{x}(n), \mathbf{u}(n)) \tag{6}$$

where **f** is a new vector-valued function. Define the output of the system as

$$\mathbf{y}(n) = \mathbf{x}_0(n) \tag{7}$$

With $\mathbf{x}_0(n)$ included in the definition of the state $\mathbf{x}(n + 1)$ and with $\mathbf{x}(n)$ dependent on the input $\mathbf{u}(n)$, we thus have

$$\mathbf{y}(n) = \mathbf{g}(\mathbf{x}(n), \mathbf{u}(n)) \tag{8}$$

where $\mathbf{g}(\cdot,\cdot)$ is another vector valued function. Equations (6) and (8) define the state-space model of the recurrent MLP.

Problem 15.3

It is indeed possible for a dynamic system to be controllable but unobservable, and vice versa. This statement is justified by virtue of the fact that the conditions for controllability and observability are entirely different, which means that there are situations where the conditions are satisfied for one and not for the other.

Problem 15.4

(a) We are given the process equation

$$\mathbf{x}(n+1) = \phi(\mathbf{W}_a\mathbf{x}(n) + \mathbf{w}_b u(n))$$

Hence, iterating forward in time, we write

$$\begin{aligned}\mathbf{x}(n+2) &= \phi(\mathbf{W}_a\mathbf{x}(n+1) + \mathbf{w}_b u(n+1)) \\ &= \phi(\mathbf{W}_a\phi(\mathbf{W}_a\mathbf{x}(n) + \mathbf{w}_b u(n)) + \mathbf{w}_b u(n+1))\end{aligned}$$

$$\begin{aligned}\mathbf{x}(n+3) &= \phi(\mathbf{W}_a\mathbf{x}(n+2) + \mathbf{w}_b u(n+2)) \\ &= \phi(\mathbf{W}_a\phi\mathbf{W}_a\phi(\mathbf{W}_a\mathbf{x}(n) + \mathbf{w}_b u(n)) + \mathbf{w}_b u(n+1)) + \mathbf{w}_b u(n+2))\end{aligned}$$

and so on. By induction, we may state that the state $\mathbf{x}(n + q)$ is a nested nonlinear function of $\mathbf{x}(n)$ and $\mathbf{u}_q(n)$, where

$$\mathbf{u}_q(n) = [u(n), u(n+1), \ldots, u(n+q-1)]^T$$

(b) The Jacobian of $\mathbf{x}(n + q)$ with respect to $\mathbf{u}_q(n)$ at the origin, is

$$\mathbf{J}_q(n) = \left[\frac{\partial\mathbf{x}(n+q)}{\partial\mathbf{u}_q(n)}\right]_{\substack{\mathbf{x}(n) = 0 \\ u(n) = 0}}$$

As an illustrative example, consider the cast of $q = 3$. The Jacobian of $\mathbf{x}(n + 3)$ with respect to $\mathbf{u}_3(n)$ is

$$\mathbf{J}_3(n) = \left[\frac{\partial \mathbf{x}(n+3)}{\partial u(n)}, \frac{\partial \mathbf{x}(n+2)}{\partial u(n+1)}, \frac{\partial \mathbf{x}(n+3)}{\partial u(n+2)}\right]_{\substack{\mathbf{x}(n) = 0 \\ u(n) = 0}}$$

From the defining equation of $\mathbf{x}(n + 3)$, we find that

$$\begin{aligned}\frac{\partial \mathbf{x}(n+3)}{\partial u(n)} &= \phi'(0)\mathbf{W}_a\phi'(0)\mathbf{W}_a\phi'(0)\mathbf{w}_b \\ &= \mathbf{AAb} \\ &= \mathbf{A}^2\mathbf{b}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathbf{x}(n+3)}{\partial u(n+1)} &= \phi'(0)\mathbf{W}_a\phi'(0)\mathbf{w}_b \\ &= \mathbf{Ab}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathbf{x}(n+3)}{\partial u(n+2)} &= \phi'(0)\mathbf{w}_b \\ &= \mathbf{b}\end{aligned}$$

All these partial derivatives have been evaluated at $\mathbf{x}(n) = 0$ and $u(n) = 0$. The Jacobian $\mathbf{J}_3(n)$ is therefore

$$\mathbf{J}_3(n) = [\mathbf{A}^2\mathbf{b}, \mathbf{Ab}, \mathbf{b}]$$

We may generalize this result by writing

$$\mathbf{J}_q(n) = [\mathbf{A}^{q-1}\mathbf{b}, \mathbf{A}^{q-2}\mathbf{b}, \ldots, \mathbf{Ab}, \mathbf{b}]$$

Problem 15.5

We start with the state-space model

$$\begin{aligned}\mathbf{x}(n+1) &= \phi(\mathbf{W}_a\mathbf{x}(n) + \mathbf{w}_b u(n)) \\ y(n) &= \mathbf{c}^T\mathbf{x}(n)\end{aligned} \tag{1}$$

where **c** is a column vector. We thus write

$$y(n+1) = \mathbf{c}^T\mathbf{x}(n+1)$$

$$= \mathbf{c}^T \phi(\mathbf{W}_a \mathbf{x}(n) + \mathbf{w}_b u(n)) \tag{2}$$

$$y(n+2) = \mathbf{c}^T \mathbf{x}(n+2)$$
$$= \mathbf{c}^T \phi(\mathbf{W}_a \phi(\mathbf{W}_a \mathbf{x}(n) + \mathbf{w}_b u(n)) + \mathbf{w}_b u(n+1)) \tag{3}$$

and so on. By induction, we may therefore state that $y(n+q)$ is a nested nonlinear function of $\mathbf{x}(n)$ and $\mathbf{u}_q(n)$, where

$$\mathbf{u}_q(n) = [u(n), u(n+1), \ldots, u(n+q-1)]^T$$

Define the q-by-1 vector

$$\mathbf{y}_q(n) = [y(n), y(n+1), \ldots, y(n+q-1)]^T$$

The Jacobian of $\mathbf{y}_q(n)$ with respect to $\mathbf{x}(n)$, evaluated at the origin, is defined by

$$\mathbf{J}_q(n) = \left[\frac{\partial \mathbf{y}_q^T(n)}{\partial \mathbf{x}(n)}\right]_{\substack{\mathbf{x}(n) = \mathbf{0} \\ u(n) = 0}}$$

As an illustrative example, consider the case of $q = 3$, for which we have

$$\mathbf{J}_3(n) = \left[\frac{\partial y(n)}{\partial \mathbf{x}(n)}, \frac{\partial y(n+1)}{\partial \mathbf{x}(n)}, \frac{\partial y(n+2)}{\partial \mathbf{x}(n)}\right]_{\substack{\mathbf{x}(n) = 0 \\ u(n) = 0}}$$

From (1), we readily find that

$$\frac{\partial y(n)}{\partial \mathbf{x}(n)} = \mathbf{c}$$

From (2), we find that

$$\frac{\partial y(n+1)}{\partial \mathbf{x}(n)} = \mathbf{c}(\phi'(0)\mathbf{W}_a)^T$$
$$= \mathbf{c}\mathbf{A}^T$$

From (3), we finally find that

$$\frac{\partial y(n+2)}{\partial \mathbf{x}(n)} = \mathbf{c}(\phi'(0)\mathbf{W}_a)^T(\phi'(0)\mathbf{W}_a)$$

$$= \mathbf{c}\mathbf{A}^T\mathbf{A}^T$$

$$= \mathbf{c}(\mathbf{A}^T)^2$$

All these partial derivatives have been evaluated at the origin. We thus write

$$\mathbf{J}_3(n) = [\mathbf{c}, \mathbf{c}(\mathbf{A}^T, \mathbf{c}\mathbf{A}^T)^2]$$

By induction, we may now state that the Jacobian $\mathbf{J}_q(n)$ for observability is, in general,

$$\mathbf{J}_q(n) = [\mathbf{c}, \mathbf{c}\mathbf{A}^T, \mathbf{c}(\mathbf{A}^T)^2, \ldots, \mathbf{c}(\mathbf{A}^T)^{q-1}]$$

where $\mathbf{c}$ is a column vector and $\mathbf{A} = \phi'(0)\mathbf{W}_a$.

Problem 15.6

We are given a nonlinear dynamic system described by

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{x}(n), \mathbf{u}(n)) \tag{1}$$

Suppose $\mathbf{x}(n)$ is N-dimensional and $\mathbf{u}(n)$ is m-dimensional. Define a new nonlinear dynamic system in which the input is of additive form, as shown by

$$\mathbf{x}'(n+1) = \mathbf{f}'(\mathbf{x}'(n)) + \mathbf{u}'(n) \tag{2}$$

where

$$\mathbf{x}'(n) = \begin{bmatrix} \mathbf{x}(n) \\ \mathbf{u}(n-1) \end{bmatrix} \tag{3}$$

$$\mathbf{u}'(n) = \begin{bmatrix} \mathbf{0} \\ \mathbf{u}(n) \end{bmatrix} \tag{4}$$

and

$$\mathbf{f}'(\mathbf{x}'(n)) = \begin{bmatrix} \mathbf{f}(\mathbf{x}(n), \mathbf{u}(n)) \\ \mathbf{0} \end{bmatrix} \tag{5}$$

Both $\mathbf{x}'(n)$ and $\mathbf{u}'(n)$ are ($N + m$)-dimensional, and the first N elements of $\mathbf{u}'(n)$ are zero. From these definitions, we readily see that

$$\mathbf{x}'(n+1) = \begin{bmatrix} \mathbf{x}(n+1) \\ \mathbf{u}(n) \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{f}(\mathbf{x}(n), \mathbf{u}(n)) \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{u}(n) \end{bmatrix}$$

which is in perfect agreement with the description of the original nonlinear dynamic system defined in (1).

Problem 15.7

(a) The state-space model of the local activation feedback system of Fig. P15.7a depends on how the linear dynamic component is described. For example, we may define the input as

$$\mathbf{z}(n) = \begin{bmatrix} x(n-1) \\ \mathbf{Bu}(n) \end{bmatrix} \tag{1}$$

where **B** is a (p-1)-by-(p-1) matrix and

$$\mathbf{u}(n) = [u(n), u(n-1), \dots, u(n-p+2)]^T$$

Let **w** denote the synaptic weight vector of the single neuron in Fig. P15.7a, with w_1 being the first element and $\mathbf{w}_0$ denoting the rest. we may then write

$$x(n) = \mathbf{w}^T\mathbf{z}(n) + b$$
$$= [w_1, \mathbf{w}_0^T]\begin{bmatrix} x(n-1) \\ \mathbf{Bu}(n) \end{bmatrix} + b$$
$$= w_1 x(n-1) + \mathbf{B}'\mathbf{u}'(n) \tag{2}$$

where

$$\mathbf{u}'(n) = \begin{bmatrix} \mathbf{u}'(n) \\ 1 \end{bmatrix}$$

and

$$\mathbf{B}' = [\mathbf{w}_0^T\mathbf{B}, b]$$

The output $y(n)$ is defined by

$$y(n) = \varphi(x(n)) \tag{3}$$

Equations (2) and (3) define the state-space model of Fig. P15.7a, assuming that its linear dynamic component is described by (1).

(b) Consider next the local output feedback system of Fig. 15.7b. Let the linear dynamic component of this system be described by (1). The output of the whole system in Fig. 15.7b is then defined by

$$\begin{aligned} x(n) &= \phi(\mathbf{w}^T\mathbf{z}(n) + b) \\ &= \phi\left([w_1, \mathbf{w}_0^T]\begin{bmatrix} x(n-1) \\ \mathbf{Bu}(n) \end{bmatrix} + b\right) \\ &= \phi(w_1 x(n-1) + \mathbf{B}'\mathbf{u}'(n)) \end{aligned} \tag{4}$$

where w_1, $\mathbf{w}_0$, $\mathbf{B}'$, and $\mathbf{u}'(n)$ are all as defined previously. The output $y(n)$ of Fig. P15.7b is

$$y(n) = x(n) \tag{5}$$

Equations (4) and (5) define the state-space model of the local output feedback system of Fig. P15.7b, assuming that its linear dynamic component is described by (1).

The process (state) equation of the local feedback system of Fig. P15.7a is linear but its measurement equation is nonlinear, and conversely for the local feedback system of Fig. P15.7b. These two local feedback systems are controllable and observable, because they both satisfy the conditions for controllability and observability.

Problem 15.8

We start with the state equation

$$\mathbf{x}(n+1) = \phi(\mathbf{W}_a\mathbf{x}(n) + \mathbf{w}_b u(n))$$

Hence, we write

$$\begin{aligned} \mathbf{x}(n+2) &= \phi(\mathbf{W}_a\mathbf{x}(n+1) + \mathbf{w}_b u(n+1)) \\ &= \phi(\mathbf{W}_a\phi(\mathbf{W}_a\mathbf{x}(n) + \mathbf{w}_b u(n)) + \mathbf{w}_b u(n+1)) \end{aligned}$$

$$\begin{aligned}\mathbf{x}(n+3) &= \phi(\mathbf{W}_a\mathbf{x}(n+2) + \mathbf{w}_b u(n+2)) \\ &= \phi(\mathbf{W}_a\phi(\mathbf{W}_a\phi(\mathbf{W}_a\mathbf{x}(n) + \mathbf{w}_b u(n)) + \mathbf{w}_b u(n+1)) + \mathbf{w}_b u(n+2))\end{aligned}$$

and so on.

By induction, we may now state that $\mathbf{x}(n + q)$ is a nested nonlinear function of $\mathbf{x}(n)$ and $\mathbf{u}_q(n)$, and thus write

$$\mathbf{x}(n+q) = \mathbf{g}(\mathbf{x}(n)\mathbf{u}_q(n))$$

where $\mathbf{g}$ is a vector-valued function, and

$$\mathbf{u}_q(n) = [u(n), u(n+1), \ldots, u(n+q-1)]^T$$

By definition, the output is correspondingly given by

$$\begin{aligned}y(n+q) &= \mathbf{c}^T\mathbf{x}(n+q) \\ &= \mathbf{c}^T\mathbf{g}(\mathbf{x}(n)\mathbf{u}_q(n)) \\ &= \Phi(\mathbf{x}(n), \mathbf{u}_q(n))\end{aligned}$$

where $\mathbf{\Phi}$ is a new scalar-valued nonlinear function.

Problem 15.11

Consider a state-space model described by

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{x}(n), \mathbf{u}(n)) \qquad (1)$$

$$\mathbf{y}(n) = \mathbf{g}(\mathbf{x}(n)) \qquad (2)$$

Using (1), we may readily write

$$\begin{aligned}\mathbf{x}(n) &= \mathbf{f}(\mathbf{x}(n-1), \mathbf{u}(n-1))\\ \mathbf{x}(n-1) &= \mathbf{f}(\mathbf{x}(n-2), \mathbf{u}(n-2))\\ \mathbf{x}(n-2) &= \mathbf{f}(\mathbf{x}(n-3), \mathbf{u}(n-3))\end{aligned}$$

and so on. Accordingly, the simple recurrent network of Fig. 15.3 may be unfolded in time as follows:

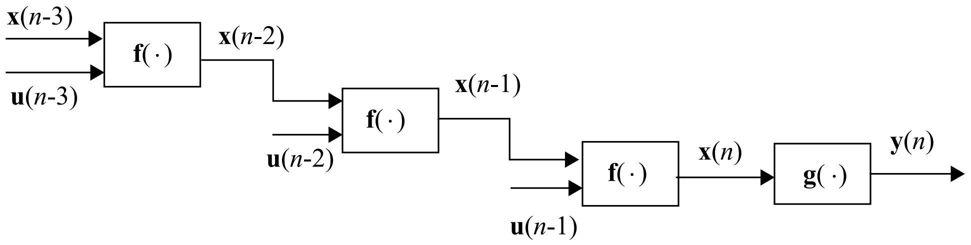


Figure : Problem 15.11

Problem 15.12

The local gradient for the hybrid form of the BPTT algorithm is given by

$$\delta_j(l) = \begin{cases} \varphi'(v_j(l))e_j(l) & \text{for } l = n \\ \varphi'(v_j(l))\left(e_j(l) + \sum_k w_{kj}(l)\delta_l(l+1)\right) & \text{for } n-h < l < n \\ \varphi'(v_j(l))\sum_k w_{kj}(l)\delta_l(l+1) & \text{for } n-h < l < j-h' \end{cases}$$

where h' is the number of additional steps taken before performing the next BPTT computation, with $h' < h$.

Problem 15.13

(a) The nonlinear state dynamics of the real-time recurrent learning algorithm of described in (15.48) and (15.52) olf the text may be reformulated in the equivalent form:

$$\frac{\partial y_j(n+1)}{\partial w_{kl}(n)} = \varphi'(v_j(n)) \sum_{i \in A \in B} w_{ji}(n) \frac{\partial \xi_i(n)}{\partial w_{kl}(n)} + \delta_{kj} \xi_l(n) \tag{1}$$

where δ_{kj} is the Kronecker delta and $y_j(n + 1)$ is the output of neuron j at time $n + 1$. For a teacher-forced recurrent network, we have

$$\xi_i(n) = \begin{cases} u_i(n) \text{ if } i \in A \\ d_i(n) \text{ if } i \in C \\ y_i(n) \text{ if } i \in B\text{-}C \end{cases} \tag{2}$$

Hence, substituting (2) into (1), we get

$$\frac{\partial y_j(n+1)}{\partial w_{kl}(n)} = \varphi'(v_j(n)) \sum_{i \in B\text{-}C} w_{ji}(n) \frac{\partial y_i(n)}{\partial w_{kl}(n)} + \delta_{kj} \xi_l(n) \tag{3}$$

(b) Let

$$\pi_{kl}^{j}(n) = \frac{\partial y_i(n)}{\partial w_{kl}(n)}$$

Provided that the learning-rate parameter η is small enough, we may put

$$\pi_{kl}^{j}(n+1) = \frac{\partial y_i(n+1)}{\partial w_{kl}(n+1)} \approx \frac{\partial y_i(n+1)}{\partial w_{kl}(n)}$$

Under this condition, we may rewrite (3) as follows:

$$\pi_{kl}^{j}(n+1) = \phi'(v_j(n)) \sum_{i \in B\text{-}C} w_{ji}(n) \pi_{kl}^{j}(n) + \delta_{kj} \xi_l(n) \tag{4}$$

This nonlinear state equation is the centerpiece of the RTRL algorithm using teacher forcing.