InfoLab

IC612: Data Warehousing and Data Mining

Lecture 9:
Data Classification: Advanced Concepts (1)

Min-Soo Kim

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

Bayes Classifier

- A probabilistic framework for solving classification problems
- Conditional Probability:

$$P(C \mid A) = \frac{P(A,C)}{P(A)}$$

$$P(A \mid C) = \frac{P(A,C)}{P(C)}$$

- Bayes theorem:

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$

Example of Bayes Theorem

- **Given:**
  - A doctor knows that meningitis causes stiff neck 50% of the time
  - Prior probability of any patient having meningitis is 1/50,000
  - Prior probability of any patient having stiff neck is 1/20

- **If a patient has stiff neck, what's the probability he/she has meningitis?**

$$P(M \mid S) = \frac{P(S \mid M)P(M)}{P(S)} = \frac{0.5 \times 1/50000}{1/20} = 0.0002$$

Bayesian Classifiers

- Consider each attribute and class label as random variables

- Given a record with attributes (A_1, A_2,…,A_n)
  - Goal is to predict class C
  - Specifically, we want to find the value of C that maximizes P(C| A_1, A_2,…,A_n)

- Can we estimate P(C| A_1, A_2,…,A_n) directly from data?

Bayesian Classifiers

- **Approach:**
  - compute the posterior probability P(C | A_1, A_2, …, A_n) for all values of C using the Bayes theorem

$$P(C \mid A_1 A_2 \ldots A_n) = \frac{P(A_1 A_2 \ldots A_n \mid C) P(C)}{P(A_1 A_2 \ldots A_n)}$$

  - Choose value of C that maximizes
    P(C | A_1, A_2, …, A_n)

  - Equivalent to choosing value of C that maximizes
    P(A_1, A_2, …, A_n|C) P(C)

- **How to estimate P(A_1, A_2, …, A_n | C)?**

Naive Bayes Classifier

- **Assume independence among attributes A_i when class is given:**
  - $P(A_1, A_2, \ldots, A_n \mid C) = P(A_1 \mid C_j)\, P(A_2 \mid C_j) \ldots P(A_n \mid C_j)$
  - Can estimate $P(A_i \mid C_j)$ for all A_i and C_j.
  - New point is classified to C_j if **$P(C_j) \prod P(A_i \mid C_j)$** is maximal.

How to Estimate Probabilities from Data?

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- **Class: $P(C) = N_c/N$**
  - e.g., P(No) = 7/10,
    P(Yes) = 3/10

- **For discrete attributes:**

$$P(A_i \mid C_k) = |A_{ik}| / N_c$$

  - where $|A_{ik}|$ is number of instances having attribute A_i and belongs to class C_k
  - Examples:

P(Status=Married|No) = 4/7
P(Refund=Yes|Yes)=0

How to Estimate Probabilities from Data?

- **For continuous attributes:**
  - Discretize the range into bins
    - one ordinal attribute per bin
    - violates independence assumption
  - Two-way split: (A < v) or (A > v)
    - choose only one of the two splits as new attribute
  - Probability density estimation:
    - Assume attribute follows a normal distribution
    - Use data to estimate parameters of distribution (e.g., mean and standard deviation)
    - Once probability distribution is known, can use it to estimate the conditional probability $P(A_i|c)$

How to Estimate Probabilities from Data?

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- **Normal distribution:**

$$P(A_i \mid c_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(A_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

  - One for each (A_i,c_i) pair

- **For (Income, Class=No):**
  - If Class=No
    - sample mean = 110
    - sample variance = 2975

$$P(Income = 120 \mid No) = \frac{1}{\sqrt{2\pi}(54.54)} e^{-\frac{(120-110)^2}{2(2975)}} = 0.0072$$

Example of Naive Bayes Classifier

Given a Test Record:

$$X = (\text{Refund} = \text{No}, \text{Married}, \text{Income} = 120\text{K})$$

naive Bayes Classifier:

P(Refund=Yes|No) = 3/7
P(Refund=No|No) = 4/7
P(Refund=Yes|Yes) = 0
P(Refund=No|Yes) = 1
P(Marital Status=Single|No) = 2/7
P(Marital Status=Divorced|No)=1/7
P(Marital Status=Married|No) = 4/7
P(Marital Status=Single|Yes) = 2/7
P(Marital Status=Divorced|Yes)=1/7
P(Marital Status=Married|Yes) = 0

For taxable income:
If class=No: sample mean=110
sample variance=2975
If class=Yes: sample mean=90
sample variance=25

- P(X|Class=No) = P(Refund=No|Class=No)
  × P(Married| Class=No)
  × P(Income=120K| Class=No)
  = 4/7 × 4/7 × 0.0072 = 0.0024

- P(X|Class=Yes) = P(Refund=No| Class=Yes)
  × P(Married| Class=Yes)
  × P(Income=120K| Class=Yes)
  = 1 × 0 × 1.2 × 10^{-9} = 0

Since P(X|No)P(No) > P(X|Yes)P(Yes)

Therefore P(No|X) > P(Yes|X)

=> Class = No

Naive Bayes Classifier

- If one of the conditional probability is zero, then the entire expression becomes zero
- Probability estimation:

$$\text{Original}: P(A_i \mid C) = \frac{N_{ic}}{N_c}$$

$$\text{Laplace}: P(A_i \mid C) = \frac{N_{ic} + 1}{N_c + c}$$

$$\text{m-estimate}: P(A_i \mid C) = \frac{N_{ic} + mp}{N_c + m}$$

c: number of classes

p: prior probability

m: parameter

Example of Naive Bayes Classifier

Name	Give Birth	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	yes	mammals
python	no	no	no	no	non-mammals
salmon	no	no	yes	no	non-mammals
whale	yes	no	yes	no	mammals
frog	no	no	sometimes	yes	non-mammals
komodo	no	no	no	yes	non-mammals
bat	yes	yes	no	yes	mammals
pigeon	no	yes	no	yes	non-mammals
cat	yes	no	no	yes	mammals
leopard shark	yes	no	yes	no	non-mammals
turtle	no	no	sometimes	yes	non-mammals
penguin	no	no	sometimes	yes	non-mammals
porcupine	yes	no	no	yes	mammals
eel	no	no	yes	no	non-mammals
salamander	no	no	sometimes	yes	non-mammals
gila monster	no	no	no	yes	non-mammals
platypus	no	no	no	yes	mammals
owl	no	yes	no	yes	non-mammals
dolphin	yes	no	yes	no	mammals
eagle	no	yes	no	yes	non-mammals

A: attributes

M: mammals

N: non-mammals

$$P(A|M) = \frac{6}{7} \times \frac{6}{7} \times \frac{2}{7} \times \frac{2}{7} = 0.06$$

$$P(A|N) = \frac{1}{13} \times \frac{10}{13} \times \frac{3}{13} \times \frac{4}{13} = 0.0042$$

$$P(A|M)P(M) = 0.06 \times \frac{7}{20} = 0.021$$

$$P(A|N)P(N) = 0.004 \times \frac{13}{20} = 0.0027$$

Give Birth	Can Fly	Live in Water	Have Legs	Class
yes	no	yes	no	?

P(A|M)P(M) > P(A|N)P(N)

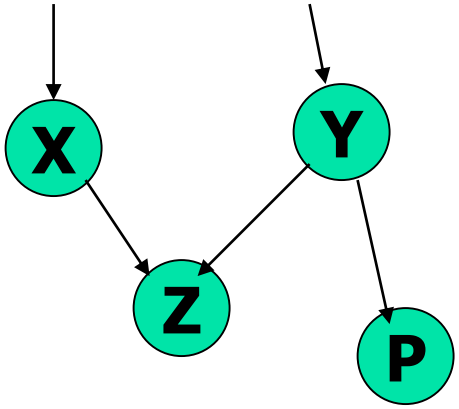
=> Mammals

Naive Bayes (Summary)

- Robust to isolated noise points
- Handle missing values by ignoring the instance during probability estimate calculations
- Robust to irrelevant attributes
- Independence assumption may not hold for some attributes
  - Use other techniques such as Bayesian Belief Networks (BBN)

Bayesian Belief Networks

- **Bayesian belief network** (also known as **Bayesian network, probabilistic network**): allows *class conditional independencies* between *subsets* of variables
- Two components: (1) A *directed acyclic graph* (called a structure) and (2) a set of *conditional probability tables* (CPTs)
- A (*directed acyclic*) graphical model of *causal influence* relationships
  - Represents dependency among the variables
  - Gives a specification of joint probability distribution



- ❑ Nodes: random variables
- ❑ Links: dependency
- ❑ X and Y are the parents of Z, and Y is the parent of P
- ❑ No dependency between Z and P
- ❑ Has no loops/cycles

Example of Bayesian Belief Network

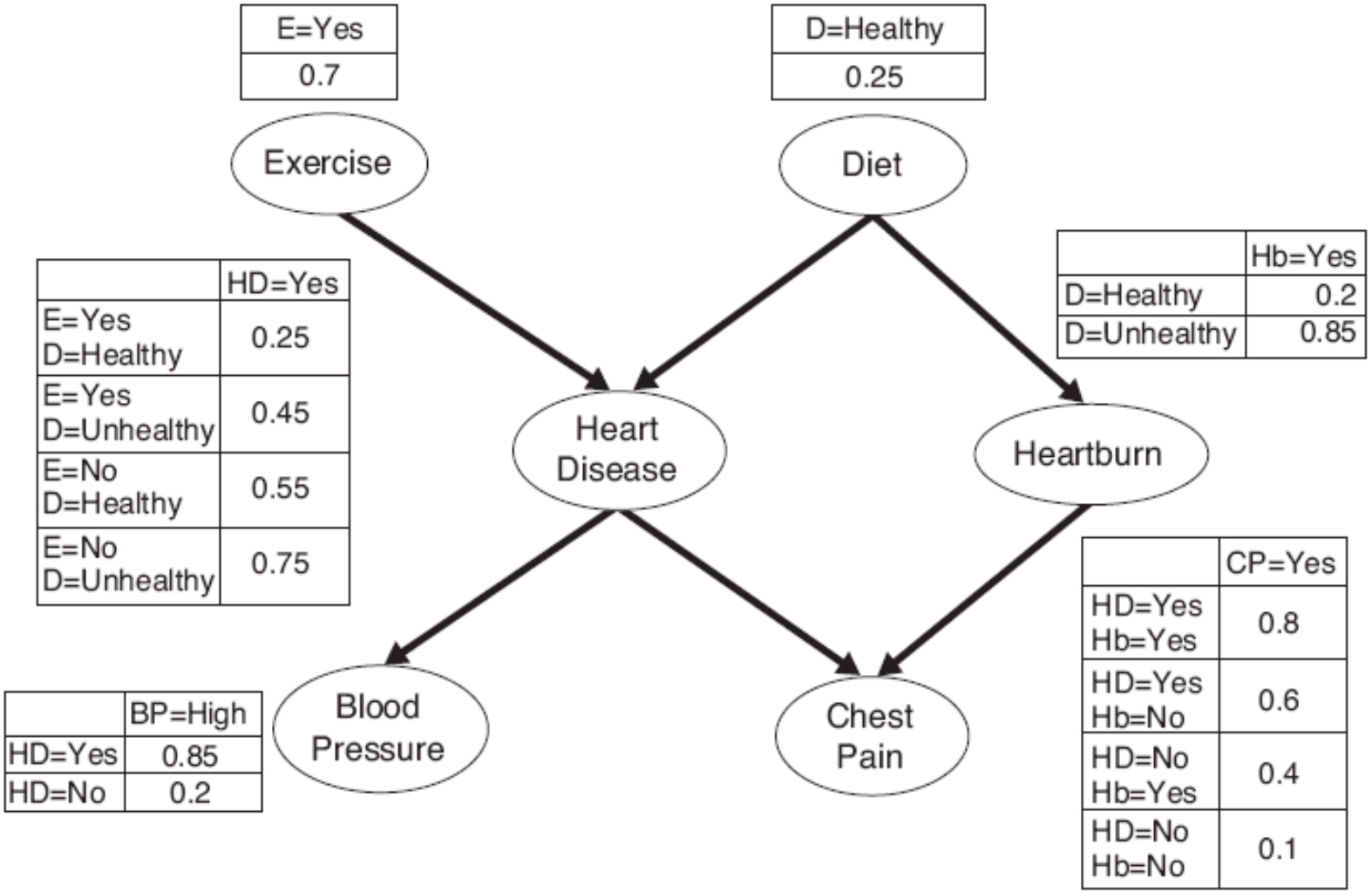


Figure 5.13. A Bayesian belief network for detecting heart disease and heartburn in patients.

Case 1: No Prior Information

$$
\begin{aligned}
P(\texttt{HD} = \texttt{Yes}) &= \sum_{\alpha}\sum_{\beta} P(\texttt{HD} = \texttt{Yes}|E = \alpha, D = \beta)P(E = \alpha, D = \beta) \\
&= \sum_{\alpha}\sum_{\beta} P(\texttt{HD} = \texttt{Yes}|E = \alpha, D = \beta)P(E = \alpha)P(D = \beta) \\
&= 0.25 \times 0.7 \times 0.25 + 0.45 \times 0.7 \times 0.75 + 0.55 \times 0.3 \times 0.25 \\
&\quad + 0.75 \times 0.3 \times 0.75 \\
&= 0.49.
\end{aligned}
$$

$$
P(\texttt{HD} = \texttt{no}) = 1 - P(\texttt{HD} = \texttt{yes}) = 0.51
$$

Example of Bayesian Belief Network

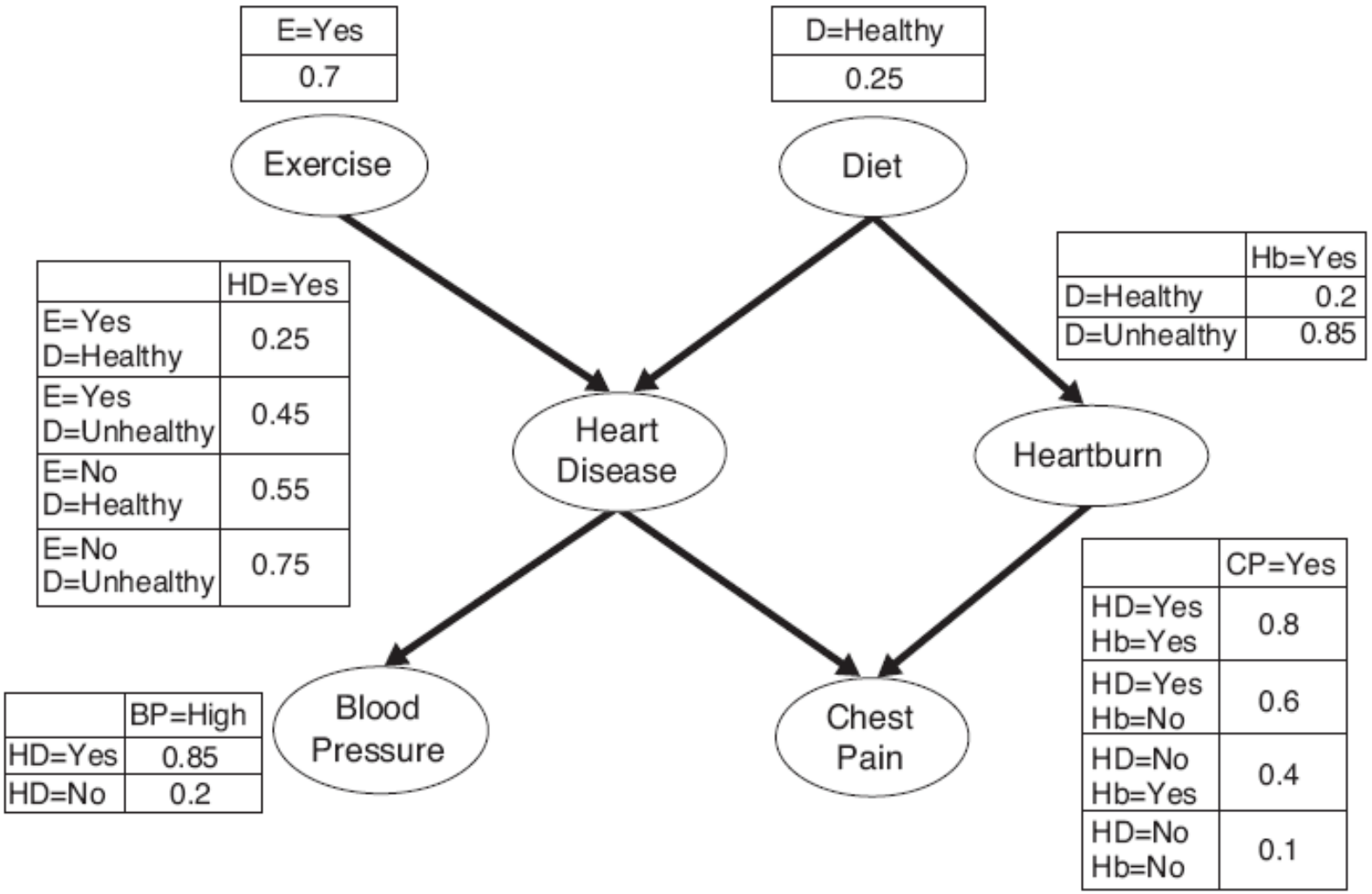


Figure 5.13. A Bayesian belief network for detecting heart disease and heartburn in patients.

Case 2: High Blood Pressure

$$P(\mathtt{BP}=\mathtt{High}) = \sum_{\gamma} P(\mathtt{BP}=\mathtt{High}|\mathtt{HD}=\gamma)P(\mathtt{HD}=\gamma)$$
$$= 0.85\times 0.49 + 0.2\times 0.51 = 0.5185.$$

$$P(\mathtt{HD}=\mathtt{Yes}|\mathtt{BP}=\mathtt{High}) = \frac{P(\mathtt{BP}=\mathtt{High}|\mathtt{HD}=\mathtt{Yes})P(\mathtt{HD}=\mathtt{Yes})}{P(\mathtt{BP}=\mathtt{High})}$$
$$= \frac{0.85\times 0.49}{0.5185} = 0.8033.$$

$$P(\mathtt{HD}=\mathtt{No}|\mathtt{BP}=\mathtt{High}) = 1 - 0.8033 = 0.1967$$

Example of Bayesian Belief Network

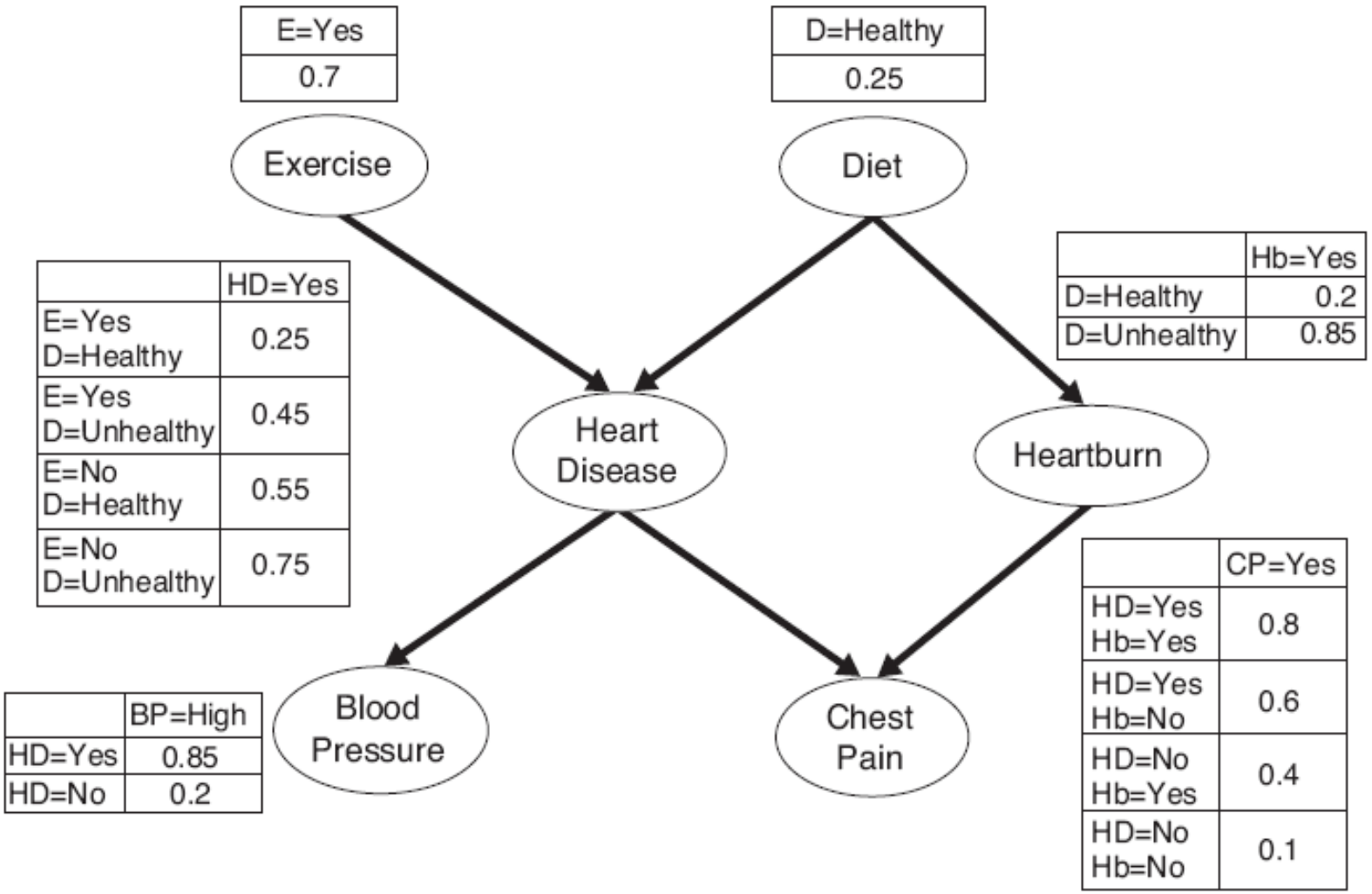


Figure 5.13. A Bayesian belief network for detecting heart disease and heartburn in patients.

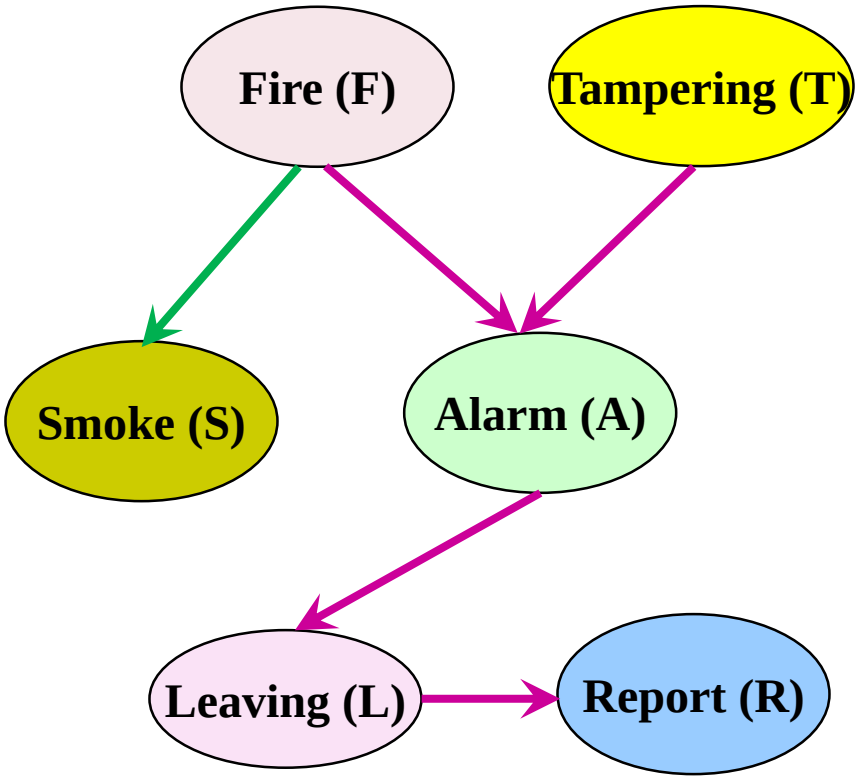
Case 3: High Blood Pressure, Healthy Diet, and Regular Exercise

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$

$$
\begin{aligned}
& P(\mathrm{HD} = \mathrm{Yes} | \mathrm{BP} = \mathrm{High}, D = \mathrm{Healthy}, E = \mathrm{Yes}) \\
= & \left[\frac{P(\mathrm{BP} = \mathrm{High} | \mathrm{HD} = \mathrm{Yes}, D = \mathrm{Healthy}, E = \mathrm{Yes})}{P(\mathrm{BP} = \mathrm{High} | D = \mathrm{Healthy}, E = \mathrm{Yes})} \right] \\
& \times P(\mathrm{HD} = \mathrm{Yes} | D = \mathrm{Healthy}, E = \mathrm{Yes}) \\
= & \frac{P(\mathrm{BP} = \mathrm{High} | \mathrm{HD} = \mathrm{Yes}) P(\mathrm{HD} = \mathrm{Yes} | D = \mathrm{Healthy}, E = \mathrm{Yes})}{\sum_\gamma P(\mathrm{BP} = \mathrm{High} | \mathrm{HD} = \gamma) P(\mathrm{HD} = \gamma | D = \mathrm{Healthy}, E = \mathrm{Yes})} \\
= & \frac{0.85 \times 0.25}{0.85 \times 0.25 + 0.2 \times 0.75} \\
= & 0.5862,
\end{aligned}
$$

$$P(\mathrm{HD} = \mathrm{No} | \mathrm{BP} = \mathrm{High}, D = \mathrm{Healthy}, E = \mathrm{Yes}) = 1 - 0.5862 = 0.4138$$

A Bayesian Network and Some of Its CPTs



CPT: Conditional Probability Tables

Fire	Smoke	$\Theta_{s\|f}$
True	True	.90
False	True	.01

Fire	Tampering	Alarm	$\Theta_{a\|f,t}$
True	True	True	.5
True	False	True	.99
False	True	True	.85
False	False	True	.0001

CPT shows the conditional probability for each possible combination of its parents

Derivation of the probability of a pa rticular combination of values of **X**, from CPT:

$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(x_i))$$

Model Building

Algorithm 5.3 Algorithm for generating the topology of a Bayesian network.

1: Let $T = (X_1, X_2, \ldots, X_d)$ denote a total order of the variables.
2: **for** $j = 1$ to d **do**
3: Let $X_{T(j)}$ denote the j^{th} highest order variable in T.
4: Let $\pi(X_{T(j)}) = \{X_{T(1)}, X_{T(2)}, \ldots, X_{T(j-1)}\}$ denote the set of variables preceding $X_{T(j)}$.
5: Remove the variables from $\pi(X_{T(j)})$ that do not affect X_j (using prior knowledge).
6: Create an arc between $X_{T(j)}$ and the remaining variables in $\pi(X_{T(j)})$.
7: **end for**

Training Bayesian Networks: Several Scenarios

- Scenario 1: Given both the network structure and all variables observable: *compute only the CPT entries*
- Scenario 2: Network structure known, some variables hidden: *gradient descent* (greedy hill-climbing) method, i.e., search for a solution along the steepest descent of a criterion function
  - Weights are initialized to random probability values
  - At each iteration, it moves towards what appears to be the best solution at the moment, w.o. backtracking
  - Weights are updated at each iteration & converge to local optimum

Training Bayesian Networks: Several Scenarios

- Scenario 3: Network structure unknown, all variables observable: search through the model space to *reconstruct network topology*

- Scenario 4: Unknown structure, all hidden variables: No good algorithms known for this purpose

- D. Heckerman. A Tutorial on Learning with Bayesian Networks. In *Learning in Graphical Models,* M. Jordan, ed. MIT Press, 1999.

InfoLab

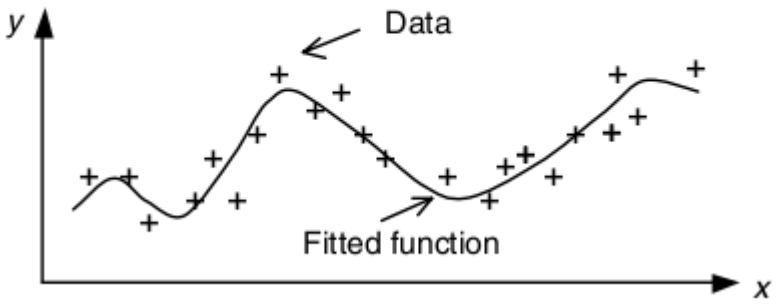
Fundamentals of Neural Networks and Models for Linear Data Analysis

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

Neural networks

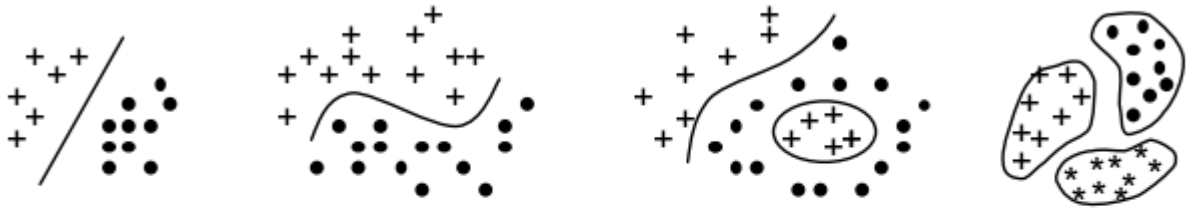
- **Evolving field with origins in neurobiology**
- **Model attempting to mimic some of the basic information processing methods found in the brain**
  - capturing essential linear and nonlinear trends in complex data
  - providing reliable predictions for new situations containing even noisy and partial information
- **Being useful in solving complex problems**
  - prediction (or function approximation)
  - pattern classification
  - clustering
  - forecasting
  - …

Function approximation

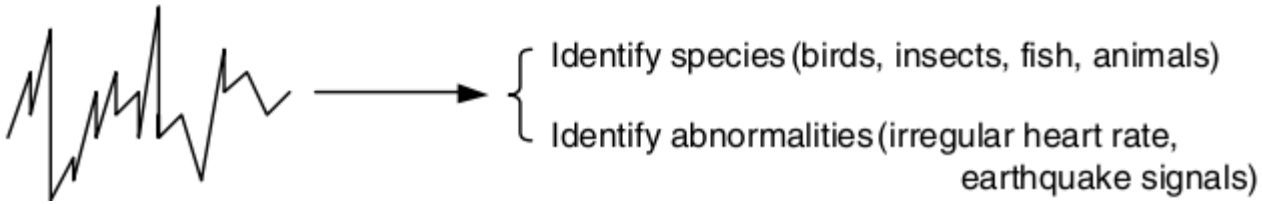


Classification

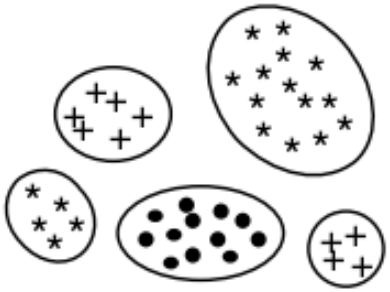
(1) Data classification: assign data to a class



(2) Signal classification: assign time-series data to a class



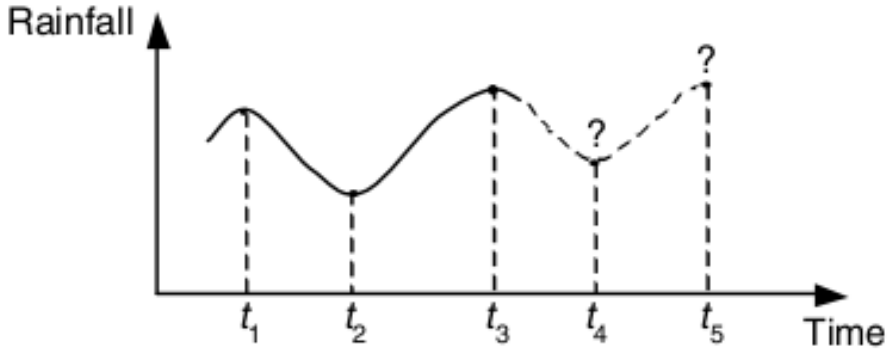
Unsupervised clustering: find unknown clusters in data



-- Species assemblages

-- Protein structure

Forecasting: predict next outcomes of a time series



Haykin states

- **“A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use”**

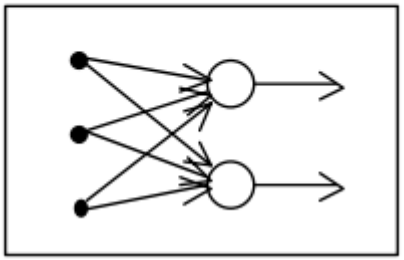
- **“It resembles the brain in two respects:”**
  - ① Knowledge is acquired by the network through a learning process

  - ① Interconnection strengths between neurons, known as synaptic weights or weights, are used to store knowledge

Various neural networks

Single-layer perceptron

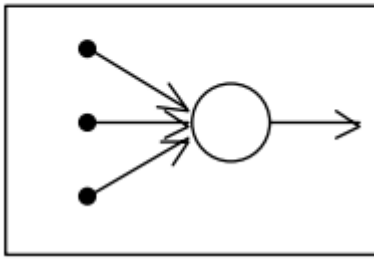
Linear classifier



(a)

Linear neuron

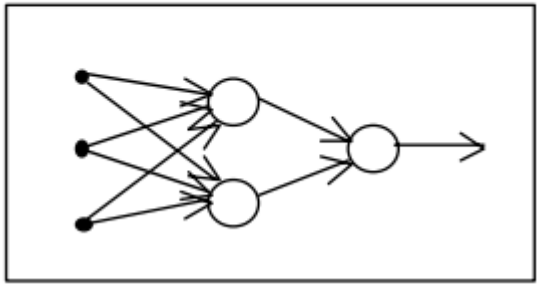
Linear predictor/classifier



(b)

Multilayer perceptron

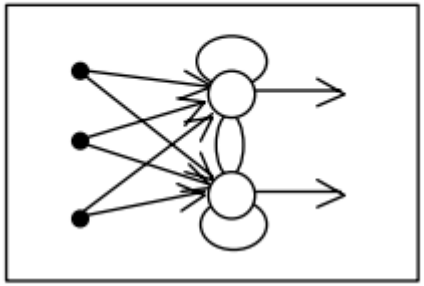
Nonlinear predictor/classifier



(c)

Competitive networks

Unsupervised classifier

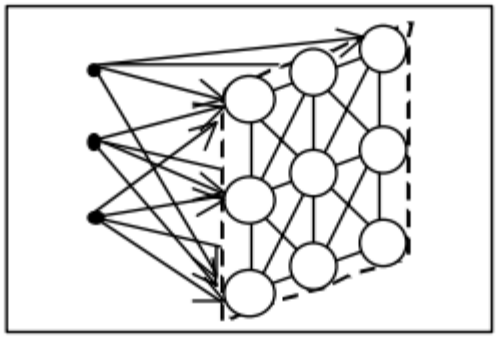


(d)

SOM

Unsupervised clustering/topology presentation

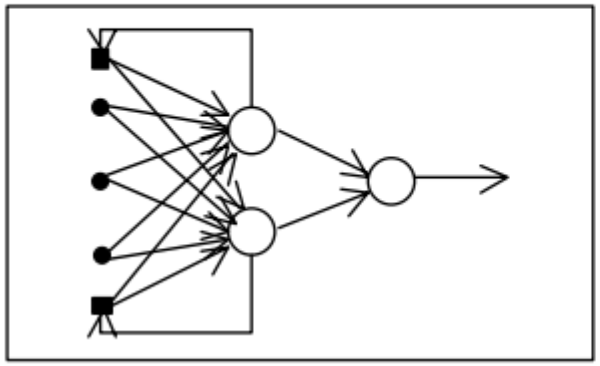
(e)

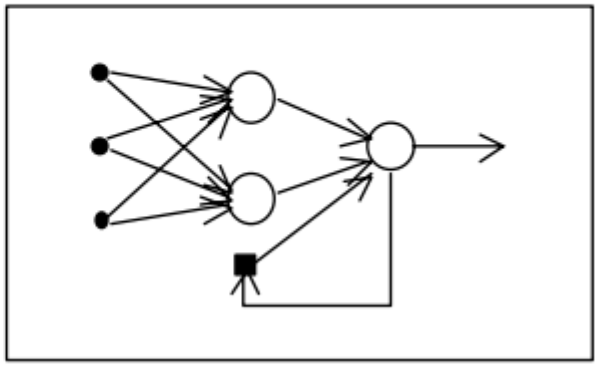


Recurrent petworks

Time-series forecasting

(f)





Weights in neural networks

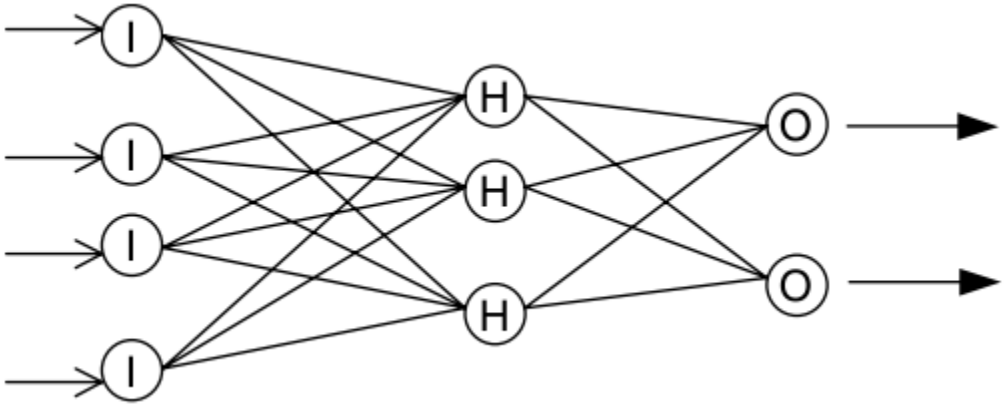
- **Links connecting inputs to neurons and neurons to outputs**

- **They facilitate a structure for flexible learning that allows a network to freely follow the patterns in the data**
  - the flexible structure is what makes them capable of solving such a variety of complex problems

- **Neural networks are parametric models involving the estimation of optimum parameters**

An example multilayer neural network

- A set of weights in the input-hidden layer
- A set of weights in the hidden-output layer



Solving problems using weights

- **The input layer transmits input data to the hidden neurons through input-hidden layer weights**
  - Inputs are weighted by the corresponding weights before they are received by the hidden neurons

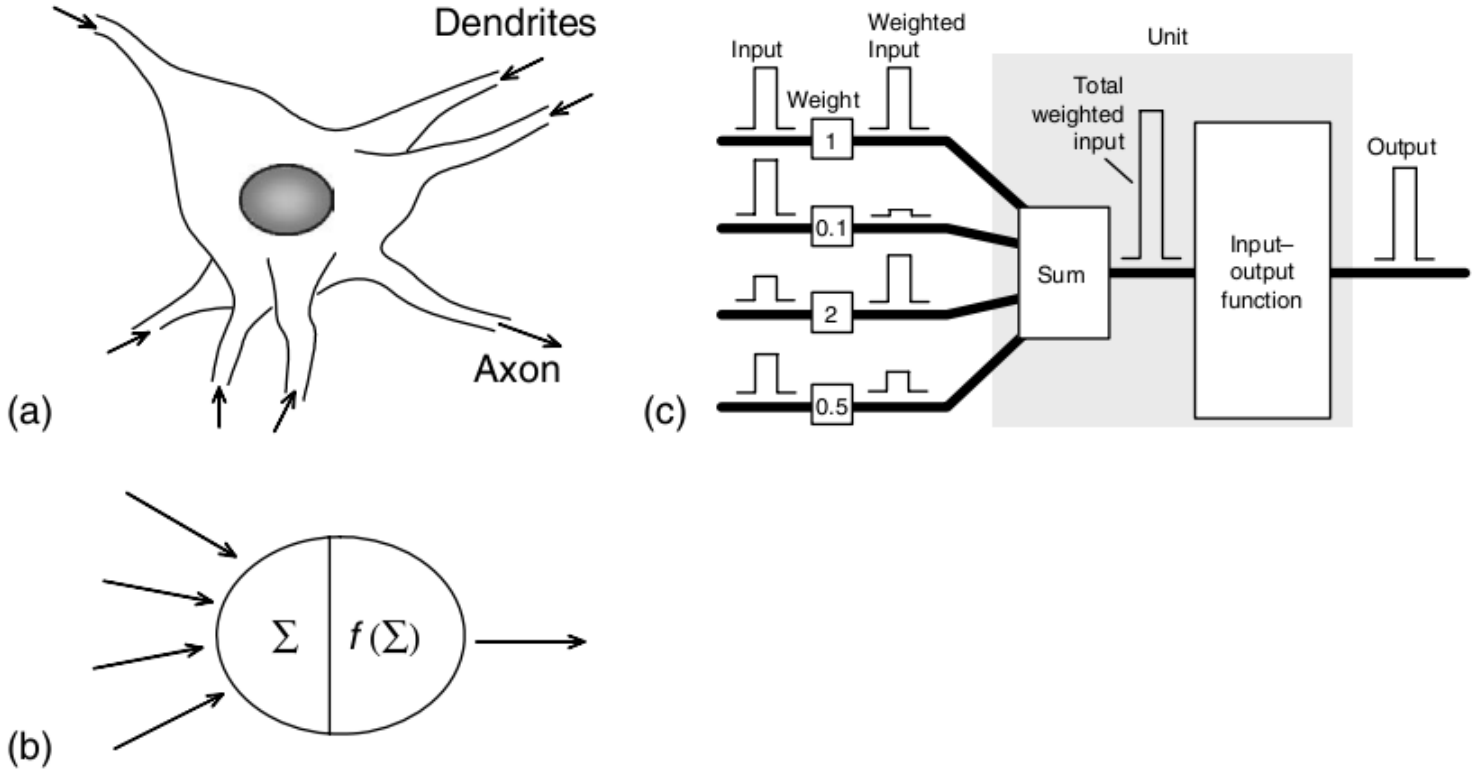
- **The neurons in the hidden layer accumulate and process the weighted inputs before sending their output to the output neurons via the hidden-output layer weights**
  - hidden-neuron output is weighted by the corresponding weights and processed to produce the final output

Learning weights

- This structure is trained to learn by repeated exposure to examples (input–output data) until the network produces the correct output
- Learning involves incrementally changing the connection strengths (weights) until the network learns to produce the correct output
- The final weights are the optimized parameters of the network

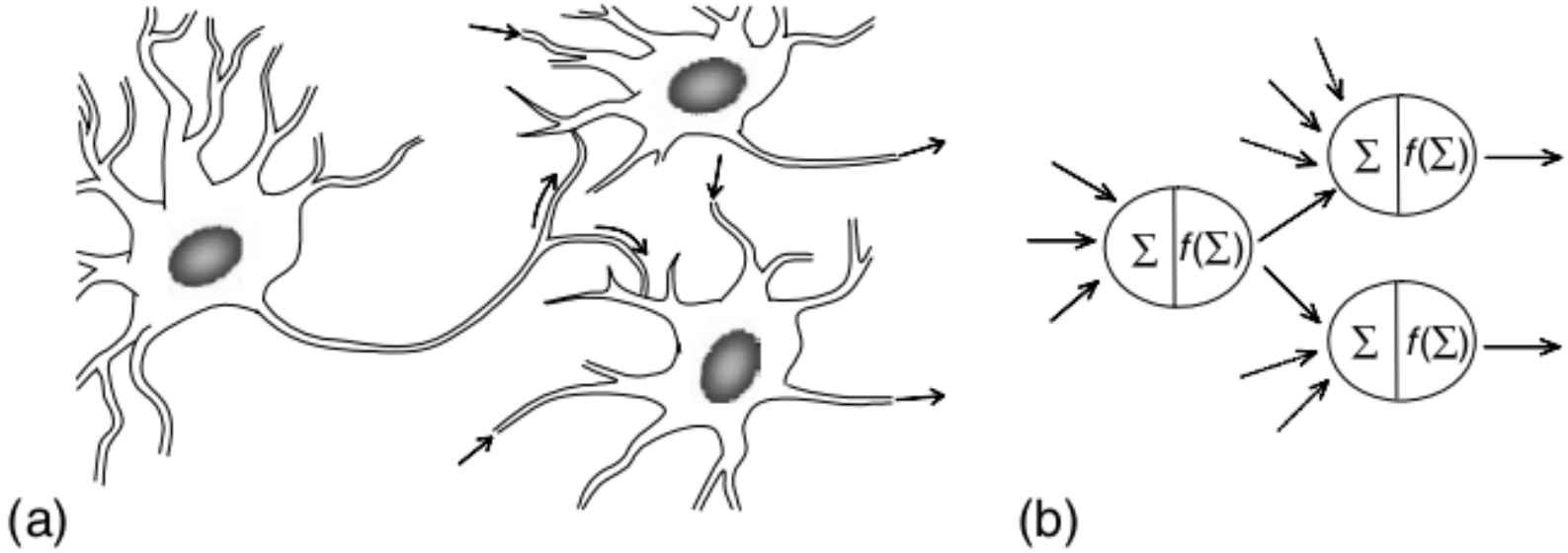
A biological neuron and its representation

- (a) biological neuron
- (b) neuron model
- (c) detailed workings of a single neuron

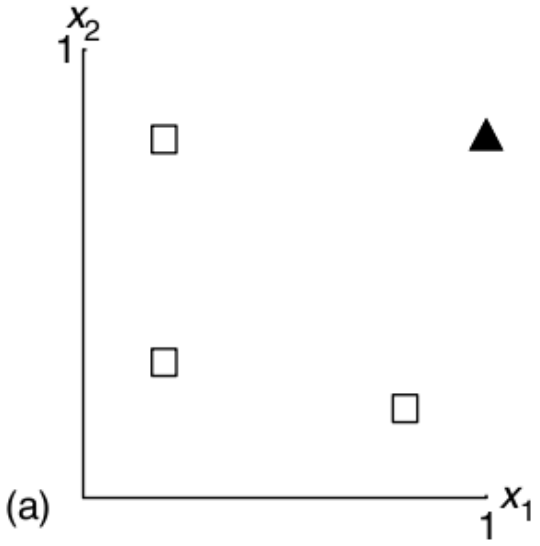


Communication between neurons

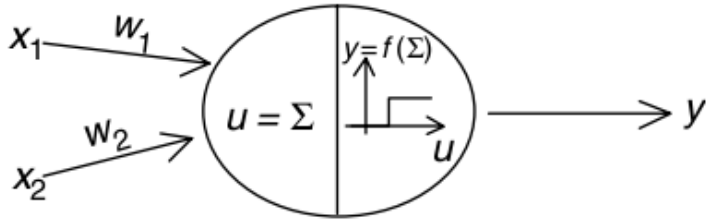
- (a) network of three biological neurons
- (b) neural network model



Threshold neuron as a simple classifier



(a)



(b)

$$\Sigma = u = w_1 x_1 + w_2 x_2$$

$$u = x_1 + x_2$$

$$f(\Sigma) = y = \begin{cases} 0 & u < 1.3 \\ 1 & u \geq 1.3 \end{cases}$$

Table 2.1 Classification Data

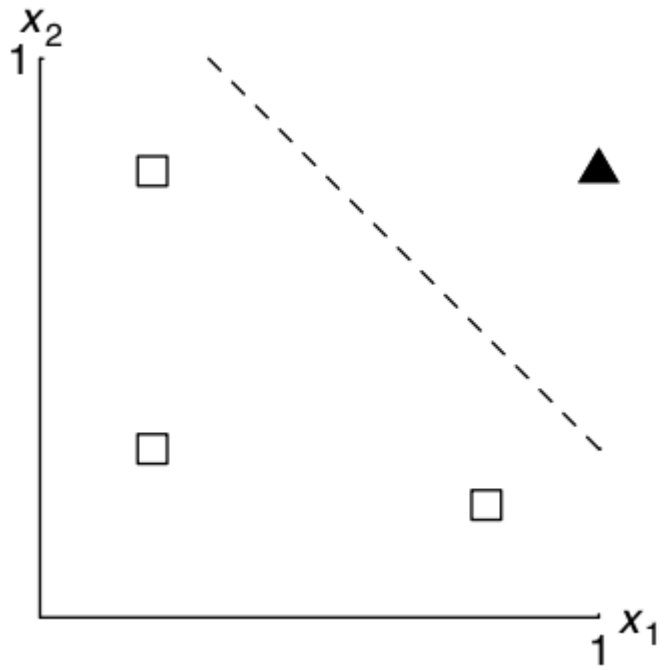
x_1	x_2	t
0.2	0.3	0
0.2	0.8	0
0.8	0.2	0
1.0	0.8	1

$$u = x_1 + x_2 = 1.3$$

$$x_2 = 1.3 - x_1.$$

Table 2.2 Performance of the Threshold Classifier

Input (x_1, x_2)	u	y
(0.2, 0.3)	0.5	0
(0.2, 0.8)	1.0	0
(0.8, 0.2)	1.0	0
(1.0, 0.8)	1.8	1



Vector representations

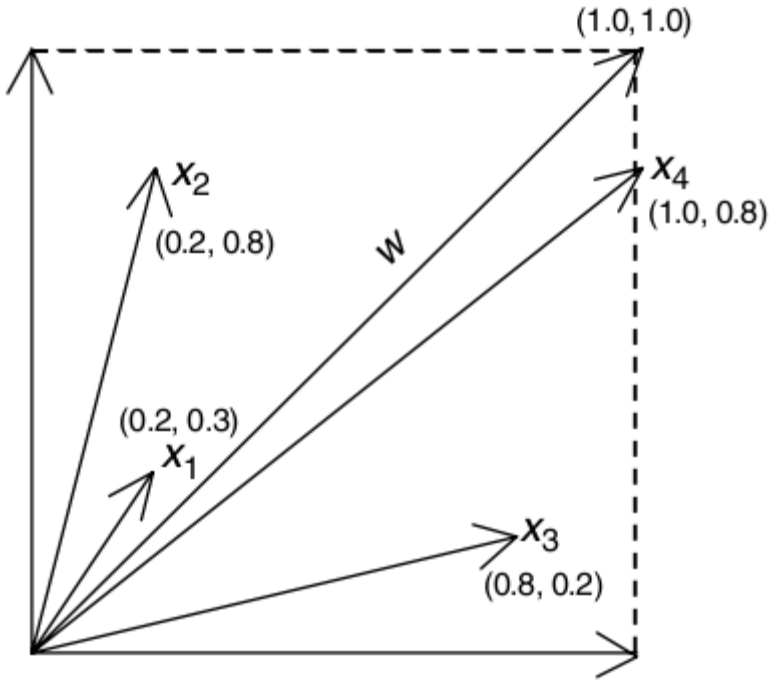
$$\mathbf{x} = \{x_1, x_2\}$$

$$\mathbf{x}_1 = \{0.2, 0.3\},\ \mathbf{x}_2 = \{0.2, 0.8\},\ \mathbf{x}_3 = \{0.8, 0.2\},\ \mathbf{x}_4 = \{1.0, 0.8\}$$

$$\mathbf{w} = \{w_1, w_2\}$$

$$u = \mathbf{w} \cdot \mathbf{x} = \{w_1, w_2\} \cdot \{x_1, x_2\}$$
$$= w_1 x_1 + w_2 x_2$$

For $\mathbf{w} = \{1, 1\}$, $\quad u = x_1 + x_2$



Hebbian learning

- **Threshold neuron does not learn**
- **In 1949, Donald Hebb, a psychologist, proposed a mechanism whereby learning can take place in neurons in a learning environment**
  - In his book *The Organization of Behavior*, Hebb
- **He defined a method to update weights between neurons (called Hebbian learning)**
- **He contributed the following three key points**

Key point 1

- He stated that the information in a network is stored in weights or connections between the neurons

Key point 2

- He postulated that the weight change between two neurons is proportional to the product of their activation values (neuron outputs)

- It enables a mathematical formulation of the concept that stronger excitation between neurons leads to the growth in weights between them

Key point 3

- He proposed a neuron assembly theory
- It suggests that as learning takes place by repeatedly and simultaneously activating a group of weakly connected neurons
- The strength and patterns of the weights between them undergo incremental changes
- It leads to the formation of assemblies of strongly connected neurons

Motivation of Hebbian idea

- Motivated by well-known concepts of classical, or Pavlovian, conditioning
- Pavlovian conditioning: through repeated exposure to a stimulus, learning takes place in the brain
- Learning involves the formation of new connections between neurons that grow in strength through repeated exposure to the stimulus
- Hebbian allows neurons to learn by adjusting their weights in a learning environment

Formulation of Hebbian learning

- If two neurons have activations (or, if x excites y), the connection strength between them increases
- The change in weight between two neurons, Δw, is proportional to the product of x and y

$$\Delta w \propto x \cdot y$$

$$\Delta w = \beta\, x \cdot y$$

$$w_{\text{new}} = w_{\text{old}} + \Delta w = w_{\text{old}} + \beta\, x \cdot y$$

- The constant β is termed the "learning rate," and determines the speed at which learning takes place

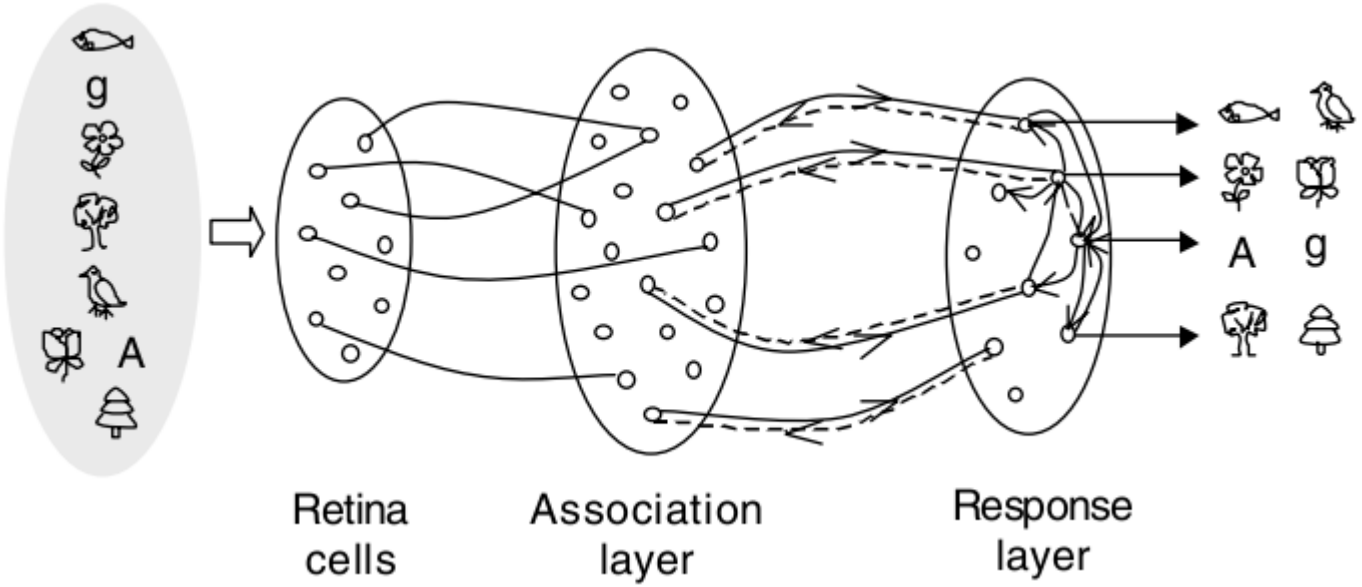
Implementation of learning in a neural assembly (perceptron)

- **Frank Rosenblatt (during the 1950s) focused on how the brain**
  - learns from experience
  - responds in similar ways to similar experiences
  - recognizes patterns
  - groups similar experiences together
  - differentiates them from dissimilar experiences
- **despite the imprecision in initial wiring in the brain**

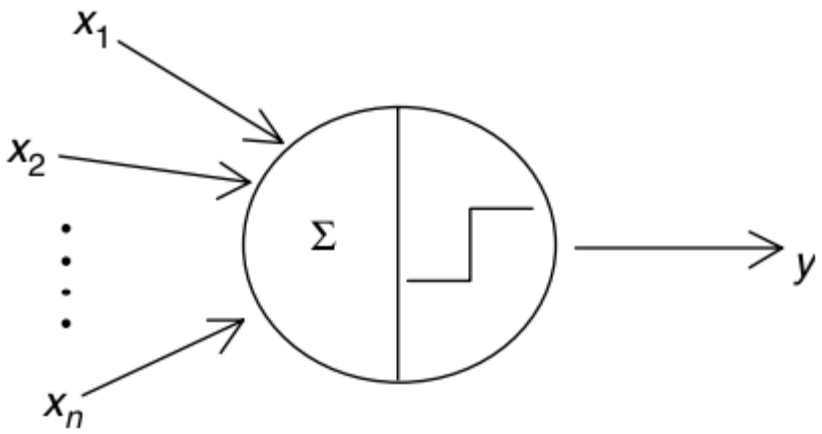
- **Perceptron : he proposed it by making threshold neurons learn using Hebbian learning**

A schematic diagram of perceptron network

- **Suppose three-layer perceptron network**
  - input layer : a set of sensory cells in the retina, randomly and partially connected to neurons in the next higher association layer
  - association and response neurons can excite or inhibit each other with bidirectional connections
- **Goal: activating the correct response neurons for each input pattern class**
  - Learning happens between association layer and response layer



Perceptron with Supervised Learning as a Classifier



$$u = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

$$y = \begin{cases} 0 & u < 0 \\ 1 & u \geq 0 \end{cases}$$

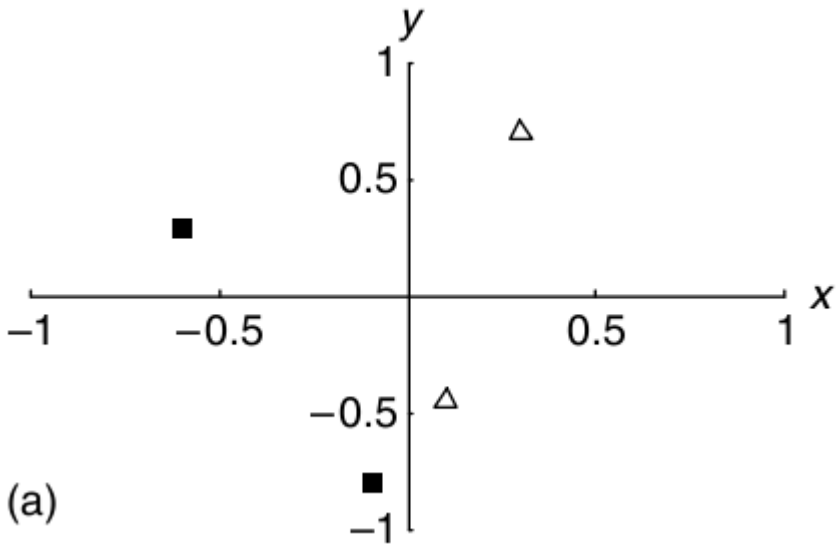
$$\text{Error} = E = t - y$$

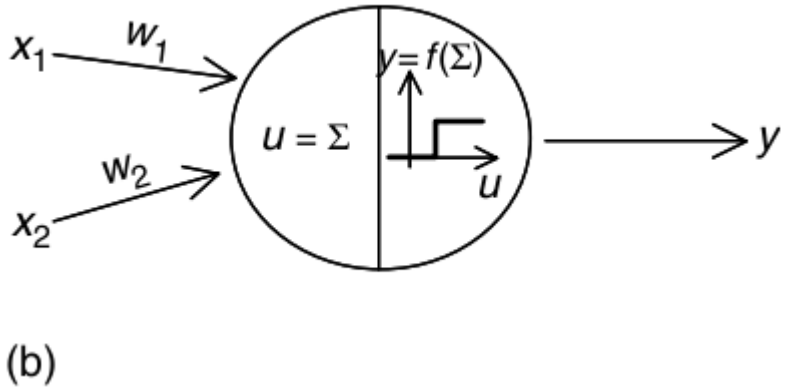
$$w_{\text{new}} = w_{\text{old}} + \beta \mathbf{x} E$$

$$w_{\text{new}} = \begin{cases} w_{\text{old}} & E = 0 \quad (\text{i.e., } t = y) \\ w_{\text{old}} + \beta \mathbf{x} & E = 1 \quad (\text{i.e., } t = 1, y = 0) \quad \text{Rule 1,} \\ w_{\text{old}} - \beta \mathbf{x} & E = -1 \quad (\text{i.e., } t = 0, y = 1) \quad \text{Rule 2} \end{cases}$$

$$\beta = 0.5 \qquad w_1^0 = 0.8 \quad \text{and} \quad w_2^0 = -0.5$$

$$\mathbf{L}^0 = \sqrt{(w_1^0)^2 + (w_2^0)^2} = \sqrt{(0.8)^2 + (-0.5)^2} = 0.94$$





Present input pattern 1—$A1$: $x_1 = 0.3$, $x_2 = 0.7$, $t = 1$; $w_1^0 = 0.8$, $w_2^0 = -0.5$

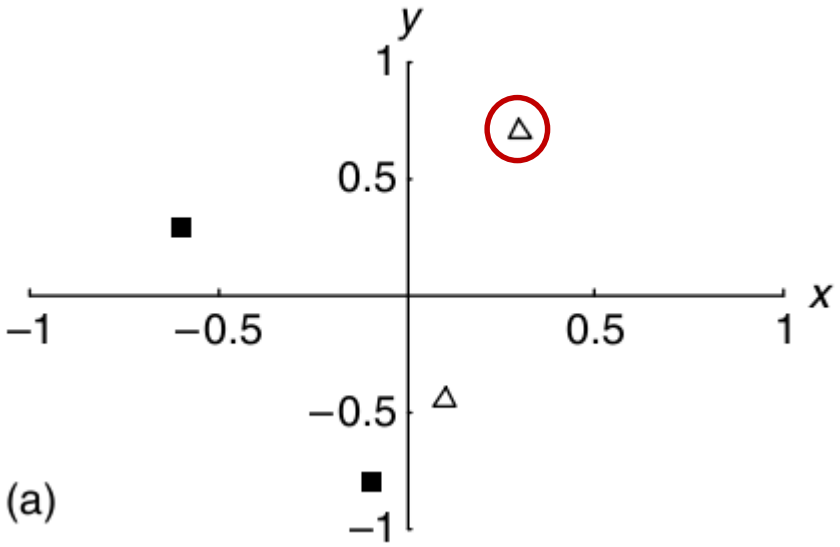
$$u = (0.8)(0.3) + (-0.5)(0.7) = -0.11,$$

$$u < 0 \Rightarrow y = 0$$

$\Rightarrow$ classification INCORRECT;

$$\Delta w_1^1 = \beta x_1 = (0.5)(0.3) = 0.15$$

$$\Delta w_2^1 = \beta x_2 = (0.5)(0.7) = 0.35$$



$$w_1^1 = w_1^0 + \Delta w_1^1 = 0.8 + 0.15 = 0.95$$

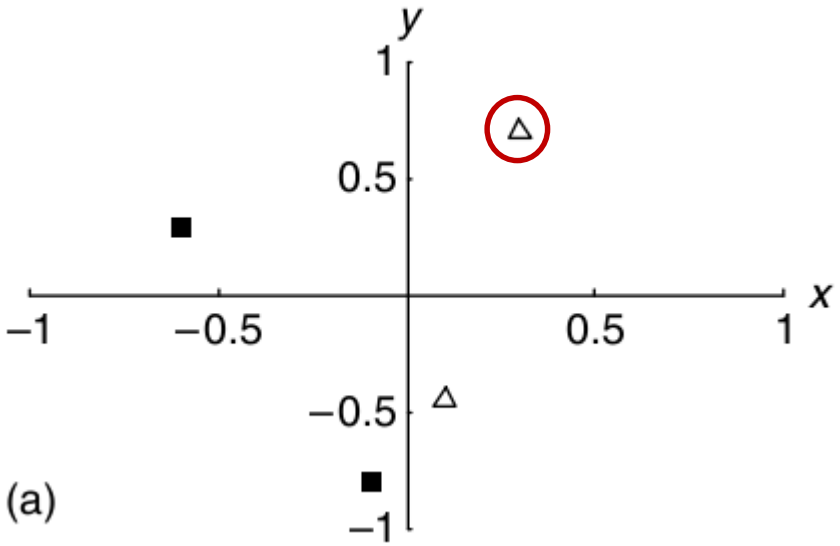
$$w_2^1 = \Delta w_2^0 + w_2^1 = -0.5 + 0.35 = -0.15$$

$$\mathbf{w}^1 = [w_1^1, w_2^1] = [0.8, -0.5] + [0.15, 0.35] = [0.95, -0.15]$$

$$\mathbf{L}^1 = \sqrt{(w_1^1)^2 + (w_2^1)^2} = \sqrt{0.95^2 + (-0.15)^2} = 0.96$$

$$u = (0.3)(0.95) + (0.7)(-0.15) = 0.18 > 0$$

$$\Rightarrow y = 1 \Rightarrow \text{classification CORRECT.}$$



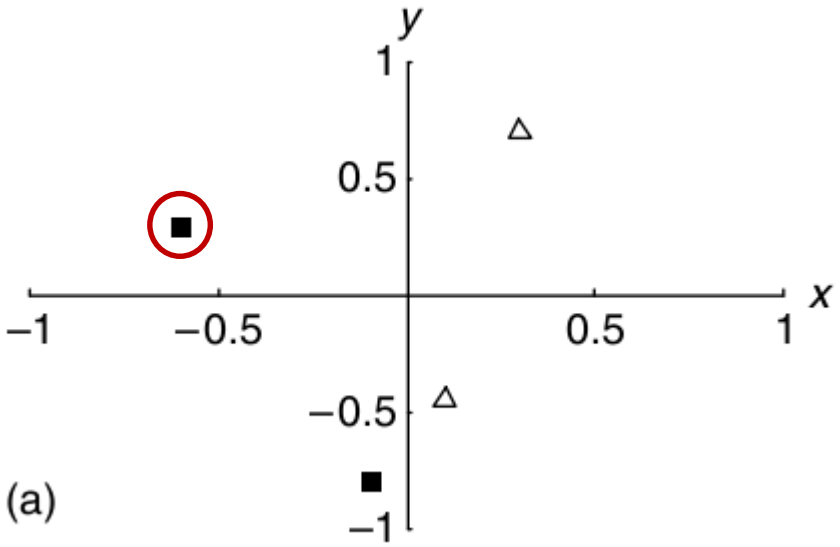
Present input pattern 2—$B1$: $x_1 = -0.6,\ x_2 = 0.3,\ t = 0;$

$$w_1^1 = 0.95,\ w_2^1 = -0.15,$$

$$u = (-0.6)(0.95) + (0.3)(-0.15) = -0.615$$

$$u < 0 \Rightarrow y = 0$$

⇨ classification CORRECT ⇨ weights are not modified.



Present input pattern 3—$B2$: $x_1 = -0.1$, $x_2 = -0.8$,
$t = 0$; $w_1^1 = 0.095$, $w_2^1 = -0.15$.

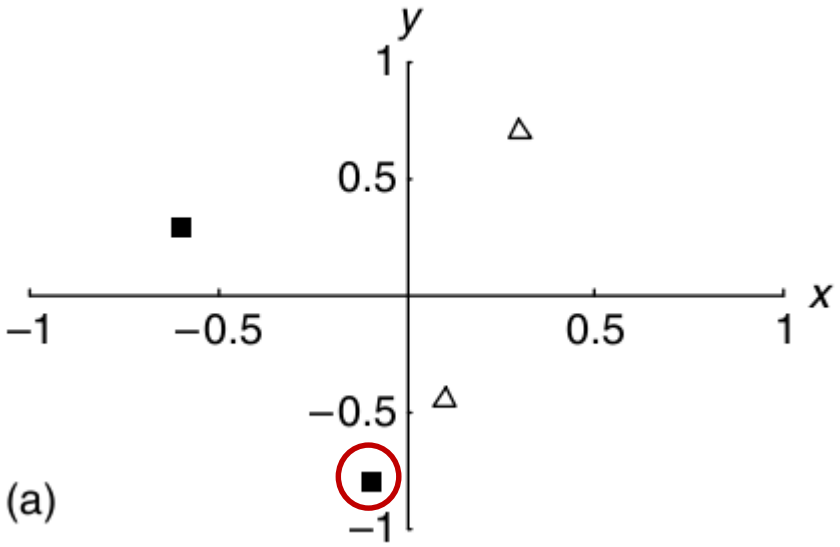
$$u = (-0.1)(0.95) + (-0.8)(-0.15) = 0.025$$

$$u > 0 \Rightarrow y = 1$$

⇨classification INCORRECT

$$\Delta w_1^2 = -\beta x_1 = -(0.5)(-0.1) = 0.05$$

$$\Delta w_2^2 = -\beta x_2 = -(0.5)(-0.8) = 0.4.$$



$$w_1^2 = w_1^1 + \Delta w_1^2 = 0.95 + 0.05 = 1.0$$

$$w_2^2 = w_2^1 + \Delta w_2^2 = -0.15 + 0.4 = 0.25.$$

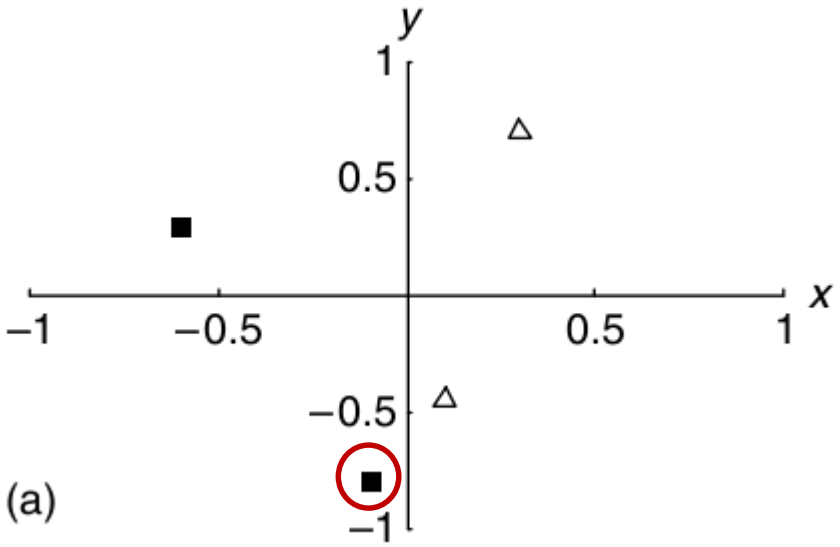
$$\mathbf{w}^2 = [w_1^2, w_2^2] = [0.95, -0.15] + [0.05, 0.4] = [1.0, 0.25]$$

$$\mathbf{L}^2 = \|\mathbf{w}^2\| = \sqrt{1.0^2 + 0.25^2} = 1.03$$

u = (-0.1)(1.0) + (-0.8)(0.25) = -0.3

u < 0 ⇒ y = 0

⇒ classification CORRECT



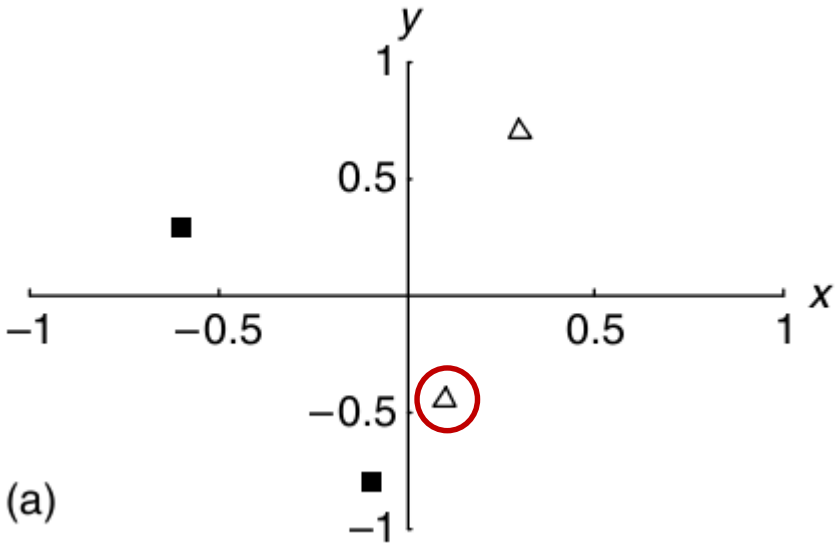
Present input pattern 4—$A2$: $x_1 = 0.1$, $x_2 = -0.45$, $t = 1$; $w_1^2 = 1.0$, $w_2^2 = 0.25$.

$$u = (0.1)(1.0) + (-0.45)(0.25) = -0.0125$$

$u < 0 \Rightarrow y = 0$ underprediction, classification INCORRECT

$$\Delta w_1^3 = \beta x_1 = (0.5)(0.1) = 0.05$$

$$\Delta w_2^3 = \beta x_2 = (0.5)(-0.45) = -0.225.$$



$$w_1^3 = w_1^2 + \Delta w_1^3 = 1.0 + 0.05 = 1.05$$

$$w_2^3 = w_2^2 + \Delta w_2^3 = 0.25 - 0.225 = 0.025$$

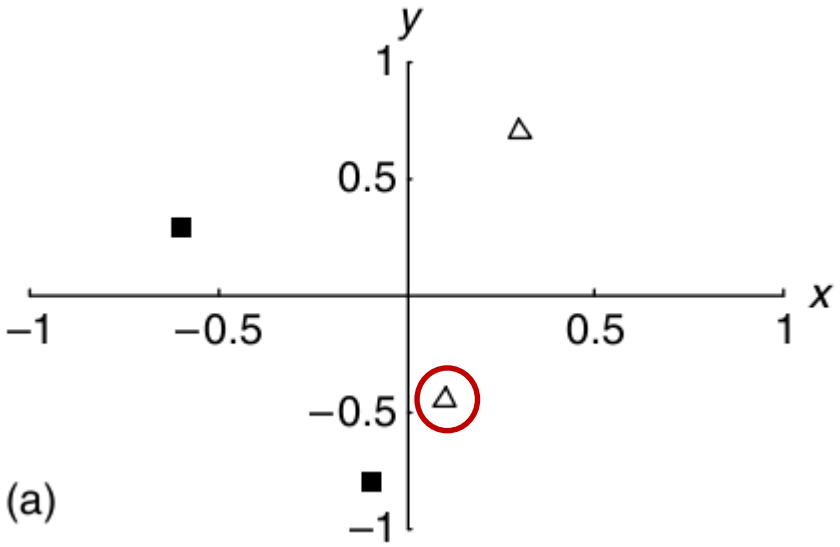
$$\mathbf{W}^3 = [w_1^3, w_2^3] = [1.0, 0.25] + [0.05, -0.225] = [1.05, 0.025].$$

$$\mathbf{L}^3 = \sqrt{1.05^2 + 0.025^2} = 1.05.$$

u = (0.1)(1.05) + (-0.45)(0.025) = 0.09375

u > 0 ⇒ y = 1

⇒ classification CORRECT



$$w_1^0 = 0.8 \quad \text{and} \quad w_2^0 = -0.5$$

$$\mathbf{w}^1 = [w_1^1, w_2^1] = [0.8, -0.5] + [0.15, 0.35] = [0.95, -0.15]$$

$$\mathbf{w}^2 = [w_1^2, w_2^2] = [0.95, -0.15] + [0.05, 0.4] = [1.0, 0.25]$$

$$\mathbf{W}^3 = [w_1^3, w_2^3] = [1.0, 0.25] + [0.05, -0.225] = [1.05, 0.025].$$

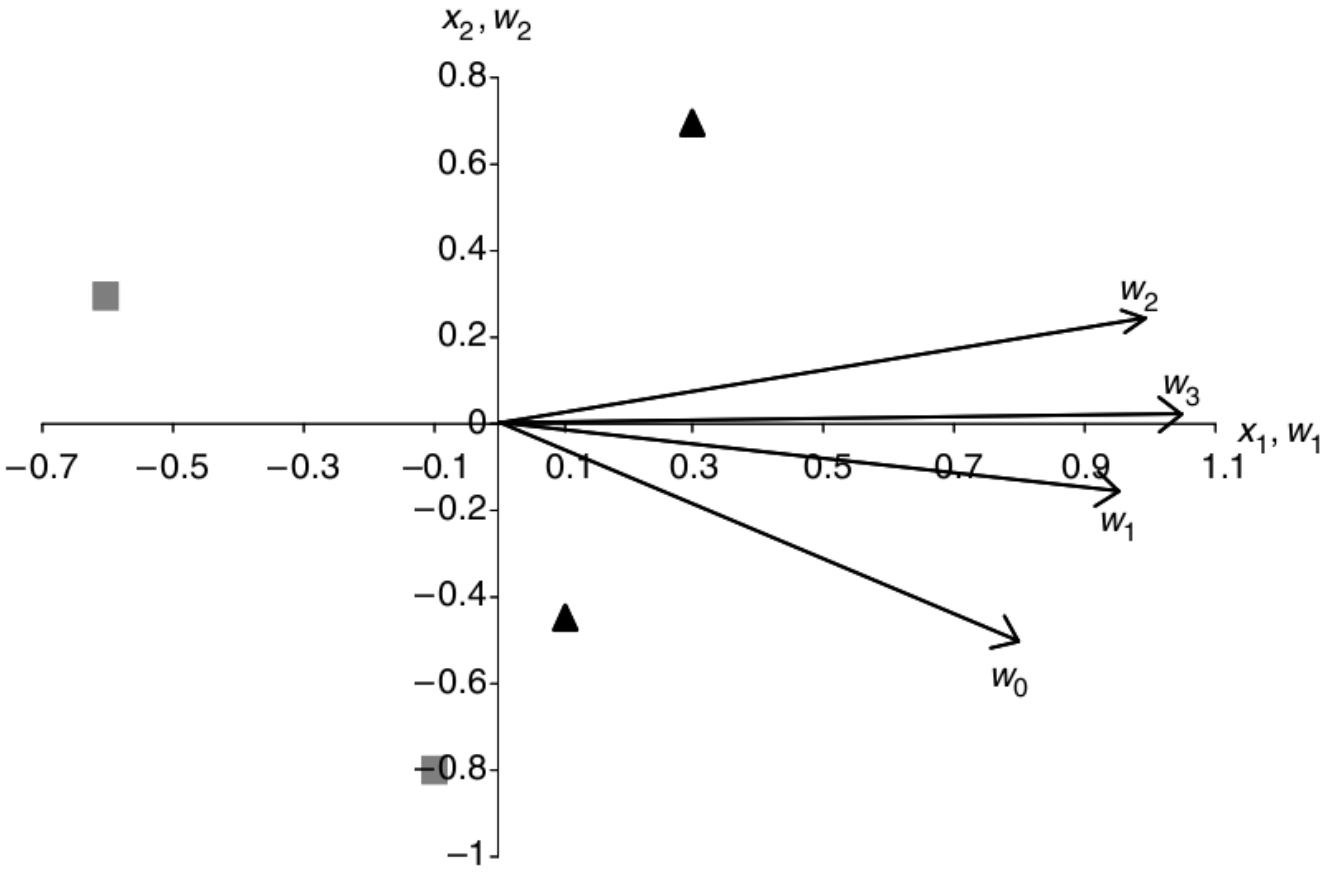


Table 2.3 Classification Accuracy of the Trained Perceptron

Input Pattern	x_1, x_2	t	u	y	*Classification Accuracy*
A1	0.3, 0.7	1	$0.3 \times 1.05 + 0.7 \times 0.025 = 0.3325 > 0$	1	Correct
B1	−0.6, 0.3	0	$-0.6 \times 1.05 + 0.3 \times 0.025 = -0.6225 < 0$	0	Correct
B2	−0.1, −0.8	0	$-0.1 \times 1.05 + (-0.8) \times 0.025 = -0.125 < 0$	0	Correct
A2	0.1, −0.45	1	$0.1 \times 1.05 + (-0.45) \times 0.025 = 0.09375 > 0$	1	Correct

$$u = w_1 x_1 + w_2 x_2 = 0$$

$$x_2 = -\left(\frac{w_1}{w_2}\right) x_1.$$

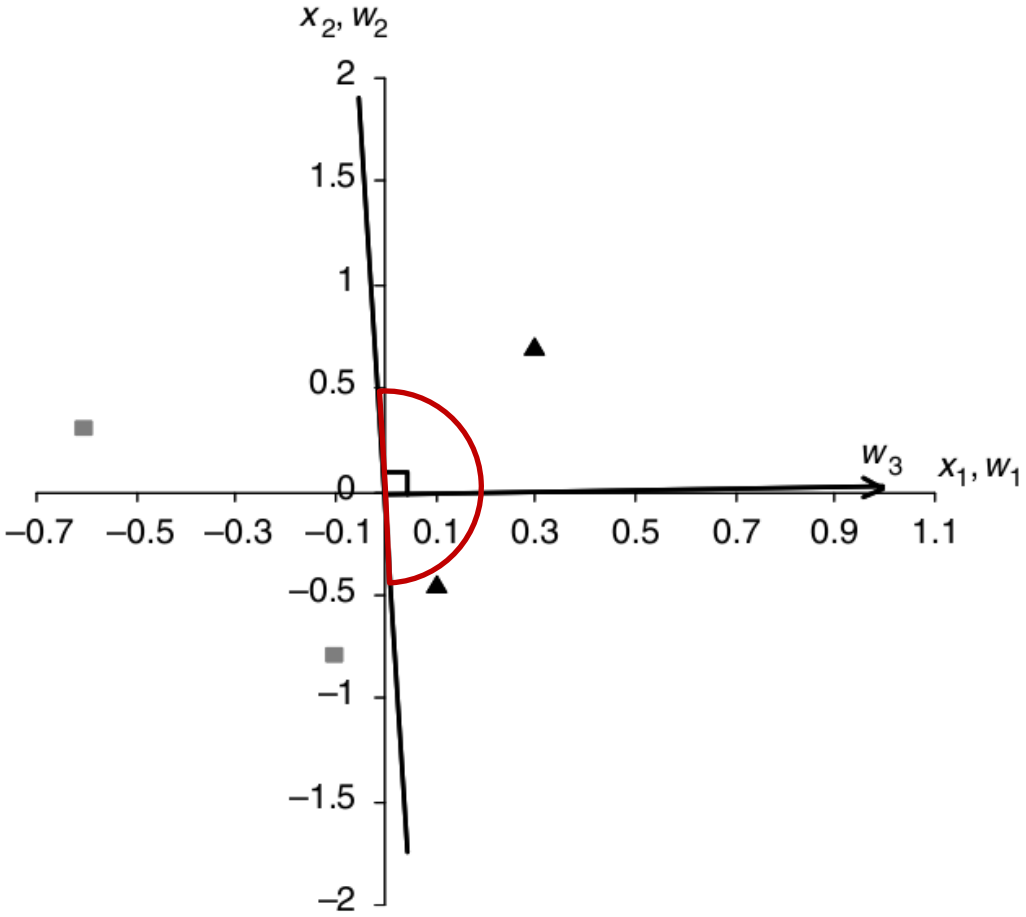
$$x_2 = -(1.05/0.025)x_1 = -42x_1,$$

$$x_2 = -42x_1.$$

$$u = x_1 w_1 + x_2 w_2$$

$$u = \|\mathbf{x}\|\|\mathbf{w}\|\cos\theta$$

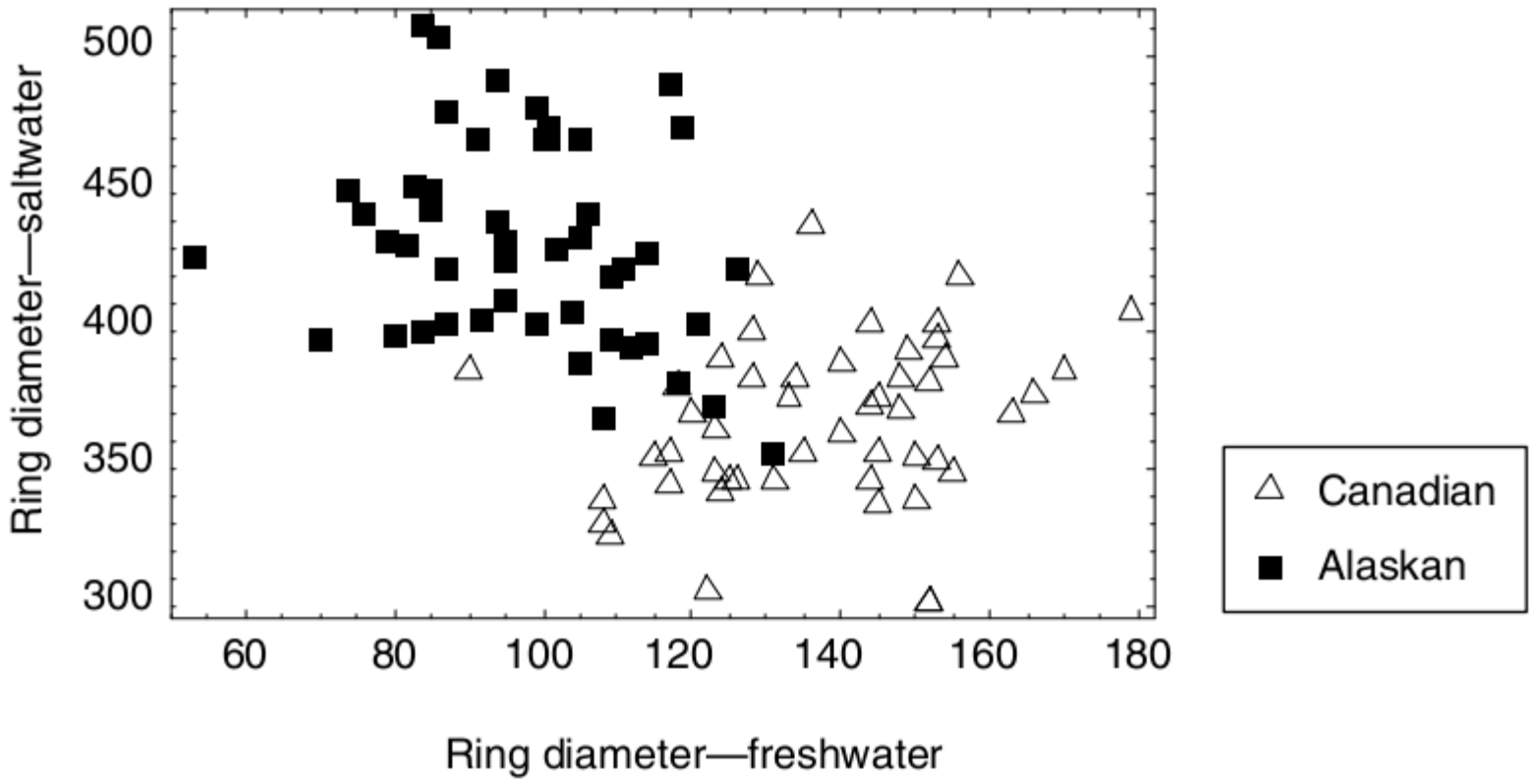
u ≥ 0 when -90° ≤ θ ≤ 90°



A Practical Example of Perceptron on a Larger Realistic Data Set: Identifying the Origin of Fish from the Growth-Ring Diameter of Scales

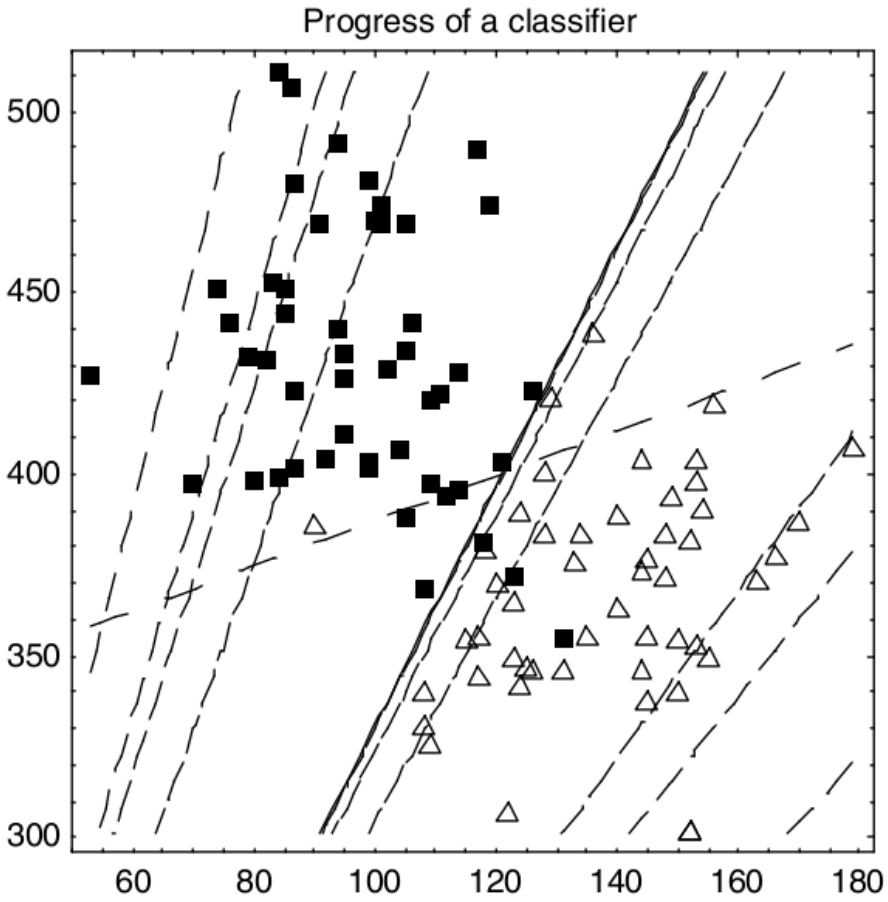
- **Fifty fish from each place of origin were caught, and the growth-ring diameter of the scales was measured**
  - for the time they lived in freshwater
  - for the subsequent time they lived in saltwater

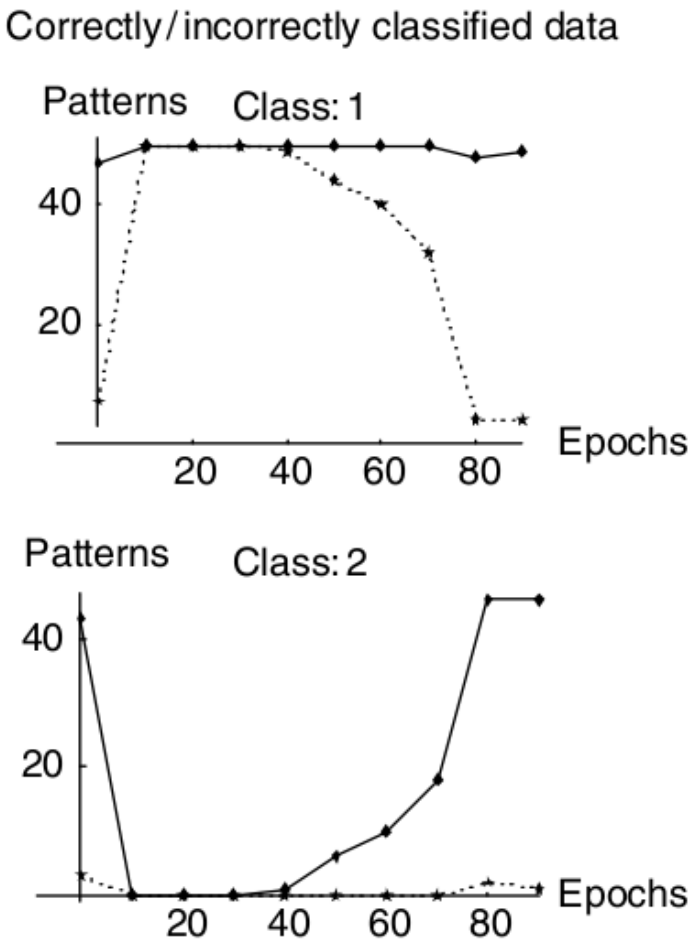
- **Problem : identifying the origin (Alaskan or Canadian) of a fish from its growth-ring diameter in freshwater and saltwater**


Ring diameter—saltwater
500
450
400
350
300
60
80
100
120
140
160
180
Ring diameter—freshwater
Canadian
Alaskan

Epoch

- One pass of all input patterns through the perceptron

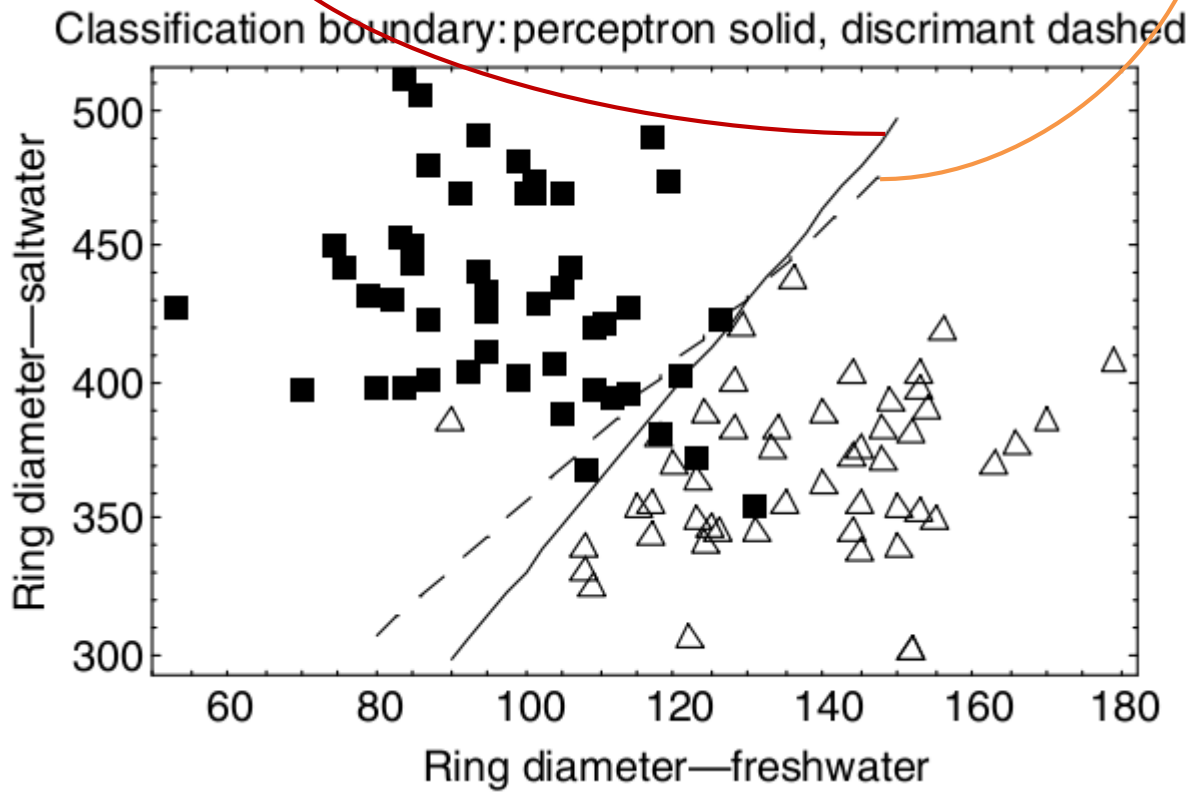




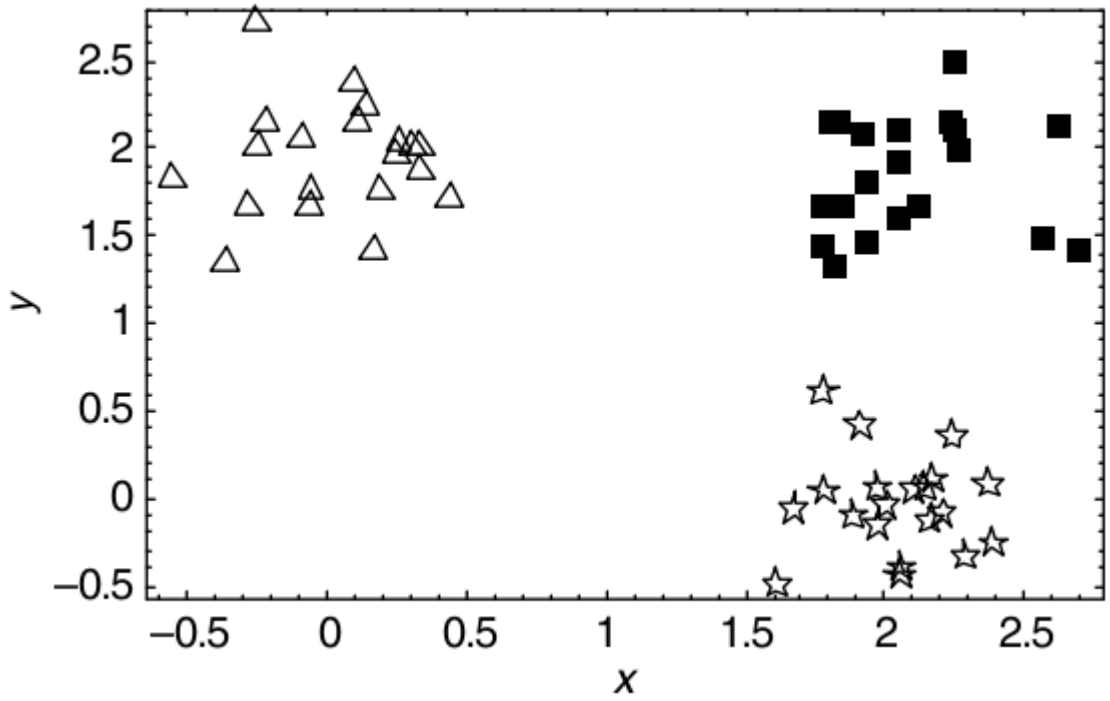
Comparison of Perceptron with Linear Discriminant Function Analysis

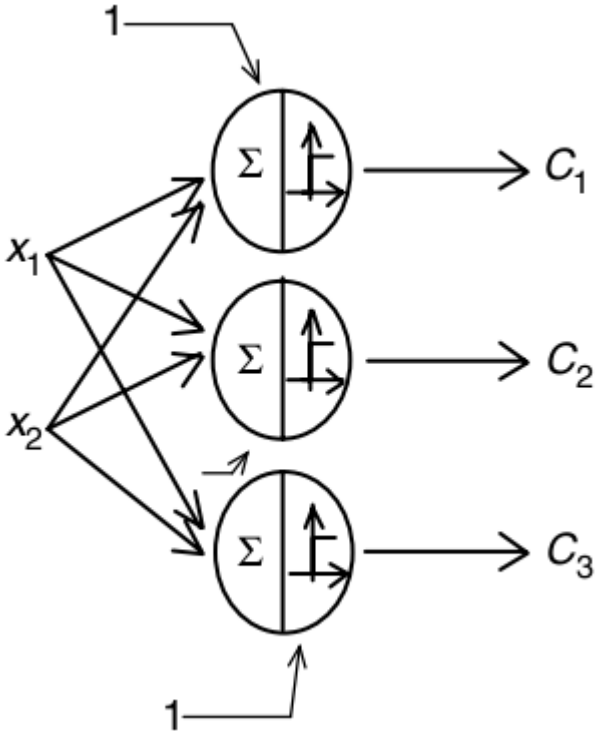
$x_2 = 0.026 + 3.31x_1$

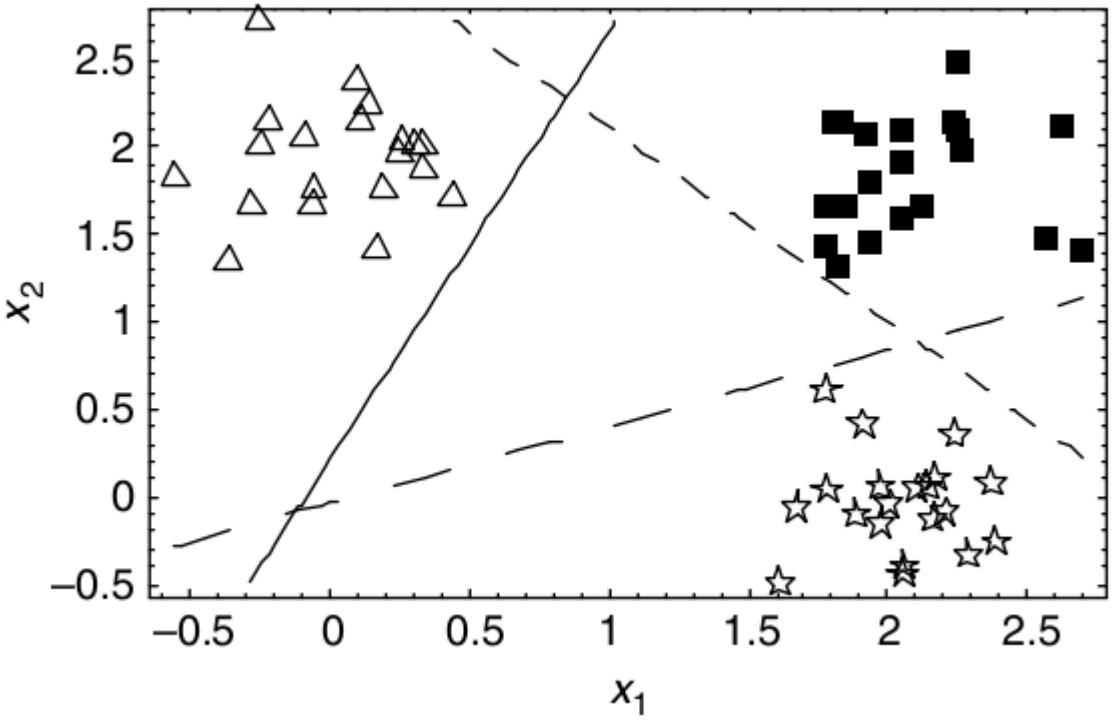
$x_2 = 106.9 + 2.5x_1$



Multi-Output Perceptron for Multicategory Classification



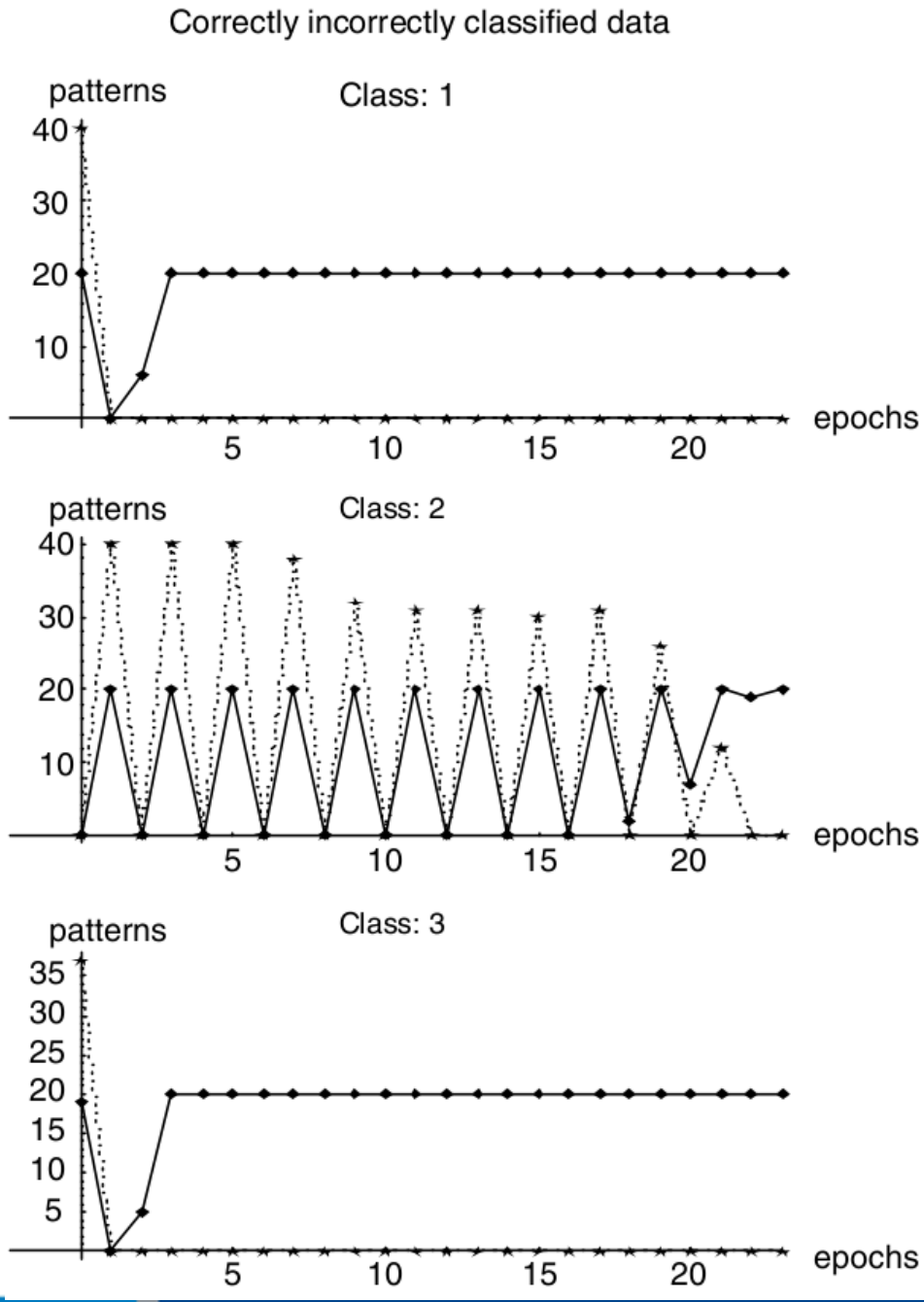




$$-3.8 - 42.8x_1 + 17.5x_2 = 0$$
$$-78.2 + 26.9x_1 + 24.3x_2 = 0$$
$$-1.35 + 16.6x_1 - 38.0x_2 = 0.$$

Table 2.4 Inputs and the Corresponding Output from the Multi-Output Perceptron

Input	*Output*	*Target*
(−0.56, 1.84)	(1, 0, 0)	(1, 0, 0)
(2.23, 2.15)	(0, 1, 0)	(0, 1, 0)
(2.00, −0.04)	(0, 0, 1)	(0, 0, 1)

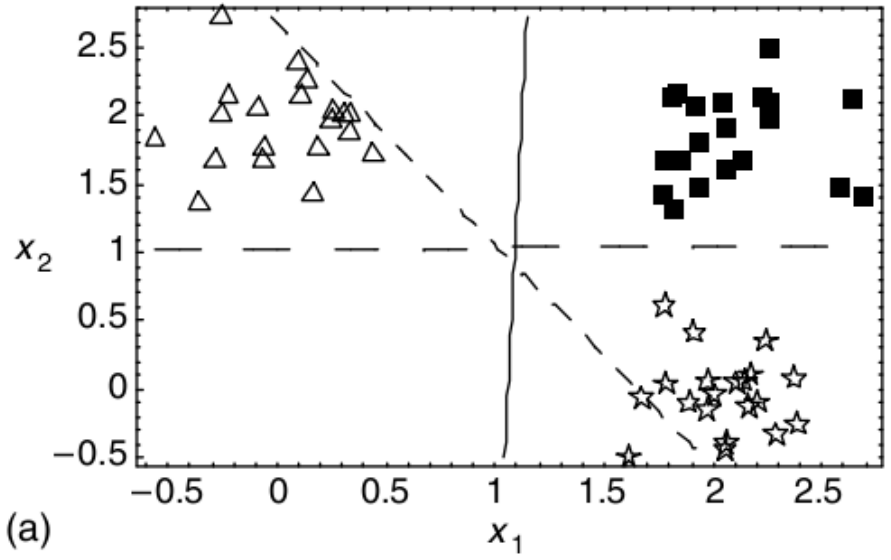


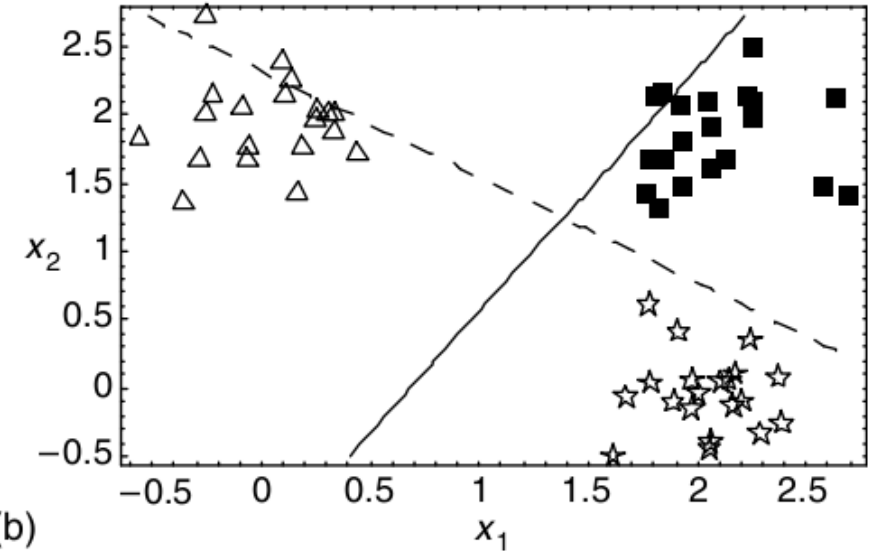
Comparison with multiple discriminant classifier

$$-31.4 + 29.7x_1 - 0.953x_2 = 0$$

$$-49.5 + 30.0x_1 + 18.5x_2 = 0$$

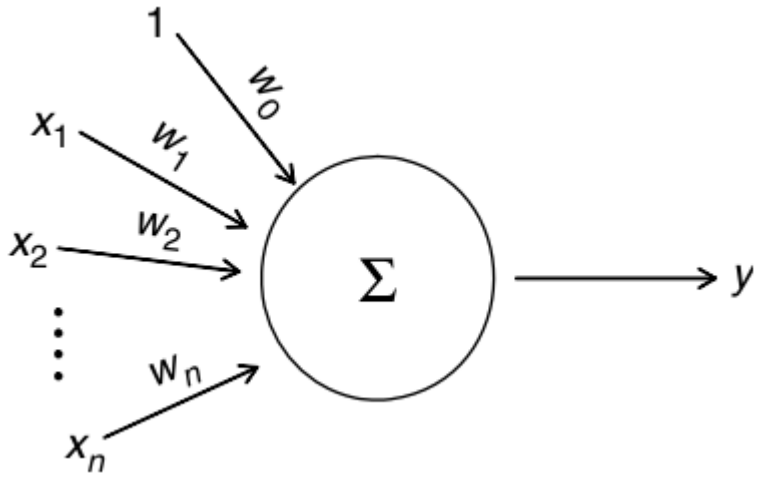
$$-20.6 - 0.231x_1 + 20.1x_2 = 0$$





Linear Neuron for Linear Classification and Prediction

- **Output is a linear sum of weighted inputs**
- **Activation function in the neuron is linear (continuous)**
  - as opposed to a threshold in perceptron
- **Capable of both linear classification and linear function approximation**



$$u = \sum_{1}^{n} w_i x_i + w_0 = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + w_0$$

$$y = u;$$

$$y = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + w_0$$

Two key attributes

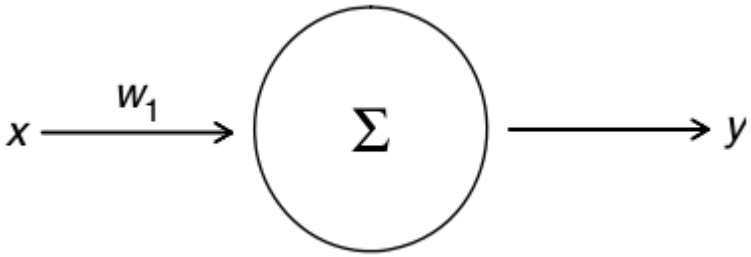
- **It uses supervised learning (as in perceptron)**
  - the neuron must have the desired (target) output for the inputs on which it is trained
- **Learning is based on the delta rule or gradient descent, which adjusts the weights in the direction in which error goes down most steeply**

Learning with the Delta Rule

$$u = w_1 x$$
$$y = u = w_1 x.$$

$$E = t - y = t - w_1 x$$

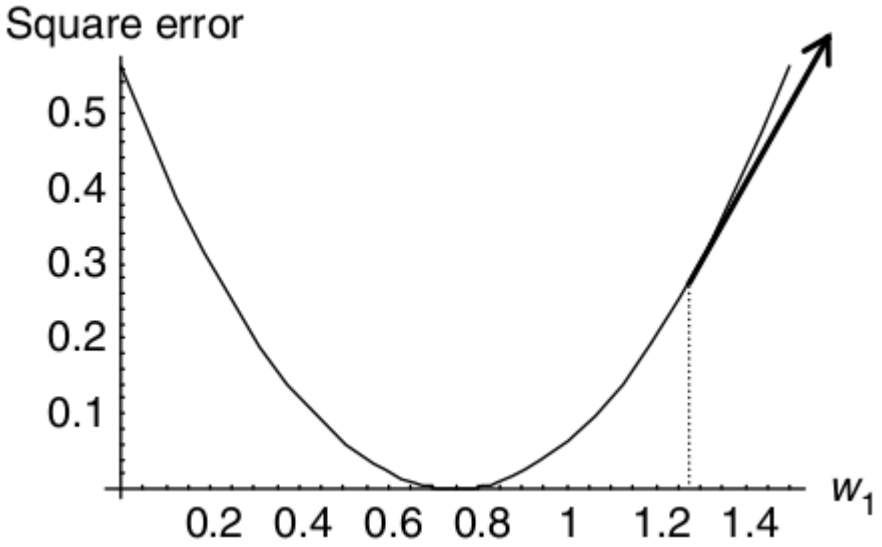
$$\varepsilon = \frac{1}{2} E^2 = \frac{1}{2}(t - w_1 x)^2$$



- **Delta rule** : **reach the optimum weight** **from the initial weight values by** **descending down** **the bowl in the** **opposite direction to the gradient** **during training**

- **To find the gradient at a particular weight**
  - **differentiate the square error function with respect to w**

$$\frac{d\varepsilon}{dw_1} = \frac{2}{2}(t-y)(-x) = -Ex$$



$$\varepsilon = \frac{1}{2}E^2 = \frac{1}{2}(t - w_1 x)^2$$

```
x = 2.0
t = 1.53
```

$$\Delta w_1 \propto Ex$$

$$\Delta w_1 = \beta\, Ex$$

$$w_1^{i+1} = w_1^i + \Delta w_1 = w_1^i + \beta\, Ex$$

$$w_0^{i+1} = w_0^i + \beta\, E$$

$$w_j^{i+1} = w_j^i + \beta\, x_j E$$

Example-by-example learning vs. batch learning

- **Training can take many epochs to complete learning**
  - epoch : one pass of the whole training dataset
  - learning can be performed after **each input pattern (iteration)**, or after **an epoch**

- **Example-by-example learning : adjusting the weights after each presentation of an input pattern**
  - for some problems, this can cause weights to oscillate due to the fact that the adjustment required by one input vector may be canceled by that of another input
  - however, this method works well for some other problems

- **Epoch or batch training : more popular because it generally provides stable solutions**

Batch learning

- **Goal : reducing the average error over all the patterns, which is called the mean square error (MSE)**

$$\mathrm{MSE} = \frac{1}{2n}\sum_{i=1}^{n} E_i^2$$

- **Method**
  - obtain the error gradient for each input pattern as it is processed
  - average them at the end of the epoch
  - use this average value to adjust the weights using the delta rule

$$\Delta w_1 = \beta\left(\frac{1}{n}\sum_{i=1}^{n} E_i x_i\right)$$

Linear Neuron as a Classifier

$$u = w_1 x_1 + w_2 x_2$$

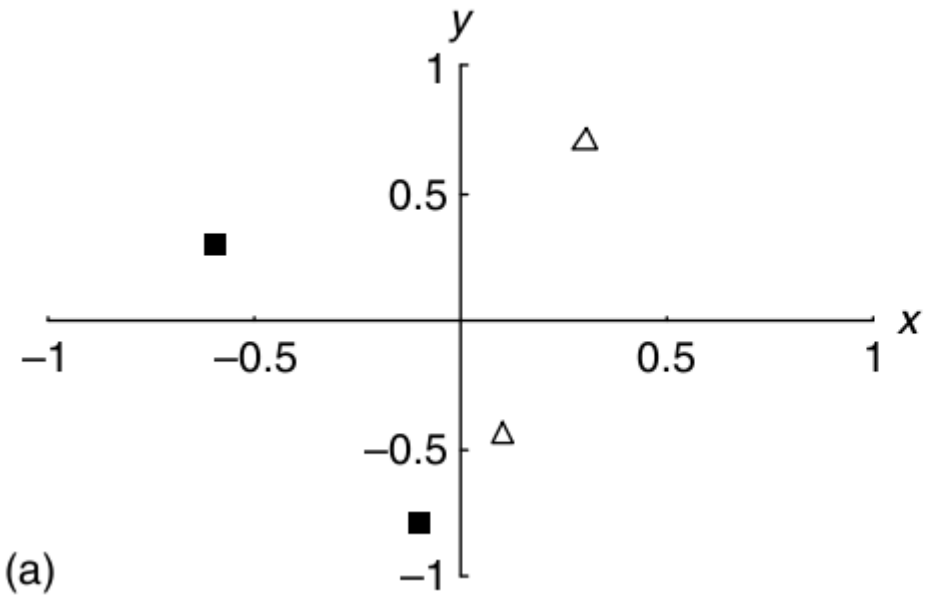
$$y = u$$

$$E = t - y$$

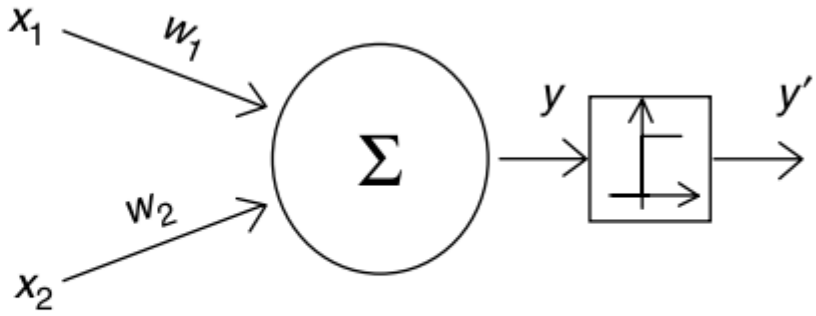
$$\Delta w = \beta E x$$

$$w_{\text{new}} = w_{\text{old}} + \Delta w.$$

$$y' = \begin{cases} 1 & y \geq 0 \\ 0 & y < 0. \end{cases}$$



(a)



(b)

Present input pattern 1—$A1$: $x_1 = 0.3,\ x_2 = 0.7,\ t = 1;\ w_1^0 = 0.8,$

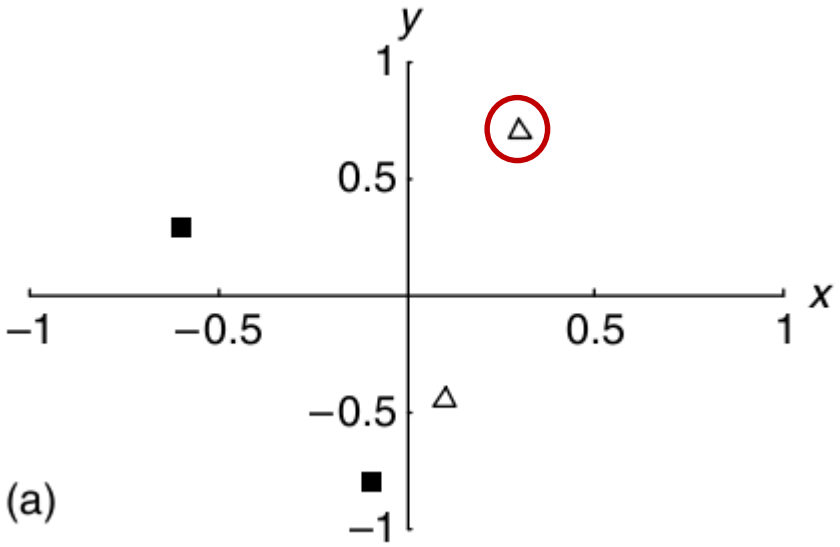
$$w_2^0 = -0.5.$$

$$u = (0.3)(0.8) + (0.7)(-0.5) = -0.11$$

$$y = u = -0.11 < 0 \Rightarrow \text{Category } 0$$

$$\Rightarrow \text{WRONG CLASSIFICATION}$$

$$E = 1 - (-0.11) = 1.11.$$



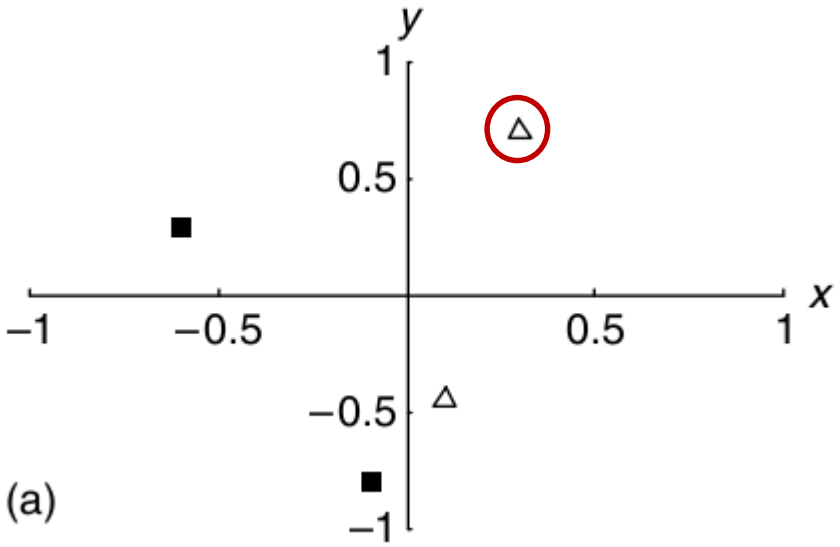
$$\Delta w_1^1 = (0.5)(1.11)(0.3) = 0.1665$$

$$\Delta w_2^1 = (0.5)(1.11)(0.7) = 0.3885.$$

$$w_1^1 = 0.8 + 0.1665 = 0.9665$$

$$w_2^1 = -0.5 + 0.3885 = -0.1115.$$

$$\mathbf{w}^1 = [0.9665, -0.1115].$$

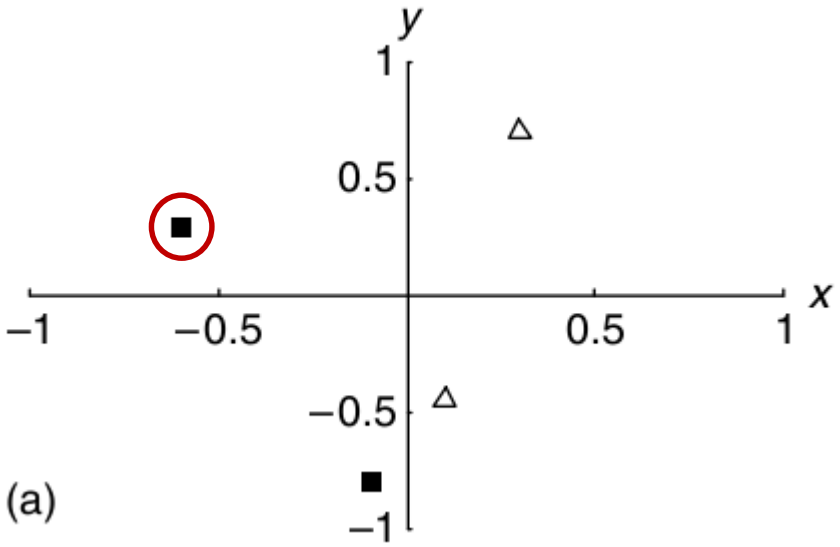


Present input pattern 2—$B1$: $x_1 = -0.6,\ x_2 = 0.3,\ t = 0;\ w_1^1 = 0.9665,$
$w_2^1 = -0.1115.$

$$u = (-0.6)(0.9665) + (0.3)(-0.1115) = -0.61335$$

$$y = u = -0.61335 < 0 \Rightarrow \text{Category } 0$$

⇨CORRECT CLASSIFICATION ⇨ WEIGHTS DO NOT CHANGE.

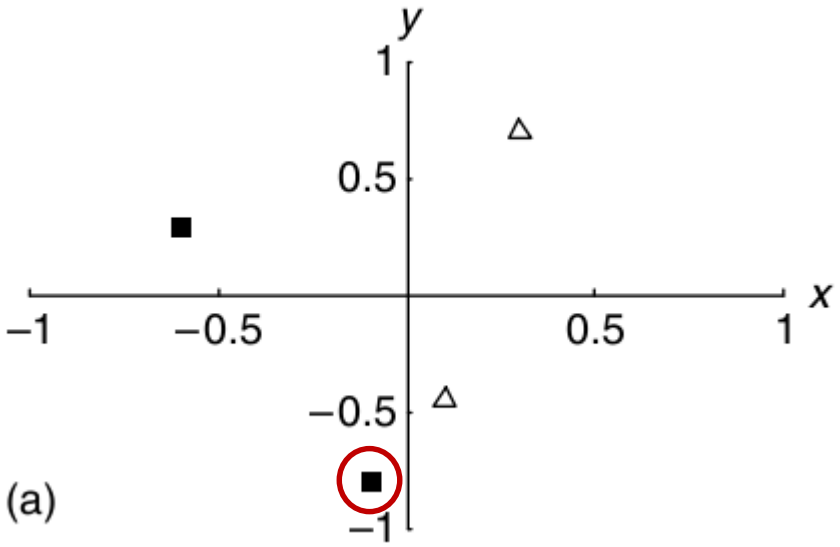


Present input pattern 3—$B2$: $x_1 = -0.1$, $x_2 = -0.8$, $t = 0$; $w_1^1 = 0.9665$, $w_2^1 = -0.1115$.

$$u = (-0.1)(0.9665) + (-0.8)(-0.1115) = -0.00745$$

$$y = u = -0.00745 < 0 \Rightarrow \text{Category } 0$$

⇨CORRECT CLASSIFICATION ⇨ WEIGHTS DO NOT CHANGE.

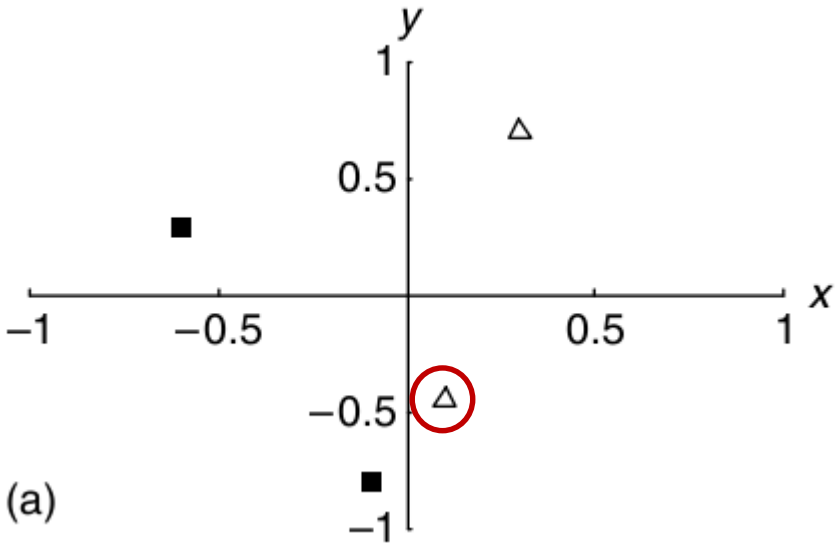


Present input pattern 4—$A2$: $x_1 = 0.1$, $x_2 = -0.45$, $t = 1$; $w_1^1 = 0.9665$, $w_2^1 = -0.1115$.

$$u = (0.1)(0.9665) + (-0.45)(-0.1115) = 0.1468$$

$$y = u = 0.1468 > 0 \Rightarrow \text{Category 1}$$

⇨CORRECT CLASSIFICATION ⇨ WEIGHTS DO NOT CHANGE.

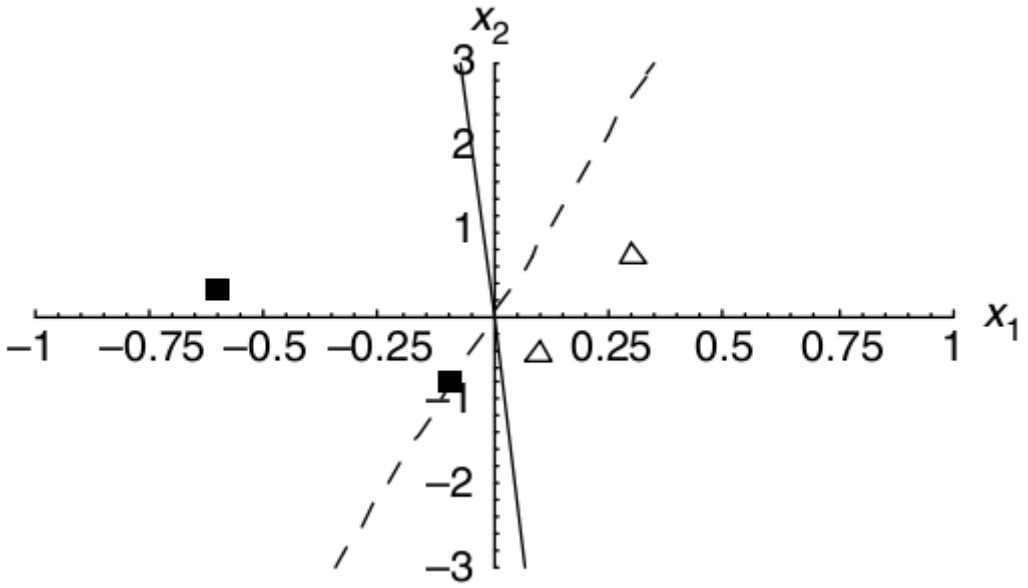


$$y = u = w_1 x_1 + w_2 x_2 = 0$$

$$x_2 = -(w_1 x_1)/w_2$$

$$x_2 = -(0.9665 x_1)/(-0.1115)$$

$$x_2 = 8.67 x_1.$$



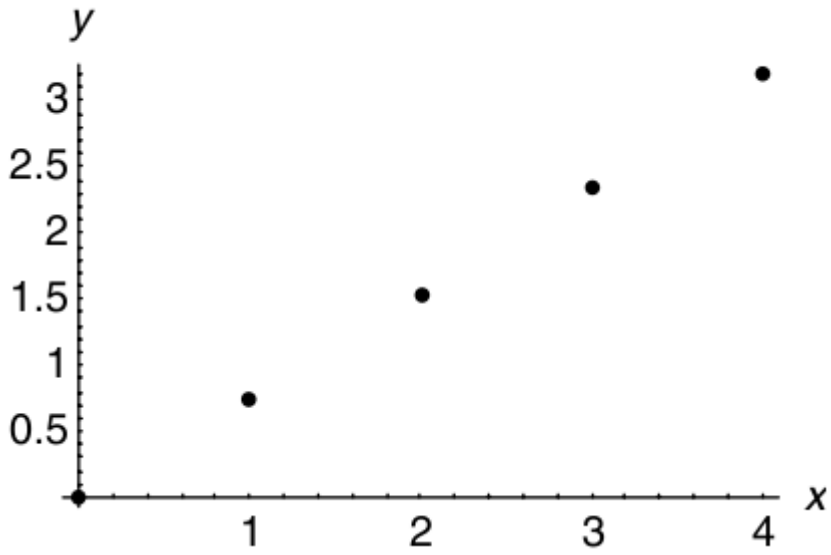
Example of linear neuron as a predictor

- Data was generated from the function y = 0.8x

$$y = wx$$

Table 2.5 Linear Function Data

x	t
0	0
1.0	0.75
2.0	1.53
3.0	2.34
4.0	3.2



Example-by-example learning

$$y = wx$$

$$E = t - y = t - wx$$

$$\Delta w = \beta Ex$$

$$w_{\text{new}} = w_{\text{old}} + \Delta w.$$

For the first input: $y = 0.5 \times 1 = 0.5$

$$E = t - y = 0.75 - 0.5 = 0.25$$

$$w^1 = w + \beta xE = 0.5 + 0.1 \times 0.25 \times 1 = 0.5 + 0.025 = 0.525$$

For the second input:

$$y = 0.525 \times 2 = 1.05$$

$$E = t - y = 1.53 - 1.05 = 0.48$$

$$w^2 = 0.525 + 0.1 \times 0.48 \times 2 = 0.525 + 0.096 = 0.621$$

For the third input:

$$y = 0.621 \times 3 = 1.863$$

$$E = 2.34 - 1.863 = 0.477$$

$$w^3 = 0.621 + 0.1 \times 0.477 \times 3 = 0.621 + 0.1431 = 0.7641$$

For the fourth input:

$$y = 0.7641 \times 4 = 3.0564$$

$$E = 3.2 - 3.0564 = 0.1436$$

$$w^4 = 0.7641 + 0.1 \times 0.1436 \times 4 = 0.7641 + 0.05744 = 0.8215$$

Batch learning

$$y = wx$$

$$E = t - y = t - wx$$

$$\Delta w = \beta E x$$

$$w_{\text{new}} = w_{\text{old}} + \Delta w.$$

If batch learning is employed, Δw is defined as $\beta\left(\frac{1}{n}\sum_{i=1}^{n} E_i x_i\right)$

For the first epoch:

$y_1 = 0.5 \times 1 = 0.5$

$E_1 = t - y_1 = 0.75 - 0.5 = 0.25$

$y_2 = 0.5 \times 2 = 1.0$

$E_2 = t - y_2 = 1.53 - 1.0 = 0.53$

$y_3 = 0.5 \times 3 = 1.5$

$E_3 = t - y_3 = 2.34 - 1.5 = 0.84$

$y_4 = 0.5 \times 4 = 2.0$

$E_4 = t - y_4 = 3.2 - 2.0 = 1.2$

$$\Delta w^1 = \beta\left(\frac{1}{n}\sum_{i=1}^{n} E_i x_i\right) = 0.1 \times \left[\frac{1}{4}\left(E_1x_1 + E_2x_2 + E_3x_3 + E_4x_4\right)\right]$$

$$= 0.1 \times \left[\frac{1}{4}(0.25 \times 1 + 0.53 \times 2 + 0.84 \times 3 + 1.2 \times 4)\right]$$

$$= 0.2158.$$

$$w^1 = w + \Delta w = 0.5 + 0.2158 = 0.7158$$

For the second epoch:

$$y_1 = 0.7158 \qquad E_1 = 0.0342$$

$$y_2 = 1.4316 \qquad E_2 = 0.0984$$

$$y_3 = 2.1474 \qquad E_3 = 0.1926$$

$$y_4 = 2.8632 \qquad E_4 = 0.3368$$

$$\Delta w^2 = 0.0539$$

$$w^2 = 0.7697.$$

For the third epoch:

$$y_1 = 0.7697 \qquad E_1 = -\,0.0197$$

$$y_2 = 1.5394 \qquad E_2 = -\,0.0094$$

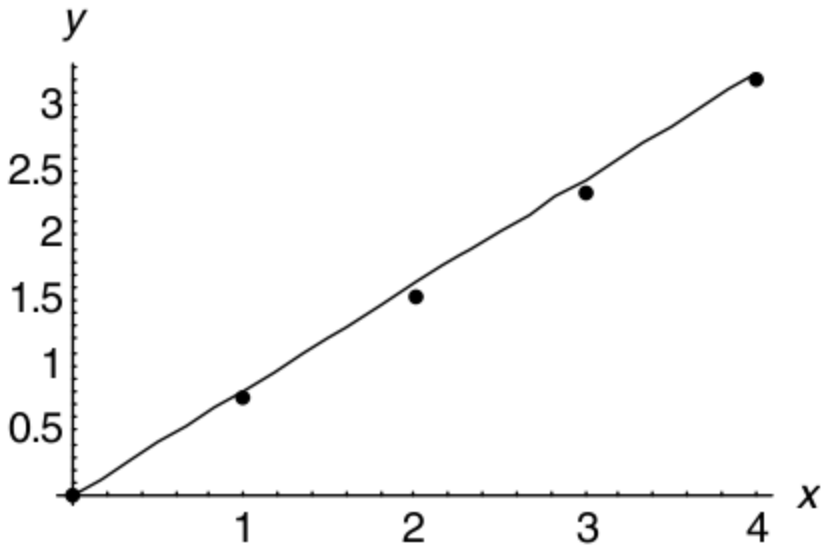
$$y_3 = 2.3091 \qquad E_3 = 0.0309$$

$$y_4 = 3.0788 \qquad E_4 = 0.1212$$

$$\Delta w^3 = 0.013475$$

$$w^3 = 0.783175.$$

$$y = 0.821x$$



- **Examine the overall prediction error in the form of MSE over all the input patterns by averaging the error**

$$MSE = \frac{1}{2n}\sum_{i=1}^{n} E_i^2$$

$$x_1 = 1,\ y_1 = 0.821,\ t_1 = 0.75,\ (t_1 - y_1)^2 = (0.75 - 0.821)^2 = 0.00504$$

$$x_2 = 2,\ y_2 = 0.821 \times 2 = 1.642,\ t_2 = 1.53,\ (t_2 - y_2)^2 = (1.53 - 1.642)^2 = 0.0125$$

$$x_3 = 3,\ y_3 = 0.821 \times 3 = 2.463,\ t_3 = 2.34,\ (t_3 - y_3)^2 = (2.34 - 2.463)^2 = 0.0151$$

$$x_4 = 4,\ y_4 = 0.821 \times 4 = 3.284,\ t_4 = 3.2,\ (t_4 - y_4)^2 = (3.2 - 3.284)^2 = 0.0071,$$

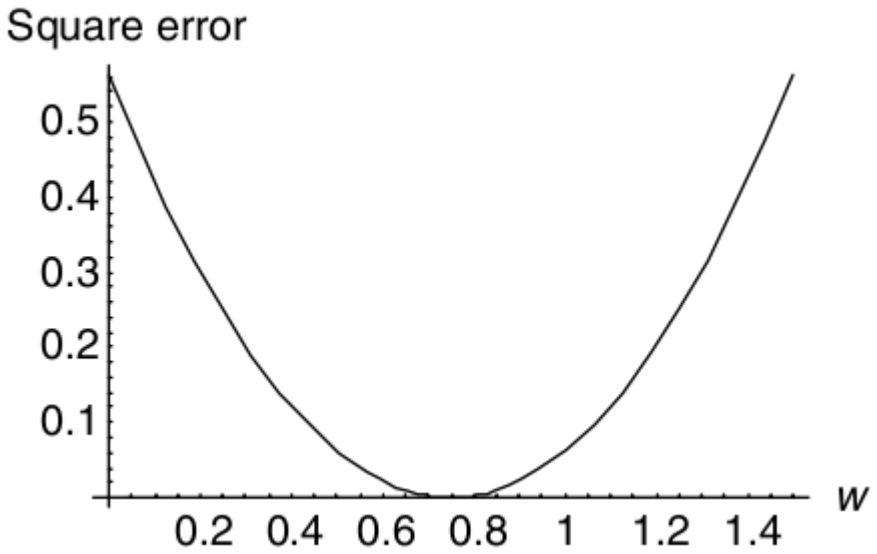
$$\text{MSE} = (0.00504 + 0.0125 + 0.0151 + 0.0071)/(2 \times 4) = 0.00497.$$

Error surface for a one-input linear neuron without bias

$$E = t - y = t - wx$$

$$\varepsilon = E^2 = (t - wx)^2.$$

$$\varepsilon = (0.75 - w)^2$$

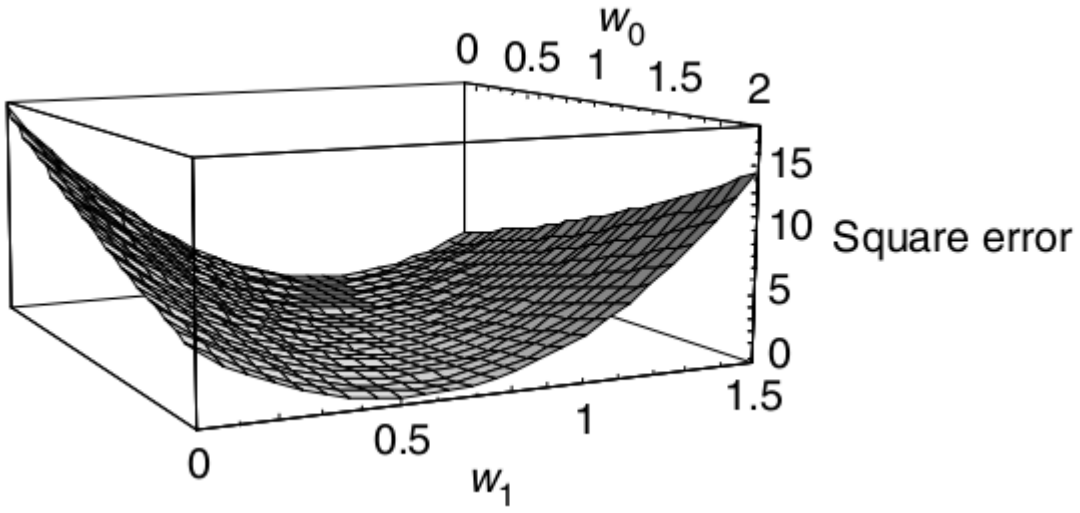


Error surface for a one-input linear neuron with bias

$$y = w_0 + w_1 x.$$

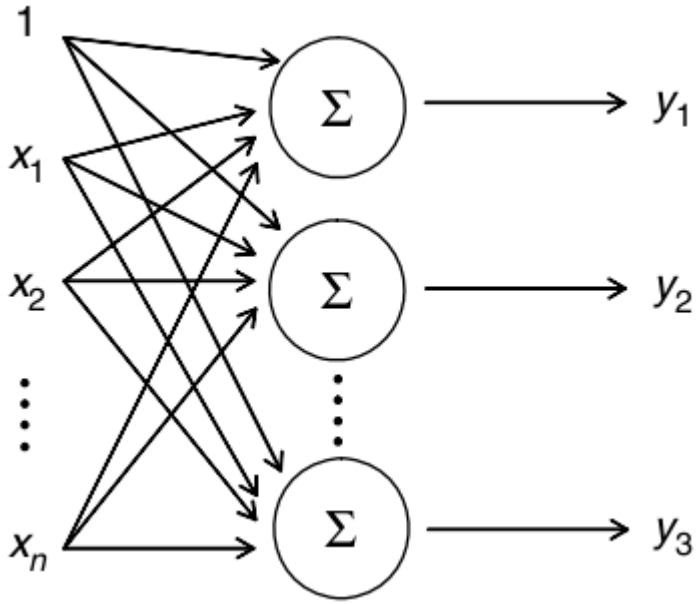
$$\varepsilon = E^2 = [t - (w_0 + w_1 x)]^2$$

$$\varepsilon = E^2 = [4.2 - (w_0 + 4w_1)]^2$$



Multiple Linear Neuron Models

$$y_i = b_0 + w_{i1}x_1 + w_{i2}x_2 + \cdots + w_{in}x_n$$



InfoLab

Neural Networks for Nonlinear Pattern Recognition

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

Methods for nonlinear analysis

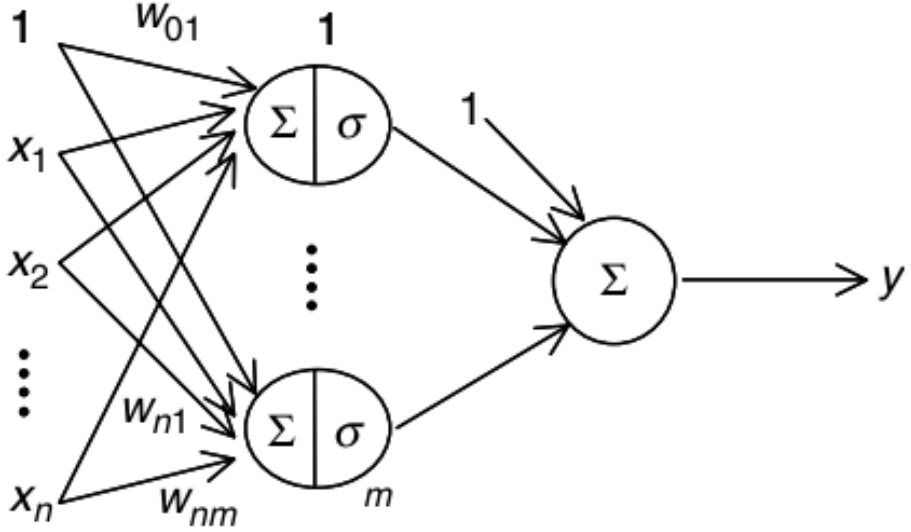
- **Multilayer perceptron (MLP) networks**
  - similar to RBF networks in that the intermediate processing is done by one or more hidden layers with nonlinear activation functions
  - RBFs use Gaussian functions as activation functions
  - MLPs use a range of activation functions
- **Radial basis function (RBF) networks**
- **Support vector machines (SVMs)**
- **Generalized model for data handling (GMDH)**
- **Generalized regression neural network (GRNN)**
- **Generalized neural network (GNN)**
  - one hidden neuron: perform linear analysis
  - the other hidden neuron: perform nonlinear analysis

MLP network

- **It is a powerful extension of the perceptron**
- **Founding concepts and studies were developed in the 1980s**
- **It is a universal approximator**
  - due to their ability to approximate any nonlinear relationship between inputs and outputs to any degree of accuracy
- **The power comes from the hidden layer of neurons**
- **The hidden layer consists of one or many nonlinear neurons**

Layout of an MLP network

- **One or many hidden neurons**
  - because they are not exposed to the external environment (data)
- **Each hidden neurons passes its weighted sum through a nonlinear transfer function (denoted by σ)**
- **Many choices for the neuron activation (transfer) function**



Learning (training) in MLPs

- **Network output is compared with the desired output**
- **If the absolute difference (i.e., error) is larger than an acceptable threshold**
- **The error is backpropagated through the network**
- **This process adjusts**
  - the weights between the input and hidden layers and
  - those between the hidden and output layers

  using an appropriate learning method

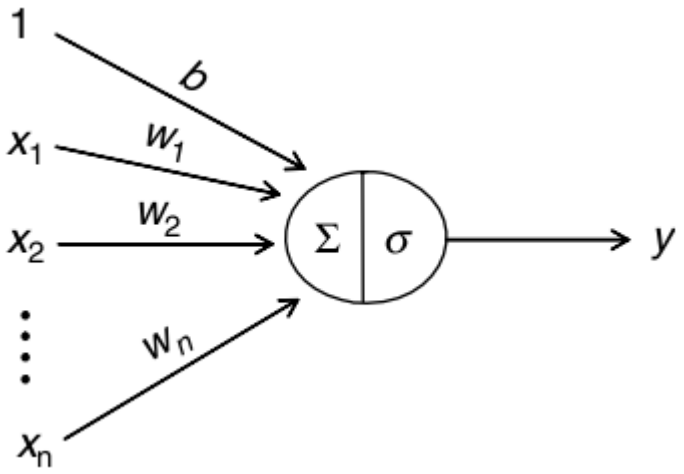
Complexity of MLPs

- **For simple nonlinear problems, one or few hidden neurons may be sufficient**

- **However, for highly nonlinear problems involving many input variables, a larger number of neurons may be necessary**
  - to correctly approximate the desired input–output relationship

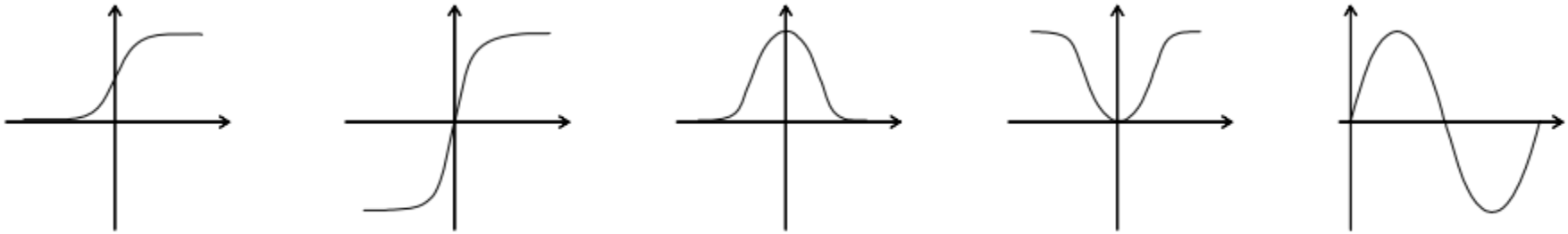
Nonlinear Neurons

- **MLP derives its power from nonlinear processing in the hidden neurons**
- **Output of neuron :** $$\sigma\left(\sum_{j=1}^{n} w_j x_j + b\right)$$
- **Most widely used function for σ : sigmoid**
  - a family of curves that includes logistic and hyperbolic tangent functions



Neuron Activation Functions

- **Nonlinear**
  - the output of the function varies nonlinearly with the input
- **Continuous**
  - there are no sharp peaks or gaps in the function
  - they can be differentiated throughout
  - it is possible to implement the delta rule
- **Bounded**
  - output never reaches very large values, regardless of the input
  - it mimics the biological neurons



(a) logistic (sigmoid)

(b) hyperbolic-tangent (sigmoid)

(c) Gaussian

(d) Gaussian complement

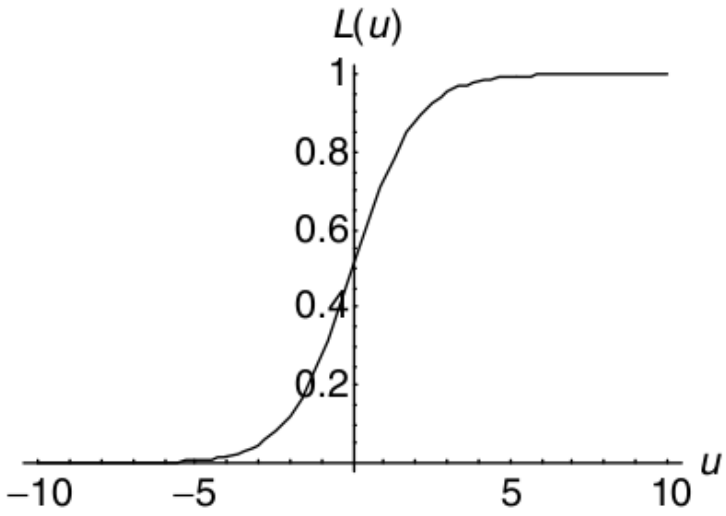
(e) sine

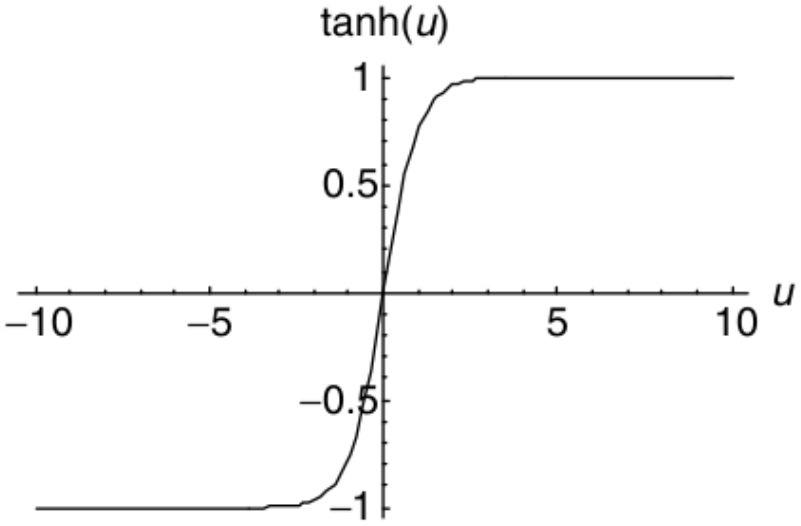
Sigmoid functions

- **Family of S-shaped functions**
- **Logistic function : range [0, 1]**
- **Hyperbolic tangent function : range [-1,1]**
  - the slope is higher than that of the logistic function (at u=0)
  - it reaches the bounds more quickly than the logistic function

$$y = L(u) = \frac{1}{1 + e^{-u}}$$

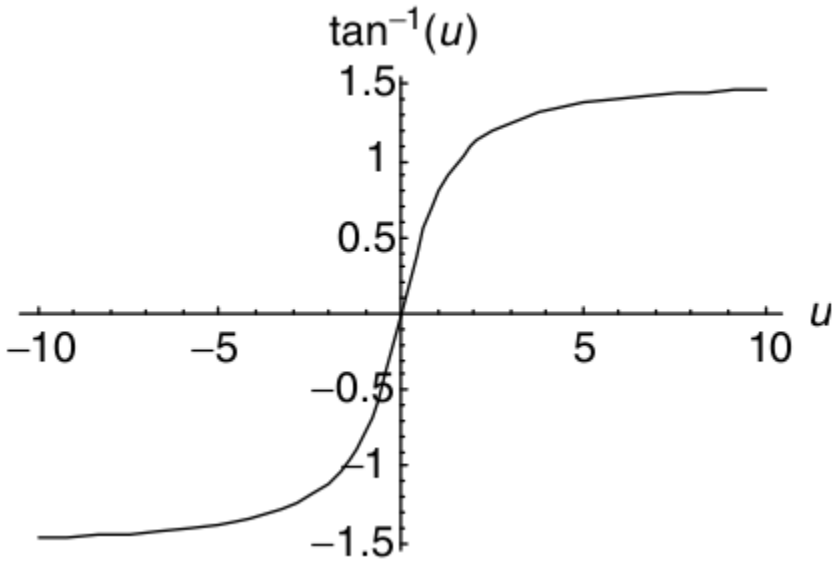
$$\tanh(u) = \frac{1 + e^{-u}}{1 - e^{-u}}$$





Inverse tan (tan-1 or arctan) functions

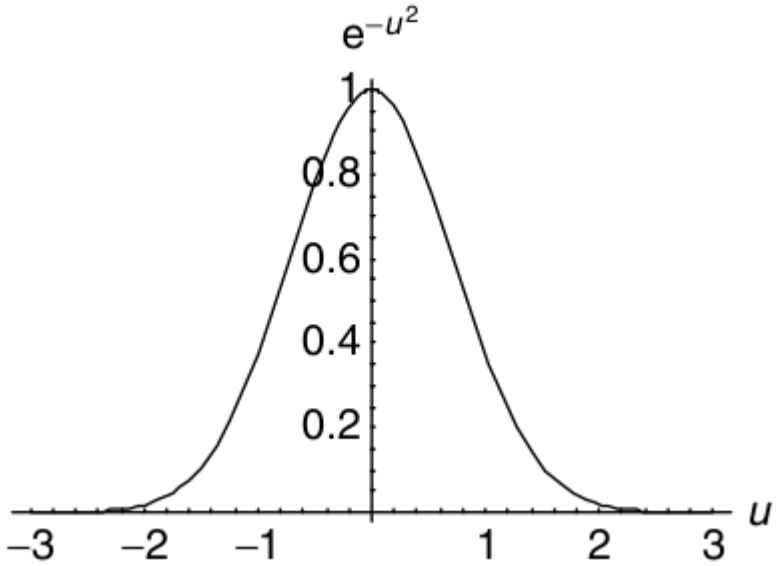
- a more gradual variation than the above two functions
- a slope at boundary in between those of the logistic and hyperbolic tangent functions



Gaussian functions

- **Standard normal curve : range [0,1]**
  - highly sensitive to u values around zero
  - almost insensitive to those at the tails
  - amplify the mid-range of the input distribution
  - more sensitive to the weighted inputs that are close to zero when used in a neuron

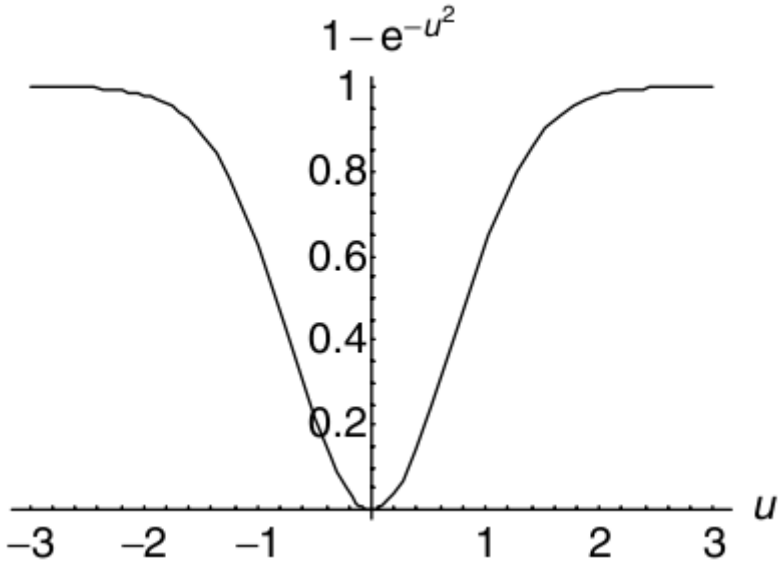
$$y = e^{-u^2}$$



Gaussian complement : range [0, 1]

- peaks at the tails and a value of zero when u=0
- a larger output for the inputs at the upper and lower ends
- more sensitive to the weighted inputs that are at the two extreme ends, when used in a neuron

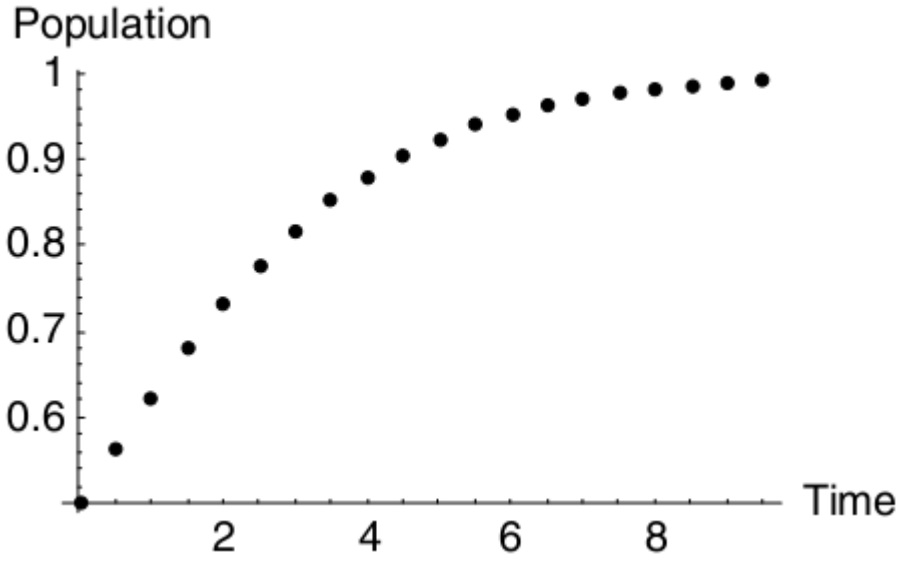
$$y = 1 - e^{-u^2}$$

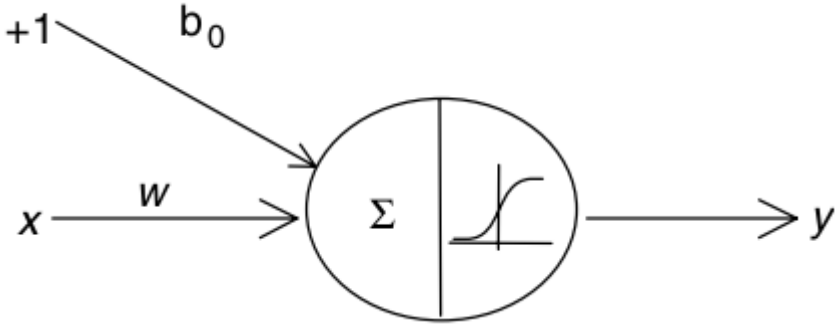


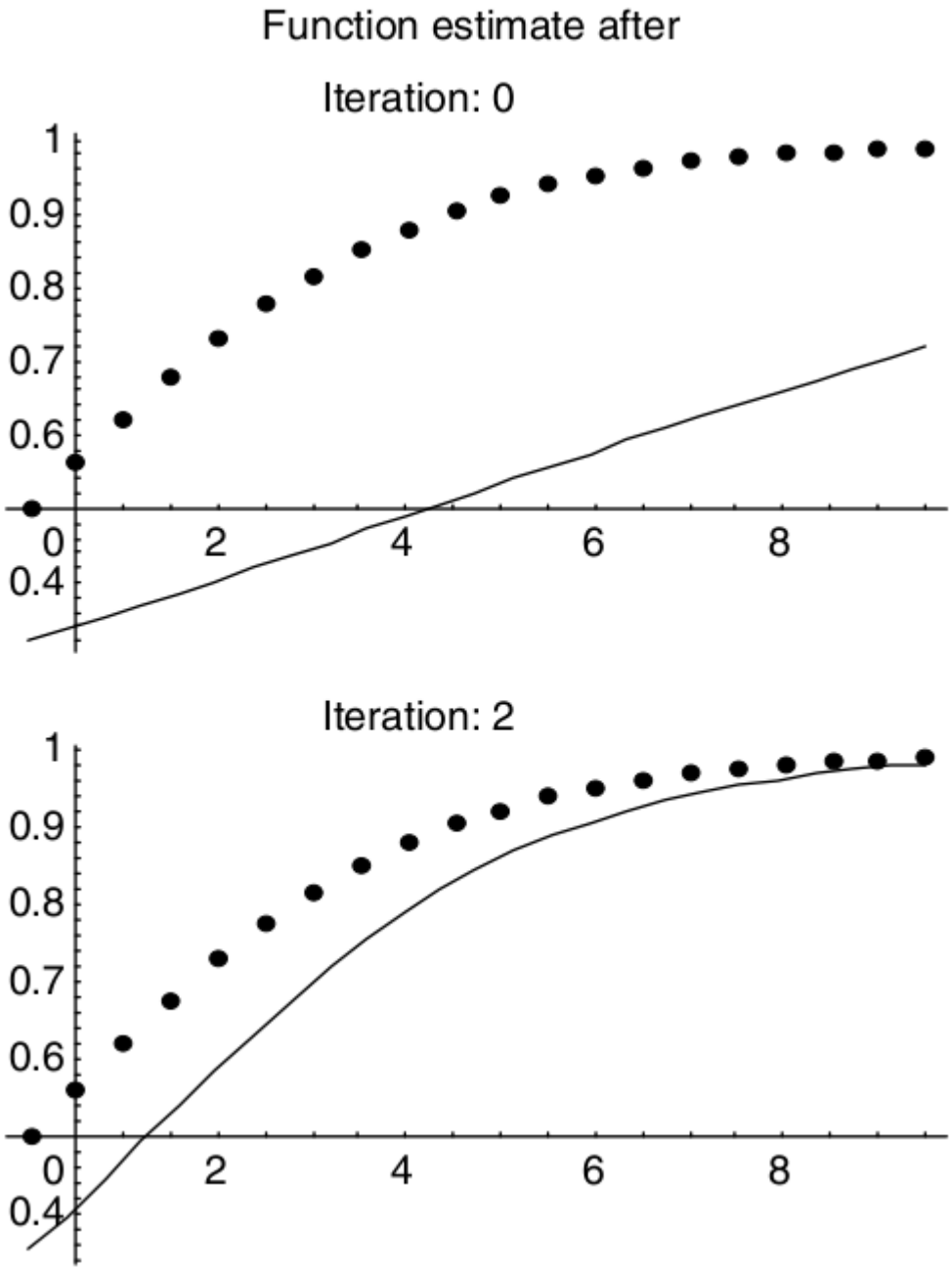
Example: Population Growth Modeling Using a Nonlinear Neuron

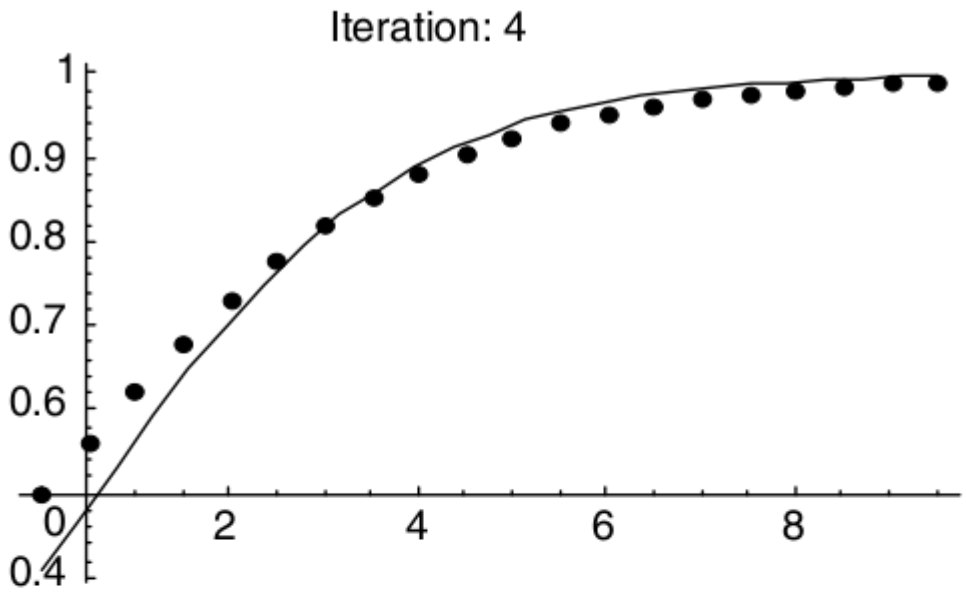
$$y = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n)}}$$

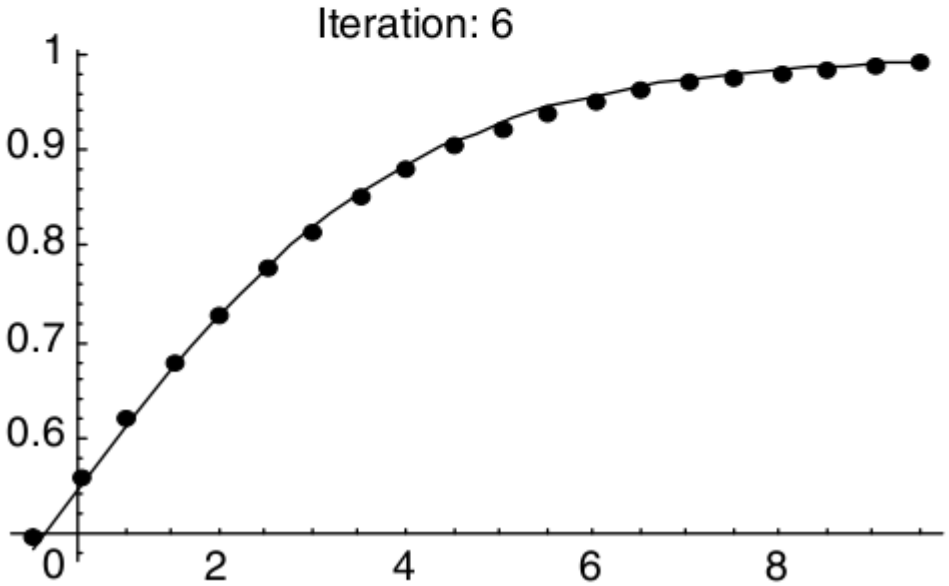
$$y = \frac{1}{1 + e^{-0.5x}}$$





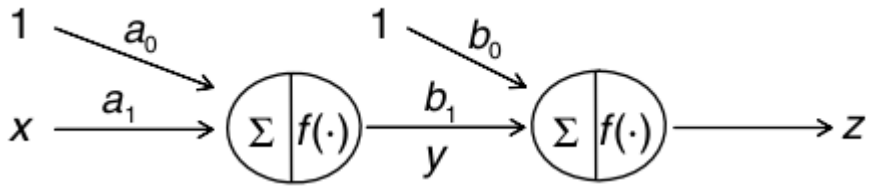

Function estimate after
Iteration: 0
1
0.9
0.8
0.7
0.6
0.4
0
2
4
6
8
Iteration: 2
1
0.9
0.8
0.7
0.6
0.4
0
2
4
6
8


Iteration: 4
1
0.9
0.8
0.7
0.6
0.4
0
2
4
6
8

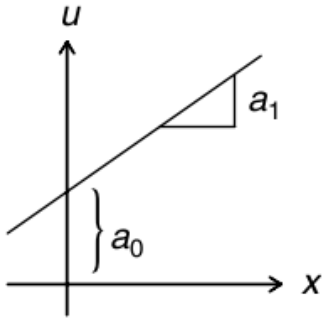

Iteration: 6
1
0.9
0.8
0.7
0.6
0
2
4
6
8

One-Input Multilayer Nonlinear Networks

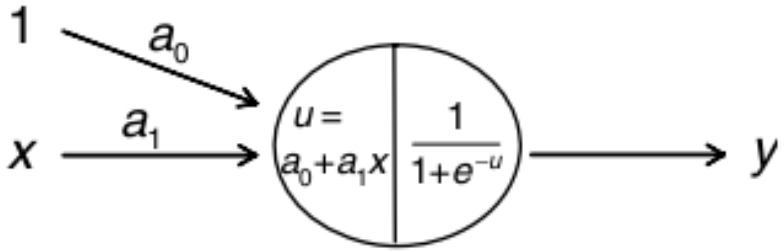
- **Processing with a Single Nonlinear Hidden Neuron**



$$u = a_0 + a_1 x$$



$$y = \frac{1}{1 + e^{-(a_0 + a_1 x)}}$$



Output of logistic function

case 1: $a_0 = 0, \ a_1 = 1$

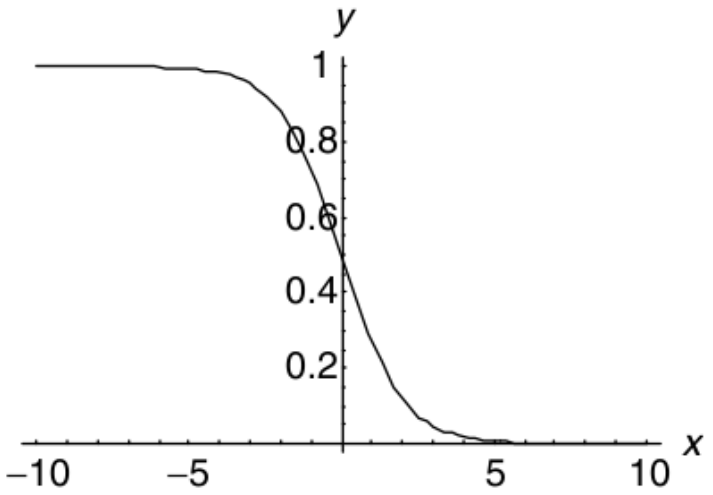
$$y = \frac{1}{1 + e^{-x}}$$ no bias input

$$x = 0, y = 0.5$$

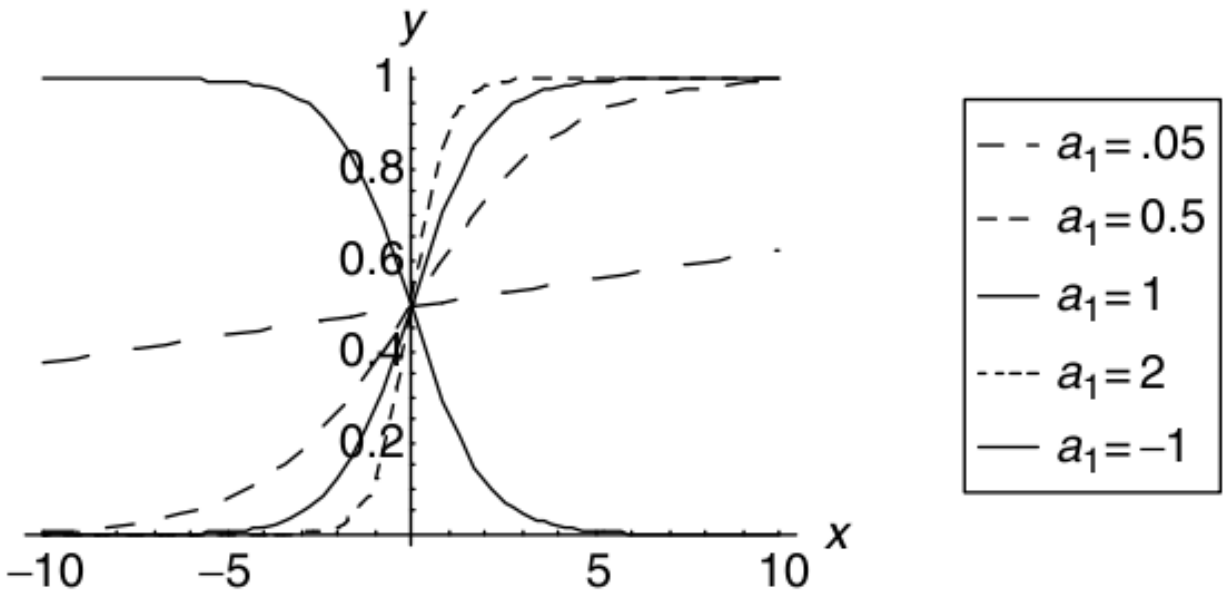
---

case 2: $a_0 = 0 \ and \ a_1 = -1$

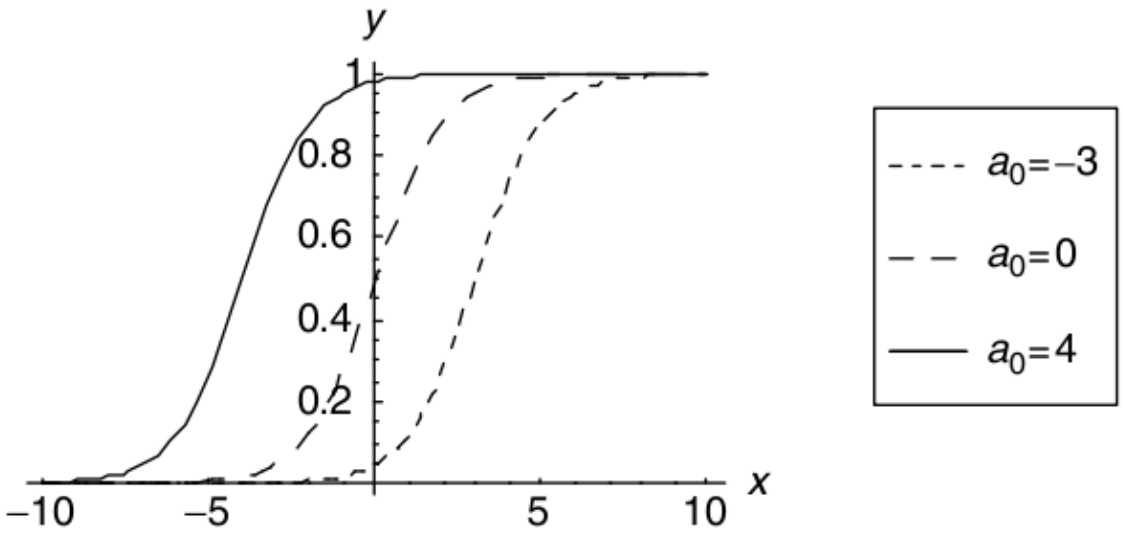
$$y = \frac{1}{1 + e^{x}}$$



case 3: $a_0 = 0,\ a_1\ varies\ from\ -1\ to\ 2$

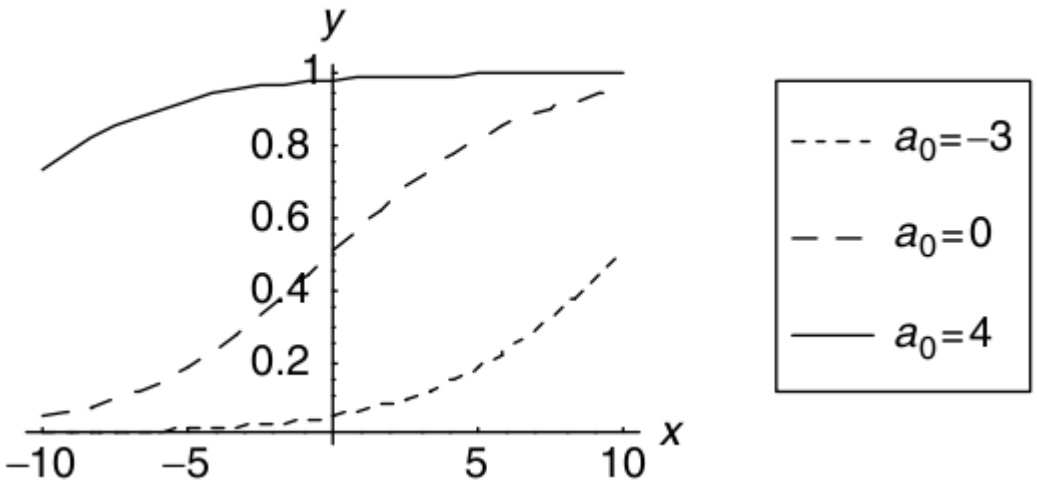


case 4: $a_1 = 1,\ a_0 = -3,\ 0,\ 4$



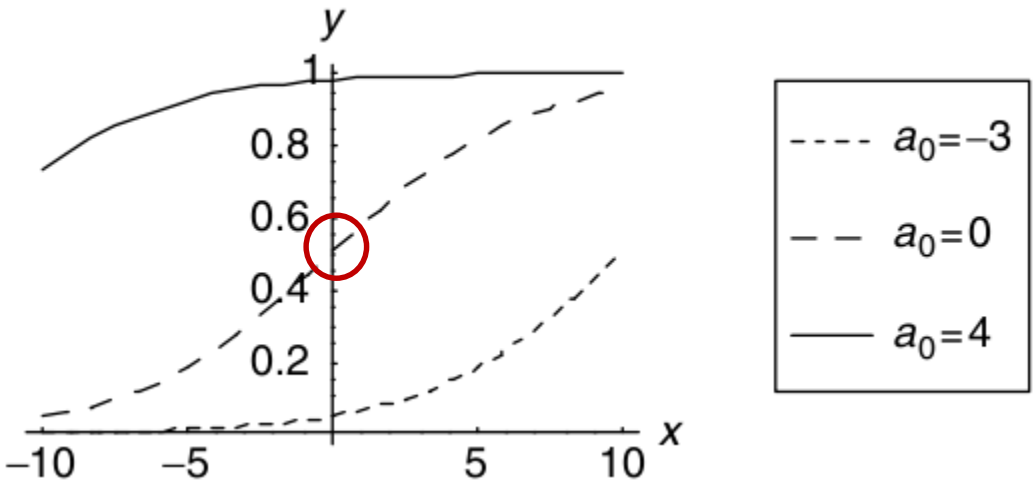
---

case 5: $a_1 = 0.3,\ a_0 = -3,\ 0,\ 4$



Boundary point

- **The most active point on the curves (i.e., where u = 0)**



$$u = a_0 + a_1 x, \qquad a_0 + a_1 x' = 0, \qquad x' = -a_0/a_1$$

the boundary point for $a_0 = -3$ is $-(-3)/0.3 = 10$

for $a_0 = 4$ is $(-4)/0.3 = -13.3$

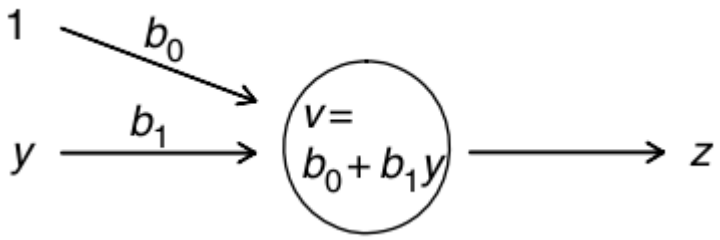
MLP networks approximating any function

- The capability of MLP is greatly enhanced by adding more neurons that act in parallel
- Each neuron processes information in a similar fashion
- But, they begin using logistic functions with different slopes and positions
- During training, each of these undergoes transformations in shape and position
- Eventually, they collectively approximate any desired function

Output of the network

- **The output is linear with respect to v**
- **But, it is still nonlinear with respect to the original input**
  - due to the nonlinear processing in the hidden neuron

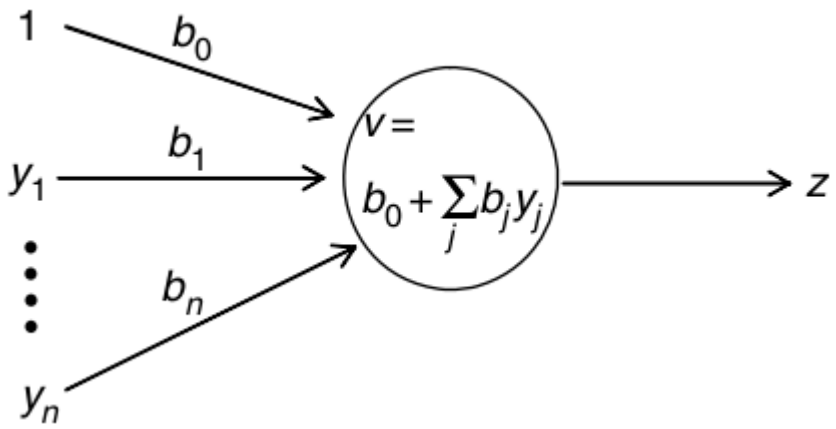
$$v = b_0 + b_1 y$$
$$z = v,$$



- **In general, there is more than one hidden neurons**

$$v = b_0 + \sum_{j=1}^{n} b_j y_j$$

$$z = v,$$

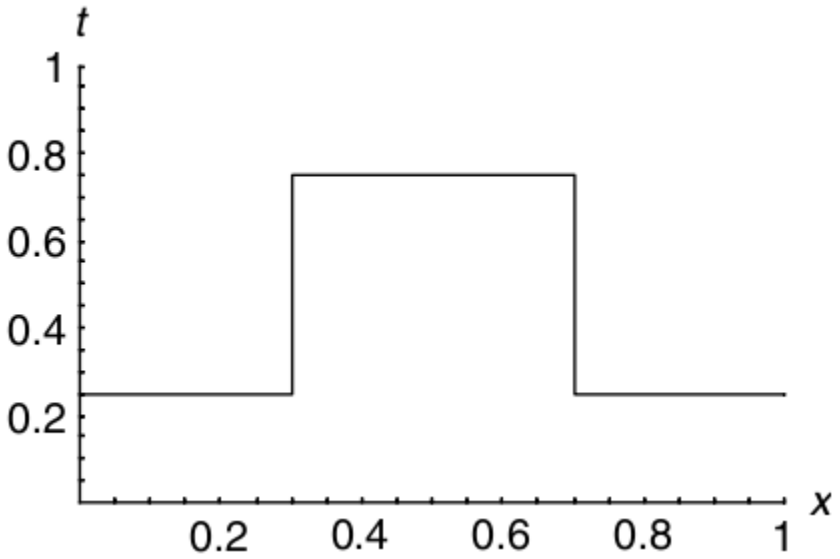


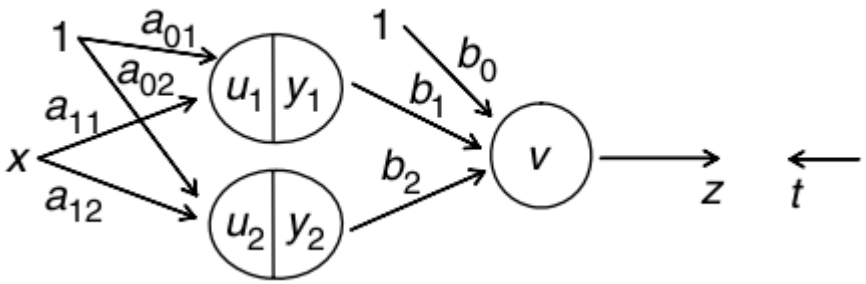
- **For classification problems, it is more appropriate to use a logistic activation function**

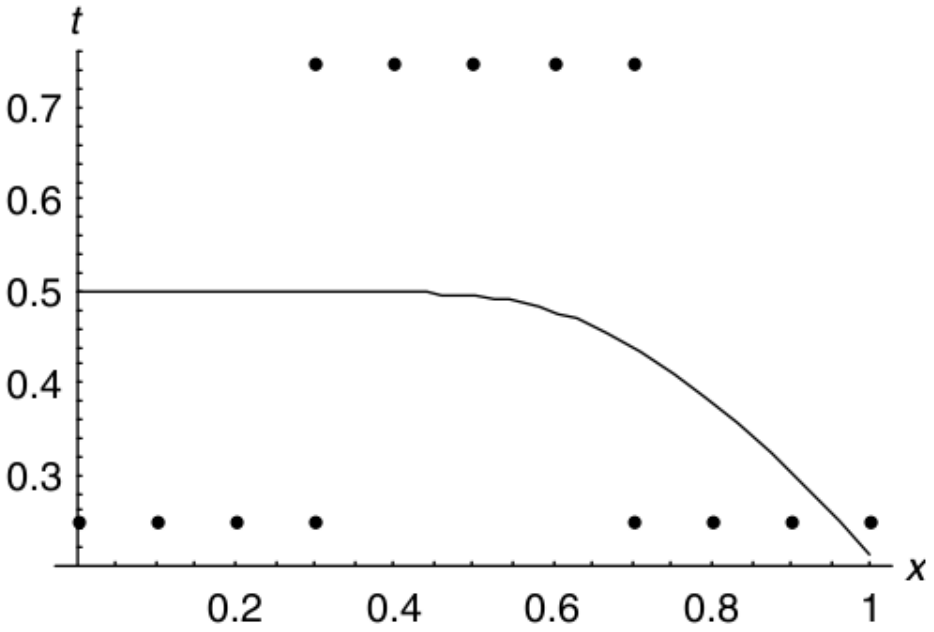
$$z = \frac{1}{1 + e^{-v}}$$

Examples: Modeling Cyclical Phenomena with Multiple Nonlinear Neurons

- **Example 1: Approximating a Square Wave**







after 15 epochs

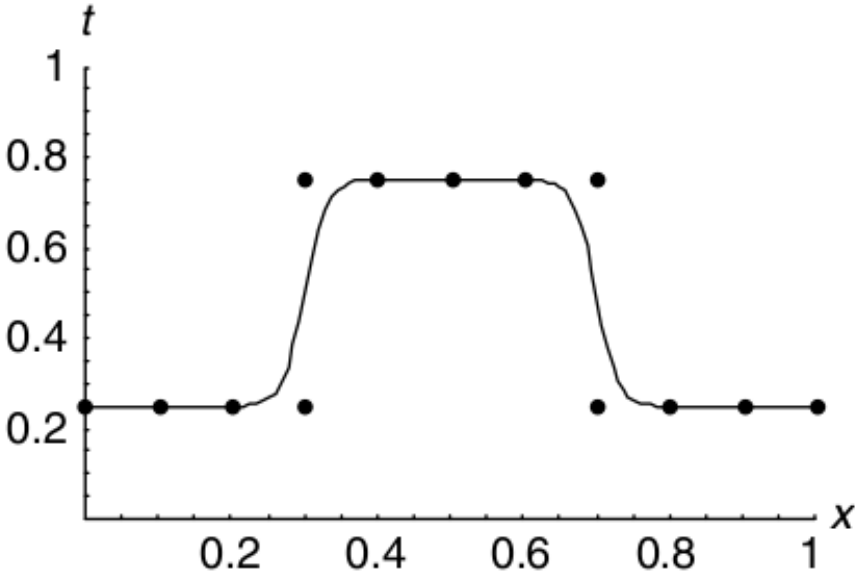


Table 3.2 Weights for the Two-Neuron Nonlinear Model Approximating Square Wave

a_{01}	a_{02}	a_{11}	a_{12}	b_0	b_1	b_2
20.8	47.6	−69	−68	0.25	−0.5	0.5

1. $x = 0,\ t = 0.25$

first hidden neuron

$$u_1 = a_{01} + a_{11}x = -20.8 + 0 = 20.8$$

$$y_1 = \frac{1}{1 + e^{-u_1}} = \frac{1}{1 + e^{-20.8}} = 0.999.$$

second hidden neuron

$$u_2 = a_{02} + a_{12}x = 47.6 + 0 = 47.6$$

$$y_2 = \frac{1}{1 + e^{-u_2}} = \frac{1}{1 + e^{-47.6}} = 1.$$

$$v = b_0 + b_1y_1 + b_2y_2 = 0.25 - (0.5 \times 0.999) + (0.5 \times 1) = 0.25$$

$$z = v = 0.25.$$

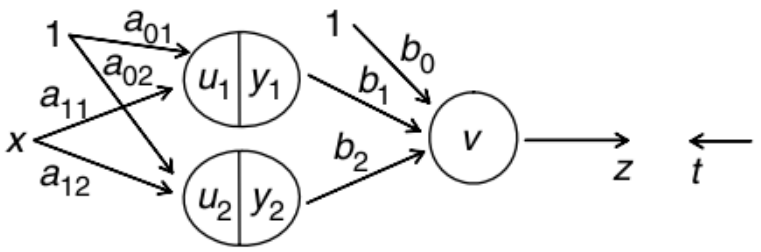


Table 3.2 Weights for the Two-Neuron Nonlinear Model Approximating Square Wave

a_{01}	a_{02}	a_{11}	a_{12}	b_0	b_1	b_2
20.8	47.6	−69	−68	0.25	−0.5	0.5

2. $x = 0.5,\ t = 0.75$

first hidden neuron

$$u_1 = a_{01} + a_{11}x = 20.8 - (69 \times 0.5) = -13.7$$

$$y_1 = \frac{1}{1 + e^{-u_1}} = \frac{1}{1 + e^{-(-13.7)}} = 1.122 \times 10^{-6}.$$

second hidden neuron

$$u_2 = a_{02} + a_{12}x = 47.6 - 68 \times 0.5 = 13.6$$

$$y_2 = \frac{1}{1 + e^{-u_2}} = \frac{1}{1 + e^{-13.6}} = 0.999.$$

$$v = b_0 + b_1 y_1 + b_2 y_2 = 0.25 - 0.5 \times 1.122 \times 10^{-6} + 0.5 \times 0.999 = 0.75$$

$$z = v = 0.75.$$

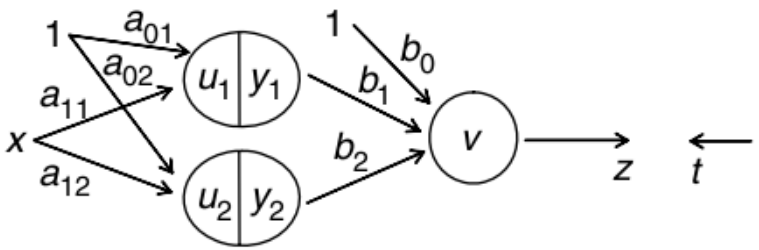


Table 3.2 Weights for the Two-Neuron Nonlinear Model Approximating Square Wave

a_{01}	a_{02}	a_{11}	a_{12}	b_0	b_1	b_2
20.8	47.6	−69	−68	0.25	−0.5	0.5

Hidden neuron outputs in relation to the input

$$u_1 = a_{01} + a_{11}x \qquad y_1 = 1/(1 + e^{-(u_1)})$$

$$y_1 = \frac{1}{1 + e^{-(a_{01} + a_{11}x)}}$$

$$u_2 = a_{02} + a_{12}x \qquad y_2 = 1/(1 + e^{-(u_2)})$$

$$y_2 = \frac{1}{1 + e^{-(a_{02} + a_{12}x)}}$$

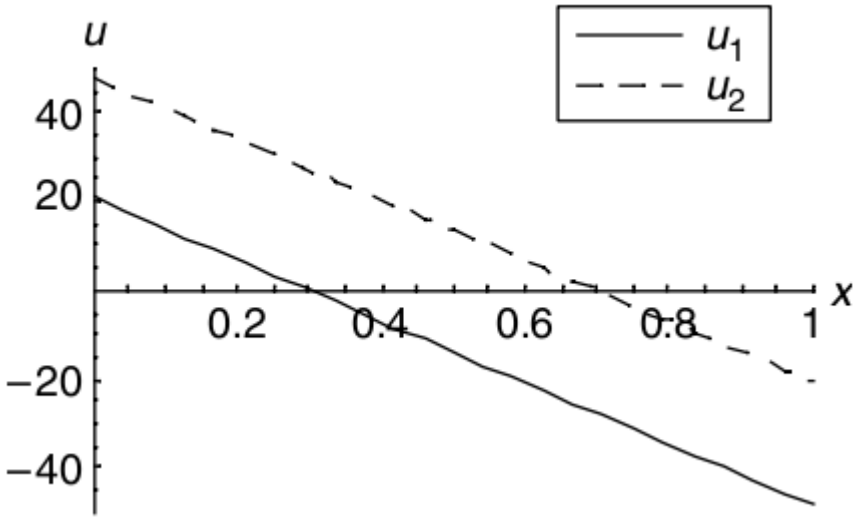
Network outputs in relation to the input

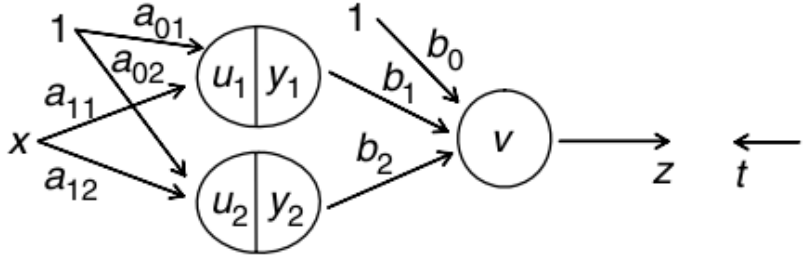
$$z = \frac{1}{1 + e^{-v}} = \frac{1}{1 + e^{-\left(b_0 + \sum_{j=1}^{2} b_j y_j\right)}}$$

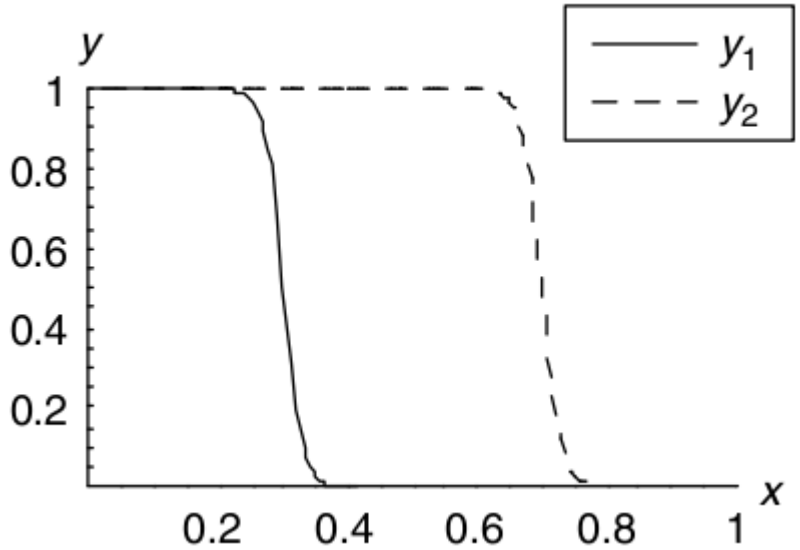
$$z = \frac{1}{1 + e^{-\left(b_0 + \sum_{j=1}^{2} b_j (1/1 + e^{-(a_{0j} + a_{1j} x)})\right)}}$$

$$= \frac{1}{1 + e^{-(b_0 + [b_1 (1/1 + e^{-(a_{01} + a_{11} x)}) + b_2 (1/1 + e^{-(a_{02} + a_{12} x)})])}}$$

$$z = \frac{1}{1 + e^{-(0.25 + [(-0.5)(1/1 + e^{-(20.8 + (-69)x)}) + 0.5(1/1 + e^{-(47.6 + (-68)x)})])}}$$


u
40
20
−20
−40
0.2
0.4
0.6
0.8
1
x
u_1
u_2

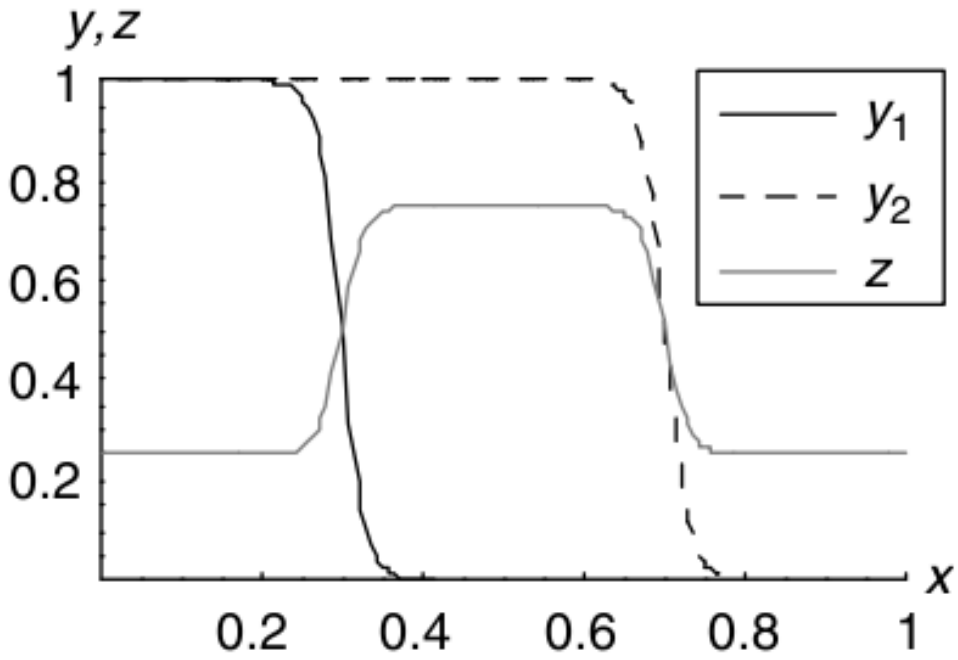

1
a_{01}
a_{02}
a_{11}
a_{12}
x
u_1 y_1
u_2 y_2
1
b_0
b_1
b_2
v
z
t

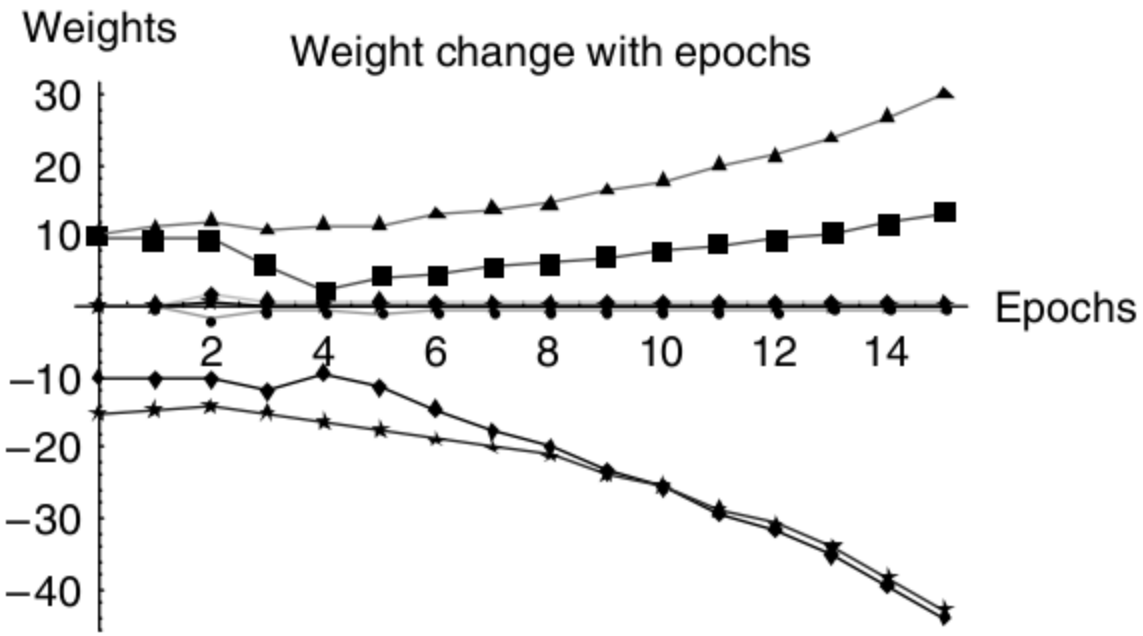

y
1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1
x
y_1
y_2

$$z = b_0 + b_1 y_1 + b_2 y_2 = 0.25 - 0.5 y_1 + 0.5 y_2$$

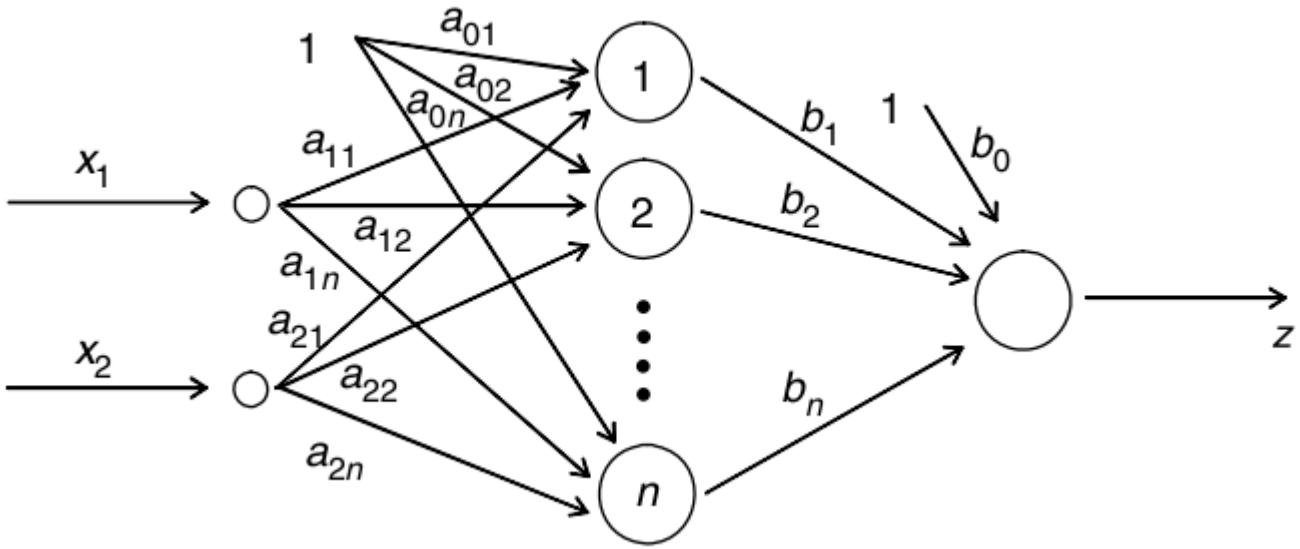
Table 3.2 Weights for the Two-Neuron Nonlinear Model Approximating Square Wave

a_{01}	a_{02}	a_{11}	a_{12}	b_0	b_1	b_2
20.8	47.6	−69	−68	0.25	−0.5	0.5

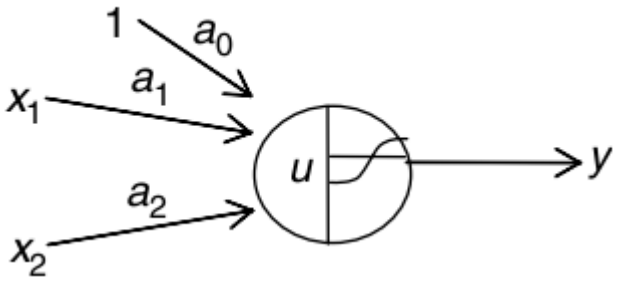



Weights
Weight change with epochs
30
20
10
−10
−20
−30
−40
2
4
6
8
10
12
14
Epochs

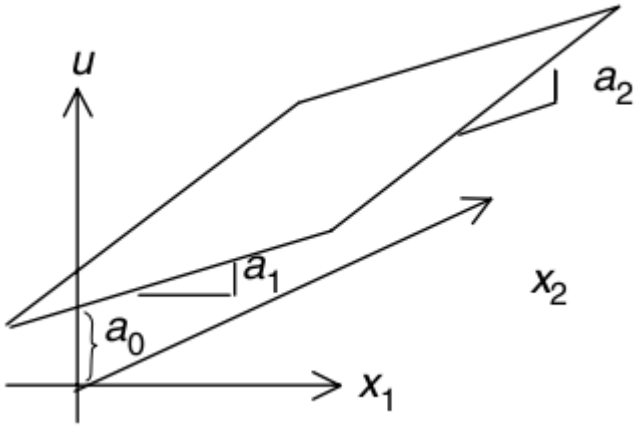
Two-Input Multilayer Perceptron Network



- **One hidden neuron of the two-dimensional nonlinear network**

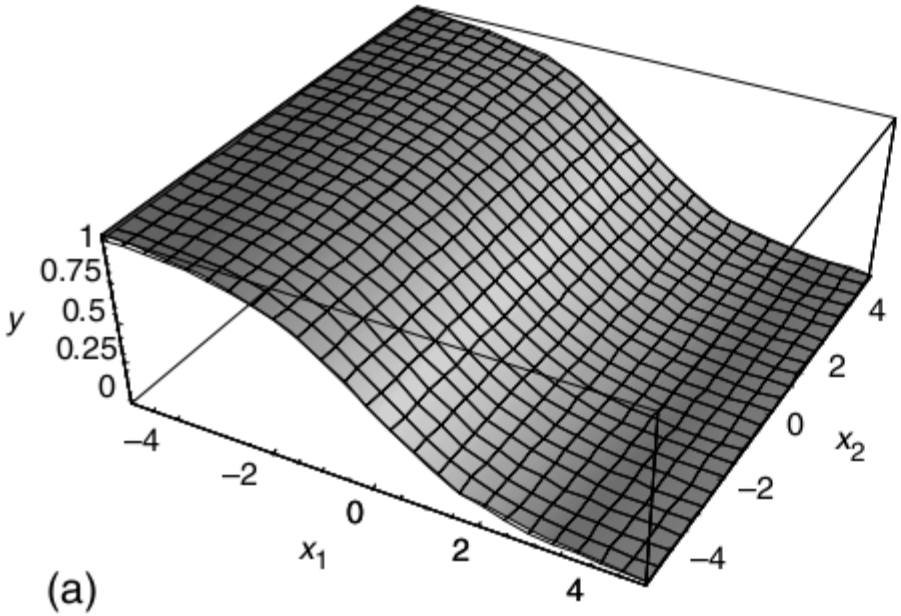


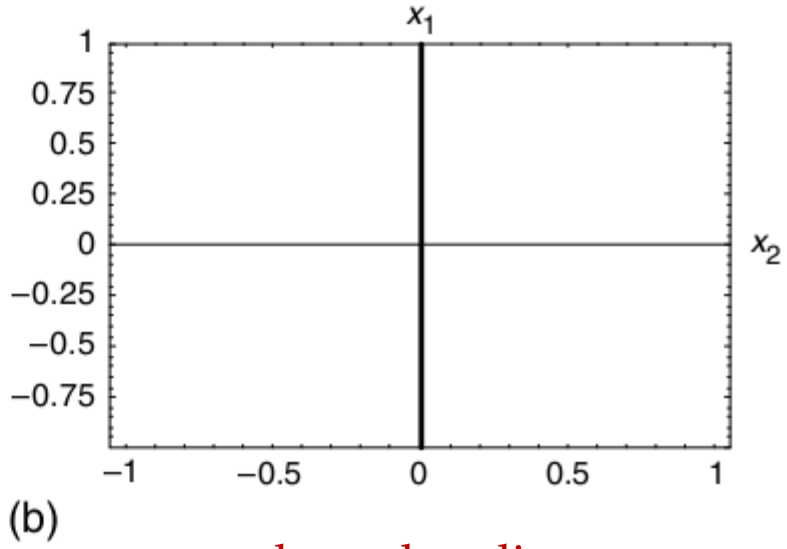
$$u = a_0 + a_1 x_1 + a_2 x_2$$



$$y = \frac{1}{1 + e^{-u}} \qquad y = \frac{1}{1 + e^{-(a_0 + a_1 x_1 + a_2 x_2)}}$$

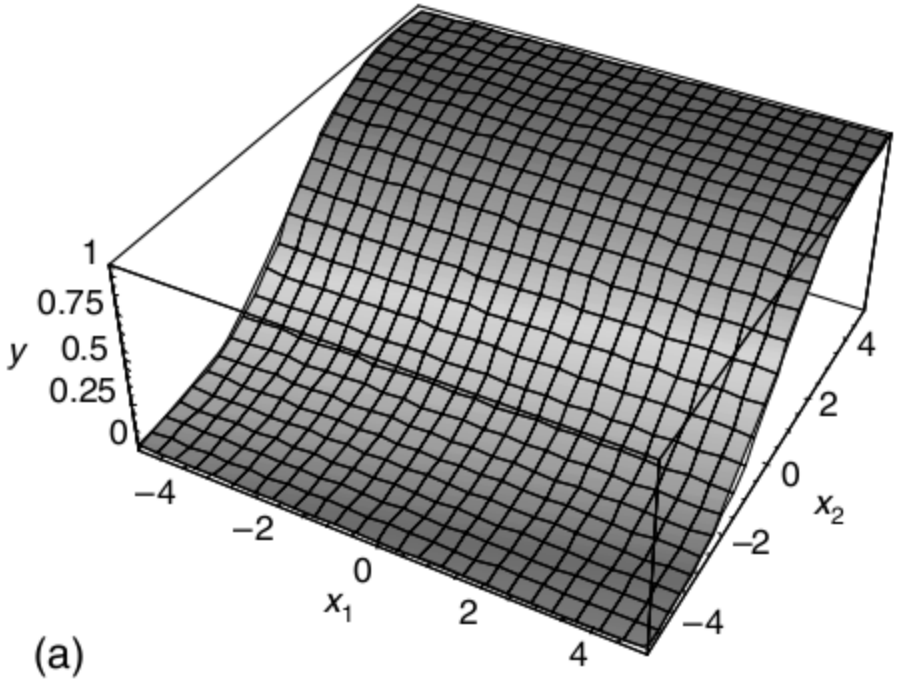
1. $a_0 = 0,\ a_1 = 1,\ a_2 = 0$ $\qquad u = a_0 + a_1 x_1 + a_2 x_2 = 0$

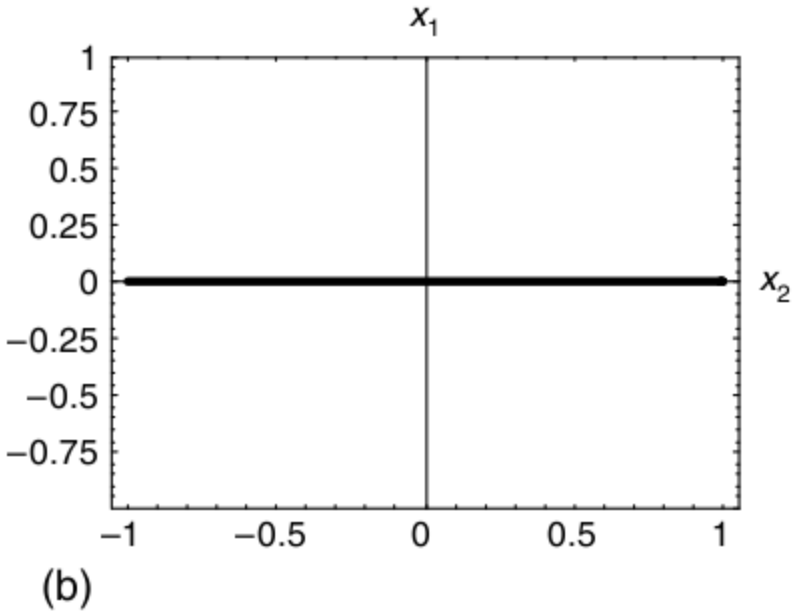




boundary line

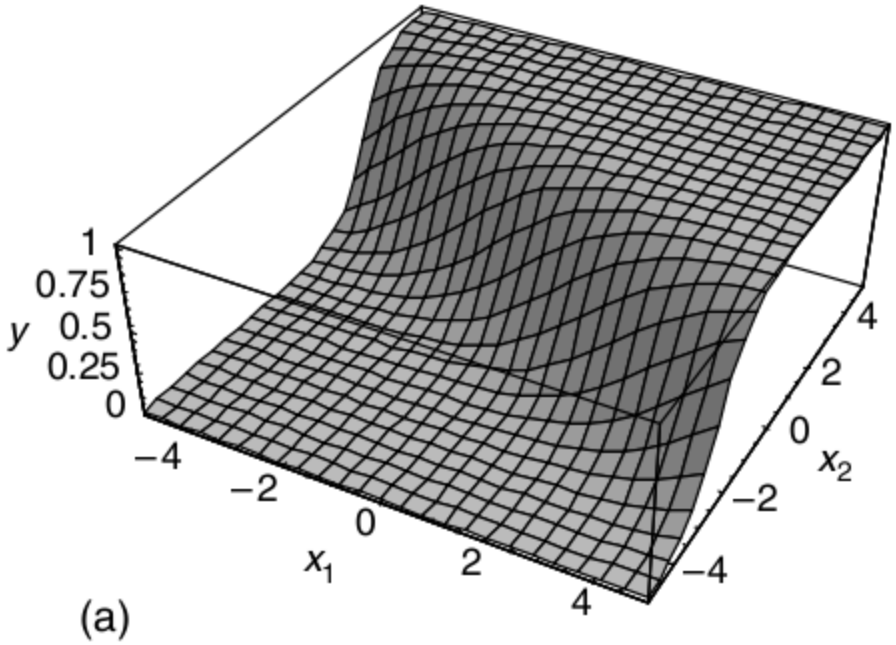
2. $a_0 = 0,\ a_1 = 0,\ a_2 = 1$

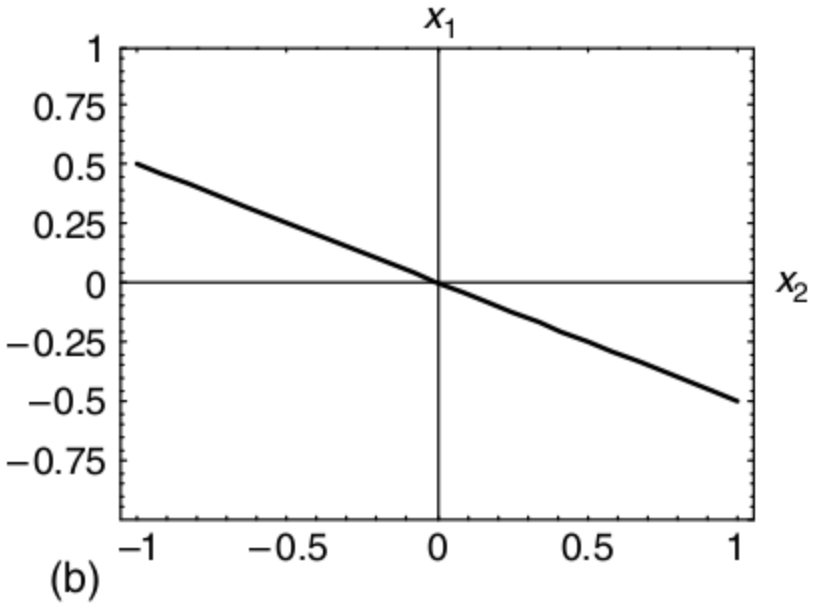




boundary line

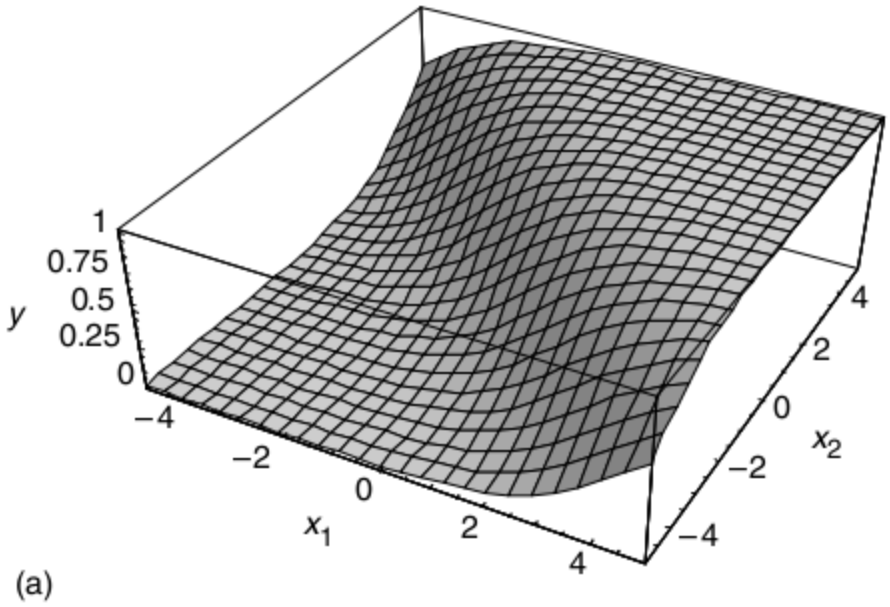
3. $a_0 = 0,\ a_1 = 1,\ a_2 = 2$



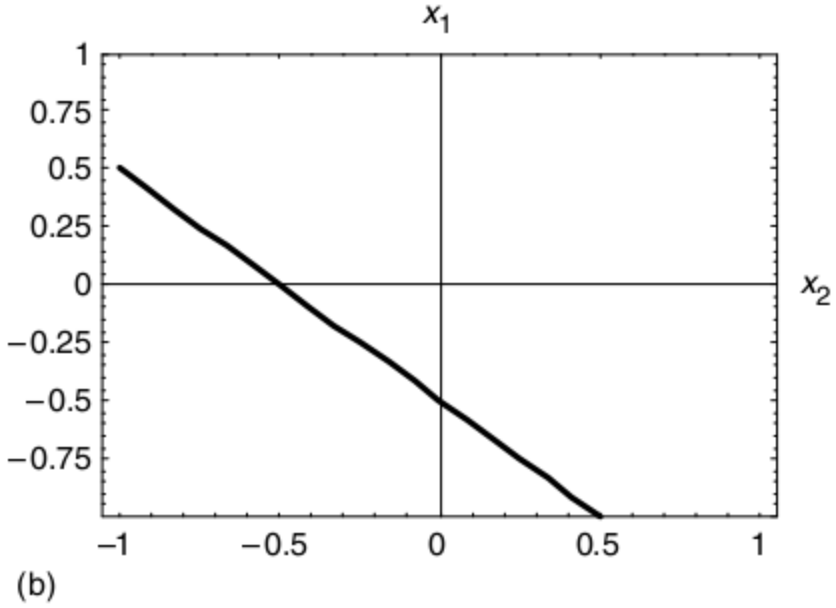


boundary line

4. $a_0 = -0.5,\ a_1 = 1,\ a_2 = -1$

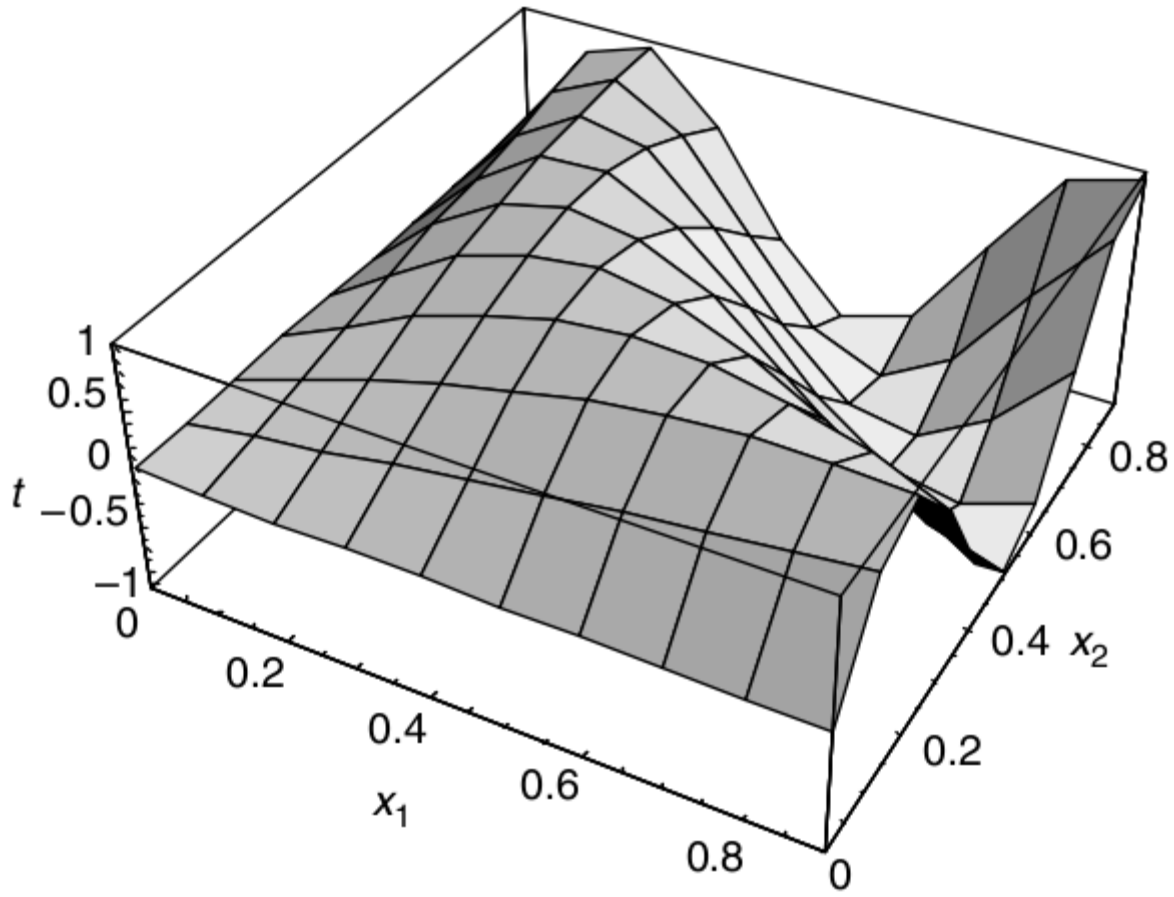


(a)

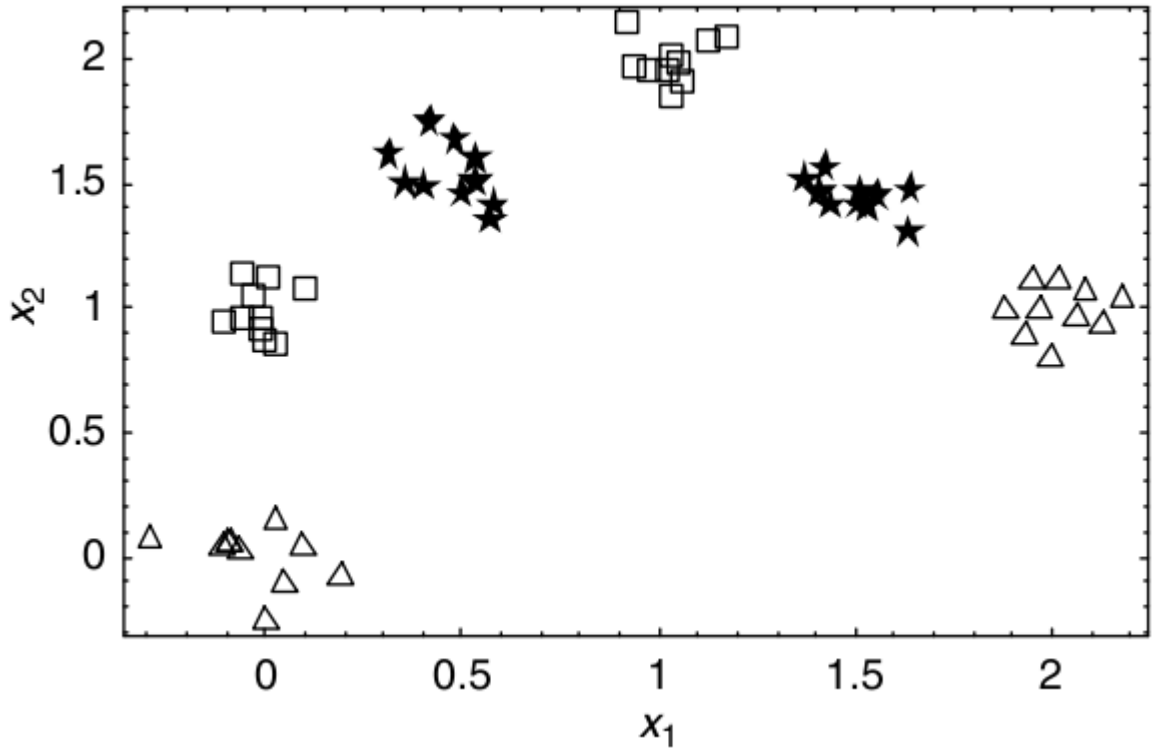


(b)

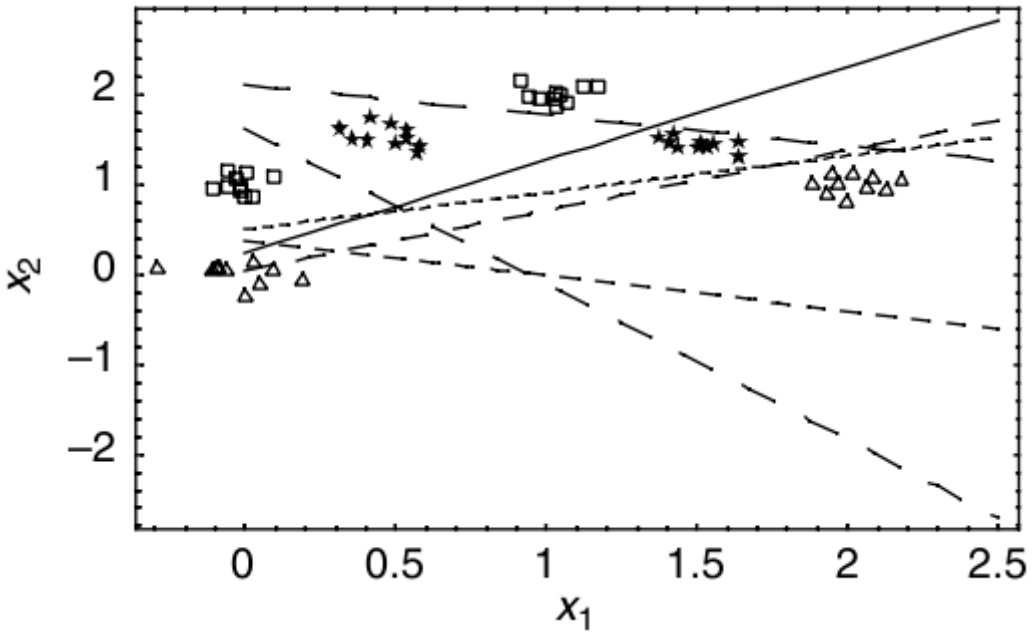
boundary line



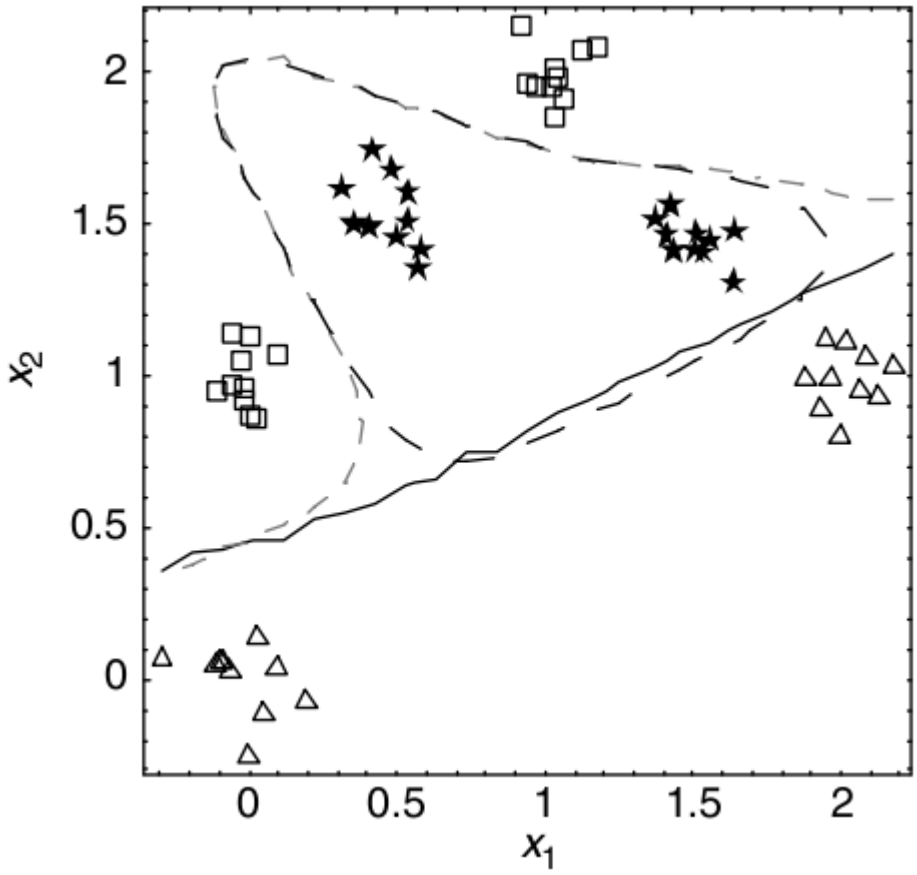
Example: Two-Dimensional Nonlinear Classification Model



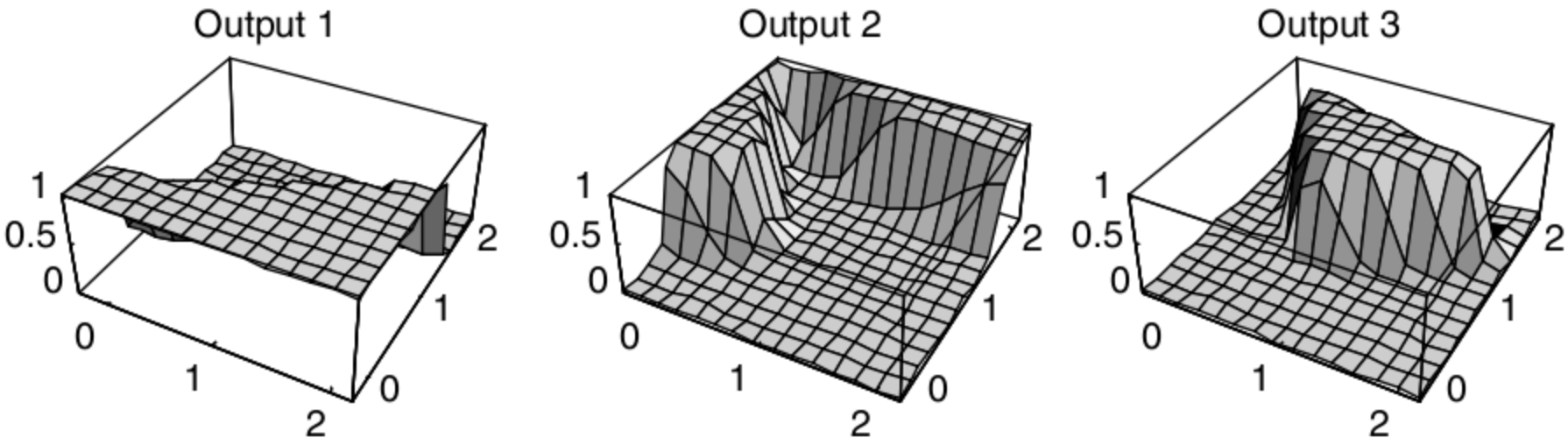
- **Boundary lines of the six hidden neurons superimposed on data**



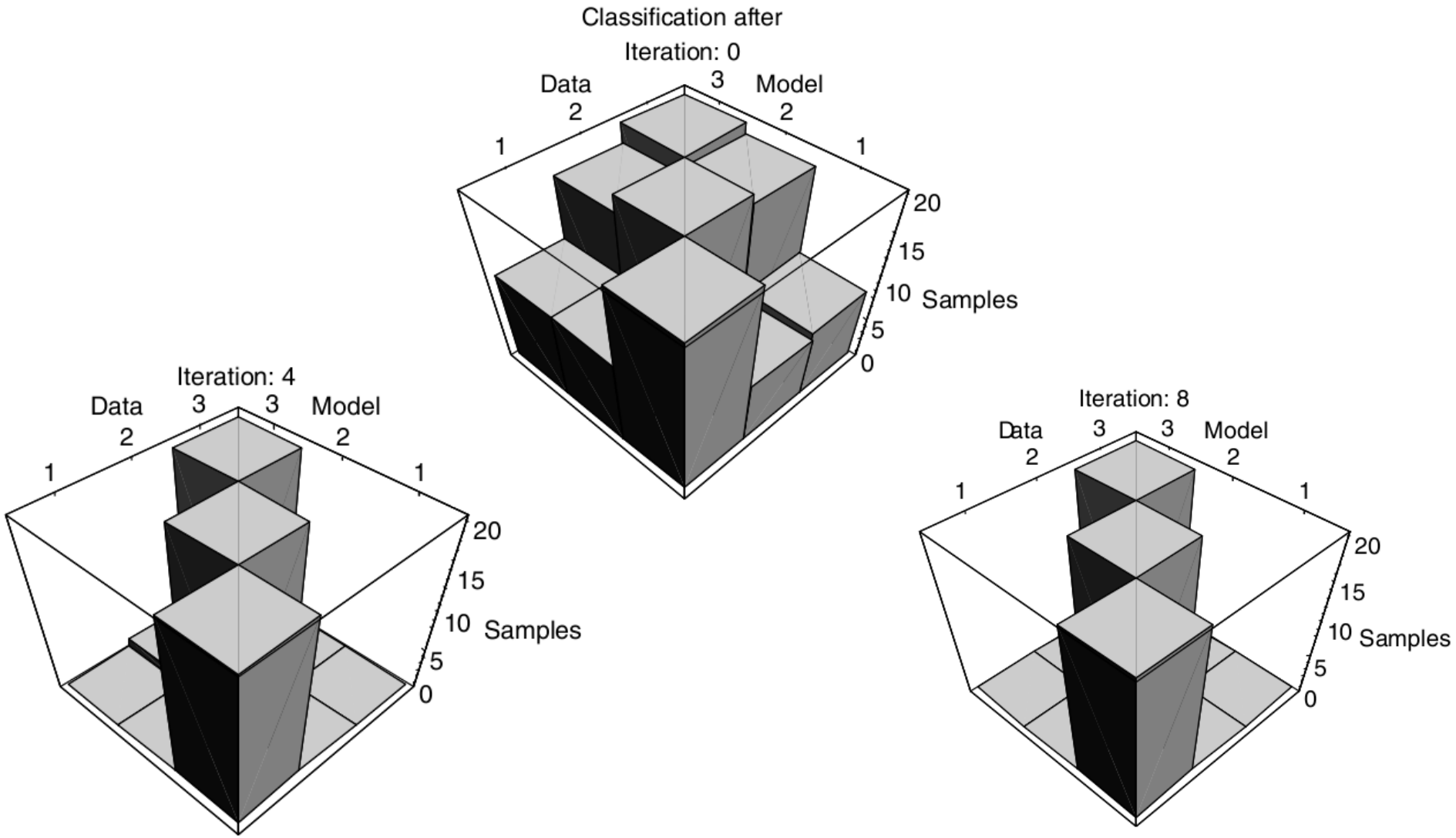
- **Final nonlinear classification boundaries created by the network**



- **Output function for each of the output neurons shown in the input space**



Progress of the MLP classifier for data belonging to three classes with two clusters in each



Multidimensional Data Modeling with Nonlinear Multilayer Perceptron Networks

$$u_j = a_{0j} + \sum_{i=1}^{n} a_{ij} x_i$$

$$y_j = f(u_j),$$

$$v_k = b_{0k} + \sum_{j=1}^{m} b_{jk} y_j$$

$$z_k = f(v_k),$$

InfoLab

Learning of Nonlinear Patterns by Neural Networks

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

Several variants of gradient descent

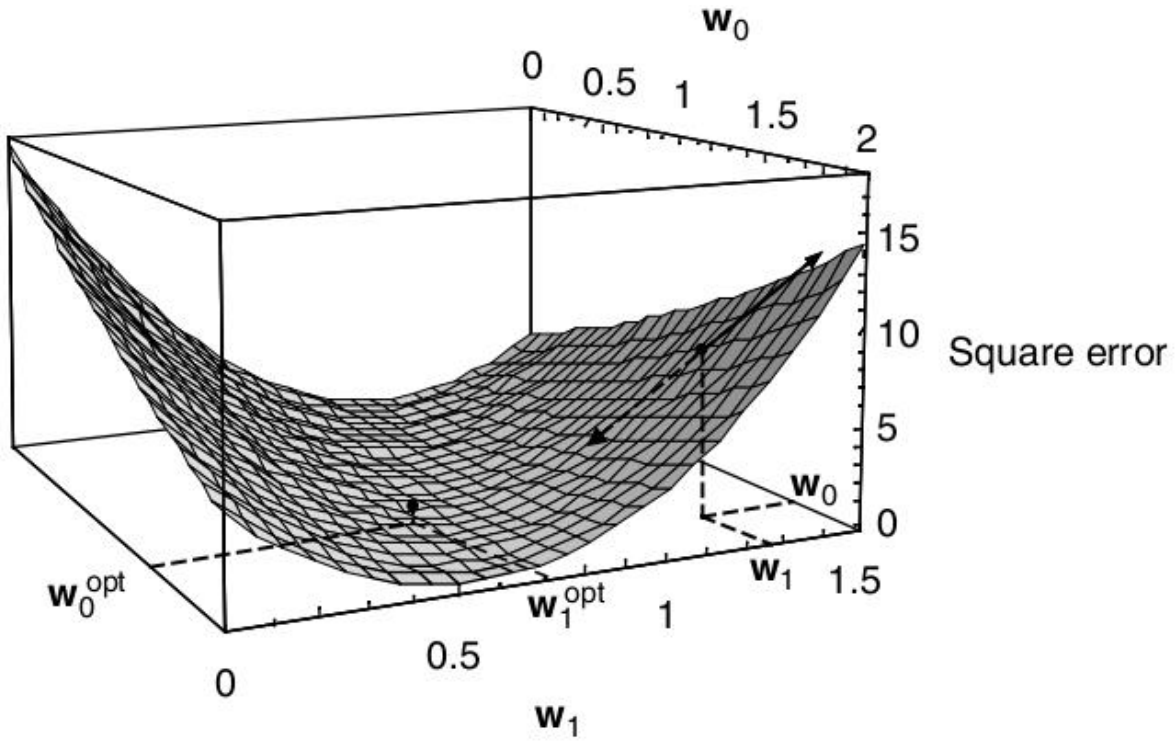
- **First-order error minimization methods**
  (based solely on the gradient of the error surface)
  - Backpropagation
  - Delta-bar-delta (or adaptive learning rate)
  - Steepest descent

- **Second-order error minimization methods**
  (the gradient descent concept is extended to include the curvature (second derivative) of the error surface)
  - QuickProp
  - Gauss–Newton
  - Levenberg–Marquardt (LM) learning methods

Supervised Training of Networks for Nonlinear Pattern Recognition

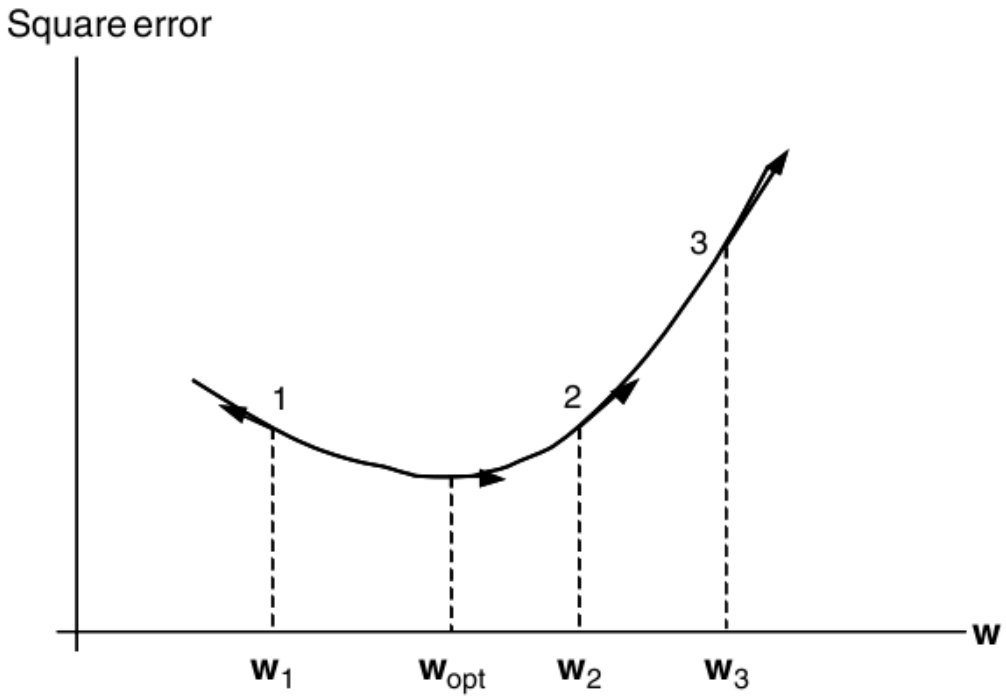
- **MSE: the most commonly used error indicator**

$$E = \frac{1}{2N} \sum_{i}^{N} (t_i - z_i)^2$$



Gradient Descent and Error Minimization

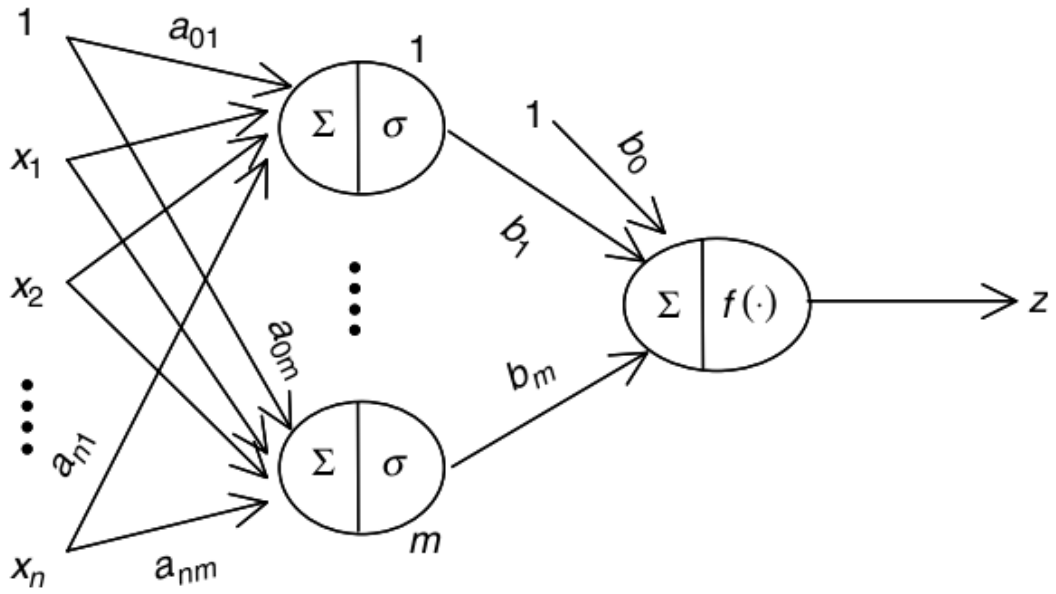
- **Gradient descent : uses the error gradient to descend the error surface**
  - change the weights in the direction in which the error decreases most rapidly, i.e., in the opposite direction to the gradient



Backpropagation Learning

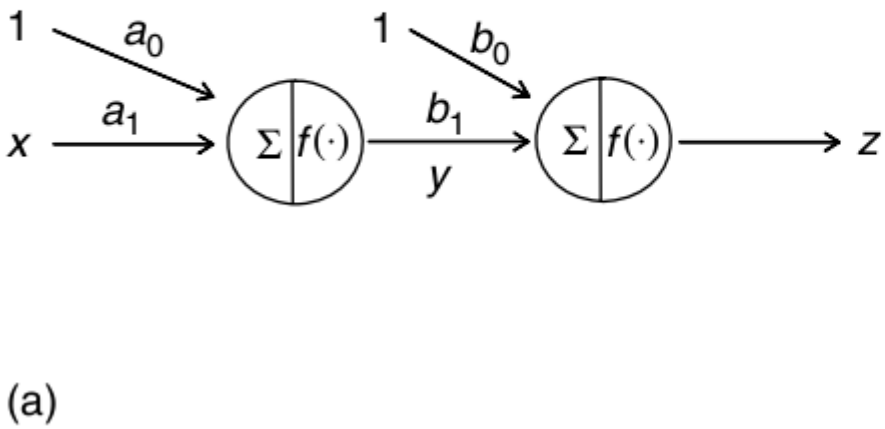
- It is possible to use the chain rule of differentiation to obtain the $\partial E / \partial b$

- It is also possible to follow this chain of association from E, z, v, y to the inputs to obtain $\partial E / \partial a$

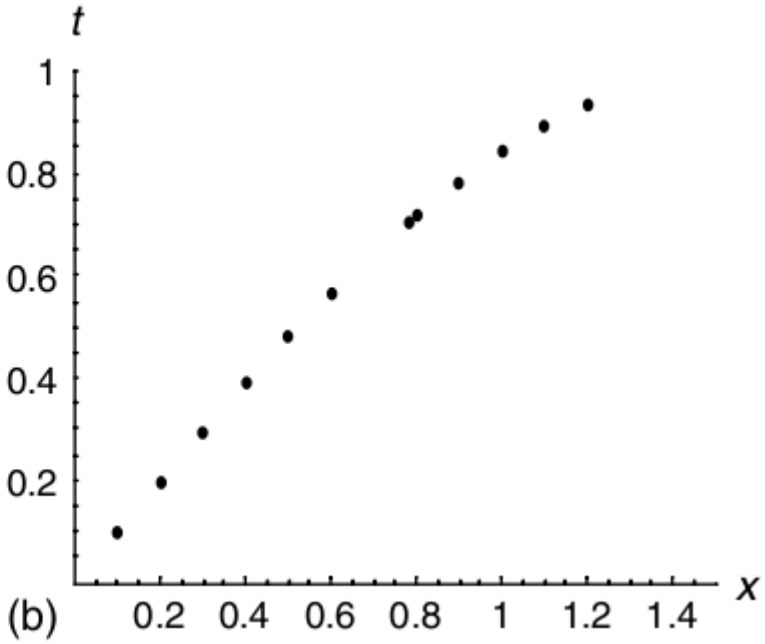


Example: Backpropagation Training (A Hand Computation)

- The function to be approximated is the first quarter of the sine wave
- Two input–output pairs {(0.7853, 0.707), (1.57, 1.0)} were chosen



(a)



(b)

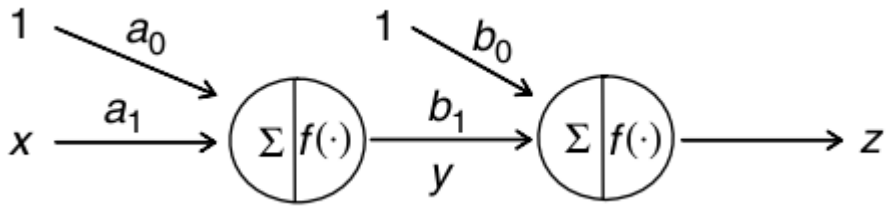


Table 4.1 Initial Weights and Two Example Input–Output Patterns (x, t)

a_0	a_1	b_0	b_1	x	t
0.3	0.2	−0.1	0.4	0.7853	0.707
				1.571	1.00

$$u = a_0 + a_1 x = 0.3 + 0.2(0.7853) = 0.457$$

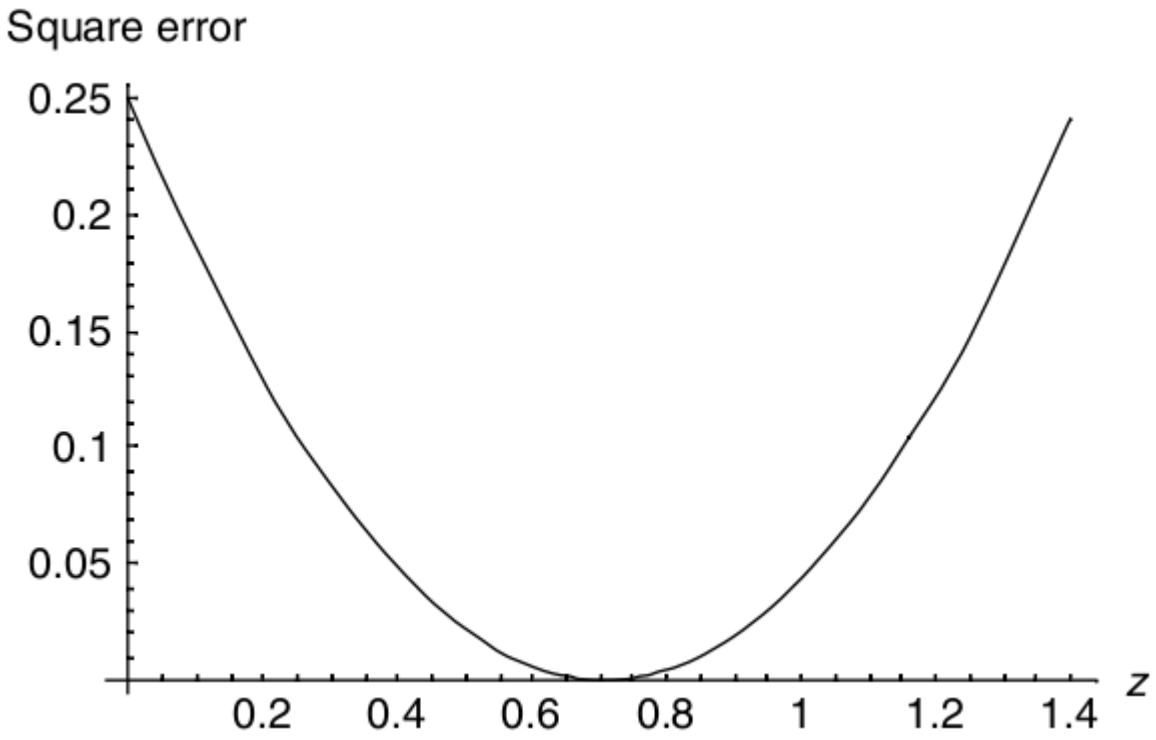
$$y = \frac{1}{1 + e^{-u}} = \frac{1}{1 + e^{-(0.457)}} = 0.612$$

$$v = b_0 + b_1 y = -0.1 + 0.4(0.612) = 0.143$$

$$z = \frac{1}{1 + e^{-v}} = \frac{1}{1 + e^{-(0.143)}} = 0.536.$$

$$E = \frac{1}{2}(z - t)^2$$

$$E = \frac{1}{2}(0.536 - 0.707)^2 = 0.0146$$



$$z = \frac{1}{1 + \mathrm{e}^{-(b_0 + b_1)\left\{\frac{1}{1 + \mathrm{e}^{-(a_0 + a_1 x)}}\right\}}}$$

$$E = \frac{1}{2}\left\{\frac{1}{1 + \mathrm{e}^{-(b_0 + b_1)\left\{\frac{1}{1 + \mathrm{e}^{-(a_0 + a_1 x)}}\right\}}} - t\right\}^2$$

Error Gradient with Respect to Output Neuron Weights

- According to the chain rule, the error derivative for any hidden-output weight b

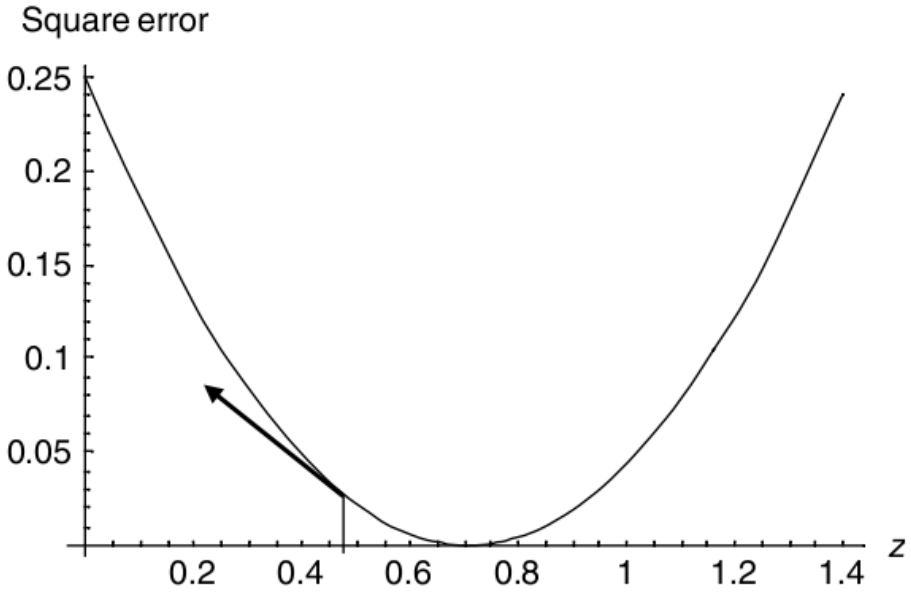
$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial v} \cdot \frac{\partial v}{\partial b}$$

- **The ∂E/∂z is basically the slope of this error surface with respect to the network output**
  - i.e., the sensitivity of the error to the network output

$$\frac{\partial E}{\partial z} = z - t$$

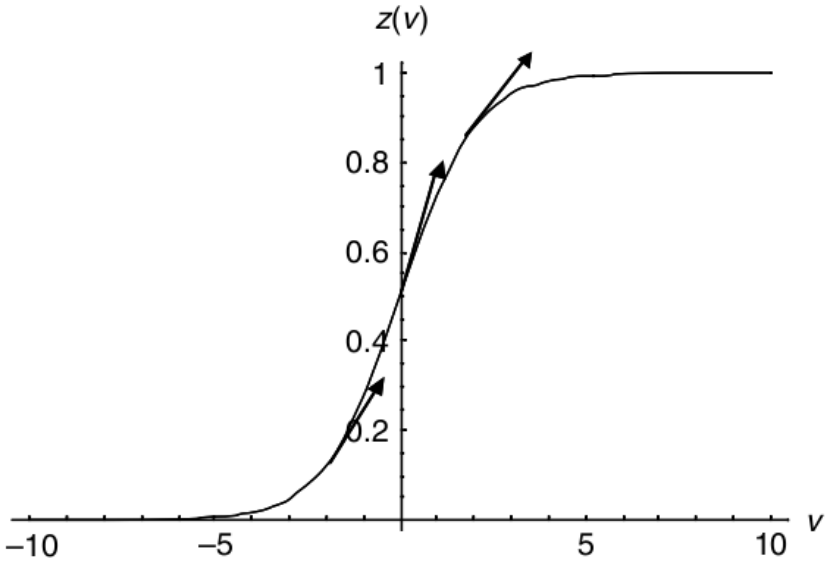
$$E = \frac{1}{2}(z-t)^2$$

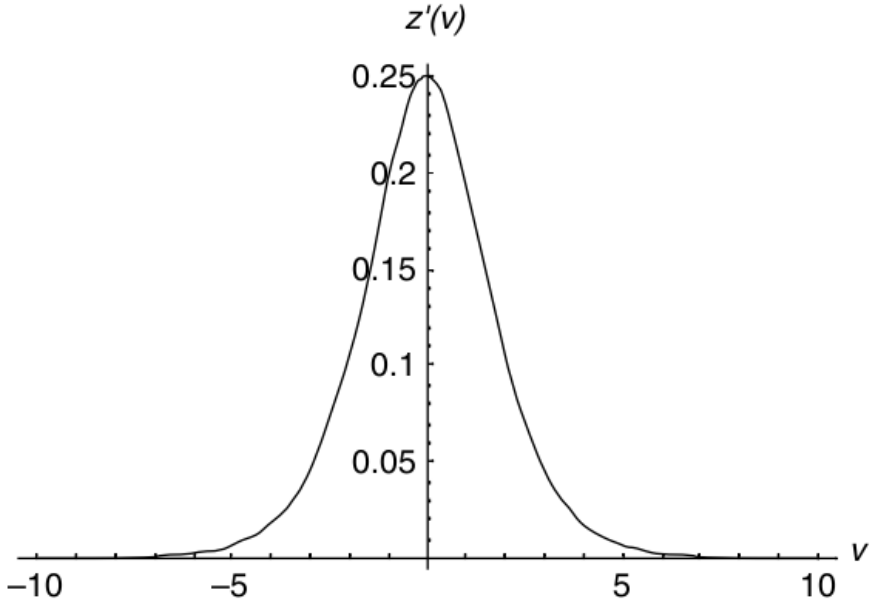
$$\frac{\partial E}{\partial z} = z - t = 0.536 - 0.707 = -0.171$$



$$z = \frac{1}{1 + e^{-v}}$$

$$z'(v) = \frac{\partial z}{\partial v} = -\frac{e^{-v}(-1)}{(1 + e^{-v})^{-2}} = \frac{e^{-v}}{(1 + e^{-v})^{-2}}$$





$$1 + e^{-v} = 1/z$$

$$e^{-v} = \frac{1}{z} - 1 = \frac{1-z}{z}$$

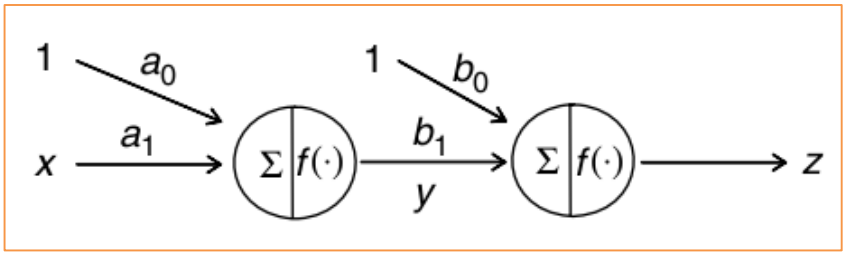
$$\frac{\partial z}{\partial v} = \frac{(1-z)/z}{1/z^2} = z(1-z)$$

- **The ∂v/∂b indicates the sensitivity of the weighted sum of the inputs v to any changes in the output neuron weights**

$$v = b_0 + b_1 y,$$

$$\frac{\partial v}{\partial b_1} = y$$

$$\frac{\partial v}{\partial b_0} = 1.$$



$$\frac{\partial E}{\partial b_0} = (z - t)\, z\, (1 - z) = p$$

$$\frac{\partial E}{\partial b_1} = (z - t)\, z\, (1 - z)\, y = py.$$

$$\frac{\partial E}{\partial b_0} = (z - t)\, z\, (1 - z) = p = (0.536 - 0.707)\,(0.536)\,(1 - 0.536) = -0.042$$

$$\frac{\partial E}{\partial b_1} = py = (-0.042)\,(0.6120) = -0.026.$$

$$\frac{\partial E}{\partial b_0} = (z - t)\, z\, (1 - z) = p = (0.536 - 0.707)\,(0.536)\,(1 - 0.536) = -0.042$$

$$\frac{\partial E}{\partial b_1} = py = (-0.042)\,(0.6120) = -0.026.$$

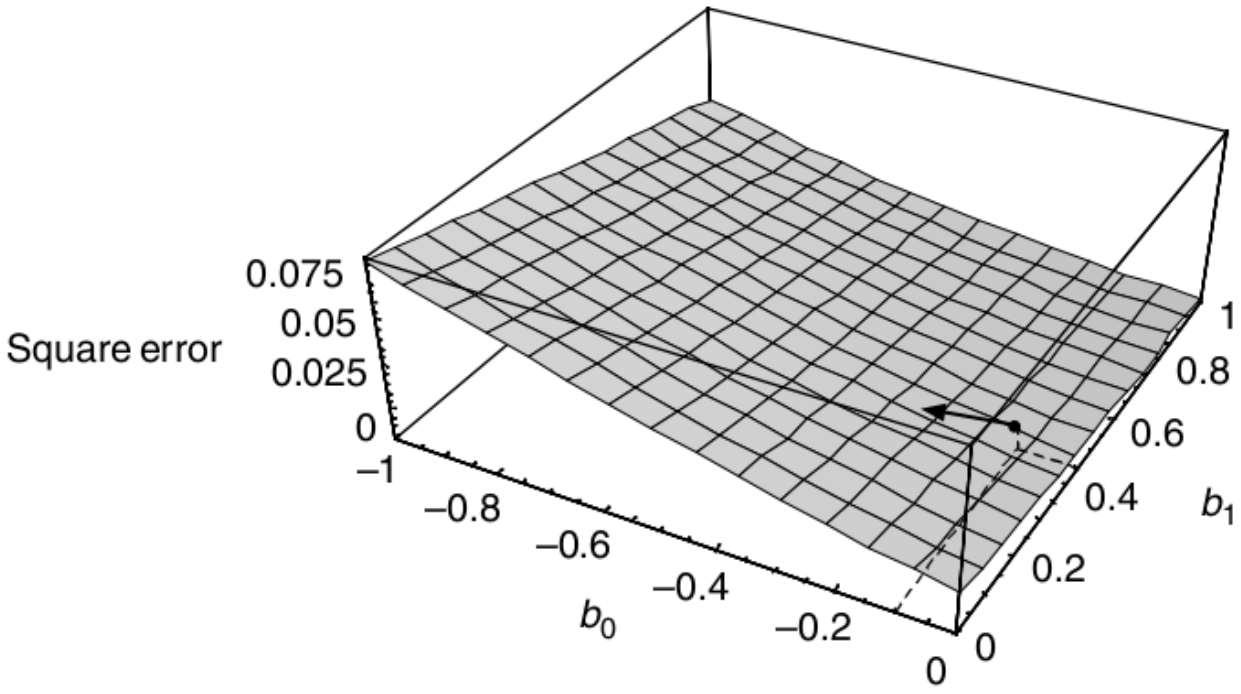
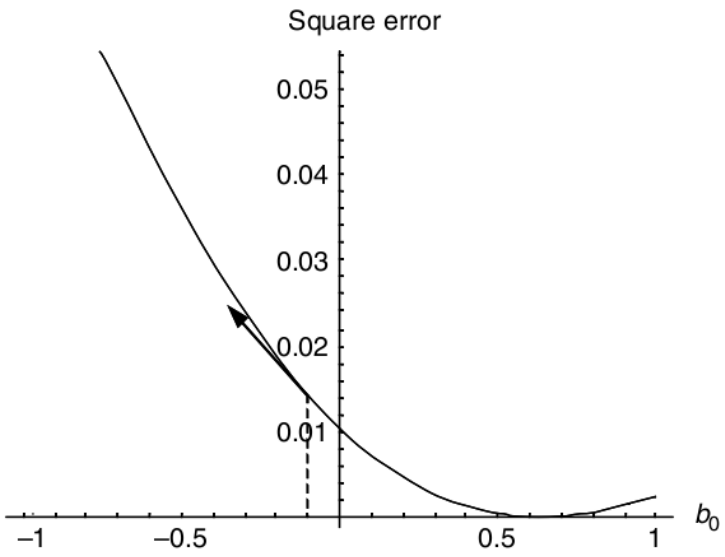


Table 4.1 Initial Weights and Two Example Input–Output Patterns (x, t)

a_0	a_1	b_0	b_1	x	t
0.3	0.2	−0.1	0.4	0.7853	0.707
				1.571	1.00



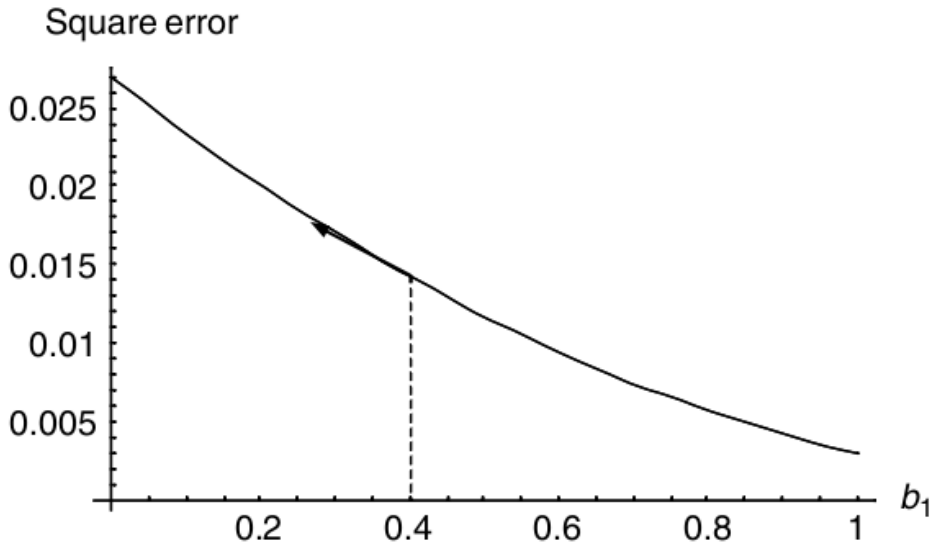
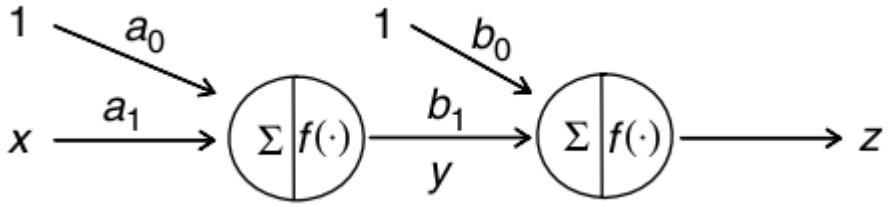


Table 4.1 Initial Weights and Two Example Input–Output Patterns (x, t)

a_0	a_1	b_0	b_1	x	t
0.3	0.2	−0.1	0.4	0.7853	0.707
				1.571	1.00

The Error Gradient with Respect to the Hidden-Neuron Weights

$$\frac{\partial E}{\partial a} = \left(\frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial v} \cdot \frac{\partial v}{\partial y}\right) \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial a}$$



$$v = b_0 + b_1 y \qquad \frac{\partial v}{\partial y} = b_1$$

$$\frac{\partial E}{\partial y} = \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial v} \cdot \frac{\partial v}{\partial y} = pb_1$$

$$\frac{\partial y}{\partial u} = y(1-y)$$

$$\frac{\partial z}{\partial v} = \frac{(1-z)/z}{1/z^2} = z(1-z).$$

$$\frac{\partial u}{\partial a_1} = x$$

$$\frac{\partial u}{\partial a_0} = 1$$

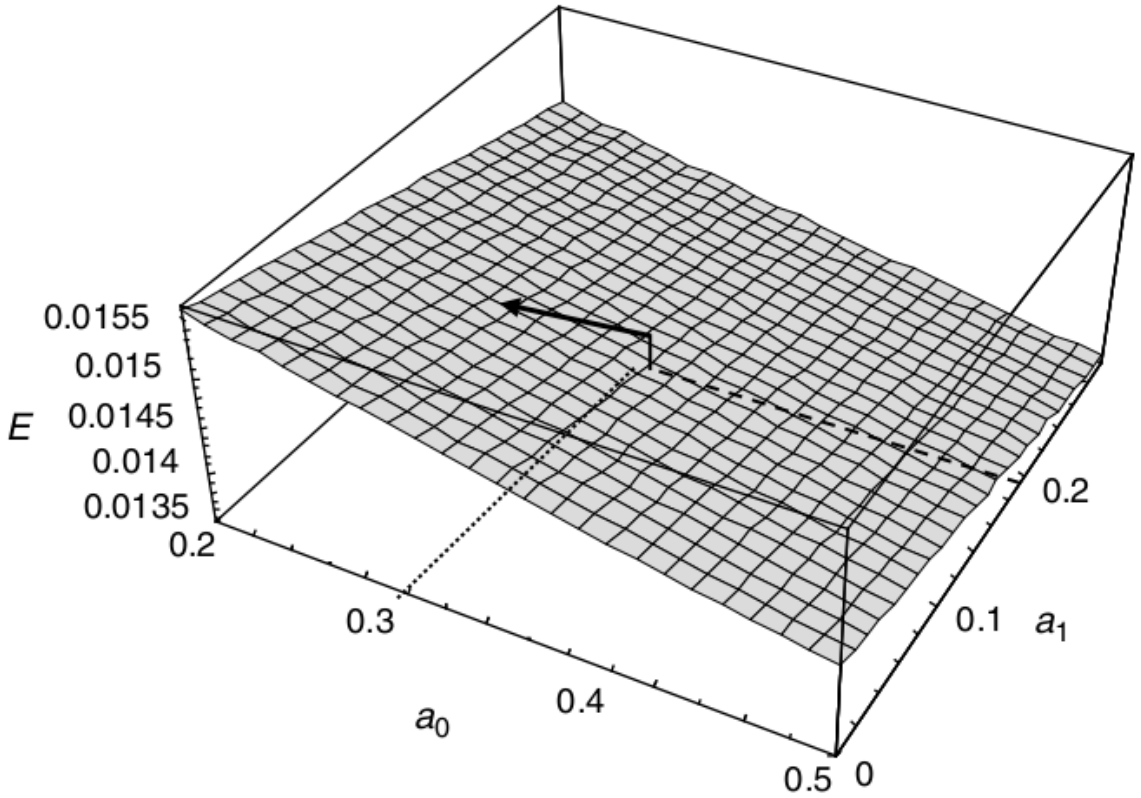
$$\frac{\partial E}{\partial a_1} = pb_1 y(1-y)x = qx$$

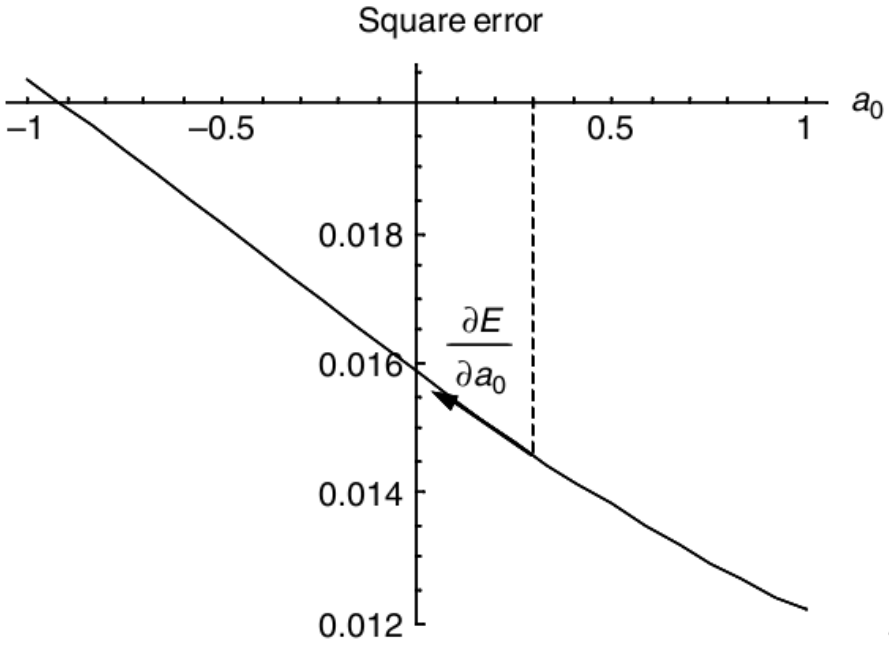
$$\frac{\partial E}{\partial a_0} = pb_1 y(1-y) = q.$$

$$p = -0.042,\ b_1 = 0.4,\ y = 0.612,\ x = 0.7853$$

$$\frac{\partial E}{\partial a_0} = pb_1y(1-y) = q = (-0.042)(0.4)(0.612)(1-0.612) = -0.004$$

$$\frac{\partial E}{\partial a_1} = pb_1y(1-y)x = qx = (-0.004)(0.7853) = -0.00314.$$





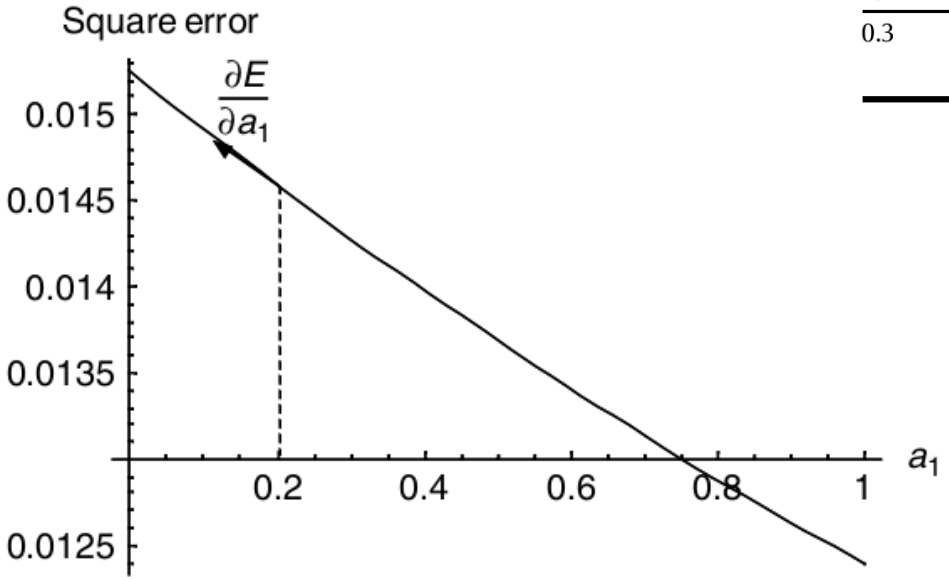


Table 4.1 Initial Weights and Two Example Input–Output Patterns (x, t)

a_0	a_1	b_0	b_1	x	t
0.3	0.2	−0.1	0.4	0.7853	0.707
				1.571	1.00

- **Repeating the procedure for the second input–output pair {x, t} = (1.571, 1.00)**

$$\frac{\partial E}{\partial a_0} = -0.0104$$

$$\frac{\partial E}{\partial a_1} = -0.0163$$

$$\frac{\partial E}{\partial b_0} = -0.1143$$

$$\frac{\partial E}{\partial b_1} = -0.0742.$$

$$z = 0.5398$$

$$\text{Square error } E = 0.1059.$$

The MSE for the two patterns are $(0.0146 + 0.1059)/2 = 0.0602$

Application of Gradient Descent in Backpropagation Learning

- **Gradient descent dictates that the error should be minimized in the direction of the steepest descent**

- **Two ways for accomplishing an incremental adjustment of all of the weights**

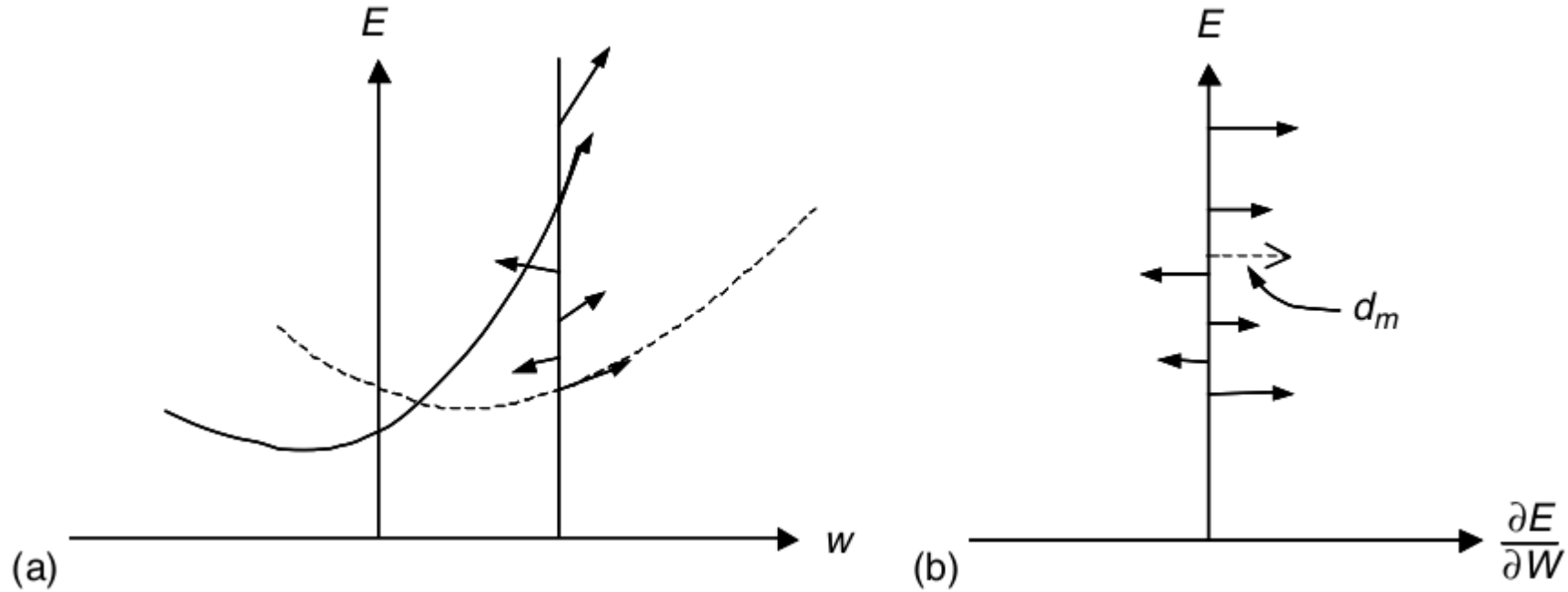
  - example-by-example (or on-line) learning : the weights are adjusted after every training pattern;

  - **batch (or off-line) learning** : learning (i.e., weight adjustment) occurs after all of the training examples have been presented to the network once

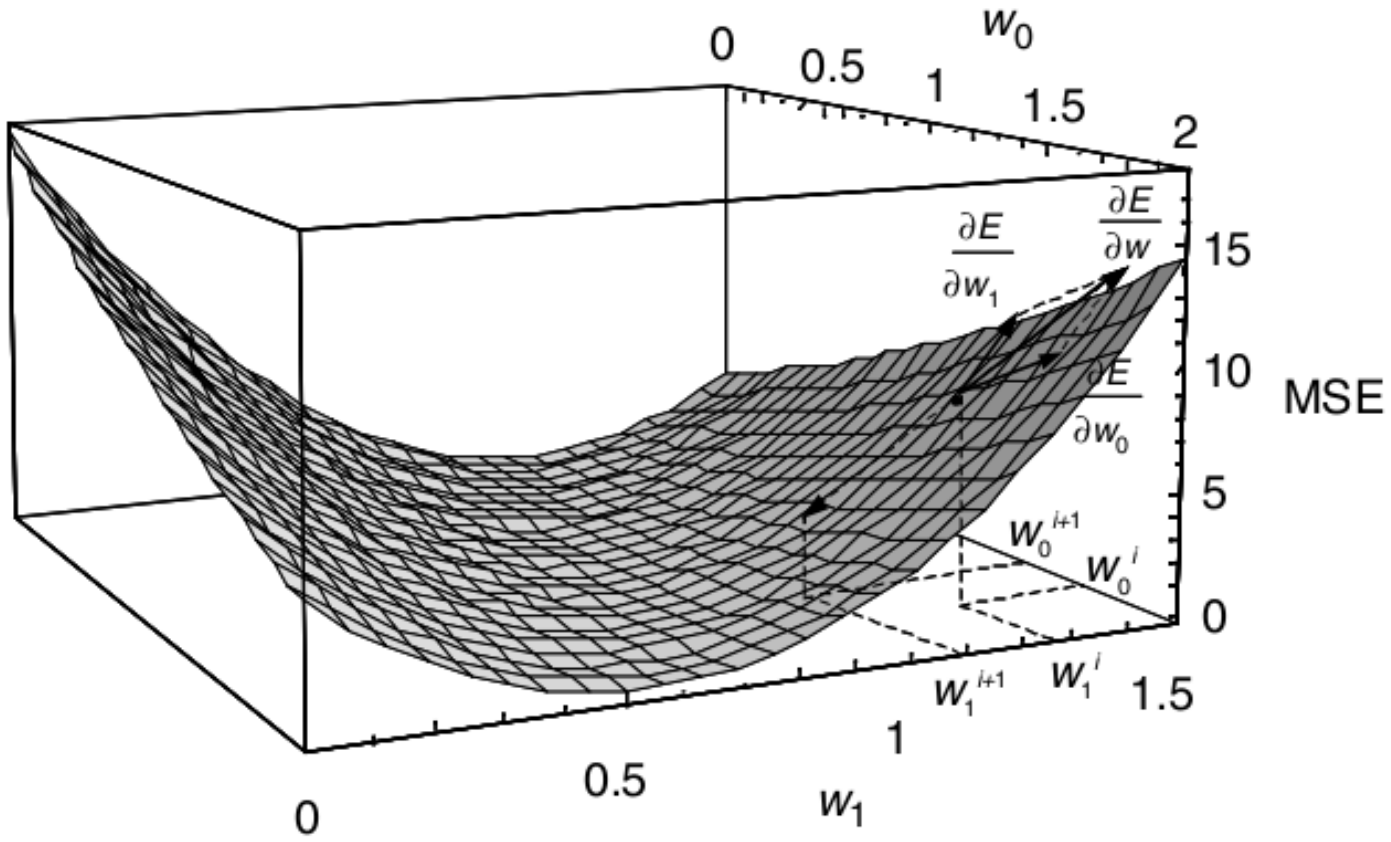
Batch Learning

- The total gradient, d_m, for m-th epoch

$$d_m = \sum_{n=1}^{N} \left[\frac{\partial E}{\partial w_m} \right]_n$$



$$E = \frac{1}{2N} \sum_{i}^{N} (t_i - z_i)^2$$

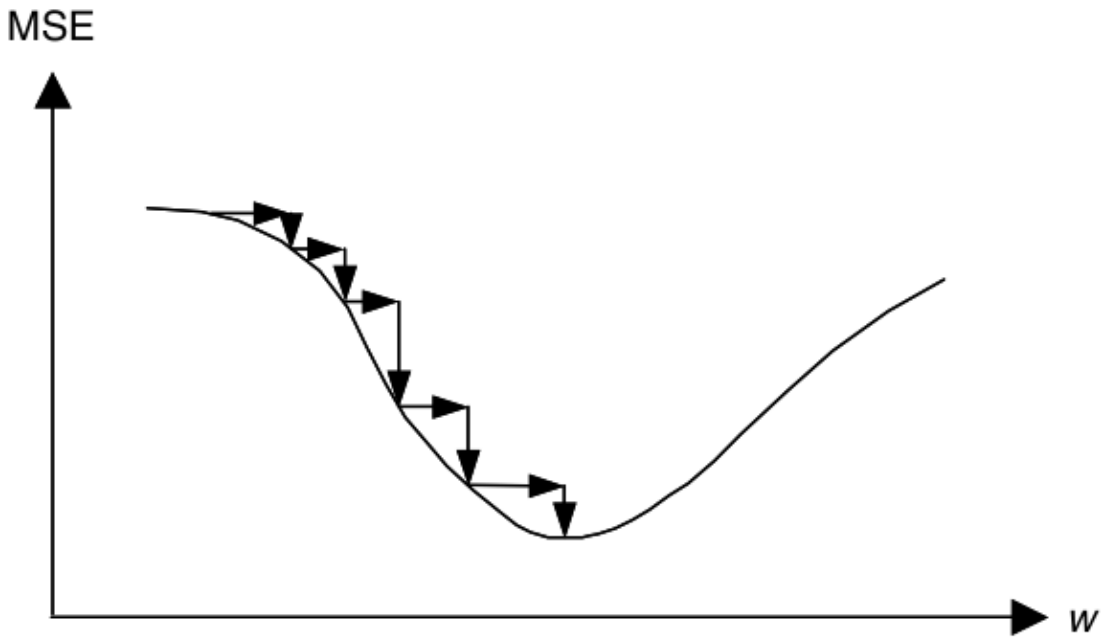


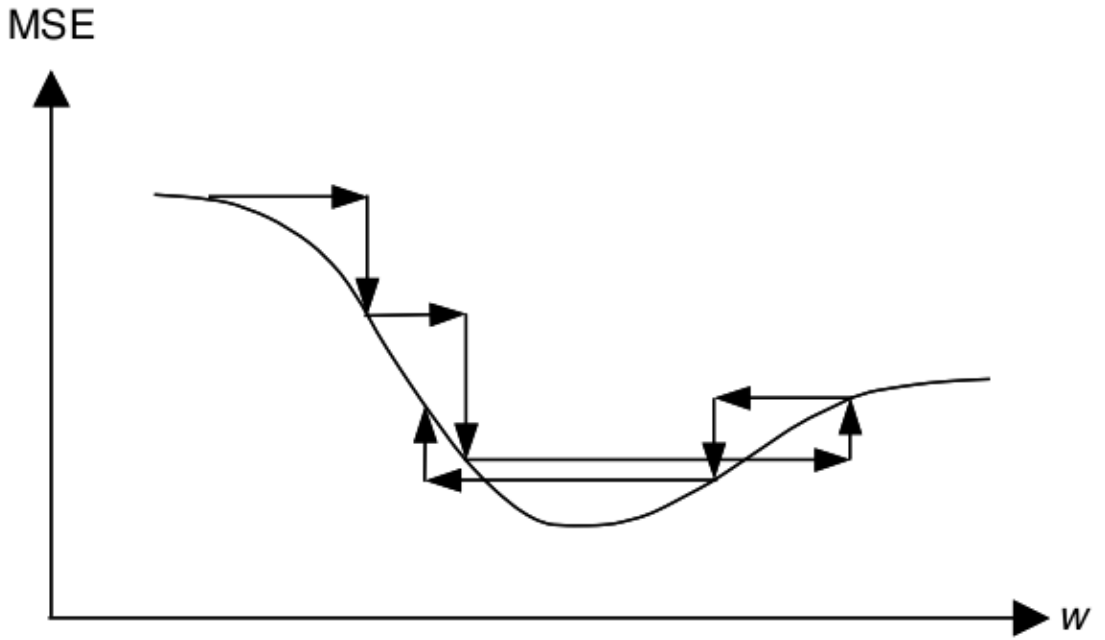
Learning Rate and Weight Update

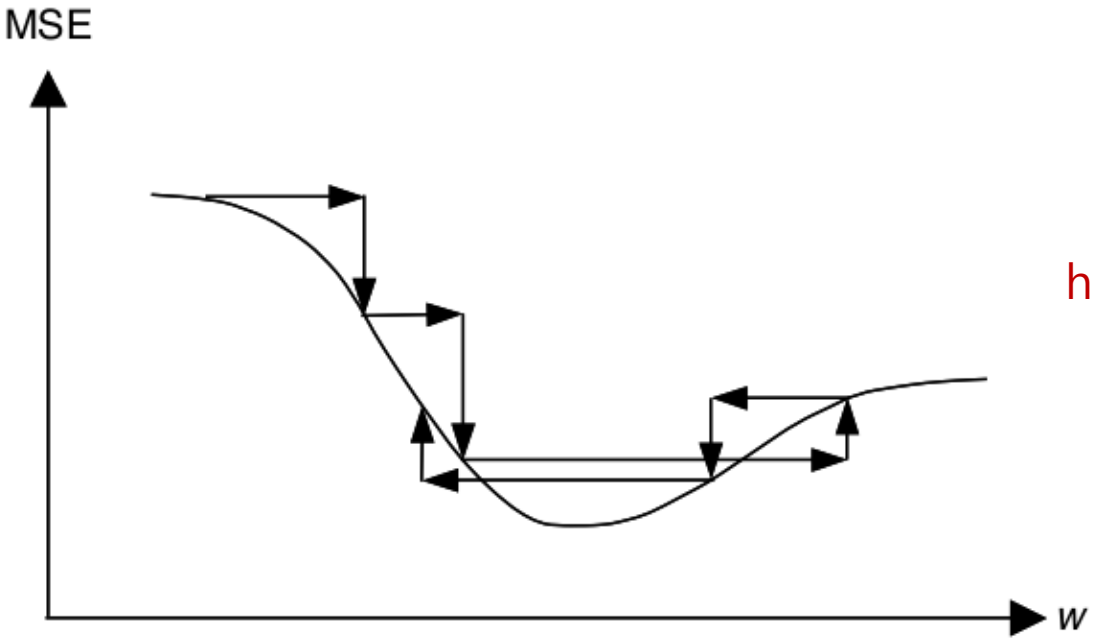
- If ε is 1.0, the distance of the descent will be equal to the total arrow length of the resultant gradient

- But, a smaller ε must be used to slowly and smoothly guide the descent towards the optimum weights (between 0 and 1)

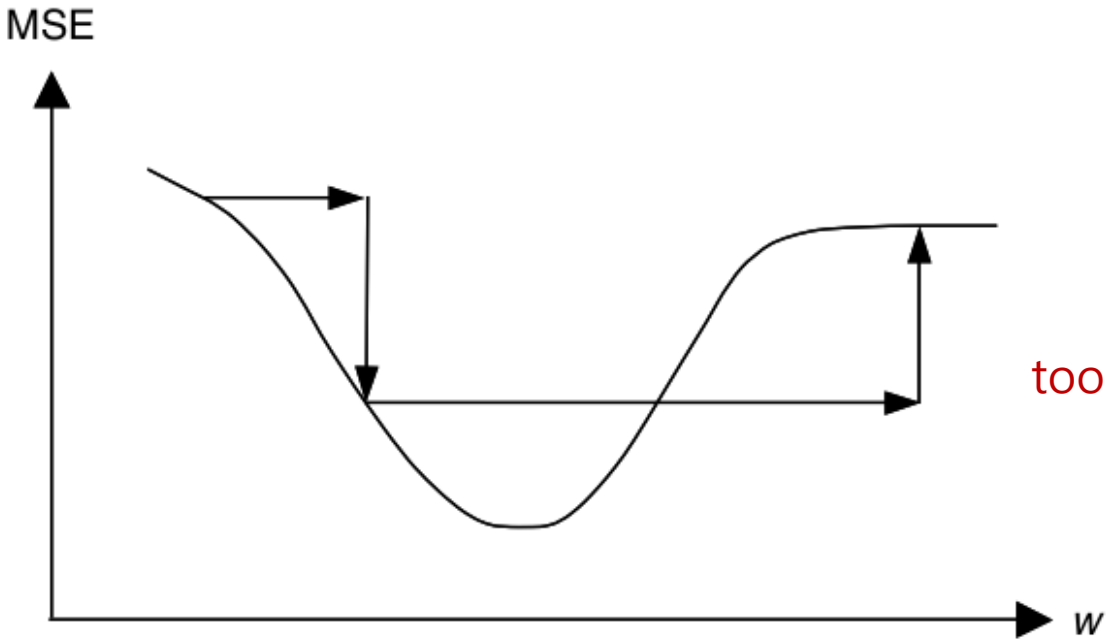
$$w_{m+1} = w_m + \Delta w_m$$
$$\Delta w_m = -\varepsilon d_m,$$


MSE
w


MSE
w



high learning rate



too high a learning rate

Example problem continued

- **Epoch 1**

$$d_1^{a_0} = \sum \frac{\partial E}{\partial a_0} = -0.004 - 0.0104 = -0.0144$$

$$d_1^{a_1} = \sum \frac{\partial E}{\partial a_1} = -0.00314 - 0.0163 = -0.01944$$

$$d_1^{b_0} = \sum \frac{\partial E}{\partial b_0} = -0.042 - 0.1143 = -0.1563$$

$$d_1^{b_1} = \sum \frac{\partial E}{\partial b_1} = -0.026 - 0.0742 = -0.1002.$$

learning rate $\varepsilon = 0.1$

$$\Delta a_0 = -\varepsilon d_1^{a_0} = -0.1(-0.0144) = 0.00144$$

$$a_0^2 = a_0 + \Delta a_0 = 0.3 + 0.00144 = 0.30144$$

$$a_1^2 = a_1 + \Delta a_1 = a_1 - \varepsilon d_1^{a_1} = 0.2 - (0.1)(-0.01944) = 0.2019$$

$$b_0^2 = b_0 + \Delta b_0 = b_0 - \varepsilon d_1^{b_0} = -0.1 - (0.1)(-0.1563) = -0.0844$$

$$b_1^2 = b_1 + \Delta b_1 = b_1 - \varepsilon d_1^{b_1} = 0.4 - (0.1)(-0.1002) = 0.410.$$

$$a_0^2 = 0.3014$$

$$a_1^2 = 0.2019$$

$$b_0^2 = -0.0844$$

$$b_1^2 = 0.410.$$

Pattern 1: $(0.7583, 0.707) \Rightarrow 0.01367$

Pattern 2: $(1.571, 1) \Rightarrow 0.1033$

$\text{MSE} = (0.01367 + 0.1033)/2 \Rightarrow 0.0585,$

smaller than the initial MSE of 0.0602

Epoch 2

$$d_2^{a_0} = \sum \frac{\partial E}{\partial a_0} = -0.00399 - 0.0105 = -0.01449$$

$$d_2^{a_1} \sum \frac{\partial E}{\partial a_1} = -0.00312 - 0.0165 = -0.0196$$

$$d_2^{b_0} = \sum \frac{\partial E}{\partial b_0} = -0.04106 - 0.1127 = -0.1538$$

$$d_2^{b_1} = \sum \frac{\partial E}{\partial b_1} = -0.0251 - 0.0732 = -0.0983,$$

$$\Delta a_0^2 = 0.00145 \quad a_0^3 = 0.3029$$

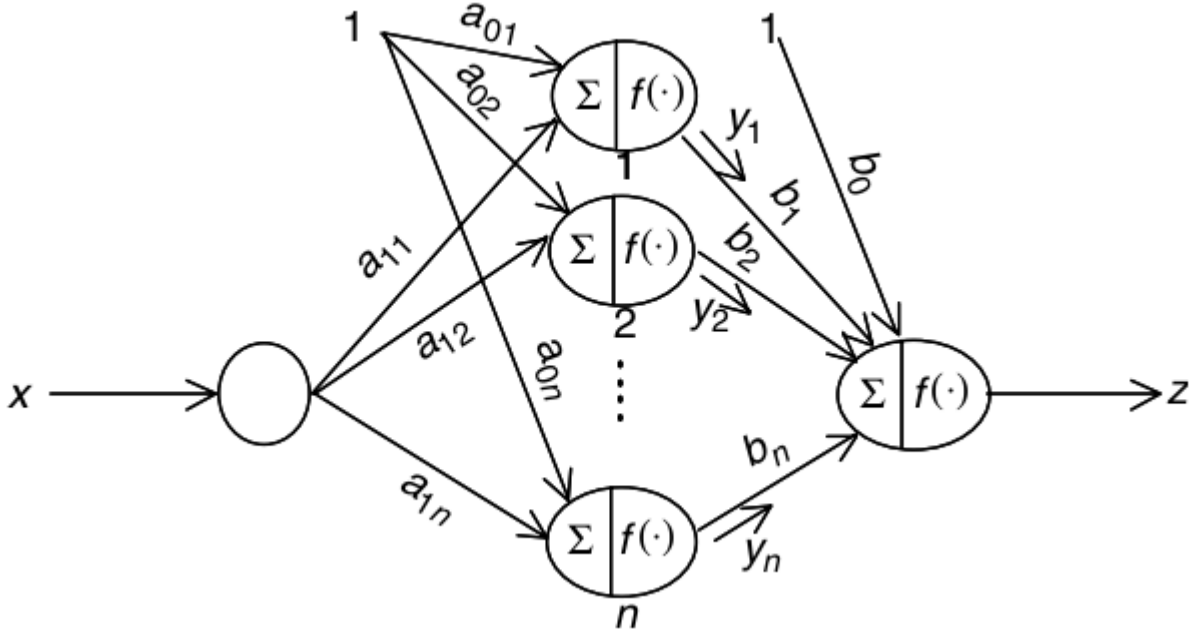
$$\Delta a_1^2 = 0.00196 \quad a_1^3 = 0.2039$$

$$\Delta b_0^2 = 0.01538 \quad b_0^3 = -0.069$$

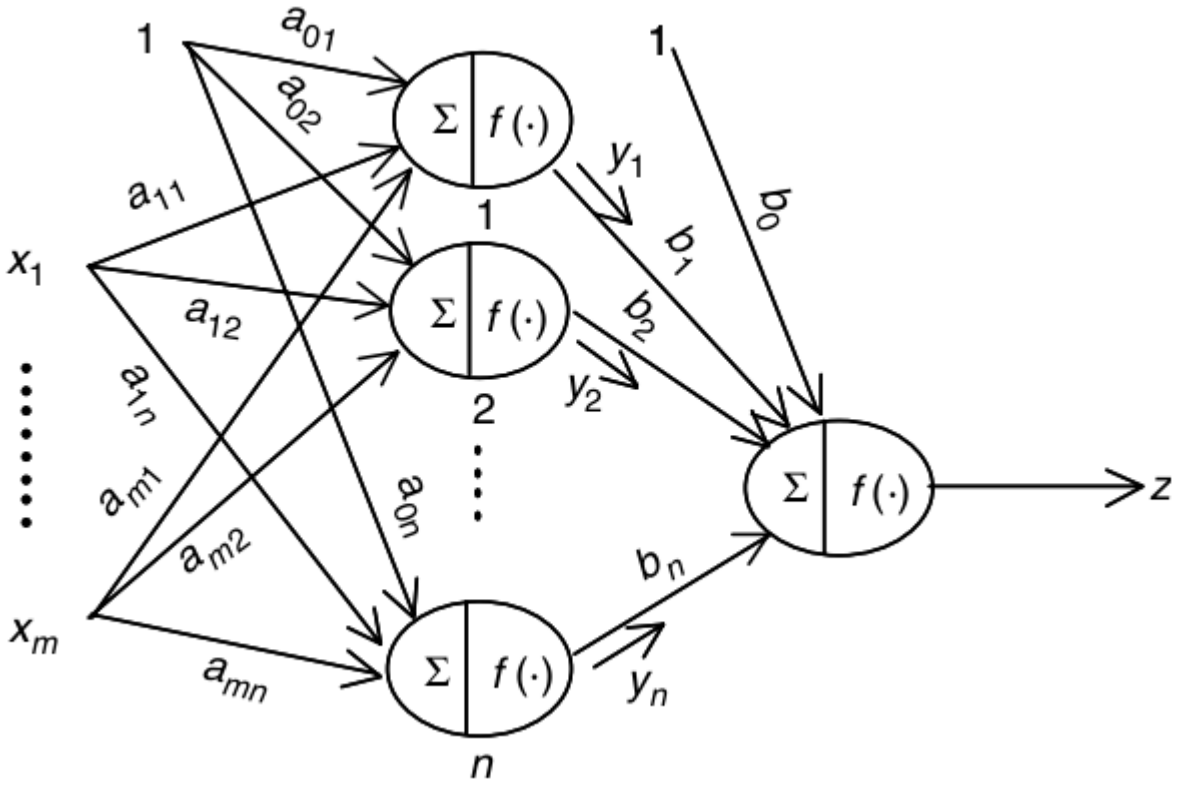
$$\Delta b_1^2 = 0.00983 \quad b_1^3 = 0.4198.$$

$$\text{MSE} = (0.0128 + 0.10085)/2 = \boxed{0.0568}$$

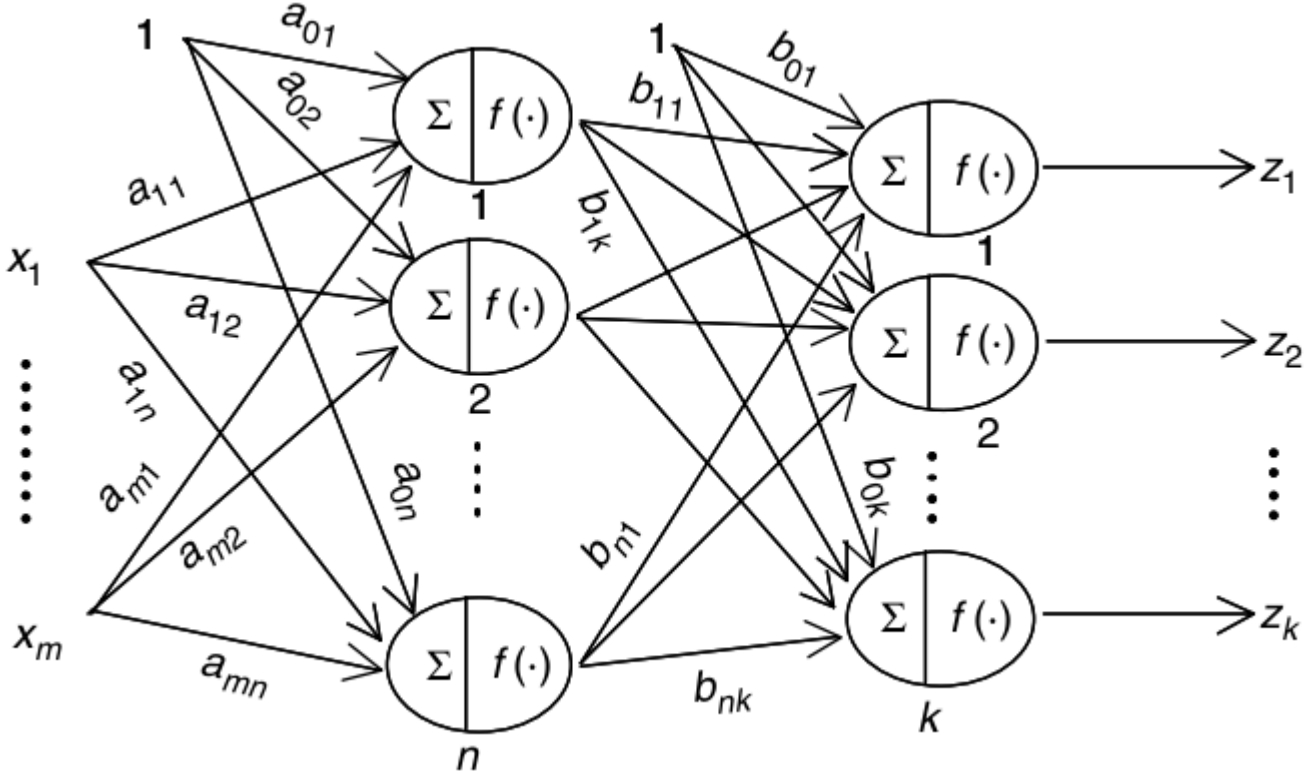
Single-Input, Multiple-Hidden Neuron, Single-Output Network



Multiple-Input, Multiple-Hidden Neuron, Single-Output Network



Multiple-Input, Multiple-Hidden Neuron, Multiple-Output Network



InfoLab

Thank you!

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology