EENG582: Artificial Neural Networks

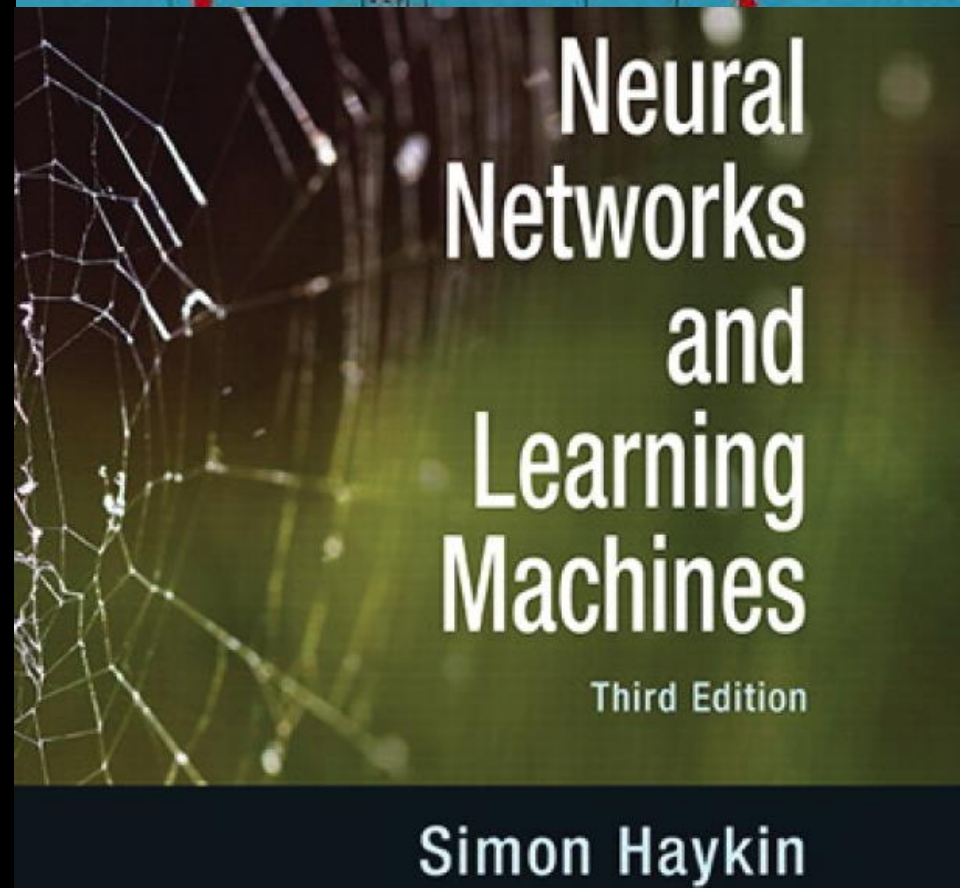
Neural Networks A Comprehensive Foundation by S. Haykin

All Sections

Problems with Solutions

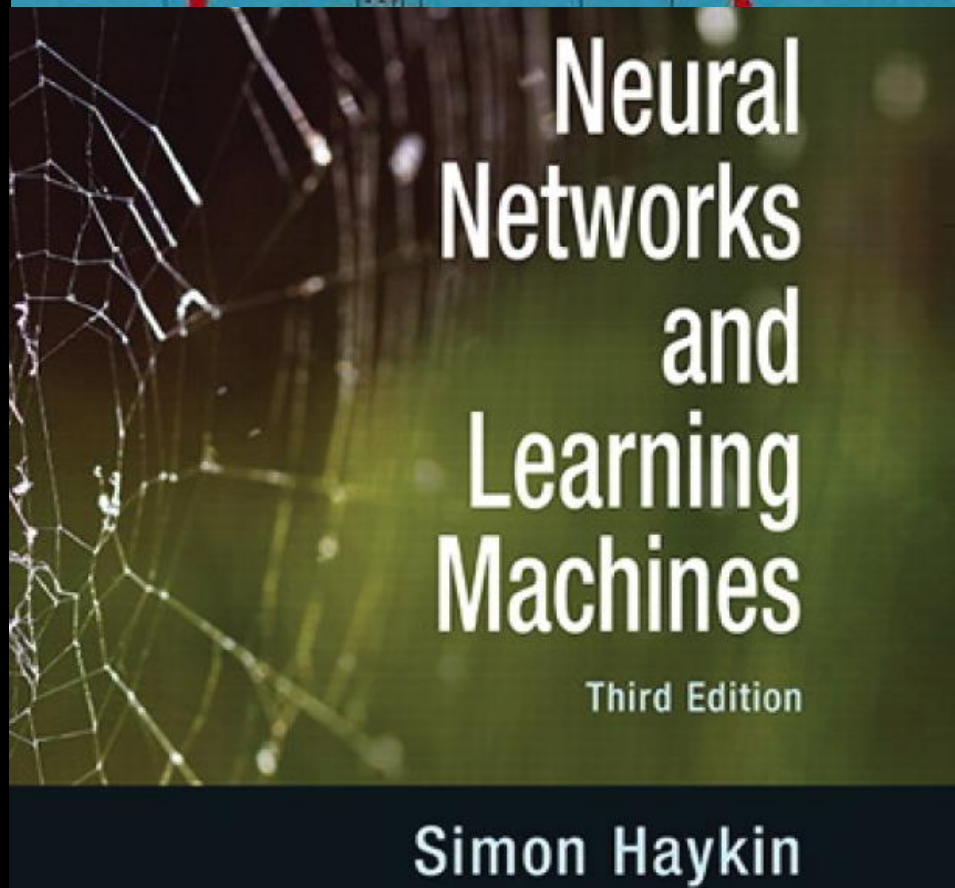
Prof. Dr. Hasan AMCA
Electrical and Electronic Engineering Department
(ee.emu.edu.tr)

Eastern Mediterranean University
(emu.edu.tr)



3 Single Layer Perceptrons

4.1 Introduction 178
4.2 Some Preliminaries 181
4.3 Back-Propagation AJgorithm . 183
4.4 Summary of the Back-Propagation Algorithm 195
4.5 XOR Problem 197
4.6 Heuristics for Making the Back -Propagation Algorithm Perfol'm Better 200
4. 7 Output Representation and Decision Rule 206
4.8 Computer Experiment 209
4.9 Feature Detection 221
4.10 Back-Propagation and Differentiation 224
4.11 Hessian Matrix 226
4.12 Generalization 227
4.13 Approximations of Functions 230
4.14 Cross-Validation 235
4.15 Network Pruning Techniques 240
4.16 Virtues and Limitations of Back-Propagation Learning 248
4.17 Accelerated Convergence of Back-Propagation Learning 255
4.18 Supervised Learning Viewed as an Optimization Problem 256
4.19 Convolutional Networks 267
4.20 Summary and Discussion 269
Notes and References 270
Problems 274



Example 2.1: Learning in Neural Nets

[Book: Artificial Neural Networks w Examples by Colin Eyfe, pp.13]

This tutorial will emphasize learning in neural nets. Recall that learning is believed to happen at the synapses (the meeting points between neurons) and that we model synaptic efficiency with weights. You are going to hand simulate a simple neural net (see Figure 1.4) performing classification according to the AND (see table) OR and XOR rules. We will use a network with three input neurons- the two required for the input values and a third neuron known as the bias neuron. The bias always fires 1 (i.e. is constant).

You will work in groups of 3 - one person in charge of selecting the input, one in charge of calculating the feedforward value of the neuron, and one person in charge of calculating the change in weights. To begin with, the group selects random (between 0 and 1) values for the three weights shown in the figure.

Example 2.1: Learning in Neural Nets

1. The INPUTER selects randomly from the set of patterns shown in the table and places the cards in the appropriate places on the table.

2. The FEED FORWARDER multiplies the weights by the input patterns to calculate the output.

$$y = \sum_{i=0}^{2} w_i x_i \tag{1.3}$$

Then y = 1 if y > 0, y = -1 if y < 0.

3. The WEIGHTCHANGER changes the weights when the value of y is not the same as the target y_T according to the formula

$$w_i = w_i + \eta \times (y_T - y) \times x_i \tag{1.4}$$

Steps (1)-(3) are repeated as often as necessary.

Example 2.1: Learning in Neural Nets

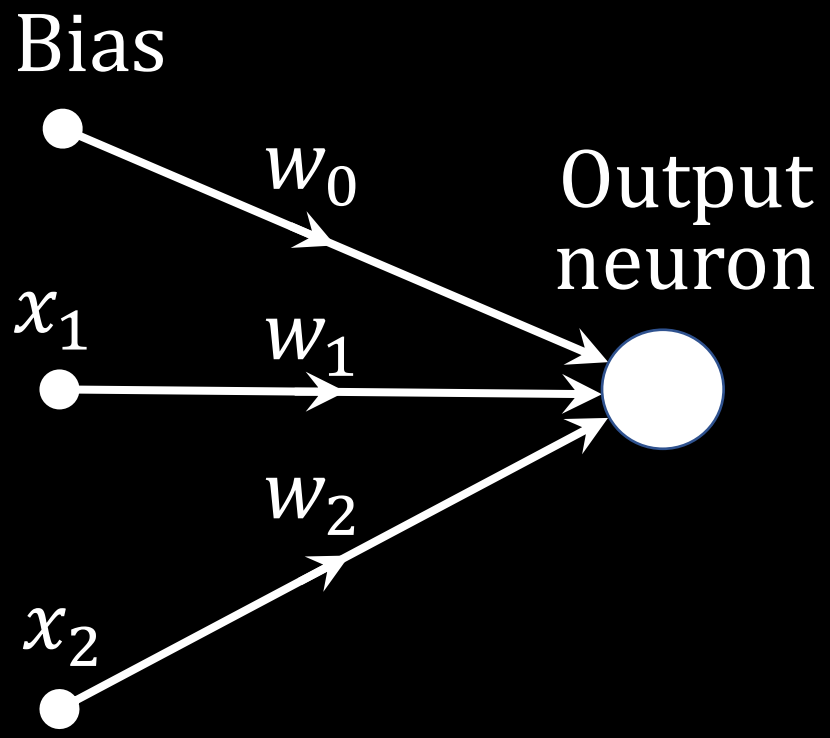


Figure 1.4: The Simple Neural Network

Bias	1st input	2nd input	Target output
1	1	1	1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	-1

Table 1.1: The values for the AND patterns

Example 2.1: Learning in Neural Nets

[Book: Artificial Neural Networks w Examples by Colin Eyfe, pp.13]

Let us have initial values $w_0 = 0.5, w_1 = 0.3, w_2 = 0.7$ and let $\eta = 0.1$.

1. "Randomly" select pattern 1. The FEEDFORWARDER calculates y = 0.5 +0.3 +0.7 =1.5. So y=1, which is the same as the target and so the WEIGHTCHANGER does nothing.

2. Imagine pattern 2 is chosen. The FEEDFORWARDER calculates y = 0.5+0.3 - 0.7 = 0.1. So y=l. Now $y_T = -1$ and so WEIGHTCHAINGER calculates

$$w_0 = w_0 + 0.1 \times (-2) \times 1 = 0.5 - 0.2 = 0.3$$

$$w_1 = w_1 + 0.1 \times (-2) \times 1 = 0.3 - 0.2 = 0.1$$

$$w_2 = w_2 + 0.1 \times (-2) \times (-1) = 0.7 + 0.2 = 0.9$$

Example 2.1: Learning in Neural Nets

3. Now, pattern 3 is chosen. The FEEDFORWARDER calculates y = 0.3-0.1 + 0.9 = 1.1. So, y=l. Now $y_T = -1$ and so WEIGHTCHAINGER calculates

$$w_0 = w_0 + 0.1 \times (-2) \times 1 = 0.3 - 0.2 = 0.1$$

$$w_1 = w_1 + 0.1 \times (-2) \times (-1) = 0.1 + 0.2 = 0.3$$

$$w_2 = w_2 + 0.1 \times (-2) \times 1 = 0.9 - 0.2 = 0.7$$

4. Now pattern 4 is chosen. The FEEDFORWARDER calculates y = 0.1- 0.3- 0.7 = -0.9. So, y= -1 and y_T= -1. Therefore, the WEIGHTCHANGER does nothing.

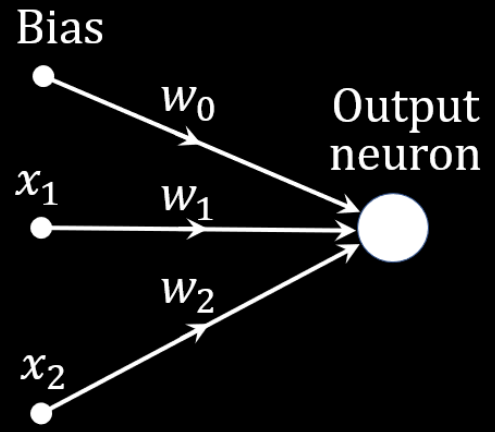


Figure 1.4: The Simple Neural Network

Bias	1st input	2nd input	Target output
1	1	1	1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	-1

Table 1.1: The values for the AND patterns

Example 2.1: Learning in Neural Nets

5. Now select pattern 2. The FEEDFORWARDER calculates y = 0.1- 0.3 + 0.7 = 0.5. So y=l. Now $y_T = -1$ and so WEIGHTCHAINGER calculates

$$w_0 = w_0 + 0.1 \times (-2) \times 1 = 0.1 - 0.2 = -0.1$$

$$w_1 = w_1 + 0.1 \times (-2) \times (-1) = 0.3 + 0.2 = 0.5$$

$$w_2 = w_2 + 0.1 \times (-2) \times 1 = 0.7 - 0.2 = 0.5$$

6. Now all of the patterns give the correct response (try it).

We can draw the solution found by using the weights as the parameters of the line a$x_1 + bx_2 + c = 0$. Using $a = w_1, b = w_2$, c $= w_0$, we get

$$0.5x_1 + 0.5x_2 - 0.1 = 0 \qquad (1.5)$$

which we can draw by getting two points.

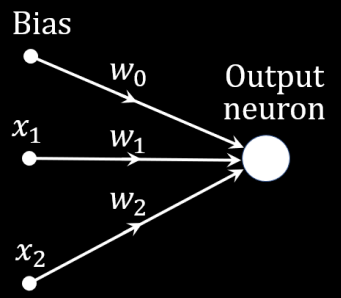


Figure 1.4: The Simple Neural Network

Bias	1st input	2nd input	Target output
1	1	1	1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	-1

Table 1.1: The values for the AND patterns

Example 2.1: Learning in Neural Nets

If $x_1 = 0$, then $0.5x_2 = 0.1$ and so $x_2 = 0.2$ which gives us one point (0,0.2). Similarly, we can find another point (0.2,0) which is drawn in Figure 1.5. Notice the importance of the BIAS weight: it allows a solution to be found which does not go through the origin; without a bias we would have to have a moving threshold.

We can see the convergence of $w_0 + w_1x_1 + w_2x_2 = 0$ in Figure 1.6. Notice that initially 2 patterns are correctly classified, very quickly a third is correctly classified and only on the fourth change are all 4 patterns correctly classified.

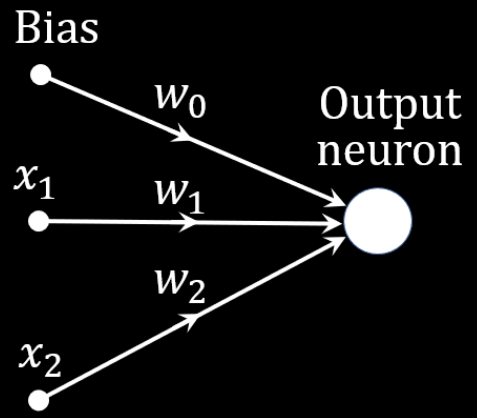


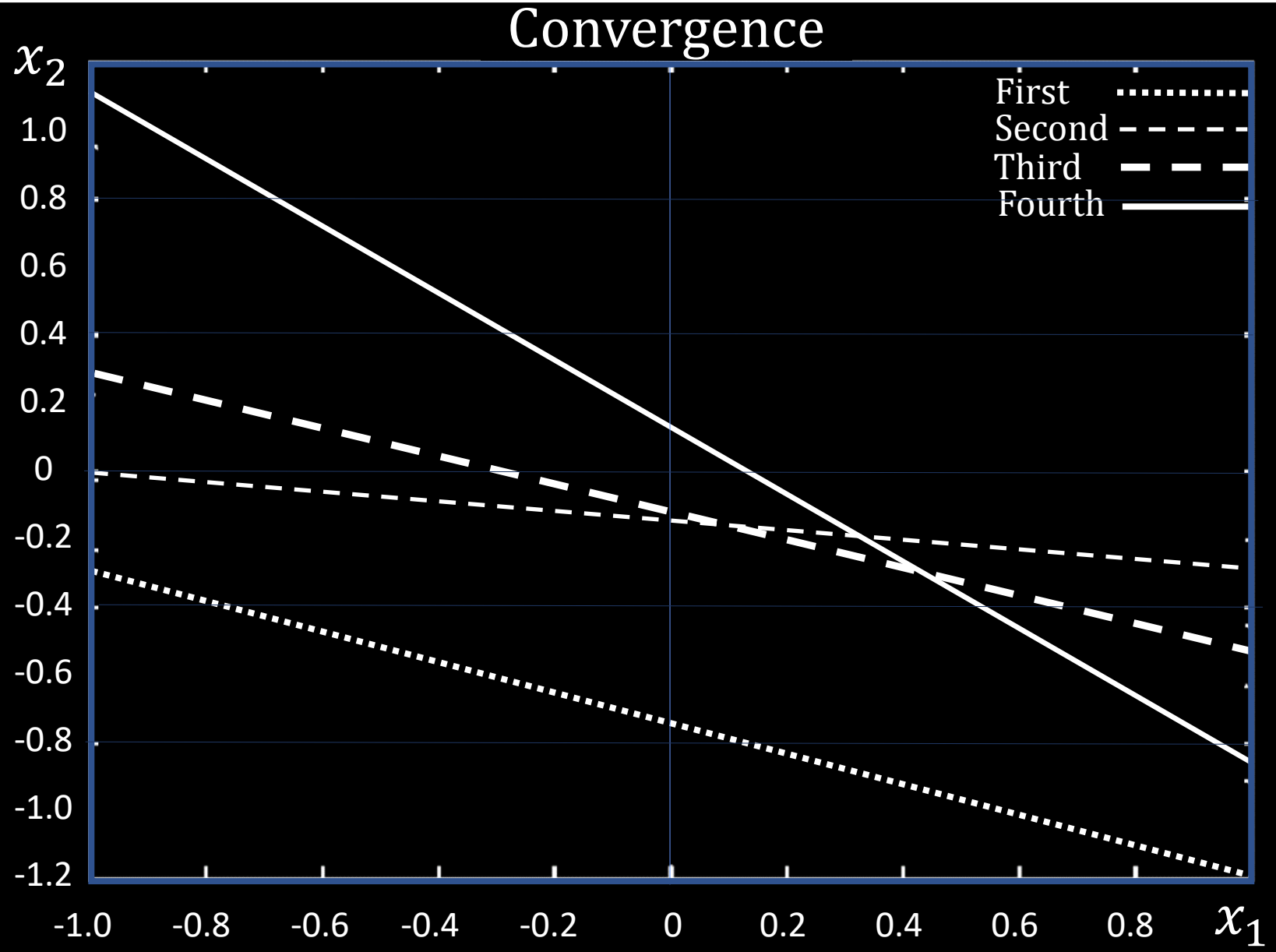
Figure 1.4: The Simple Neural Network

Bias	1st input	2nd input	Target output
1	1	1	1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	-1

Table 1.1: The values for the AND patterns

Example 2.1: Learning in Neural Nets

Figure 1.6: The iterative convergence of the network to a set of weights which can perform the correct mapping is shown diagrammatically here.



Example 2.1: Learning in Neural Nets

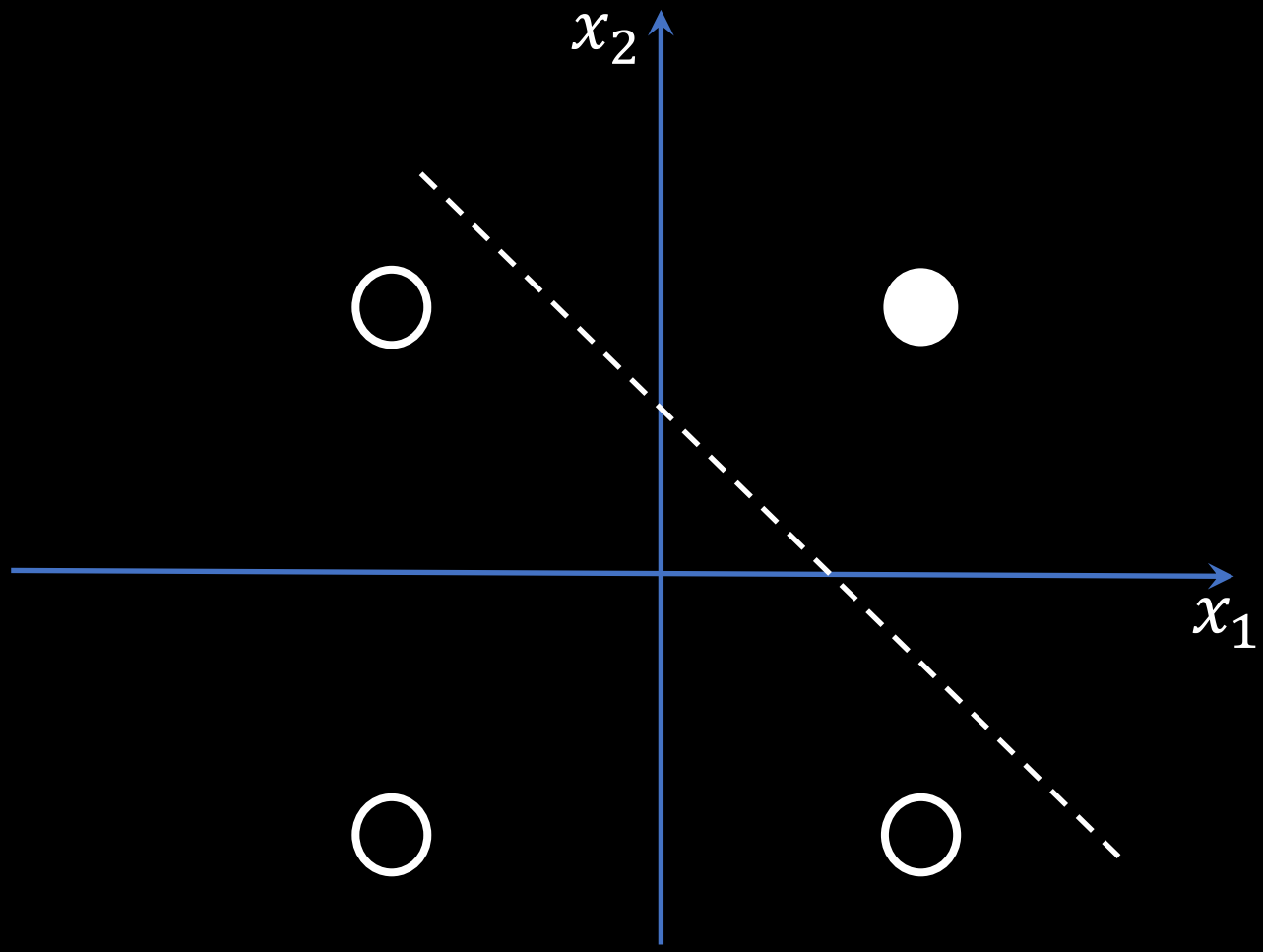


Figure 1.5: The line joining (0,0.2) and (0.2,0) cuts the 4 points into 2 sets correctly

Example 2.2: Associative Memories

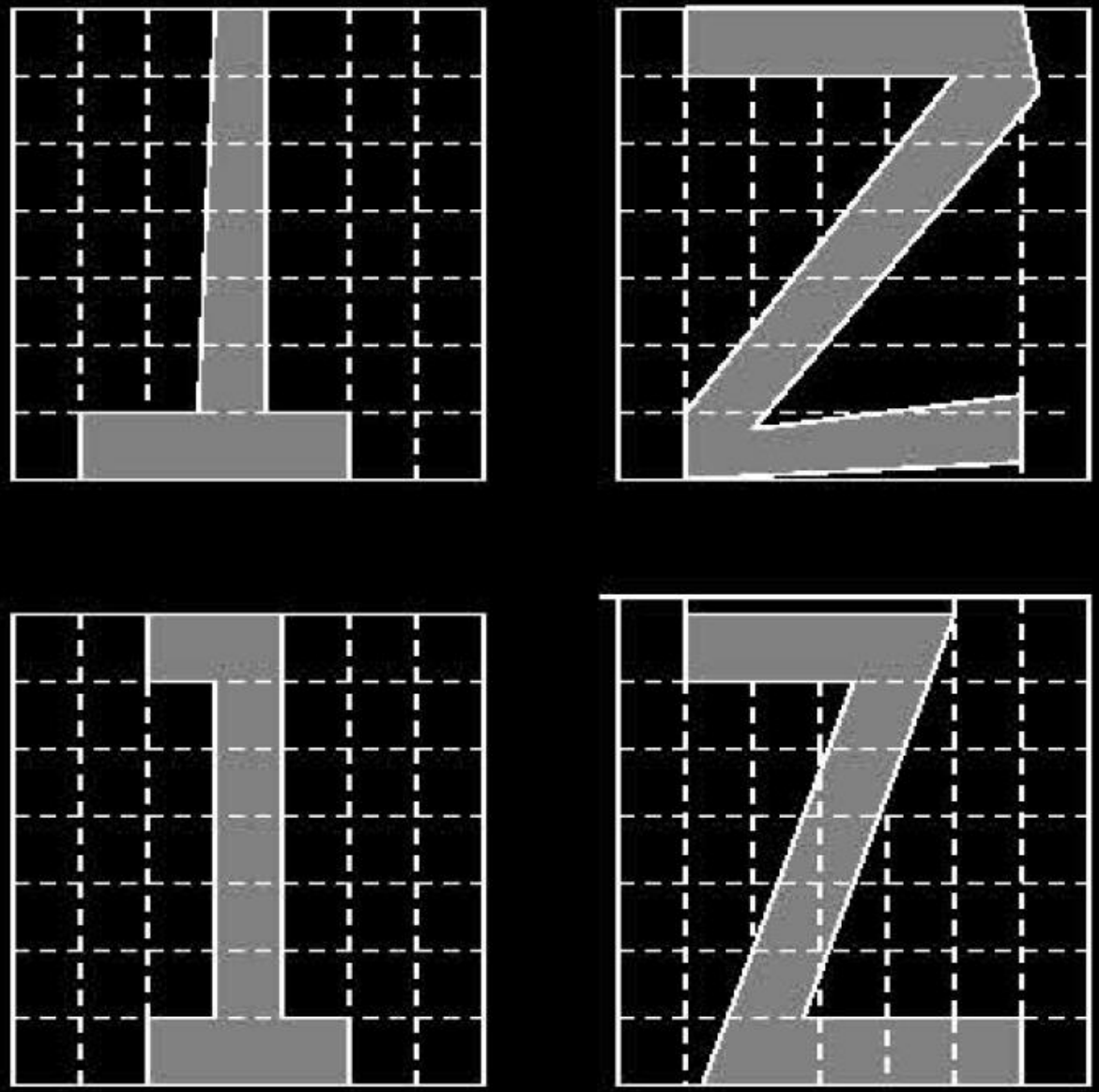
[Book: Artificial Neural Networks w Examples by Colin Eyfe, pp.20]

We would like to train an artificial neural network to differentiate between sets of patterns on a square grid of pixels. Consider Figure 2.2. We show 4 patterns two of which we wish to associate with pattern "one" and two of which we will identify as pattern "two".

To train a neural network to differentiate between these two sets of patterns (and we assume that we have many examples of "ones" and "twos"), we set up a neural network with 49 inputs and 2 outputs. Each of the 49 inputs corresponds to one pixel so that if the example drawn on the grid is such that pixel18 (say) is more than half inside the figure, the 18th element of the vector is a 1; if the 18th pixel on the other hand is more than half outside the figure, the 18th element of the vector is a 0. Therefore, our input data to the neural network is a 49 dimensional vector composed only of ones and zeroes.

Example 2.2: Associative Memories

Fig. 2.2: Sample patterns of ones and twos. Each pattern is converted to 49 dimensional vector in which dimension has value one if more than half the pixel is contained in the figure and 0 otherwise



Example 2.2: Associative Memories

The output vector is only a two-dimensional vector which will be $(1,0)^T$ if the input pattern is a "one" and $(0,1)^T$ if the input pattern is "two". Then for the kth pattern,

$$w(k) = y_k x_k^T \tag{2.5}$$

So, if the pattern is a "one", the second row of W is all zeros while if the pattern is a "two" the first row is a row of zeroes. Also, the other row in each case is simply a row containing the 49-dimensional input vector. e.g. for the first "two" pattern above

$$W = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & \dots & 1 & 0 \end{bmatrix} \tag{2.6}$$

So, the set of p training patterns P_i of type i as $M = \sum_{k=1}^{p} W(k) = \begin{bmatrix} \sum_{i \in P_1} x_i^T \\ \sum_{i \in P_2} x_i^T \end{bmatrix}$

Example 2.2: Associative Memories

So, the fact that the *y*-values are orthogonal to one another means that, in the weight matrix, the memories associated with "ones" do not interfere with the memories associated with "twos".

However, all the "ones" memories are interfering with each other; in other words, the memory which will be retained will be that which is formed by the crosstalk between patterns of the same type - the crosstalk is actually helpful in forming an idealized version of "one" and of "two".

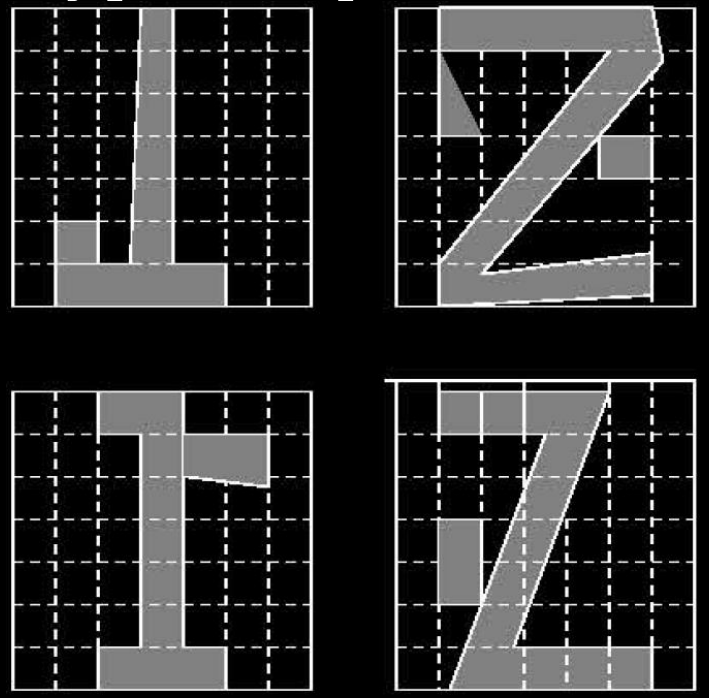
Thus, we can see that this matrix memory is proportional to the mean of the training patterns of each type which it meets during the network training. Therefore, the ability of this type of network to recognize new examples of "ones" and "twos" depends on how typical were the examples on which it was trained.

Example 2.2: Associative Memories

In general, if we put an input vector of a noisy "one" (see Figure 2.3) into the network we would hope that the output vector would more closely resemble $(1,0)^T$ than $(0,1)^T$ i.e. We should be able to identify which pattern the input is closest to by looking at the activation of the output vector.

The strength of firing of the appropriate output neuron gives a measure of how closely the input corresponds to the prototypical input vector of that class.

Figure 2.3: Noisy versions of the previous data which when input to the network will be classed as "ones« and "twos" as appropriate.



Example 2.3: Associative Memories

[Book: Artificial Neural Networks w Examples by Colin Eyfe, pp.22]

Let us create a Bidirectional Associative Memory (BAM) for a pair of memories shown in Table 2.1 in which an *X* in any position identifies that the student has that quality or that the course contains that class. We wish to create a memory which associates student *i* with course *i*

First, we may represent each student and course by a vector e.g.

$$A_1 = (0,0,1,1,0,1)^T \quad B_1 = (1,0,0,1)^T$$
$$A_2 = (0,0,1,1,1,0)^T \quad B_2 = (1,1,0,0)^T$$
$$A_3 = (0,0,1,1,1,0)^T \quad B_3 = (1,0,1,0)^T$$

The BAM is easier to construct with a bipolar coding and so we use

$$X_1 = (-1,-1,1,1,-1,1)^T \quad Y_1 = (1,-1,-1,1)^T$$
$$X_2 = (1,-1,-1,1,1,-1)^T \quad Y_2 = (1,1,-1,-1)^T$$
$$X_3 = (-1,-1,1,1,1,-1)^T \quad Y_3 = (1,-1,1,-1)^T$$

Example 2.3: Associative Memories

We can now find correlation matrices:

$$X_1 Y_1^T = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} [1 \ -1 \ -1 \ 1] = \begin{bmatrix} +1 & +1 & +1 & -1 \\ -1 & +1 & +1 & -1 \\ +1 & -1 & -1 & +1 \\ +1 & -1 & -1 & +1 \\ -1 & +1 & +1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$$

$$X_2 Y_2^T = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

$$X_3 Y_3^T = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$$

Example 2.3: Associative Memories

Let us consider initially a matrix containing only the first two pairs of memories, $A_1 - B_1$ and $A_2 - B_2$. Then we get

$$M = X_1 Y_1^T + X_2 Y_2^T = \begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} \tag{2.8}$$

which can be used in recall with

$$A_1^T M = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -4 & -2 & 4 \end{bmatrix}$$

Example 2.3: Associative Memories

which can be thresholded to give (1001). We can similarly feed B_1 to the network to get A_1 using $B_1^T M^T$. Alternatively, we can use MB_1 e.g.

$$MB_1 = \begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ 2 \\ 2 \\ -2 \\ 2 \end{bmatrix}$$

which can again be thresholded to give A_1. In addition, we can input slightly wrong stimuli and hope to get a correct response - this is the pattern completion properties of this type of network.

Example 2.3: Associative Memories

Let us input A_*^T=(1,0,1,1,0,1). Then we have

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -2 & -2 & 2 \end{bmatrix}$$

which is again thresholded to give B_1 which can be fed back as before to retrieve A_1. So, a noisy version of a pattern can be fed forward and back through the network to give an accurate version of the pattern.

Example 2.3: Associative Memories

Now we add in the third pattern $X_3 - Y_3$. Then M now becomes

$$\begin{bmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ 0 & -2 & 0 & 2 \end{bmatrix} + \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 3 & -1 & -1 \\ -3 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 \\ 3 & -1 & -1 & -1 \\ 1 & 1 & 1 & -3 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

Now, if we try to recall B_1 from A_1 we get

$$A_1^T M = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 3 & -1 & -1 \\ -3 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 \\ 3 & -1 & -1 & -1 \\ 1 & 1 & 1 & -3 \\ -1 & -1 & -1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & -5 & -1 & 3 \end{bmatrix}$$

Example 2.3: Associative Memories

which is again precisely B_1. Similarly, with A_2, A_3 and even the noisy version of A_1, A_*. However, the vectors used in the example were chosen with care, notice how the Y values are pairwise orthogonal to one another.

As we add more and more memories to our matrix, we will experience crosstalk and be unable to retrieve complete versions of our memories. Add in a fourth memory to the matrix M and attempt to retrieve all four memories.

Example 2.1

- Figure 2.23 illustrates a two-dimensional input space $\mathcal{L}$ consisting of four points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and $\mathbf{x}_4$. The decision boundaries of functions F_0 and F_1, indicated in the figure, correspond to the classes (hypotheses) 0 and 1 being true, respectively. From Fig. 2.23 we see that the function F_0 induces the dichotomy

$$\mathcal{D}_0 = \{\mathcal{G}_0 = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_4}\}, \mathcal{G}_1 = \{\mathbf{x_3}\}\}$$

- On the other hand, the function F_1 induces the dichotomy

$$\mathcal{D}_1 = \{\mathcal{G}_0 = \{\mathbf{x_1}, \mathbf{x_2}\}, \mathcal{G}_1 = \{\mathbf{x_3}, \mathbf{x_4}\}\}$$

- With the set $\mathcal{G}$ consisting of four points, the cardinality $|\mathcal{G}| = 4$. Hence,

$$\Delta_{\mathcal{F}}(\mathcal{G}) = 2^4 = 16$$

Example 2.1

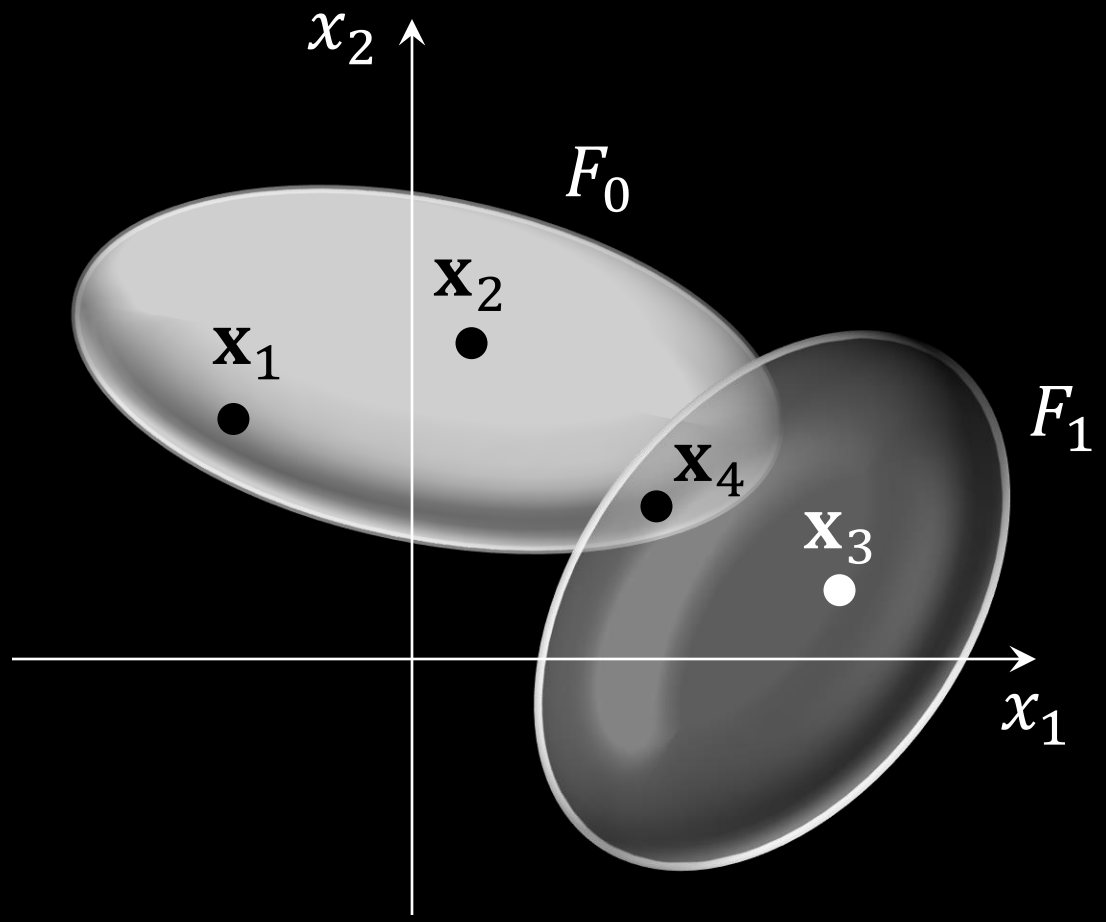


Fig. 2.23 Diagram for Example 2.1

Example 2.1

- We may now formally define the VC dimension as
  - The VC dimension of an ensemble of dichotomies $\mathcal{F}$ is the cardinality of the largest set $\mathcal{L}$ Type equation here.that is shattered by $\mathcal{F}$.
- In other words, the VC dimension of $\mathcal{F}$, written as VC dim$(\mathcal{F})$, is the largest N such that $\Delta_{\mathcal{F}}(\mathcal{G}) = 2^N$. Stated in more familiar terms, the VC dimension of the set of classification functions $\{F(\mathbf{x}, \mathbf{w})$: $\mathbf{w} \in \mathcal{W}\}$ is the maximum number of training examples that can be learned by the machine without error for all possible binary labelings of the classification functions.

Example 2.2

- To demonstrate this result, consider the situations described in Fig. 2.24 pertaining to a two dimensional input space (i.e., $m = 2$).
- In Fig. 2.24a, we have three points $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$. Three different possible labelings of these points are included in Fig. 2.24a, from which we readily see that a maximum of three lines can shatter these points.
- In Fig. 2.24b we have points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and $\mathbf{x}_4$, with points $\mathbf{x}_2$ and $\mathbf{x}_3$ labeled as 0 and points $\mathbf{x}_1$ and $\mathbf{x}_4$ labeled as 1. This time, however, we see that points $\mathbf{x}_1$ and $\mathbf{x}_4$ cannot be shattered from $\mathbf{x}_2$ and $\mathbf{x}_3$ by a line.
- The VC dimension of the decision rule described in Eq. (2.88) with $m = 2$ is therefore 3, which is in accord with the formula of Eq. (2.89).

Example 2.2

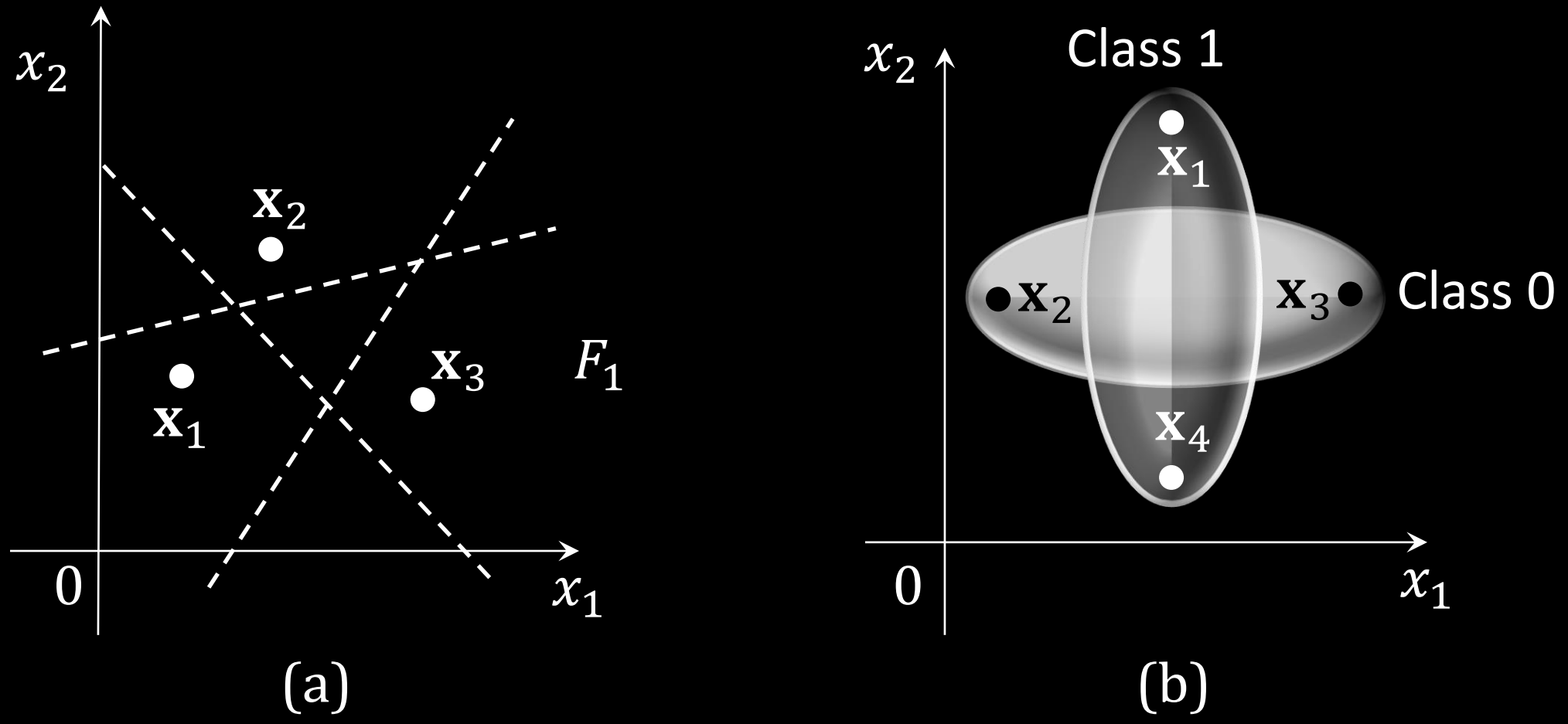


Fig. 2.24 A pair of two-dimensional data distributions for Example 2.2.

Example 2.2

- Consider a simple decision rule in an m-dimensional space $\mathcal{H}$ of input vectors, which is described by:

$$\mathcal{F}: y = \varphi\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b\right) \tag{2.88}$$

where $\mathbf{x}$ is an m-dimensional weight vector and b is a bias. The activation function φ is a threshold function; that is,

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$

The VC dimension of the decision rule in Eq. (2.88) is given by

$$VC\ \dim(\mathcal{F}) = m + 1 \tag{2.89}$$

Example 2.3

- With the VC dimension providing a measure of the capacity of a set of classification (indicator) functions, we may be led to expect that a learning machine with many free parameters would have a high VC dimension, whereas a learning machine with few free parameters would have a low VC dimension. We now present a counterexample to this statement.

- Consider the one parameter family of indicator functions, defined by

$$f(x, a) = \operatorname{sgn}(\sin(ax)), \qquad a \in \mathbb{R}$$

where sgn($\cdot$) is the signum function. Suppose we choose any number N and the requirement is to find N points that can be shattered. This requirement is satisfied by the set of functions $f(x, a)$ by choosing

$$x_i = 10^{-i}, \qquad i = 1, 2, \ldots, N$$

Example 2.3

- To separate these data points into two classes determined by the sequence

$$d_1, d_2, \ldots, d_N, \qquad d_i \in \{-1, 1\}$$

- it is sufficient that we choose the parameter a according to the formula:

$$a = \pi \left(1 + \sum_{i=1}^{N} \frac{(1 - d_i)10^i}{2} \right)$$

We thus conclude that the VC dimension of the family of indicator functions $f(x, a)$ with a single free paramenter a is infinite.

Example 3.1: Bayesian Decision Boundary

- Consider a communication system where the transmitter transmits messages $m = 0$ or $m = 1$, occurring with a priori probabilities of ¾ and ¼ respectively. The message is contaminated by a noise n, which is independent from m and takes on the values -1, 0, 1 with probabilities 1/8, 5/8 and 2/8 respectively. The received signal, or the observation, can be represented as $r = m + n$. From r, we wish to infer what the transmitted message m was (estimated state), denoted using $\hat{m}$, which also takes values on 0 or 1. When $m = \hat{m}$, the detector correctly receives the original message, otherwise an error occurs.

  a) Find the decision rule that achieves the maximum probability of correct decision. Compute the probability of error for this decision rule.

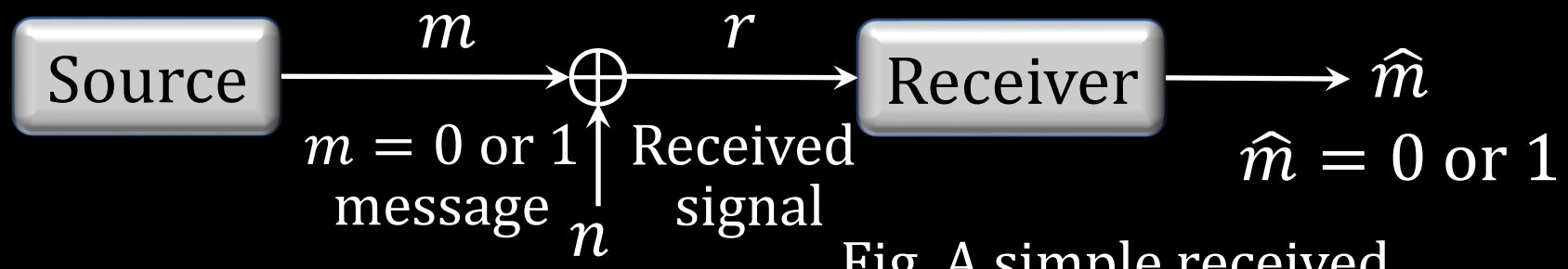


Fig. A simple received

Example 3.1: Bayesian Decision Boundary

a) Find the decision rule that achieves the maximum probability of correct decision. Compute the probability of error for this decision rule.

Solution: It is equivalent to find the decision rule that achieves the minimum probability of error. The receiver decides the transmitted message is 1, i.e., $\hat{m} = 1$ if

$$P(r|m=1).P_r(m=1) \geq P(r|m=0).P_r(m=0)$$

$$P(r|m=1) \geq P(r|m=0).3 \quad (1)$$

Otherwise, the receiver decides $\hat{m} = 0$. The likelihood functions for these two cases are

$$P(r|m=1) = \begin{cases} 1/8 & \text{if } r=0 \\ 5/8 & \text{if } r=1 \\ 2/8 & \text{if } r=2 \\ 0 & \text{otherwise} \end{cases} \qquad P(r|m=0) = \begin{cases} 1/8 & \text{if } r=-1 \\ 5/8 & \text{if } r=0 \\ 2/8 & \text{if } r=1 \\ 0 & \text{otherwise} \end{cases}$$

Example 3.1: Bayesian Decision Boundary

- Eq. 1 holds only when $r = 2$. Therefore, the decision rule can be summarized as

$$\hat{m} = \begin{cases} 1 & \text{if } r = 0 \\ 0 & \text{otherwise} \end{cases}$$

- The probability of error is as follows:

$$\begin{aligned} P_r(e) &= P_r(\hat{m} = 1|m = 0).P_r(m = 0) + P_r(\hat{m} = 0|m = 1).P_r(m = 1) \\ &= P_r(r = 2|m = 0).P_r(m = 0) + P_r(r \neq 2|m = 1).P_r(m = 1) \\ &= 0 + \left(\frac{1}{8} + \frac{5}{8}\right).\frac{1}{4} = \frac{3}{6} = 0.1875 \end{aligned} \quad (1)$$

Example 3.1: Bayesian Decision Boundary

b) Let's have the noise n be a continuous random variable, uniformly distributed between -3/4 and 2/4, and still statistically independent of m. First, plot the pdf of n. Then, find a decision rule that achieves the minimum probability of error and compute the probability of error.

Solution:

- The uniform distribution is plotted in Fig. 3.
- The decision rule is still determined by using (1)
- The likelihood functions become continuous, instead of discrete as in (a):

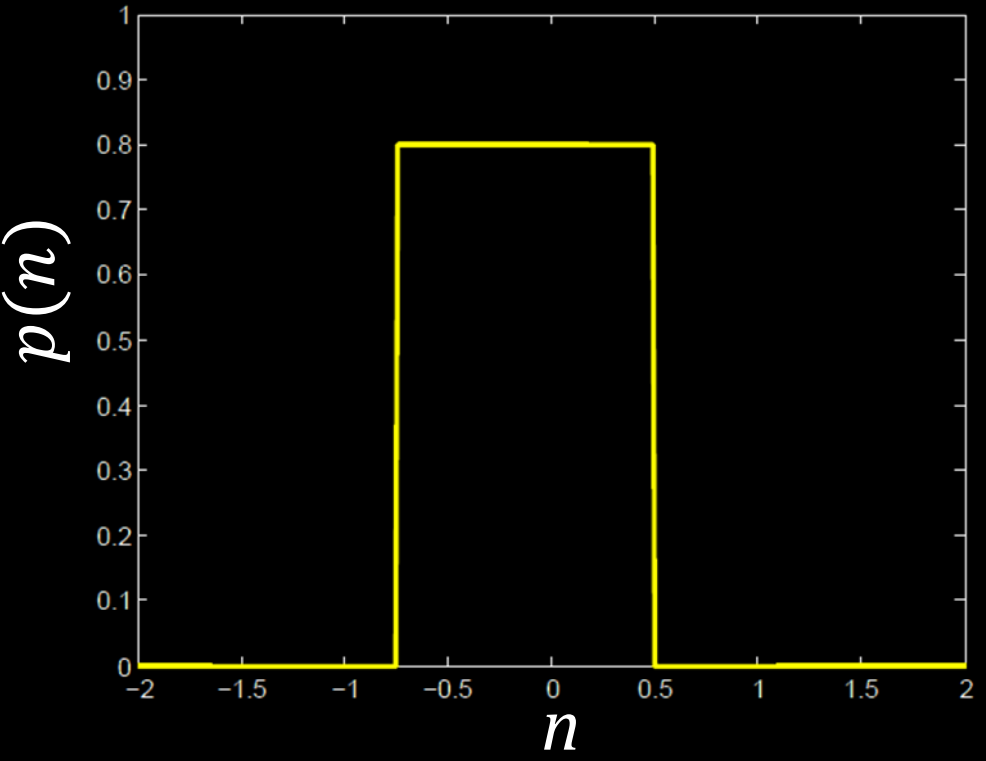


Fig. 3. The pdf of n.

Example 3.1: Bayesian Decision Boundary

$$P(r|m=1) = \begin{cases} \frac{4}{5} & if \ \frac{1}{4} < r \leq \frac{6}{4} \\ 0 & \text{otherwise} \end{cases}$$

$$P(r|m=0) = \begin{cases} \frac{4}{5} & if \ \frac{-3}{4} < r \leq \frac{2}{4} \\ 0 & \text{otherwise} \end{cases}$$

- The interesting region is where the two pdf's overlap with each other,
- namely when $1/4 < r \leq 2/4$.
- From (1), we know we should decide $\hat{m} = 0$ for this range.

Example 3.1: Bayesian Decision Boundary

- The decision rule can be summarized as

$$\hat{m} = \begin{cases} 0 & \text{if } -\frac{3}{4} < r \leq \frac{1}{4} \\ 0 & \text{if } +\frac{1}{4} < r \leq \frac{1}{4} \\ 1 & \text{if } \frac{1}{4} < r \leq \frac{2}{4} \end{cases}$$

- Note that at the decision boundaries, there is ambiguity on which decision we should make. Again, either decision won't change the probability of error, so it is acceptable to decide both ways.
- The probability of error is then,

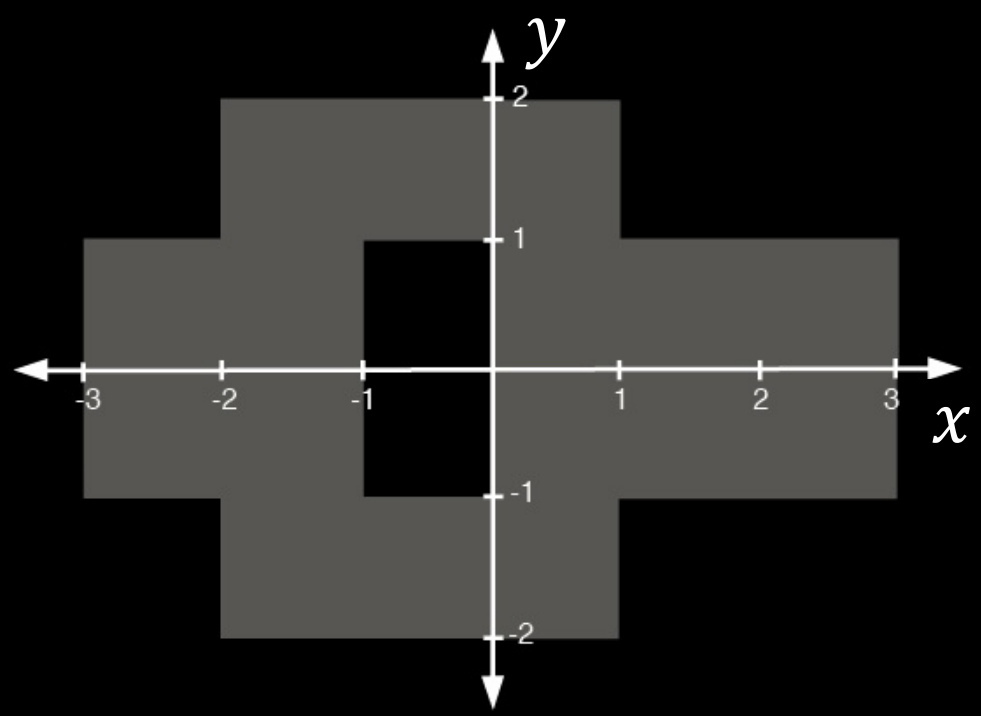
$$P_r(e) = P_r(\hat{m} = 1|m = 0).P_r(m = 0) + P_r(\hat{m} = 0|m = 1).P_r(m = 1)$$

$$P_r\left(\frac{1}{4} < r \leq \frac{2}{4}\middle|m = 1\right).P_r(m = 1) = \left(\frac{2}{4} - \frac{1}{4}\right).\frac{4}{5}.\frac{1}{4} = \frac{1}{20}$$

Example 3.2: Bayesian Decision Boundary

- Suppose x and y are random variables. Their joint density, depicted below, is constant in the shaded area and 0 elsewhere,
  a) Let w_1 be the case when $x \leq 0$, and w_2 be the case when $x > 0$. Determine the *a priori* probabilities of the two classes $P(w_1)$ and $P(w_1)$. Let y be the observation from which we infer whether w_1 or w_2 happens. Find the likelihood functions, namely, the two conditional distributions $p(y)|w_1)$ and $p(y)|w_2)$.

Fig. 5: The joint distribution of x and y.



Example 3.2: Bayesian Decision Boundary

Solution a)

- By simply counting the number of unit squares in the shaded areas on the left and right sides of the line $x = 0$; we can directly find out that there are 8 unit squares on each side. Thus, the two cases are equally likely, i.e. $\mathrm{P}(w_1) = \mathrm{P}(w_2) = 0.5$.
- It's straight-forward to obtain likelihood functions $p(y|w_1)$ and $p(y|w_2)$ by counting the number of unit squares for different ranges of y. We need to be careful with normalizing the integral of the distribution 1. See Figure 6.

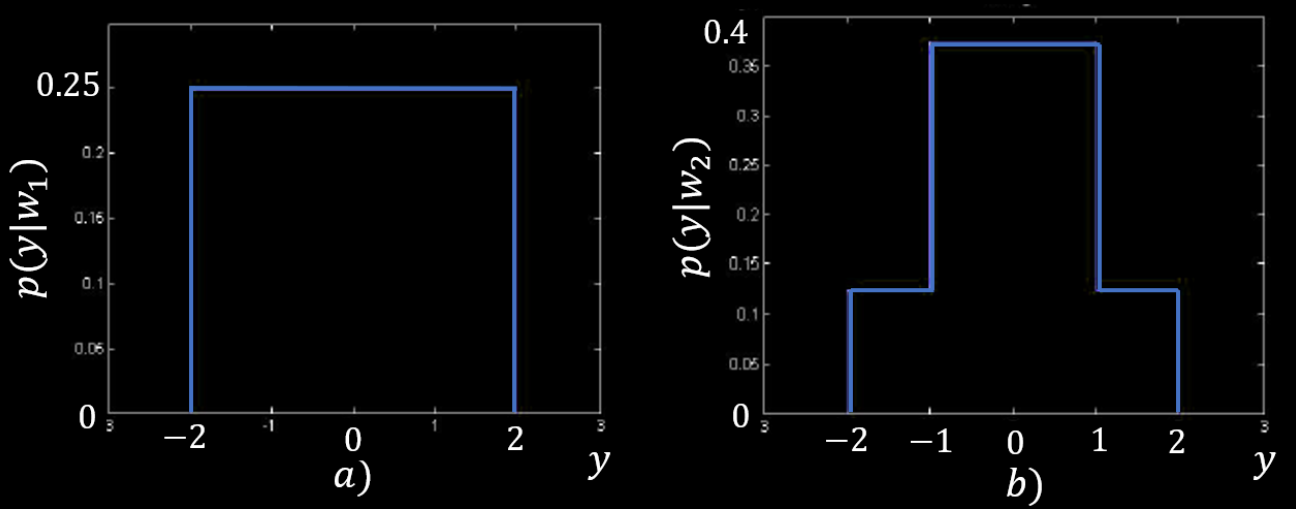


Fig. 6: Likelihood functions: (a) $p(y|w_1)$ left, (b) $p(y|w_2)$ right.

Example 3.2: Bayesian Decision Boundary

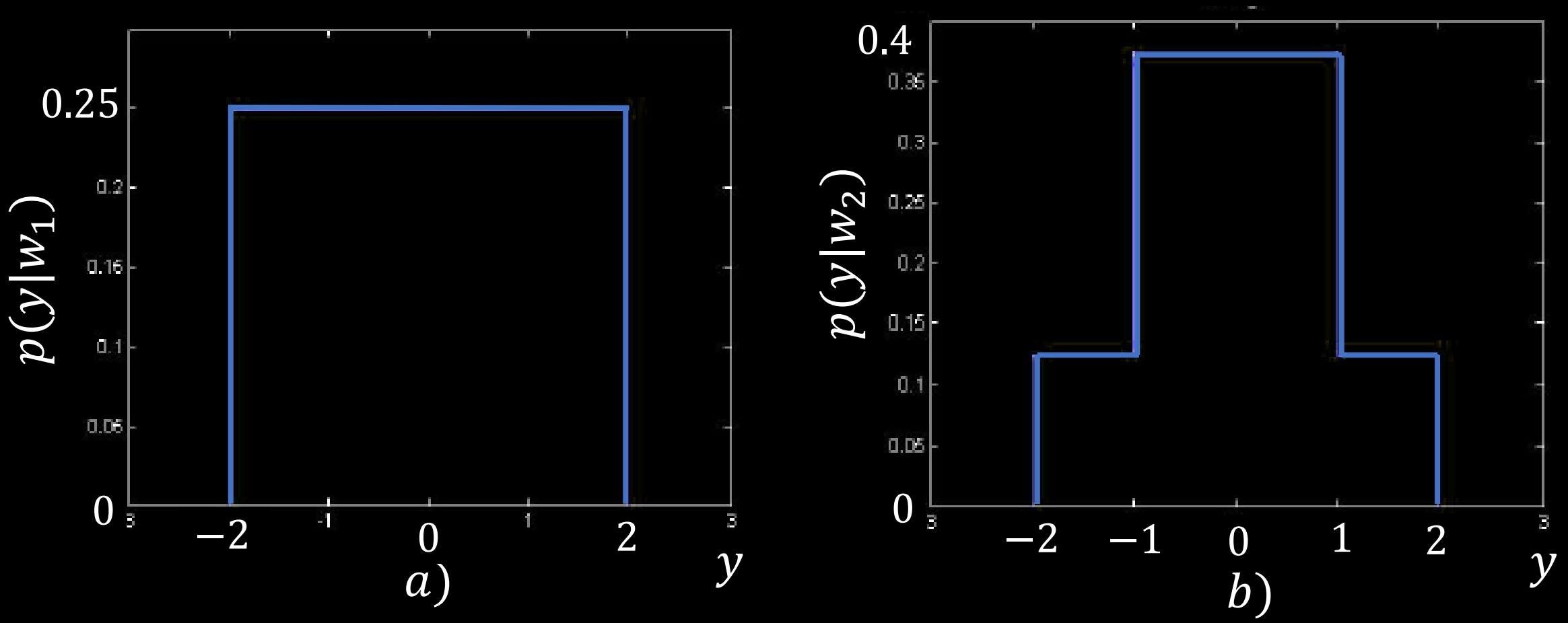


Fig. 6: Likelihood functions: (a) $p(y|w_1)$ left, (b) $p(y|w_2)$ right.

Example 3.2: Bayesian Decision Boundary

b) Find the decision rule that minimizes the probability of error and calculate what the probability of error is. Please note that there will be ambiguities at decision boundaries, but how you classify when y falls on the decision boundary doesn't affect the probability of error.

Solution b)

- As shown in (a), the *a priori* probabilities of w_1 and w_2 are identical. The minimum probability of error decision rule simply relies on the comparison of the two likelihood functions. In other words, it becomes a ML decision rule.
- The decision rule can be summarized as:

$$\widehat{w} = \begin{cases} w_1 & \text{if } -2 < y \leq -1 \quad \text{or} \quad 1 < y \leq 2 \\ w_2 & \text{if } -1 < y \leq 1 \end{cases}$$

Example 3.2: Bayesian Decision Boundary

- The probability of error is thus:

$$P_r(e) = P_r(\hat{w} = w_1|w_2)P_r(w_1) + P_r(w_1|w_2)P_r(w_2)$$
$$= \frac{1}{2}P_r(-1 < y \leq 1|w_1) + \frac{1}{2}P_r(-2 < y \leq -1, 1 < y \leq 2|w_2)$$
$$= \frac{1}{2} * \frac{1}{2} + \frac{1}{2} * \frac{1}{4} = \frac{3}{8}$$

Example 3.3: Bayesian Decision Boundary

Consider a simple classification problem. We wish to train a perceptron to differentiate between two different types of nuts. We have 3 examples of each type of nut and can measure 2 characteristics of each nut: its length and its weight. Then our data is shown in tabular form in Table 3.1.

Nut	type A - 1	type A - 2	type A - 3	type B - 1	type B - 2	type B - 3
Length (cm)	2.2	1.5	0.6	2.3	1.3	0.3
Weight (g)	1.4	1.0	0.5	2.0	1.5	1.0

Table 3.1: The lengths and weights of six instances of two types of nuts

(1, 2.2, 1.4) with associated training output 1 (equal to class A)
(1, 1.5, 1.0) with associated training output 1
(1, 0.6, 0.5) with associated training output 1
(1, 2.3, 2.0) with associated training output -1 (equal to class B)
(1, 1.3,1.5) with associated training output -1
(1, 0.3,1.0) with associated training output -1

Example 3.3: Bayesian Decision Boundary

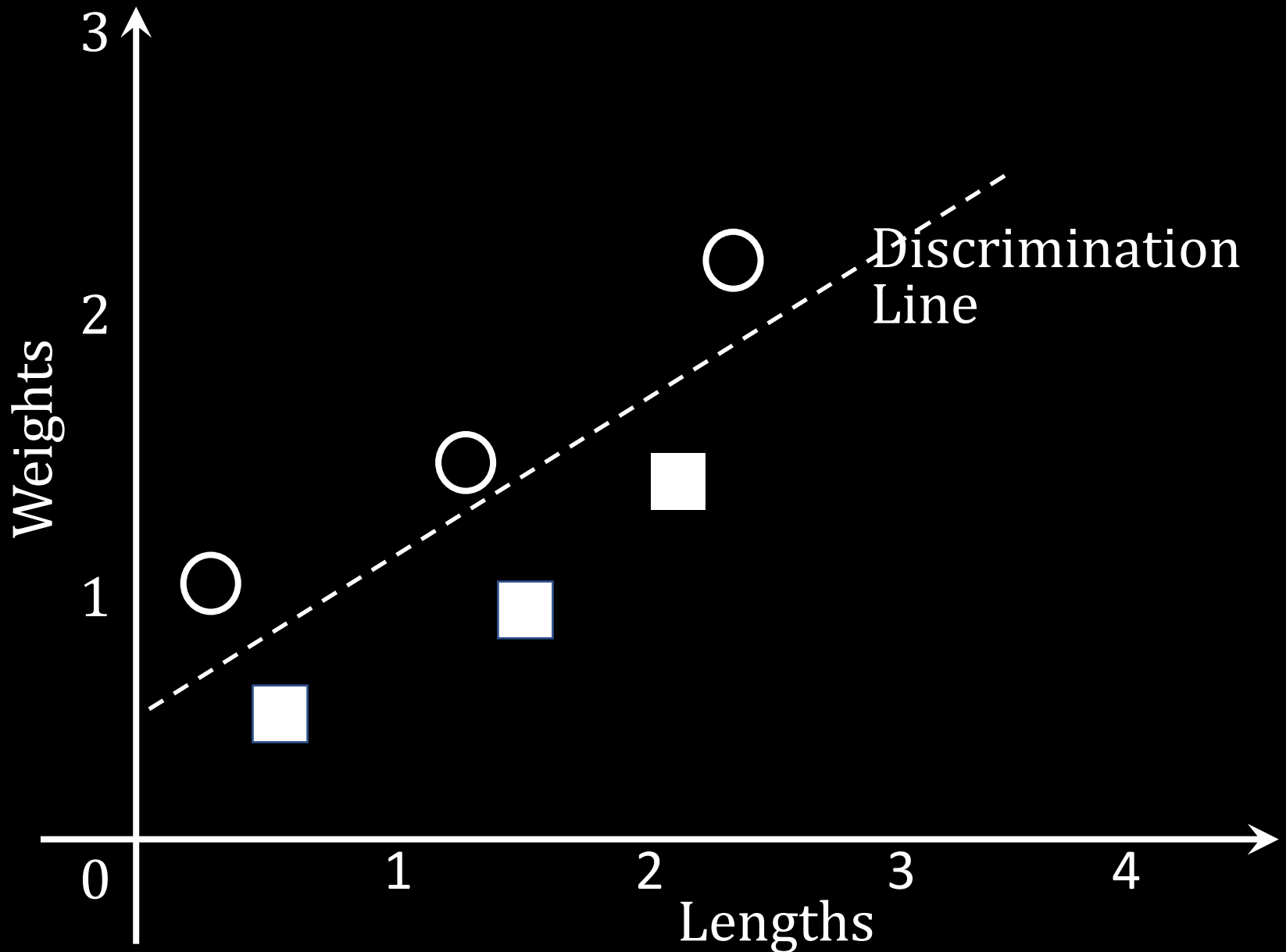
Note that the initial 1 in each case corresponds to the bias term. Its trainable weight will converge to the bias for the output neuron. So we have a network with three inputs and a single output which will tell us with a 1 if the nut is A or a -1 if the nut is a B. Therefore, our network has 3 weights.

Now we set our initial weights to small random values. Select randomly one of the patterns e.g. The third. Feed the activation forward through the weights and perform a thresholding at the output. If the output neuron is firing 1, do nothing. If, however, the output neuron is firing -1, it is wrongly classifying the nut and its weights must be changed. Change each of the three weights according to the perceptron learning rule.

The weights in this simple problem are guaranteed to converge to something like $w_0 = 0.5, w_1 = 2, w_2 = 3$. A diagrammatic representation of the solution is shown in Figure 3.2.

Example 3.3: Bayesian Decision Boundary

Figure 3.2: The two classes of nut and a discrimination line which can differentiate between the classes



Example 3.3: Bayesian Decision Boundary

Now it is clear that it is in fact trivial to differentiate between these two classes on the basis of these six instances. However, the perceptron has been shown to be capable of solving more difficult problems though we must in any real problem ensure that the data on which we are training the network is representative of the data as a whole. Only then can we state with confidence that we have a perceptron capable of distinguishing the classes.

Statisticians would say that we have created a decision surface, or discrimination line, between the two classes. Discriminant functions for the classes would be $g_A(x, y)$ and $g_B(x, y)$ where we would state that a point (x,y) belonged to class A if and only $g_A(x, y) > g_B\ (x, y)$; otherwise, (x,y) belongs to class B.

The line (or plane) which differentiates the classes is $g_A(x, y) - g_B\ (x, y) = 0$, which is the line shown in the Figure.

4.5 XOR Problem

Problem 4.1: Back-Propagation Learning

Fig. P4.1 shows a neural network involving a single hidden neuron for solving the XOR problem; this network may be viewed as an alternative to that considered in Section 4.5. Show that the network of Fig. P4.1 solves the XOR problem by constructing (a) decision regions, and (b) a truth table for the network.

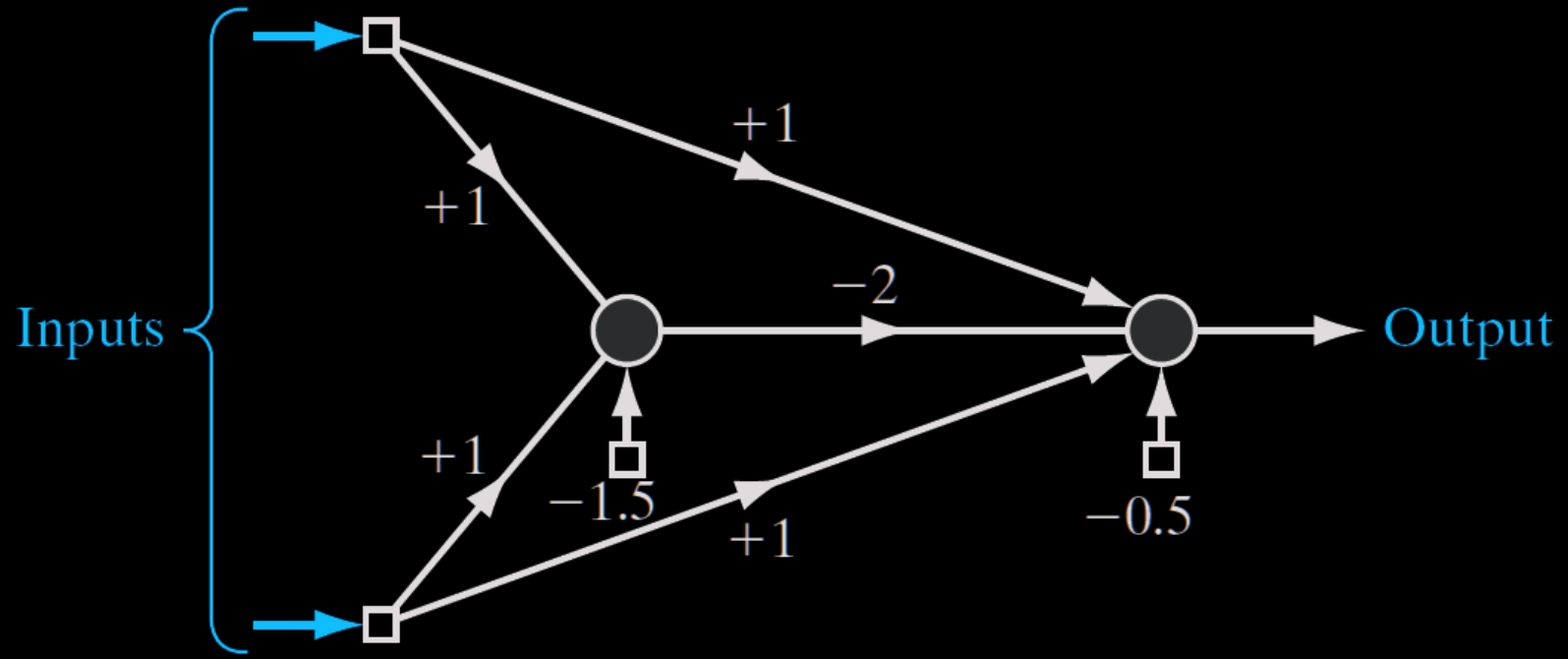


FIGURE P4.1

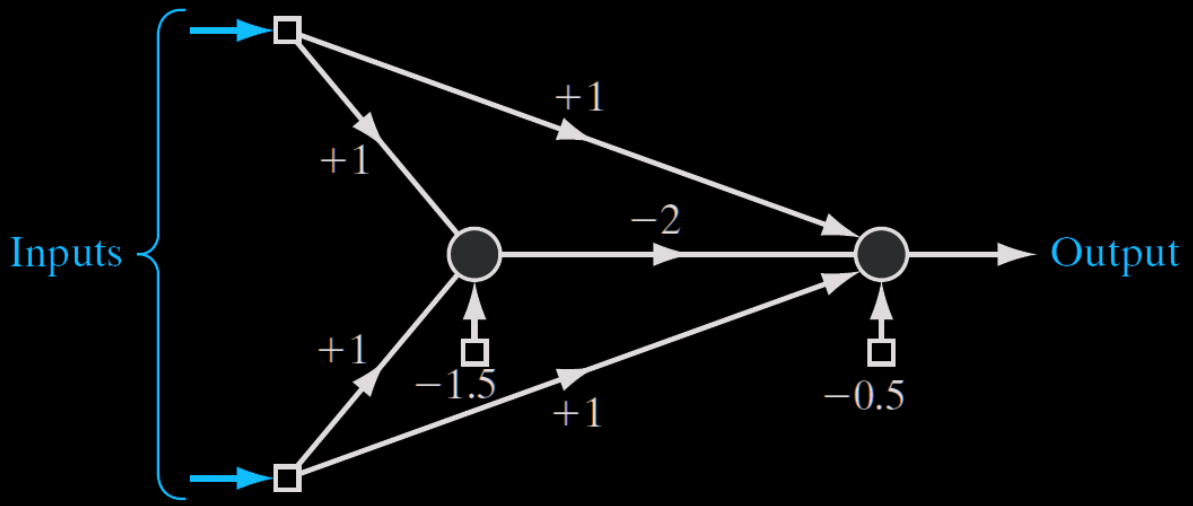
4.5 XOR Problem

Solution 4.1: Back-Propagation Learning

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

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

The induced local field of neuron 1 is $v_1 = x_1 + x_2 - 1.5$.



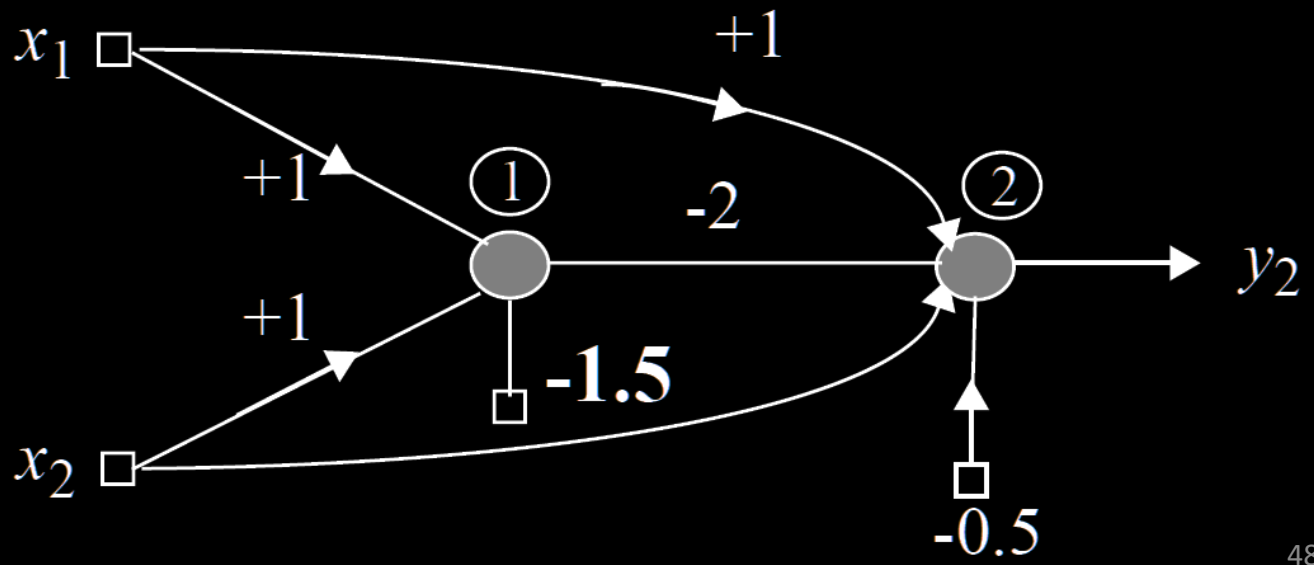


FIGURE P4.1

4.5 XOR Problem

Solution 4.1: Back-Propagation Learning

The induced local field of neuron 1 is $v_1 = x_1 + x_2 - 1.5$.

We may 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_1	0	0	0	1

The induced local field of neuron 2 is $v_2 = x_1 + x_2 - 2y_1 - 0.5$.

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

4.5 XOR Problem

Problem 4.4: Back-Propagation Learning

Consider the simple example of a network involving a single weight, for which the cost function is $\mathcal{E}(w) = k_1(w - w_0)^2 + k_2$, where w_0, k_1 and k_2 are constants.

A back-propagation algorithm with momentum α is used

to minimize $\mathcal{E}(w)$. Explore the way in which the inclusion of the momentum constant α influences the learning process, with particular reference to the number of steps required for convergence versus α.

4.5 XOR Problem

Solution 4.4: Back-Propagation Learning

In order to see the effect of the sequence of pattern presentations on the synaptic weights due to the momentum constant α we rewrite Eq. (4.41) as a time series with index t. The index t goes from the initial time 0 to the current time n. Equation (4.41) may be viewed as a first-order difference equation in the weight correction $\Delta w_{ji}(n)$. Solving this equation for $\Delta w_{ji}(n)$, we have

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t)$$

which represents a time series of length $n + 1$. From Eqs. (4.13) and (4.16), we note that the product $\delta_j(n)y_i(n)$ is equal to $-\partial \mathcal{E}(n)/\partial w_{ji}(n)$. Accordingly, we may rewrite Eq. (4.42) in the equivalent form

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)} \tag{4.43}$$

4.5 XOR Problem

Solution 4.4: Back-Propagation Learning

From Eq. (4.43) of the text we have

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

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

$$\mathcal{E}(w) = k_1(w - w_0)^2 + k_2 \tag{2}$$

Hence, taking partial derivative of (2) w.r.t. $w_{ji}(t)$ and substituting into (1) we have

$$\Delta w(n) = -2k_1\eta \sum_{t=1}^{n} \alpha^{n-t}(w(t) - w_0) \tag{3}$$

In this case, the partial derivative $\partial \mathcal{E}(w)/\partial w(t)$ has the same algebraic sign on consecutive iterations. Hence, with $0 < \alpha < 1$ 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.