EENG582: Artificial Neural Networks

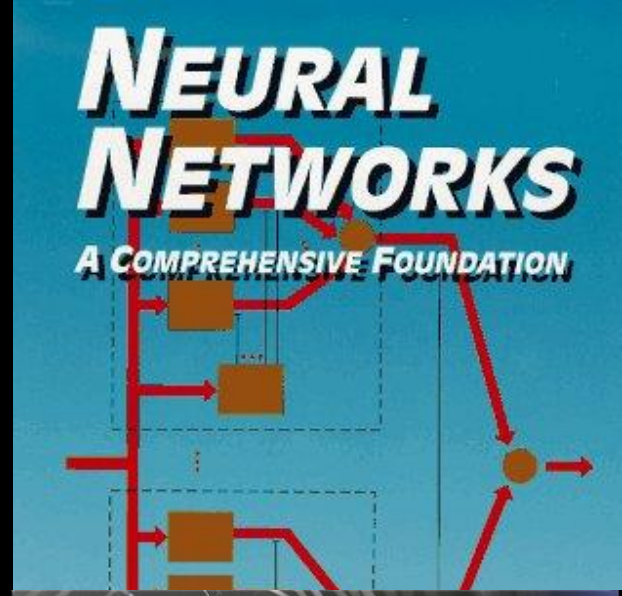
Neural Networks
A Comprehensive Foundation
by S. Haykin

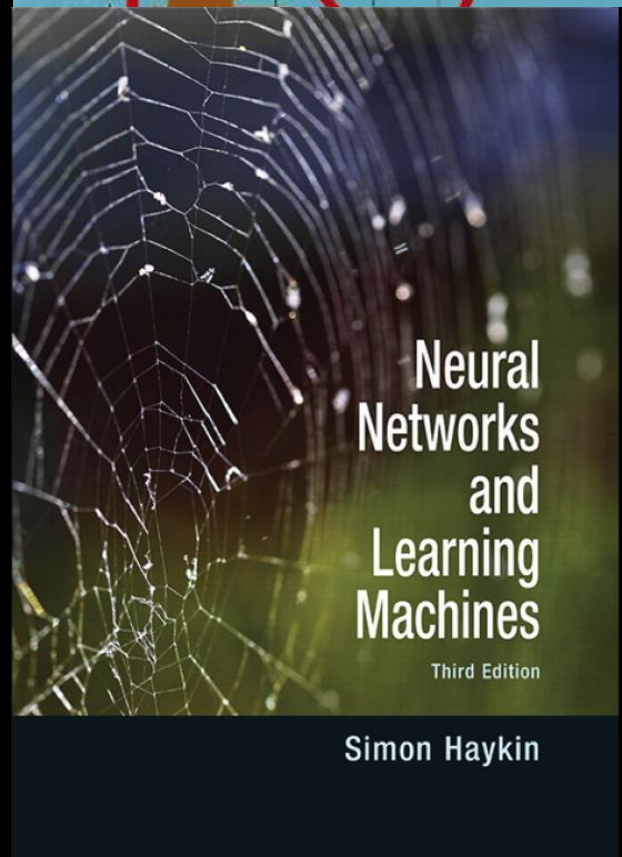
Sections 4.5 – 4.8

4 Multilayer Perceptrons

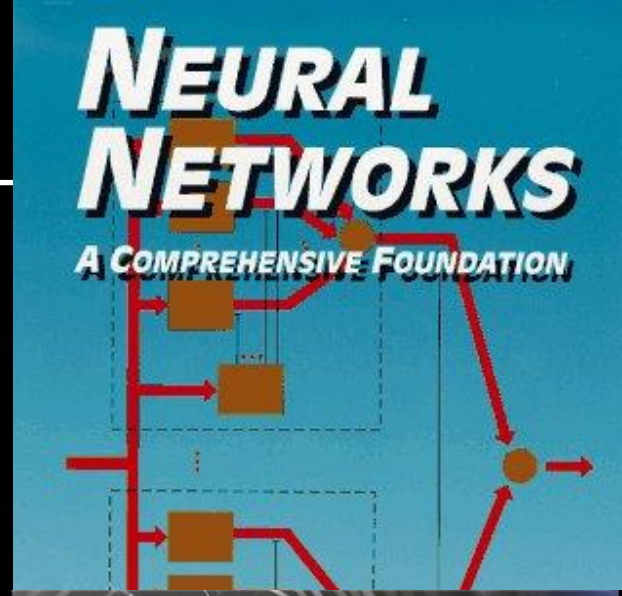
Prof. Dr. Hasan AMCA
Electrical and Electronic Engineering Department
(ee.emu.edu.tr)

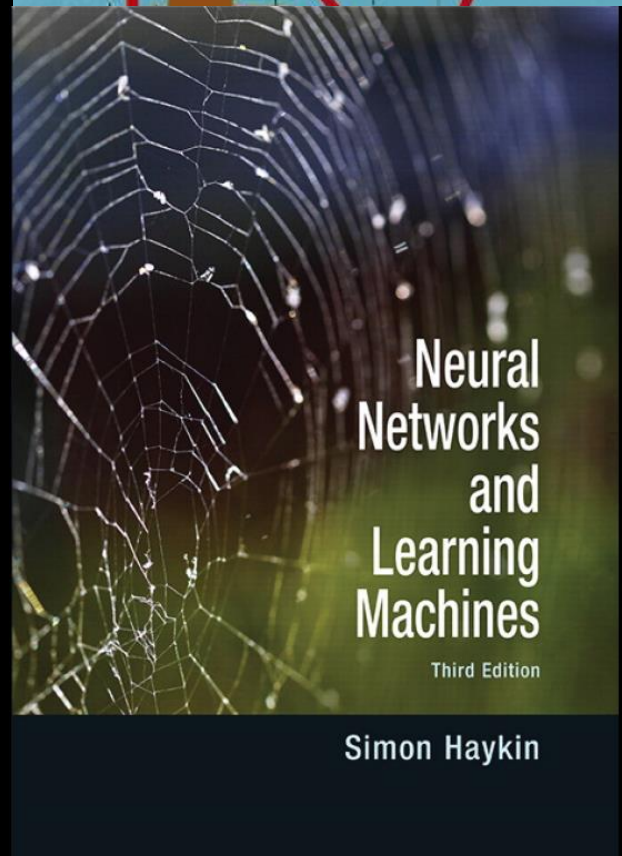
Eastern Mediterranean University
(emu.edu.tr)





3 Single Layer Perceptrons





4.5 XOR Problem

- In the elementary (single-layer) perceptron there are no hidden neurons.
- Consequently, it cannot classify input patterns that are not linearly separable.
- This situation arises in the Exclusive OR (XOR) problem, which is a special case of a classifying points in the unit hypercube where each point is either in class 0 or class 1.
- In the XOR problem, we need consider only the four corners of a unit square that correspond to the input patterns (0,0), (0,1), (1,1), and (1,0).
- The first and third input patterns are in class 0, as shown by

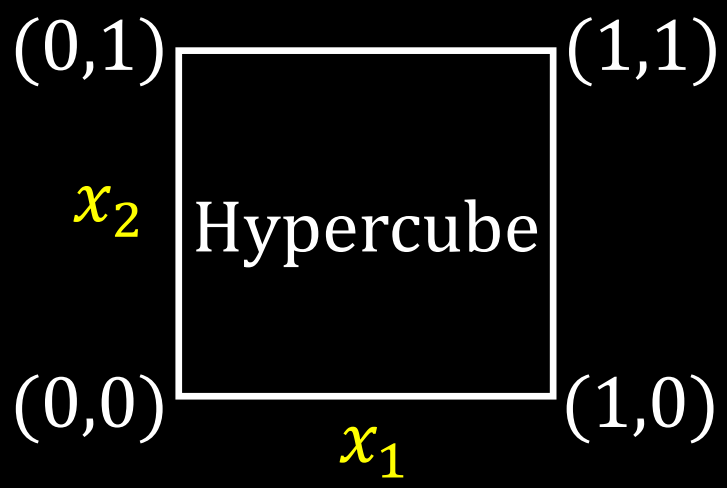
$x_1 \quad x_2 = x_1 \oplus x_2$

$0 \oplus 0 = 0$

$1 \oplus 1 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

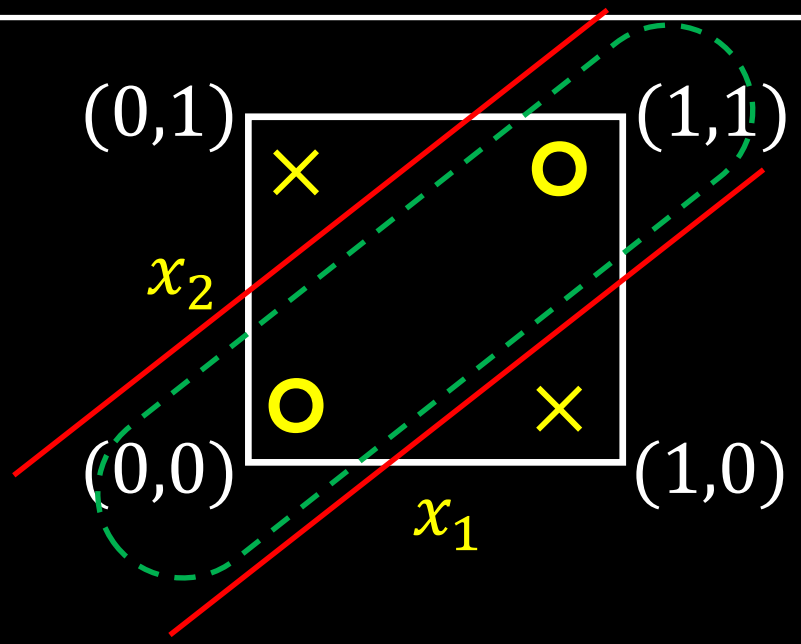


Video on
Description of the XOR Problem
https://youtu.be/kNPGXgzxoHw

Video on
Plotting the decision boundaries
https://youtu.be/s8pDf2Pt9sc

4.5 XOR Problem

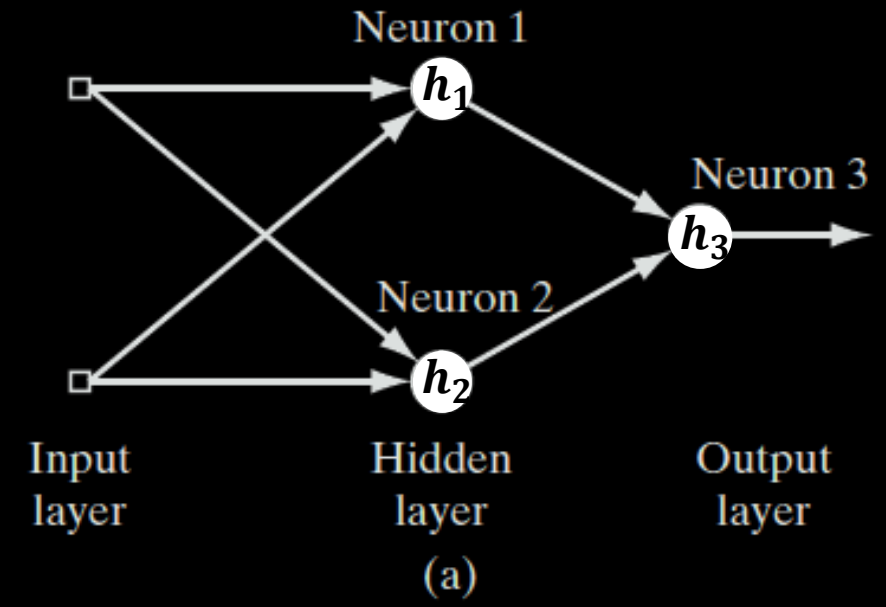
- We first recognize that the use of a single neuron with two inputs results in a straight line for a decision boundary in the input space.
  - for all points on one side of this line, the neuron outputs 1;
  - for all points on the other side of the line, it outputs 0.
- The position and orientation of the line in the input space are determined by the synaptic weights of the neuron connected to the input nodes, and the bias applied to the neuron.



- It is clear from the figure that we cannot construct a straight line for a decision boundary so that
  - (0,0) and (0,1) lie in one decision region, and
  - (0,1) and (1,0) lie in the other decision region.
- In other words, an elementary perceptron cannot solve the XOR problem.

4.5 XOR Problem

- We may solve the XOR problem by using a single hidden layer with two neurons, as in Fig. 4.8a.
- The signal-flow graph of the network is shown in Fig. 4.8b.



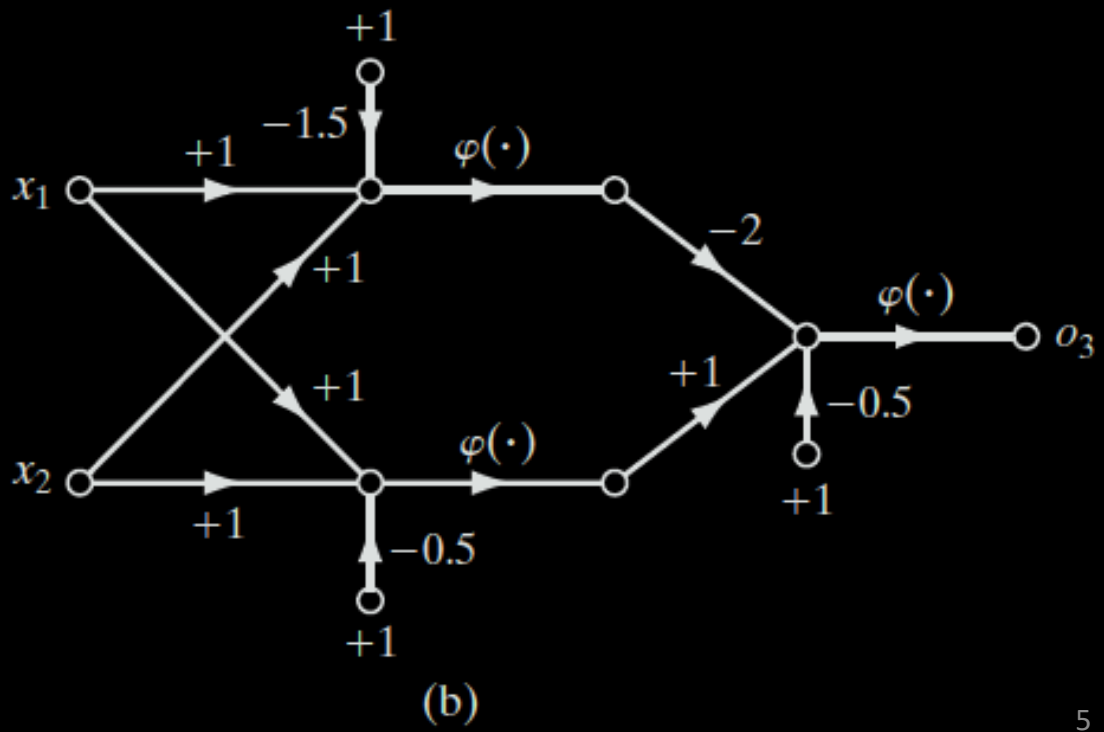
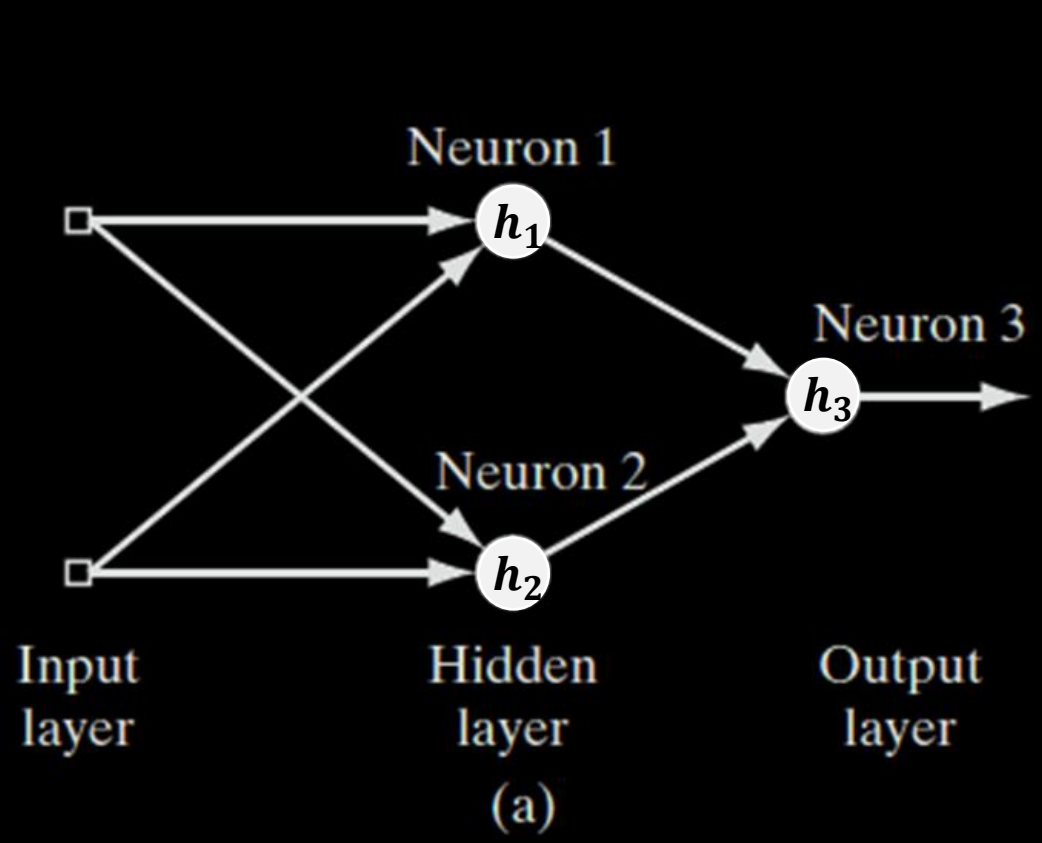


Fig. 4.8 (a) Architectural graph of network for solving the XOR problem. (b) Signal-flow graph of the network.

4.5 XOR Problem



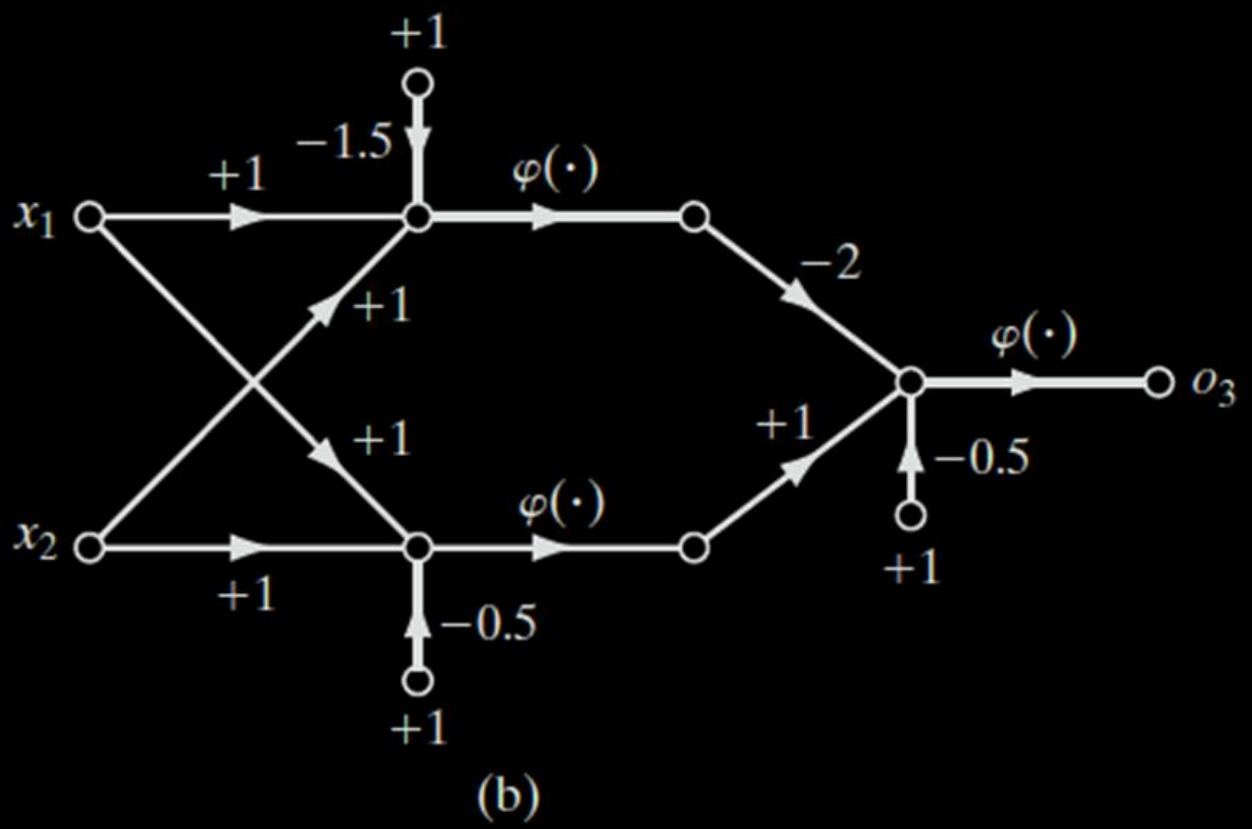


Fig. 4.8 (a) Architectural graph of network for solving the XOR problem. (b) Signal-flow graph of the network.

4.5 XOR Problem

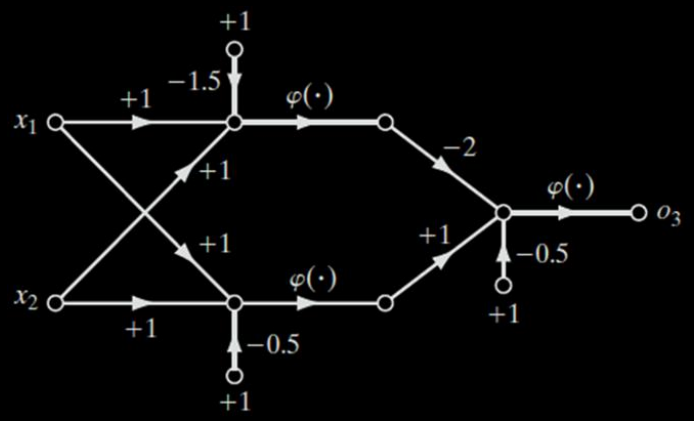
- The top neuron, labeled 1 in the hidden layer, is characterized as:

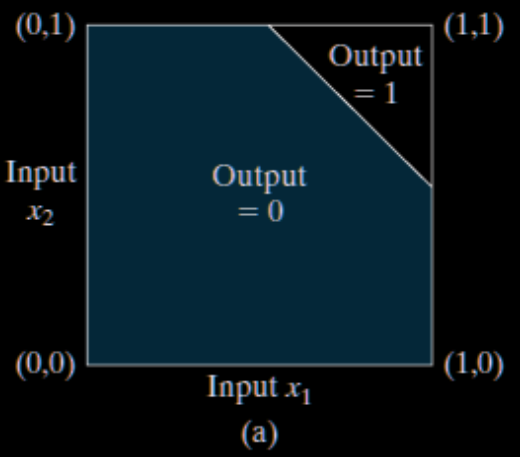
$$w_{11} = w_{12} = +1 \quad \text{and} \quad b_1 = 1.5$$

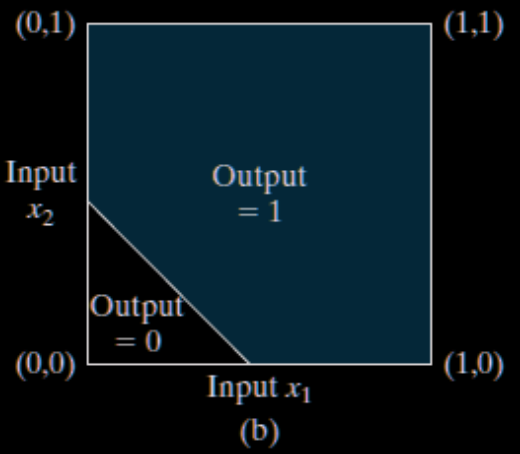
- Slope of the decision boundary constructed by this hidden neuron is equal to +1 and positioned as in Fig. 4.9a.
- The bottom neuron, labeled 2 in the hidden layer, is characterized as:

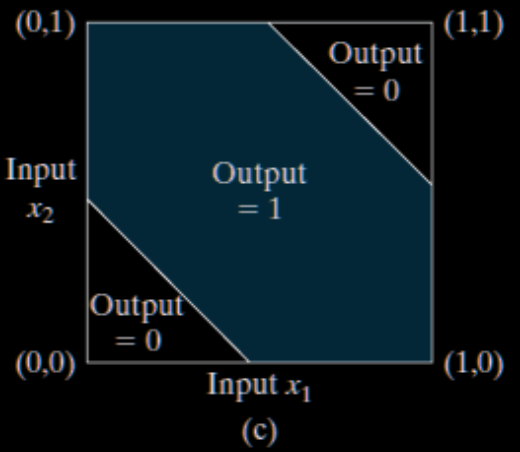
$$w_{21} = w_{22} = +1 \quad \text{and} \quad b_2 = -0.5$$

- The orientation and position of the decision boundary constructed by this second hidden neuron are as shown in Fig. 4.9b.







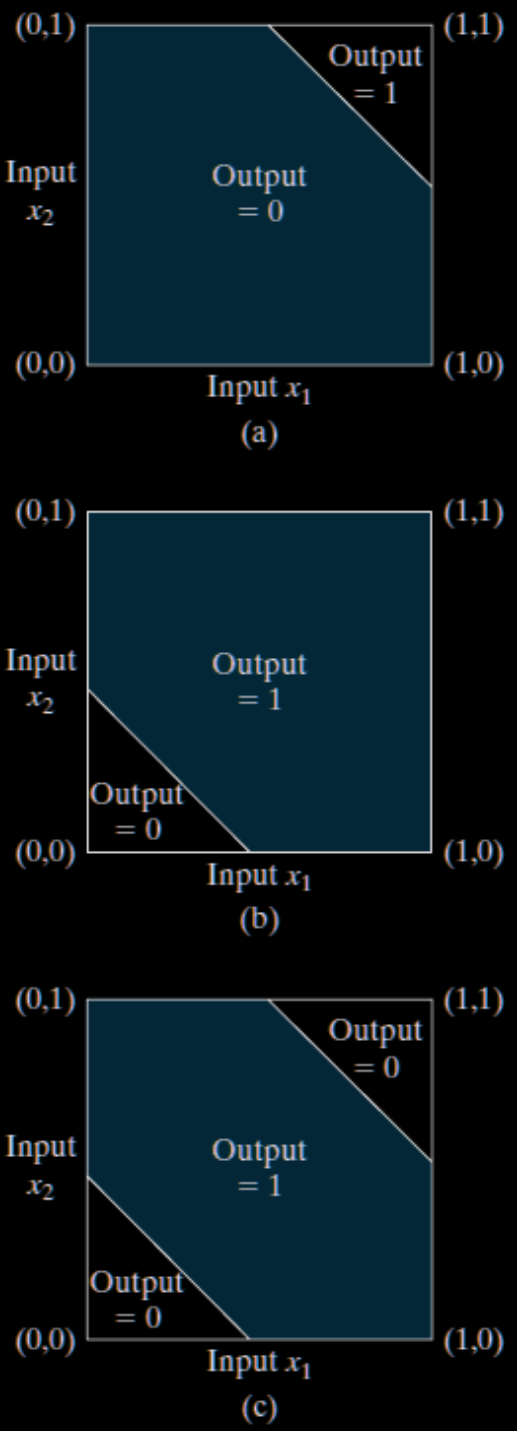


4.5 XOR Problem

- Output neuron, labeled 3 in Fig. 4.8a, is characterized as:

$$w_{31} = -2 \qquad w_{32} = +1 \qquad \text{and} \qquad b_3 = -0.5$$

- Function of the output neuron is to construct a linear combination of the decision boundaries formed by the two hidden neurons as shown in Fig. 4.9c.
- When both hidden neurons are off whenever the input pattern is (0,0), the output neuron remains off.
- When both hidden neurons are on, whenever the input pattern is (1,1), the output neuron is switched off.
- When the top hidden neuron is off and the bottom hidden neuron is on, whenever the input pattern is (0,1) or (1,0), the output neuron is switched on.



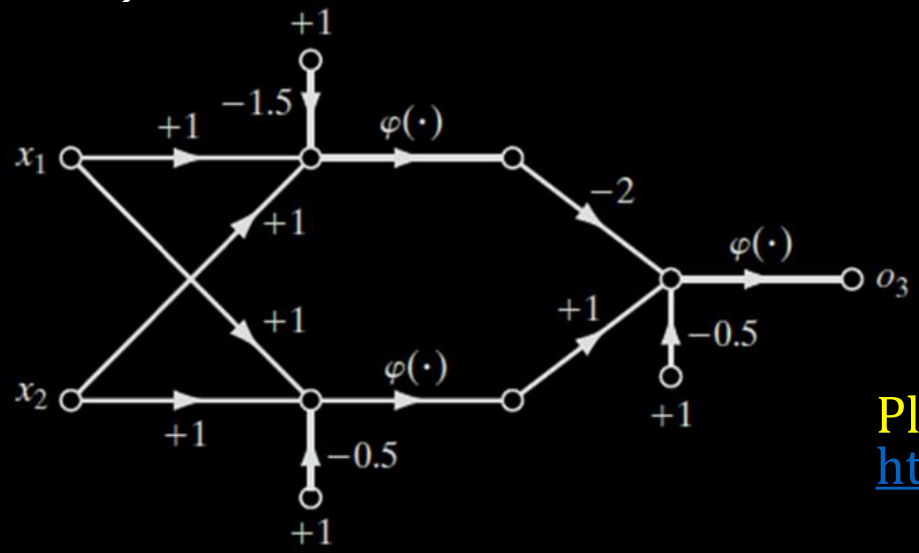
4.5 XOR Problem

- The following videos Show calculation of the outputs based on the four possible input combinations and plot the decision boundaries based on the output values by assigning 0.5 to the signum function and solving for the equations
- For $(x_1, x_2) = (0,0)$,

$$h_1 = \sigma\big((0 \times 1 + 0 \times 1) - 1.5\big) = \sigma(-1.5) \simeq 0 \quad \Rightarrow \sigma(x_1 + x_2 - 1.5) = 0.5$$

$$h_2 = \sigma\big((0 \times 1 + 0 \times 1) - 0.5\big) = \sigma(-0.5) \simeq 0 \quad \Rightarrow \sigma(x_1 + x_2 - 0.5) = 0.5$$

$$h_3 = \sigma\big((0 \times -2 + 0 \times 1) - 0.5\big) = \sigma(-0.5) \simeq 0 \quad \Rightarrow \sigma(-2x_1 + x_2 - 0.5) = 0.5$$



Video: Description of XOR Problem
https://youtu.be/kNPGXgzxoHw

Plotting the decision boundaries
https://youtu.be/s8pDf2Pt9sc

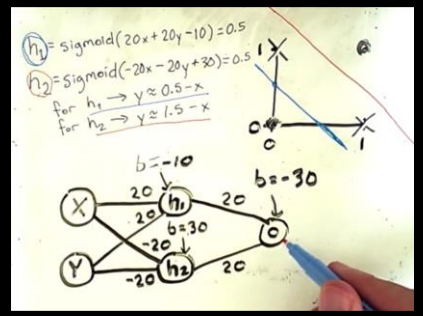
4.5 XOR Problem

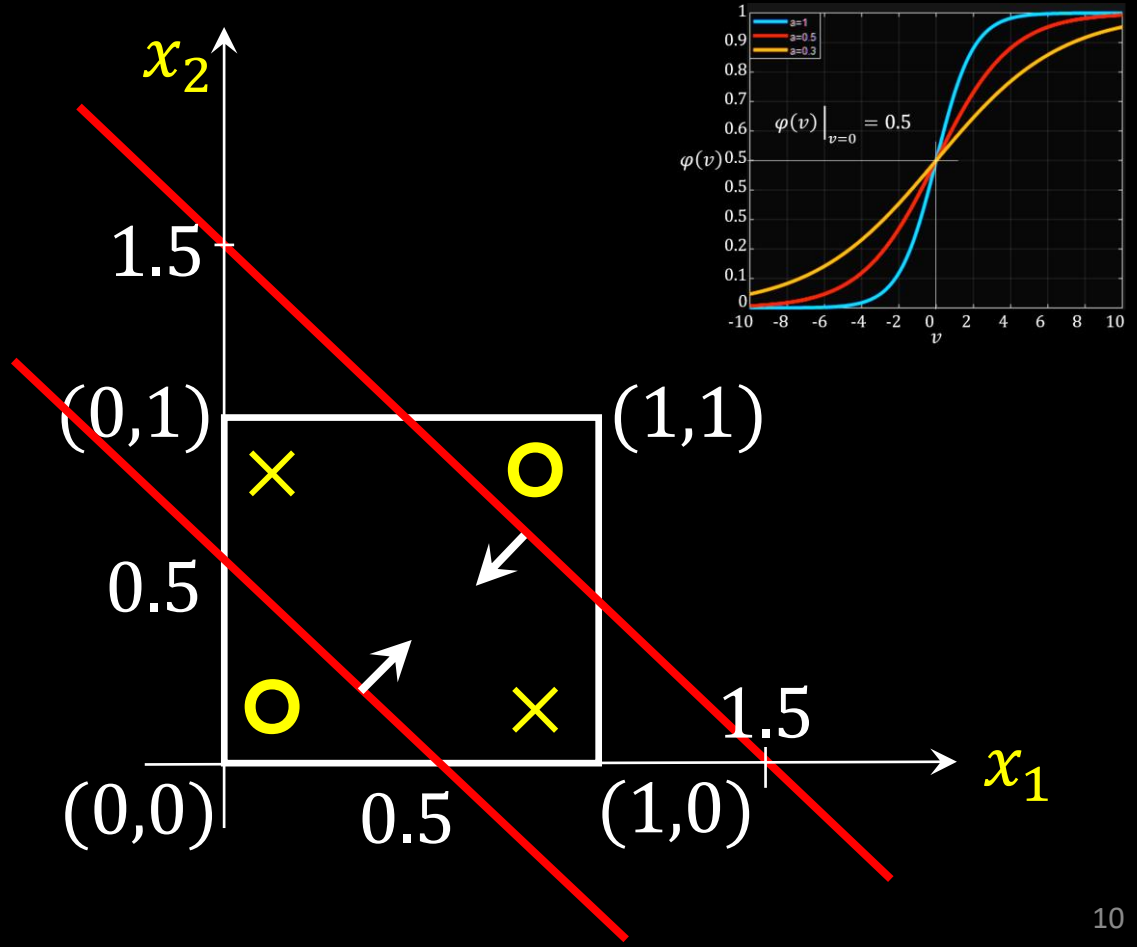
- For arbitrary inputs (x_1, x_2), we get

  $h_1 = \sigma\big((1 \times x_1 + 1 \times x_2) - 1.5\big) = 0.5$, since $\sigma(0) = 0.5$ => $x_2 = -x_1 + 1.5$

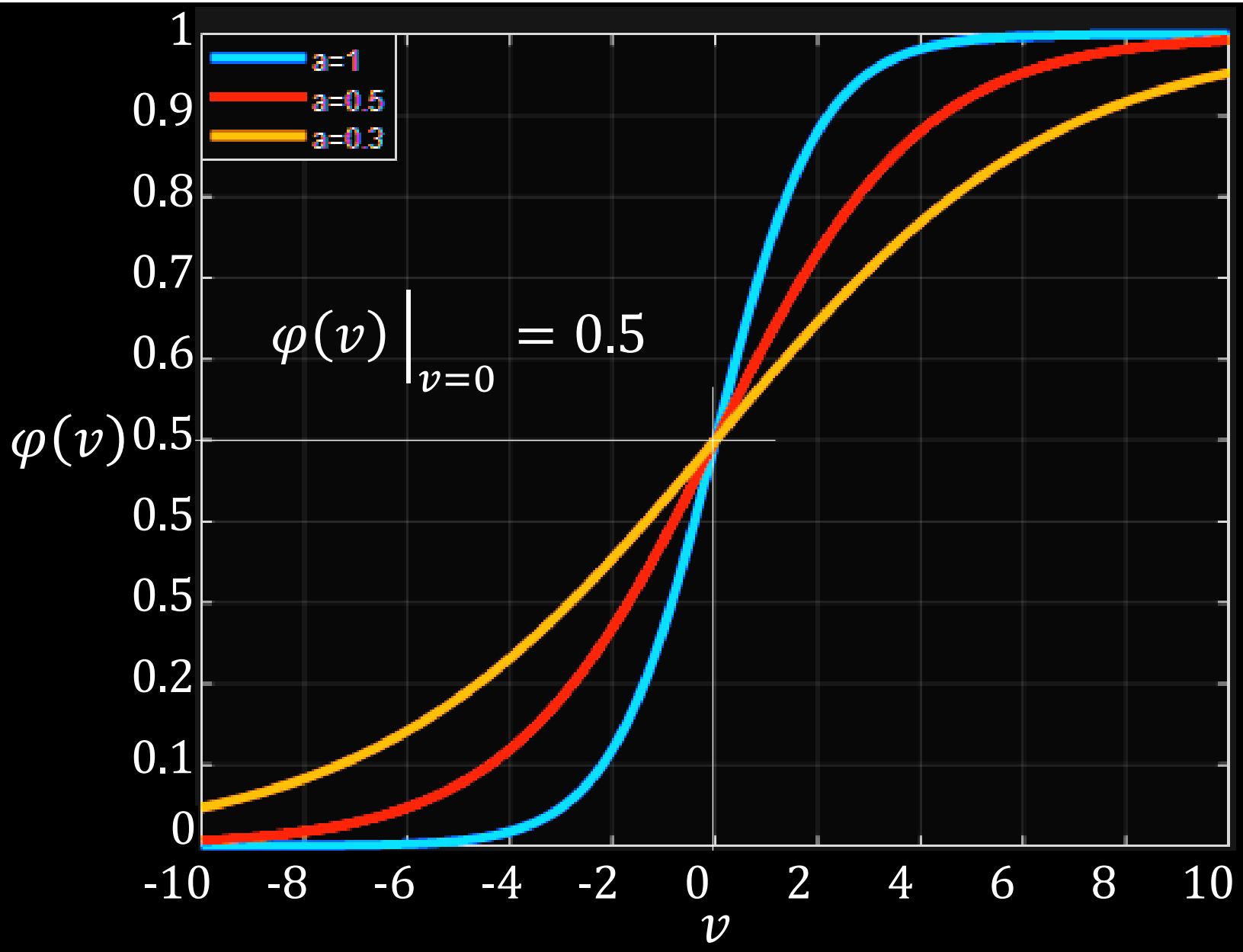
  $h_2 = \sigma\big((1 \times x_1 + 1 \times x_2) - 0.5\big) = 0.5$, since $\sigma(0) = 0.5$ => $x_2 = -x_1 + 0.5$

- Function of the output neuron is to construct a linear combination of the decision boundaries formed by the two hidden neurons h_1 and h_2 shown in the Fig. 4.8.
- These are shown by the red lines in the figure on the right.





4.5 XOR Problem



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.

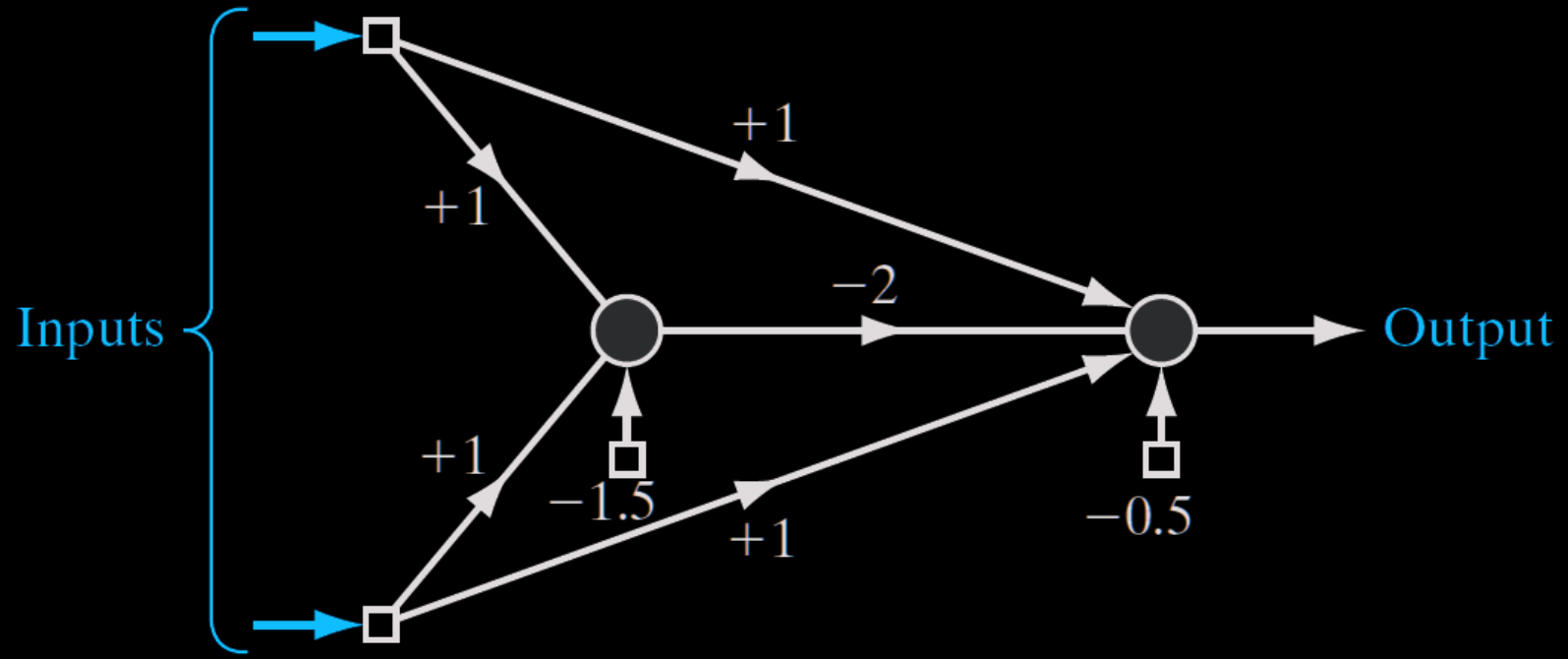


FIGURE P4.1

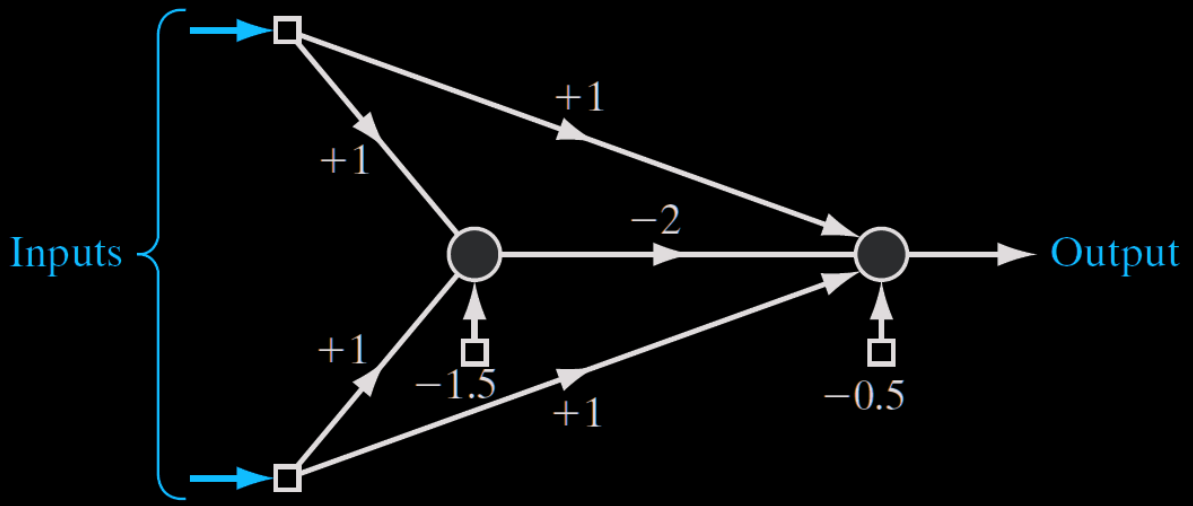
4.5 XOR Problem

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 $v_1 = x_1 + x_2 - 1.5$.



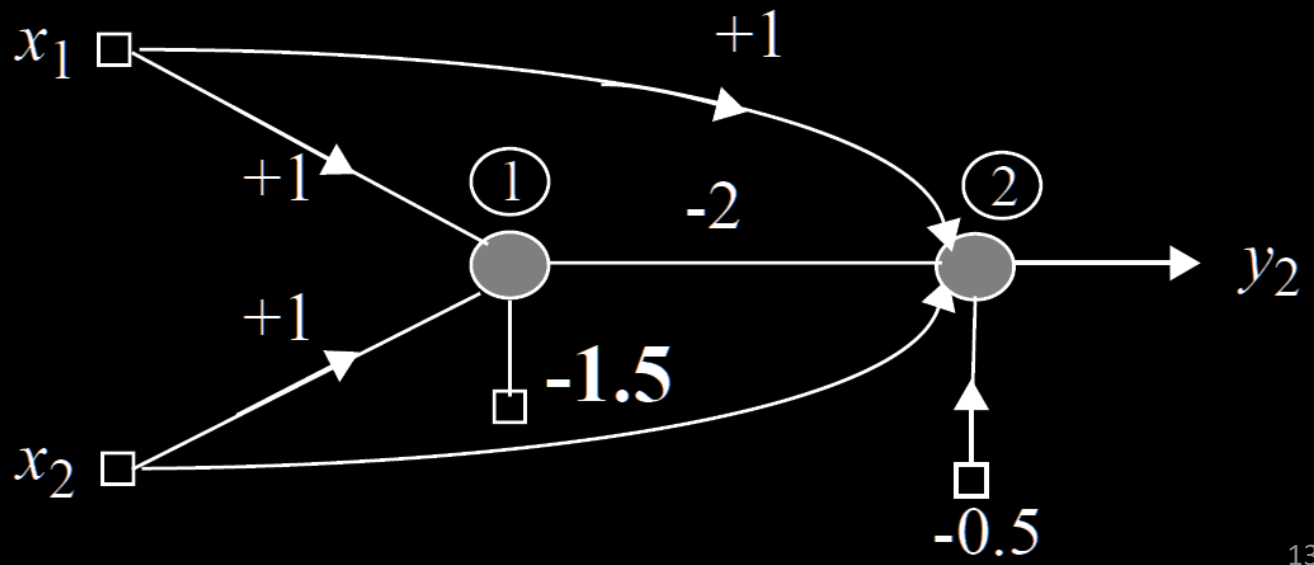


FIGURE P4.1

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

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

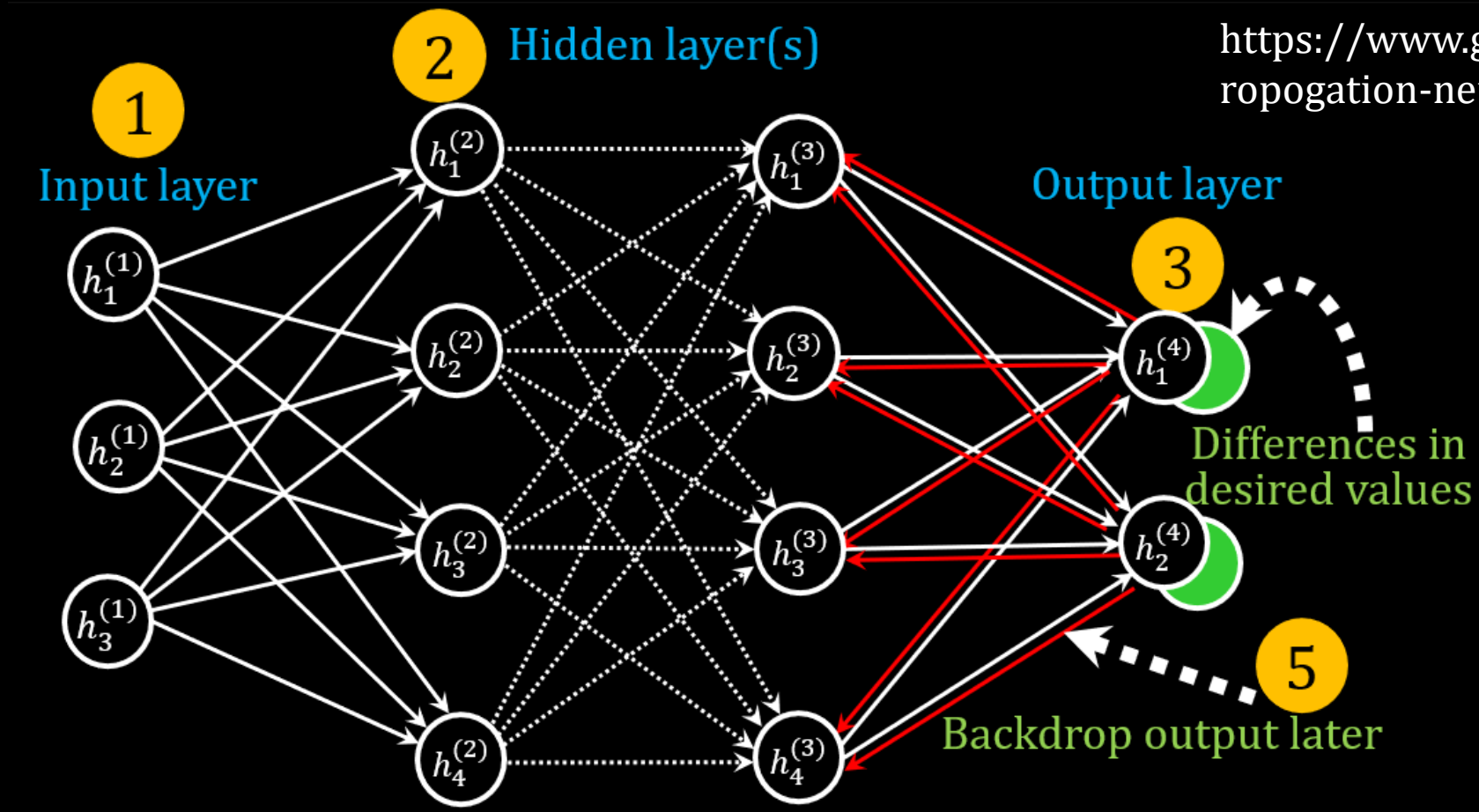
Hence, taking partial derivative of (2) w.r.t. $w_{ji}(t)$ and substituting into (1) we have

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

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.

4.6 Heuristics for Making the Back-Propagation Perform Better

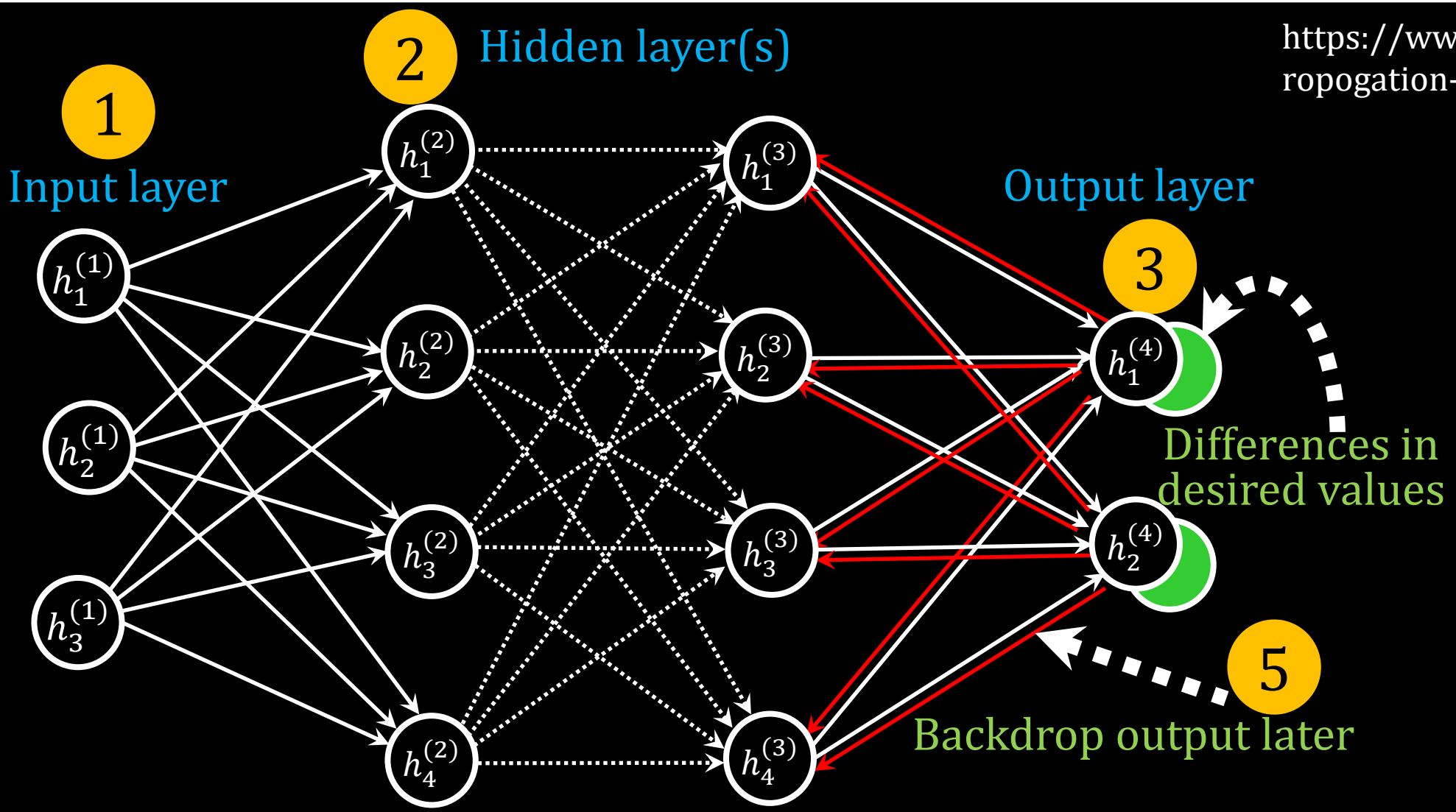
- There are methods that will significantly improve the back-propagation algorithm's performance, as described here.



https://www.guru99.com/backpropogation-neural-network.html

4.6 Heuristics for Making the Back-Propagation Perform Better

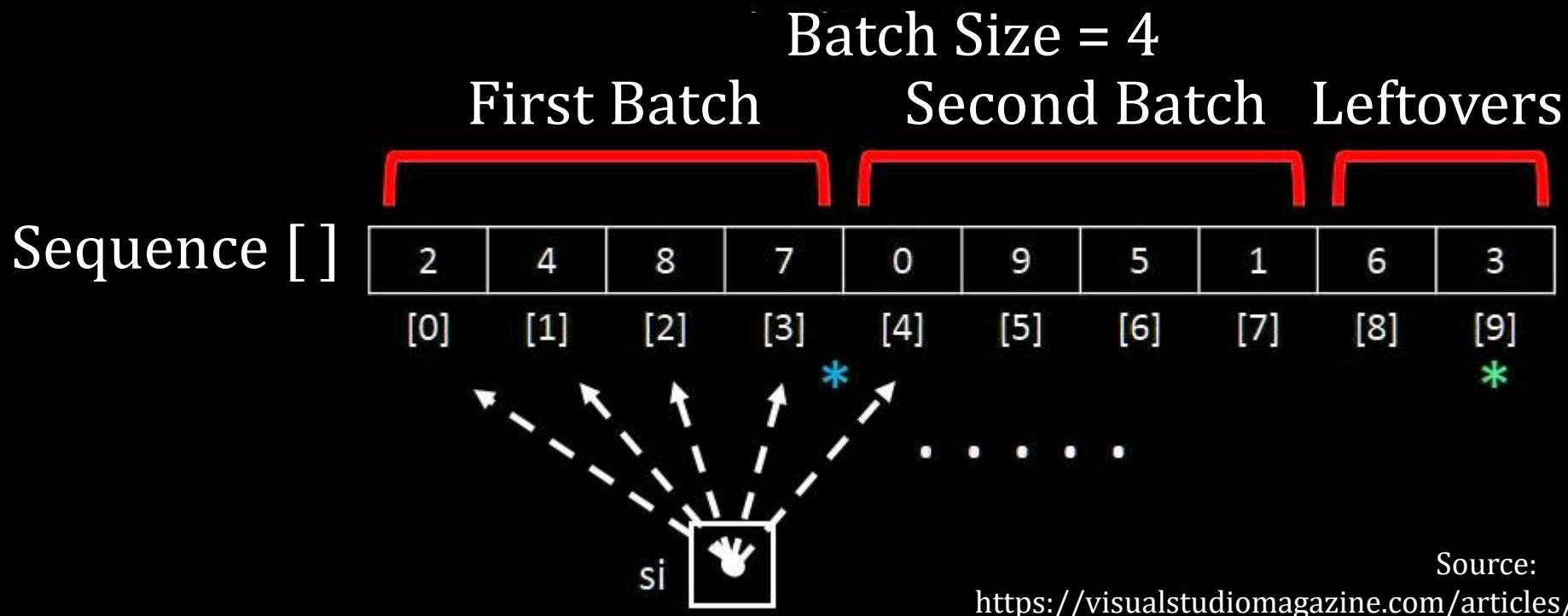
https://www.guru99.com/backpropogation-neural-network.html



4.6 Heuristics for Making the Back-Propagation Perform Better

1. Sequential versus batch update.

- The sequential mode of back-propagation learning (involving pattern-by-pattern updating) is computationally faster than the batch mode.
- This is clearer when the training data set is large and highly redundant, and the estimation of the Jacobian is required for the batch update.



Source:
https://visualstudiomagazine.com/articles/2015/07/01/~/media/ECG/visualstudiomagazine/Images/2015/07/0715vsm_McCaffreyFig2.ashx

4.6 Heuristics for Making the Back-Propagation Perform Better
2. Maximizing information content

- Use training examples with largest information content for the task at hand. Two ways of achieving this aim are:
  i. Use examples that results in the largest training error.
  ii. Use examples that is radically different from all those previously used.
- For a more refined technique, use an *emphasizing scheme*, which involves more difficult patterns than easy ones being presented to the network.
- There are two problems with the use of an emphasizing scheme :
  i. The distribution of examples within an epoch presented to the network is distorted.
  ii. The presence of an outlier or a mislabeled example can have a catastrophic consequence on the performance of the algorithm.

4.6 Heuristics for Making the Back-Propagation Perform Better

3. Activation function.

- A multilayer perceptron trained with the back-propagation algorithm may, in general, learn faster when the sigmoid activation function is antisymmetric than when it is nonsymmetric.
- An activation function $\varphi(.)$ is antisymmetric (i.e., odd function of its argument) if $\varphi(-v) = -\varphi(v)$.

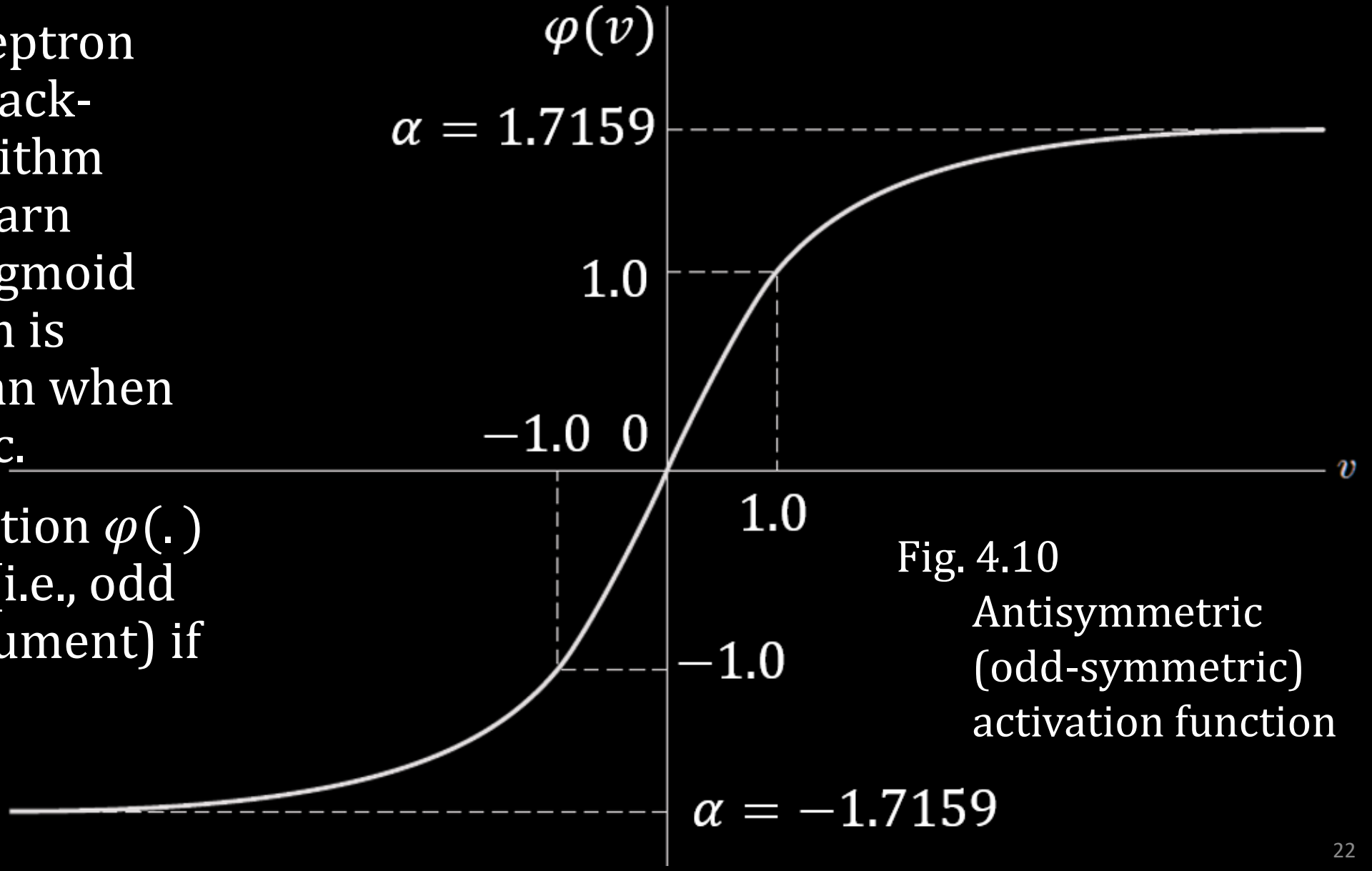


Fig. 4.10 Antisymmetric (odd-symmetric) activation function

4.6 Heuristics for Making the Back-Propagation Perform Better

4. Target values

- The target values (desired response) d_j for neuron j in the output layer of the multilayer perceptron should be offset by some amount ϵ away from the limiting value of the sigmoid activation function
  - Otherwise, the backpropagation algorithm learning process will slow down by driving the hidden neurons into saturation.
- Using the antisymmetric activation function of Fig. 4.10a for the limiting value $+\alpha$, we set $d_j = \alpha - \epsilon$, and for $-\alpha$, we set $d_j = -\alpha + \epsilon$.
- where ϵ is an appropriate positive constant.
- For the choice of $\alpha = 1.7159$, we may set $\epsilon = 0.7159$, in which case the target value (desired response) d_j can be conveniently chosen as +1, as indicated in Fig. 4.10.

4.6 Heuristics for Making the Back-Propagation Perform Better

4. Target values

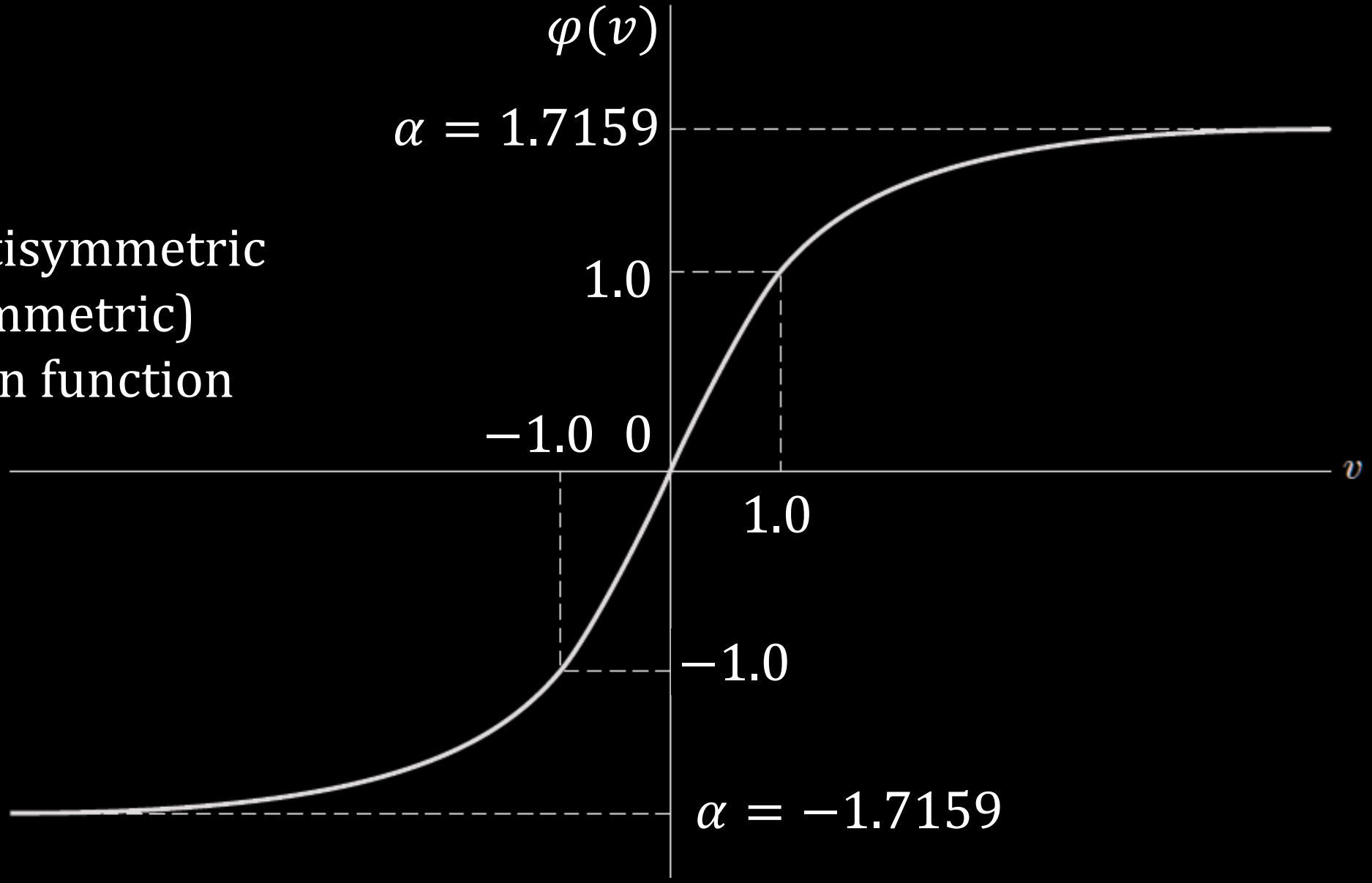


Fig. 4.10 Antisymmetric (odd-symmetric) activation function

4.6 Heuristics for Making the Back-Propagation Perform Better
5. Normalizing the inputs

- Each input variable should be preprocessed so that its mean value is close to zero, or else it is small compared to its standard deviation.
- In order to accelerate the back-propagation learning process, the normalization of the inputs should also include two other measures:
  1. The input variables contained in the training set should be uncorrelated; this can be done by using principal components analysis.
  2. The decorrelated input variables should be scaled so that their covariances are approximately equal, thereby ensuring that the different synaptic weights in the network learn at approximately the same speed.
- Fig. 4.11 illustrates the results of three normalization steps: mean removal, decorrelation, and covariance equalization, applied in that order.

5. Normalizing the inputs

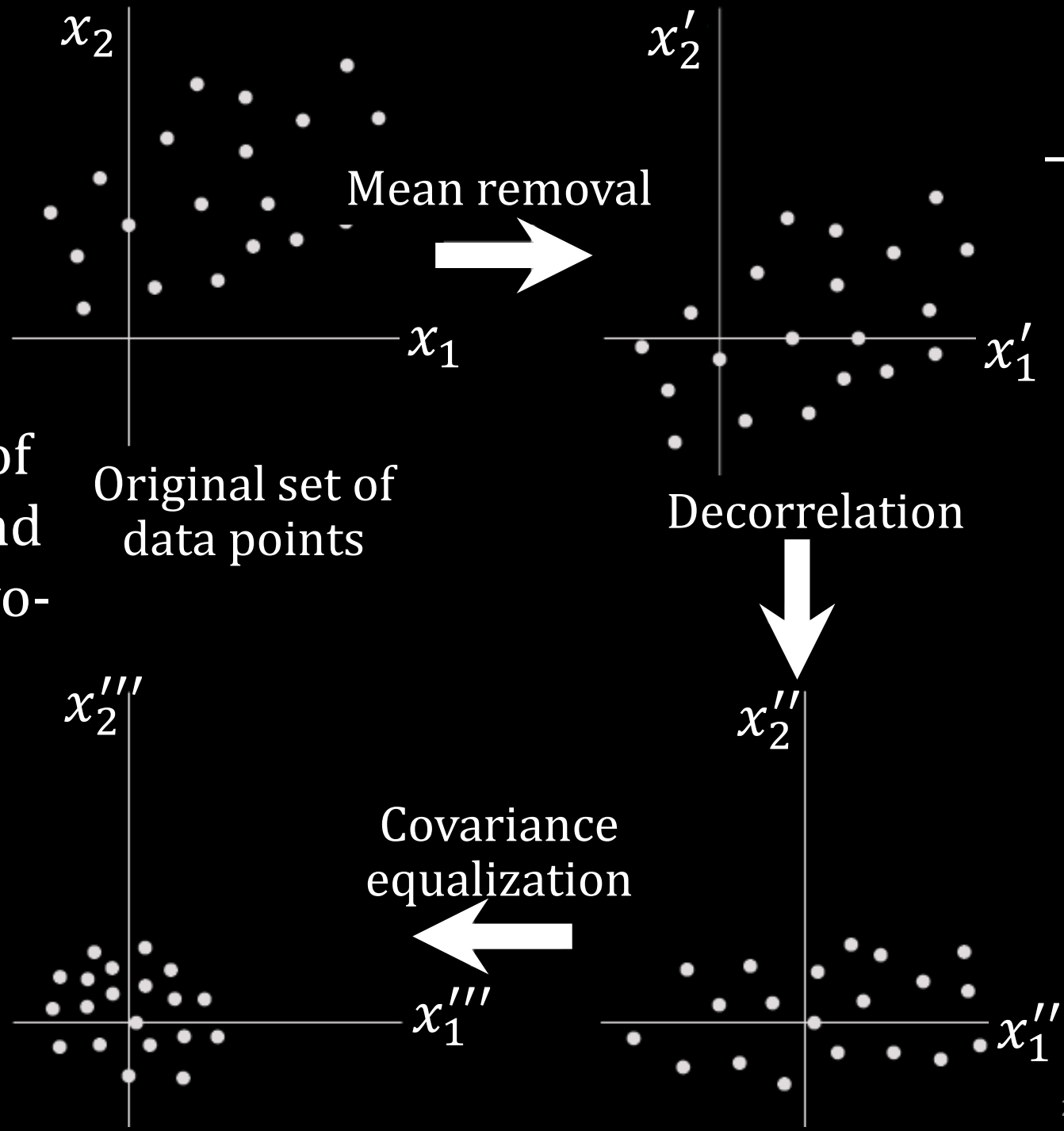


Fig. 4.11 Illustrating the operation of mean removal, decorrelation, and covariance equalization for a two-dimensional input space.

4.6 Heuristics for Making the Back-Propagation Perform Better
6. Initialization

- When the synaptic weights are assigned large initial values, it is highly likely that the neurons in the network will be driven into saturation.
- If this happens, the local gradients in the back-propagation algorithm assume small values, which in turn will cause the learning process to slow down.
- However, if the synaptic weights are assigned small initial values, the back-propagation algorithm may operate on a very flat area around the origin of the error surface;
  - this is particularly true in the case of antisymmetric activation functions such as the hyperbolic tangent function.
- For these reasons, the use of both large and small values for initializing the synaptic weights should be avoided.

4.6 Heuristics for Making the Back-Propagation Perform Better

6. Initialization

- Consider a multilayer perceptron using the hyperbolic tangent function for its activation functions.
- Let the bias applied to each neuron in the network be set to zero.
- We may then express the induced local field of neuron j as

$$v_j = \sum_{i=1}^{m} w_{ji} y_i$$

and ($\mu_y = E[y_i] = 0$, for all i) and $\sigma_y^2 = E[(y_i - \mu_i)^2] = E[(y_i)^2] = 1$ for all i and

$$E[y_i y_k] = \begin{cases} 1 & \text{for } k = i \\ 0 & \text{for } k \neq i \end{cases}$$

4.6 Heuristics for Making the Back-Propagation Perform Better
6. Initialization

- and that the synaptic weights are drawn from a uniformly distributed set of numbers with zero mean μ_w and variance σ_w as

$$\mu_w = E[w_{ji}] = 0 \text{ and } \sigma_w^2 = E\left[(w_{ji} - \mu_w)^2\right] = E\left[(w_{ji})^2\right] \quad \text{for all } (j, i) \text{ pairs.}$$

- Accordingly, we may express mean and variance of induced local field v_j as

$$\mu_v = E[v_j] = E\left[\sum_{i=1}^{m} w_{ji} y_i\right] = 0$$

and

$$\sigma_v^2 = E\left[(v_j - \mu_v)^2\right] = E\left[(v_j)^2\right] = m\sigma_w^2 \qquad (4.48)$$

where m is the number of synaptic connections of a neuron.

4.6 Heuristics for Making the Back-Propagation Perform Better
7. Learning from hints

- The process of learning from examples may include learning from hints, which is achieved by allowing prior information that we may have about the function $f(.)$ to be included in the learning process
- Such information may include
  - invariance properties, symmetries, or any other knowledge about the function $f(.)$
  - that may be used to accelerate the search for its approximate realization,
  - and more importantly, to improve the quality of the final estimate.

4.6 Heuristics for Making the Back-Propagation Perform Better

8. Learning rates

- All neurons in the multilayer perceptron should ideally learn at the same rate.
- The last layers usually have larger local gradients than the layers at the front end of the network. Hence, the learning-rate parameter η should be assigned a smaller value in the last layers than in the front layers.
- It is suggested that for a given neuron, the learning rate should be inversely proportional to the square root of synaptic connections made to that neuron.

4.7 Output Representation and Decision Rule

- For an M-class classification problem in which the union of the M distinct classes forms the entire input space, we need a total of M outputs to represent all possible classification decisions, as depicted in Fig. 4.12.
- In this figure vector $\mathbf{x}_j$ denotes the jth prototype (i.e., unique sample) of an m-dimensional random vector $\mathbf{x}$ to be classified by the multilayer perceptron.
- The kth of M possible classes to which $\mathbf{x}$ can belong is denoted by $\mathcal{C}_k$ and y_{kj} is the kth output of the network produced in response to the prototype $\mathbf{x}_j$ as

$$y_{k,j} = F_k(\mathbf{x}_j), \qquad k = 1, 2, \ldots, M \tag{4.50}$$

- where the function $F_k(.)$ defines the mapping learned by the network.

Fig. 4.1 Block diagram of a pattern classifier.

4.7 Output Representation and Decision Rule

Fig. 4.1 Block diagram of a pattern classifier.

4.7 Output Representation and Decision Rule

- For convenience of presentation, let

$$\mathbf{y}_j = \left[y_{1,j}, y_{2,j}, \ldots, y_{M,j}\right]^T = \left[F_1(\mathbf{x}_j), F_2(\mathbf{x}_j), \ldots, F_M(\mathbf{x}_j)\right]^T = \mathbf{F}(\mathbf{x}_j) \qquad (4.51)$$

- A basic question we wish to address in this section is:

> *After a multilayer perceptron is trained, what should the optimum decision rule be for classifying the M outputs of the network?*

- Any reasonable output decision rule should be based on knowledge of the vector valued function

$$\mathbf{F}: \mathbb{R}^m \ni \mathbf{x} \rightarrow \mathbf{y} \in \mathbb{R}^M \qquad (4.52)$$

4.7 Output Representation and Decision Rule

- $\mathbf{F}(.)$ should be continuous function that minimizes empirical risk functional:

$$R = \frac{1}{2N}\sum_{j=1}^{N}\left\|\mathbf{d}_j - \mathbf{F}(\mathbf{x}_j)\right\|^2 \qquad (4.53)$$

- where $\mathbf{d}_j$ is desired (target) output for the prototype $\mathbf{x}_j$, $\|.\|$ is the Euclidean norm, N is total number of examples presented to the network in training.
- Suppose that the network is trained with binary target values written as:

$$d_{kj} = \begin{cases} 1 & \text{when the prototype } \mathbf{x}_j \text{ belongs to class } \mathcal{C}_k \\ 0 & \text{when the prototype } \mathbf{x}_j \text{ does not belongs to class } \mathcal{C}_k \end{cases} \qquad (4.54)$$

4.7 Output Representation and Decision Rule

- Based on this notation class C_k is represented by M-dimensional target vector

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow k\text{th element}$$

- A multilayer perceptron classifier trained with the backpropagation algorithm on a finite set of examples may lead to an asymptotic approximation of the underlying class probabilities.
- This property may be justified on the following grounds:

4.7 Output Representation and Decision Rule

- According to the *law of large numbers,* as the size of the training set, $N \to \infty$, the weight vector $\mathbf{w}$ that minimizes cost functional R of (4.53) approaches the optimum weight vector $\mathbf{w}^*$ that minimizes the expectation of the random quantity $\frac{1}{2}\|\mathbf{d}_j - \mathbf{F}(\mathbf{x}_j)\|^2$,
  - where $\mathbf{d}$ is desired response vector and $\mathbf{F}(\mathbf{w}, \mathbf{x})$ is approximation realized by a multilayer perceptron with weight vector $\mathbf{w}$ and vector $\mathbf{x}$ as input.

4.7 Output Representation and Decision Rule

- The optimum weight vector $\mathbf{w}^*$ has the property that the corresponding vector of actual network outputs, $\mathbf{F}(\mathbf{w}^*, \mathbf{x})$, is a mean-squared, error-minimizing approximation to the conditional expectation of the desired response vector, given the input vector $\mathbf{x}$.
- For a 1 of *M pattern classification* problem, the kth element of the desired response vector equals one if the input vector $\mathbf{x}$ belongs to class $\mathcal{C}_k$ and zero otherwise.

  Hence, the conditional expectation of the desired response vector, given $\mathbf{x}$, equals the a posteriori class probability $P(\mathcal{C}_k|\mathbf{x})$, $k = 1, 2, \ldots, M$.

4.7 Output Representation and Decision Rule

- We may say that an appropriate output decision rule is the (approximate) Bayes rule generated by the a posteriori probability estimates:

Classify the random vector $\mathbf{x}$ *as belonging to class* $\mathcal{C}_k$ *if*

$$F_k(\mathbf{x}) > F_j(\mathbf{x}) \quad \text{for all } j \neq k \tag{4.55}$$

where $F_k(\mathbf{x})$ and $F_j(\mathbf{x})$ are elements of the vector-valued mapping function

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \\ \vdots \\ F_M(\mathbf{x}) \end{bmatrix}$$

4.8 Computer Experiment

- This experiment illustrates the learning behavior of a multilayer perceptron used as a pattern classifier.
- The objective of the experiment is to distinguish between two classes of "overlapping," 2-dimensional, Gaussian-distributed patterns labeled 1 and 2.
- Let $\mathcal{C}_1$ and $\mathcal{C}_2$ denote the set of events for which a random vector $\mathbf{x}$ belongs to patterns 1 and 2, respectively.

4.8 Computer Experiment

- We may then express the conditional probability density functions for the two classes:

$$\text{Class } \mathcal{C}_1: \qquad f_x(\mathbf{x}|\mathcal{C}_1) = \frac{1}{2\pi\sigma_1^2}\exp\left(-\frac{1}{2\sigma_1^2}\|\mathbf{x}-\boldsymbol{\mu_1}\|^2\right) \tag{4.56}$$

where $\boldsymbol{\mu}_1 = \text{mean vector} = [0,0]^T$, $\sigma_1^2 = \text{variance} = 1$.

$$\text{Class } \mathcal{C}_2: \qquad f_x(\mathbf{x}|\mathcal{C}_2) = \frac{1}{2\pi\sigma_2^2}\exp\left(-\frac{1}{2\sigma_2^2}\|\mathbf{x}-\boldsymbol{\mu_2}\|^2\right) \tag{4.57}$$

where $\boldsymbol{\mu}_2 = \text{mean vector} = [2,0]^T$, $\sigma_2^2 = \text{variance} = 1.5$.

- The two classes are assumed to be equiprobable; that is, $p_1 = p_2 = 1/2$.
- Fig. 4.13a shows three-dimensional plots of the two Gaussian distributions defined by (4.56) and (4.57).
- The input vector is $\mathbf{x} = [x_1, x_2]^T$ and dimensionality of input space is $m_0 = 2$.

4.8 Computer Experiment

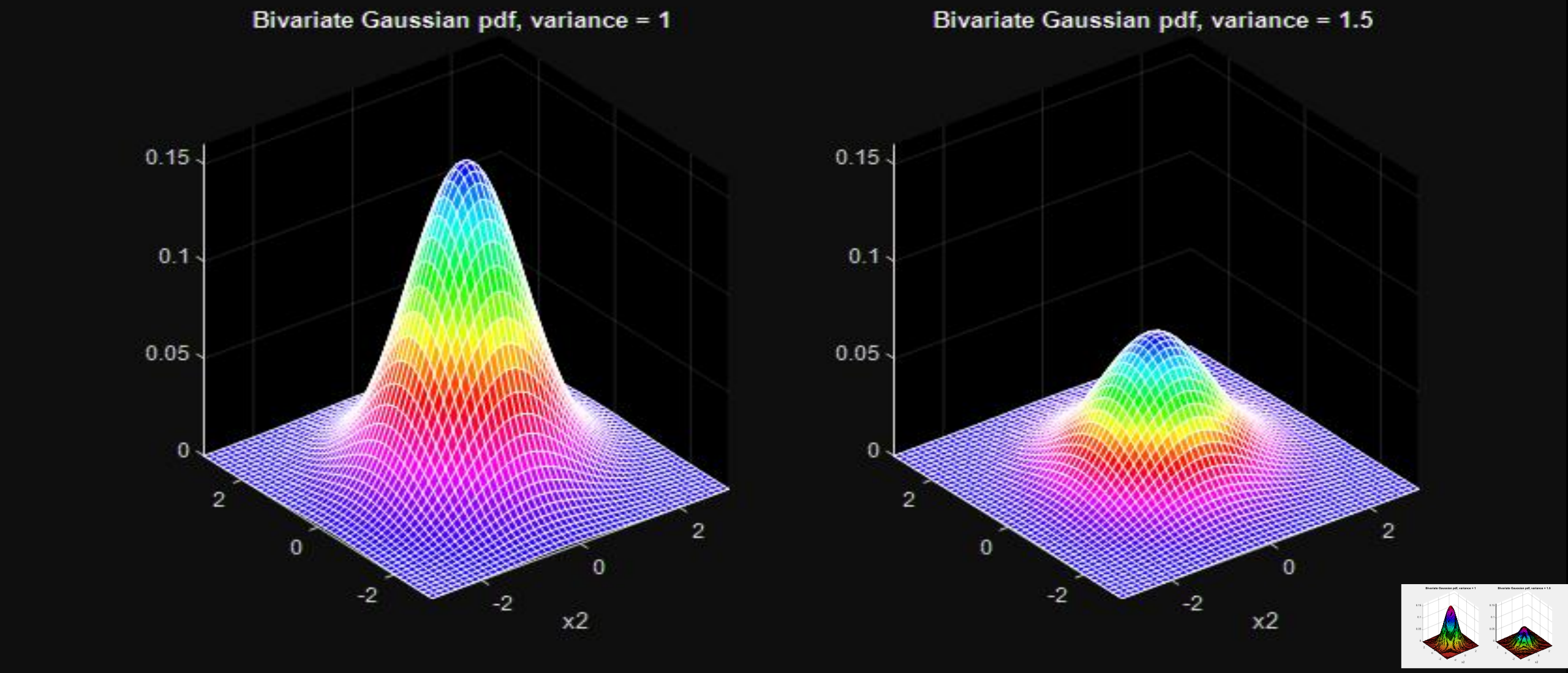


Fig. 4.13 Bivariate Gaussian pdf with 0 mean and (a) $\sigma_1 = 1$ (b) $\sigma_2 = 1.5$

4.8 Computer Experiment

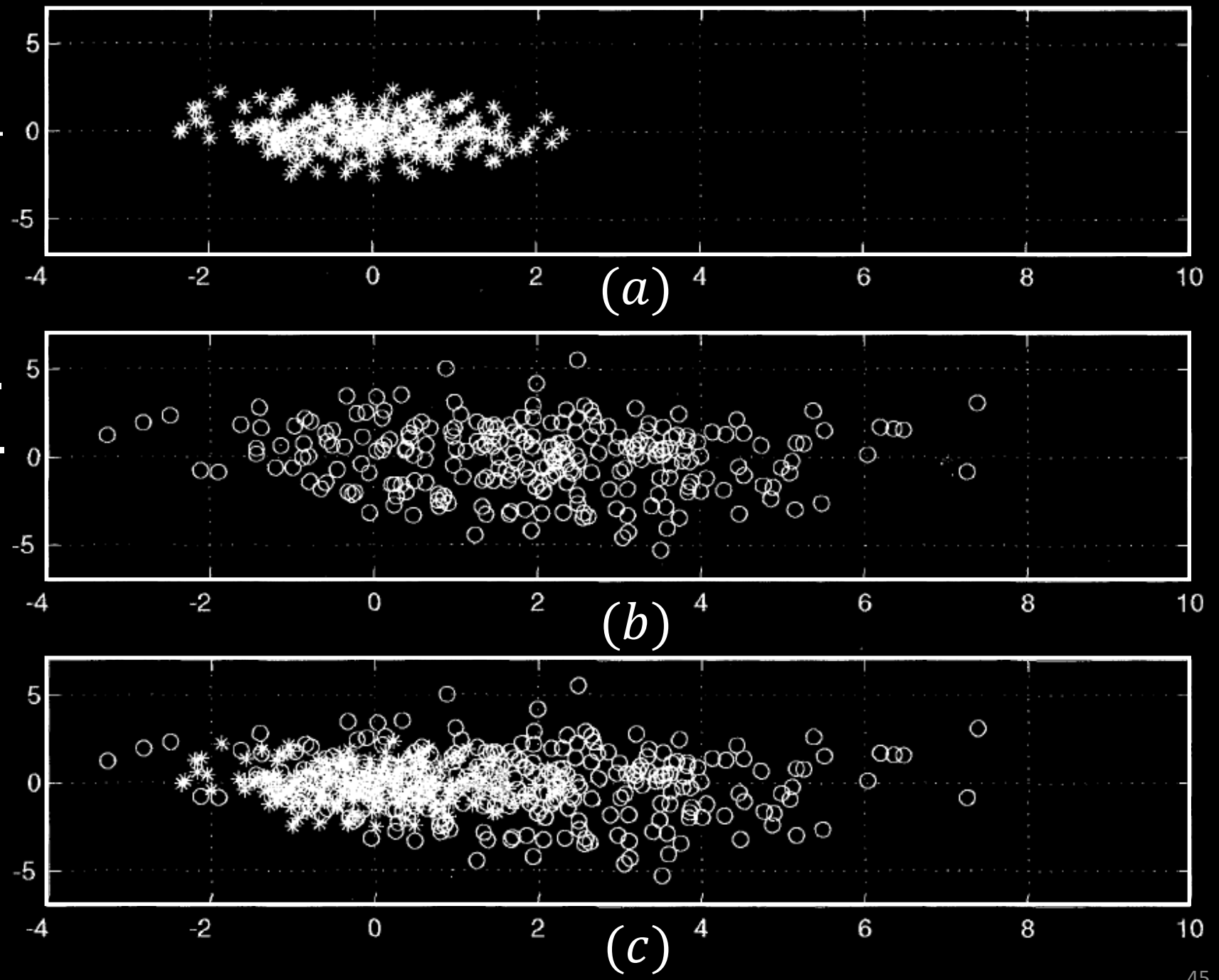
```
% PLOT OF THE BIVARIATE GAUSSIAN PDF
%correlation
% https://onlinelibrary.wiley.com/doi/pdf
% /10.1002/9781119011590.app1
% pp. 988
clear;
%rho=0.9;
rho = .1;
%variances and covariances
variance = 1;
s1=variance^2; s2=variance^2;
s12=rho*(s1*s2)^0.5;
factor=2*pi*(s1*s2)^0.5;
m=zeros(2,1); %mean vector
npoints1=50; npoints2=50;
x1=linspace(m(1)-3*s1^0.5,m(1)+3*s1^0.5,npoints1);
x2=linspace(m(2)-3*s2^0.5,m(2)+3*s2^0.5,npoints2);
[x1mesh, x2mesh]=...
meshgrid((x1-m(1))/s1^0.5,(x2-m(2))/s2^0.5);
x1x2mesh=-2*rho*x1mesh.*x2mesh;
pdfbiv=exp(-(x1mesh.^2+x1x2mesh+x2mesh.^2)...
/(2*(1-rho^2)))/(factor*(1-rho^2)^0.5);
subplot(1,2,1);
surf(x1mesh,x2mesh,pdfbiv)
%axis square
title(['Bivariate Gaussian pdf, variance = …
        ' num2str(variance)])
xlabel('x1'),xlabel('x2')
axis([-3 3 -3 3 0 .16])
```

4.8 Computer Experiment

- Fig. 4.14 shows individual scatter diagrams for classes $\mathcal{C}_1$ and $\mathcal{C}_2$ and the joint scatter diagram representing the superposition of scatter plots of 500 points taken from each of the two processes.
- This latter diagram clearly shows that the two distributions overlap each other significantly, indicating that there is inevitably a significant probability of misclassification.

4.8 Computer Experiment

Fig. 4.14
(a) Scatter plot of class $\mathcal{C}_1$.
(b) Scatter plot of class $\mathcal{C}_2$.
(c) Combined scatter plot of both classes.



End of Section 4.5 - 4.8