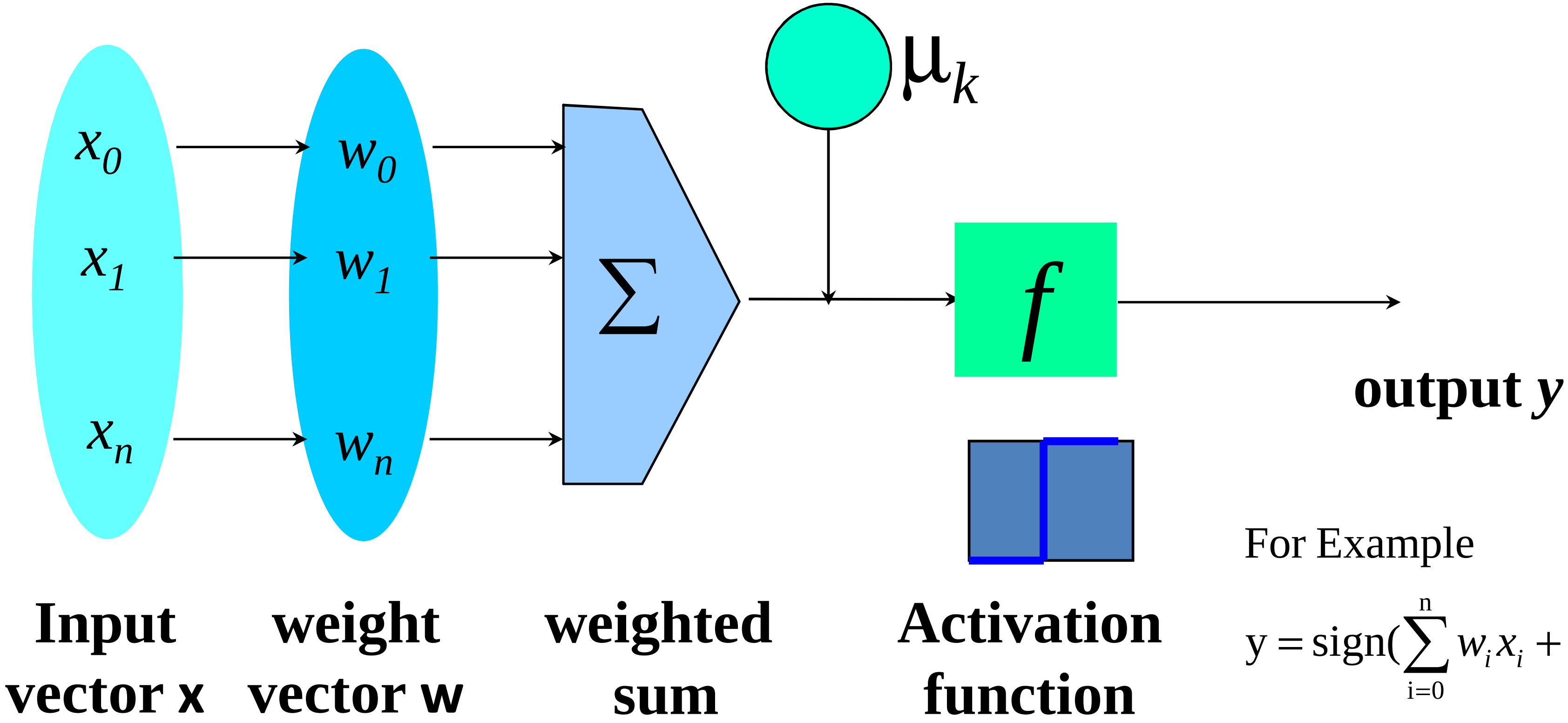
Neural Networks
Multilayer Perceptron

A Neuron (= a perceptron)



For Example

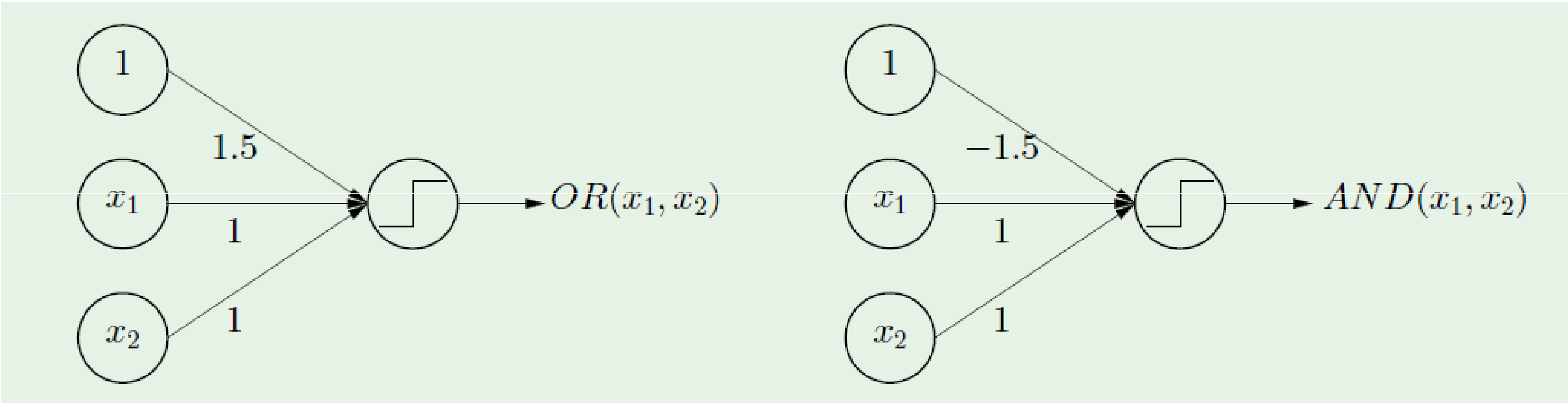
$$y = \text{sign}\left(\sum_{i=0}^{n} w_i x_i + \mu_k\right)$$

- The n-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

Perceptron - Example

Threshold >=1.5

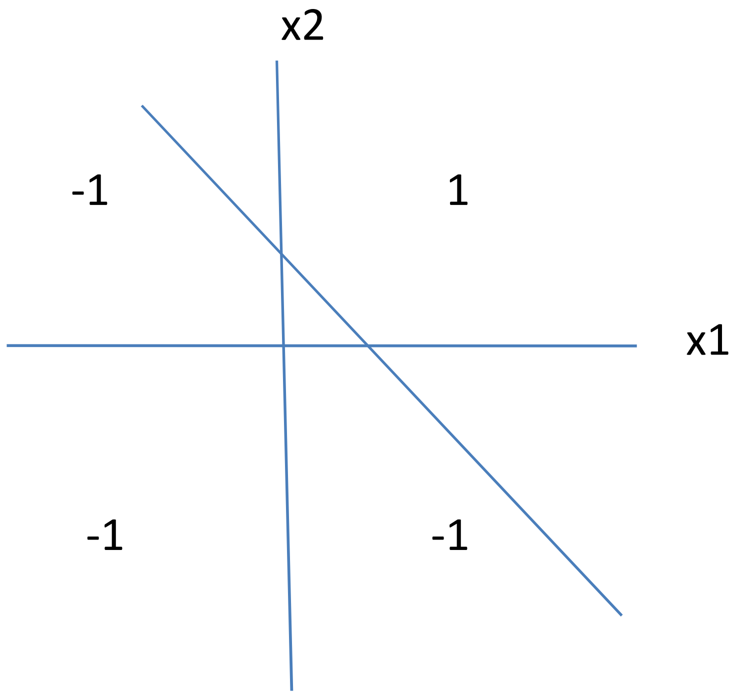
Threshold > -1.5

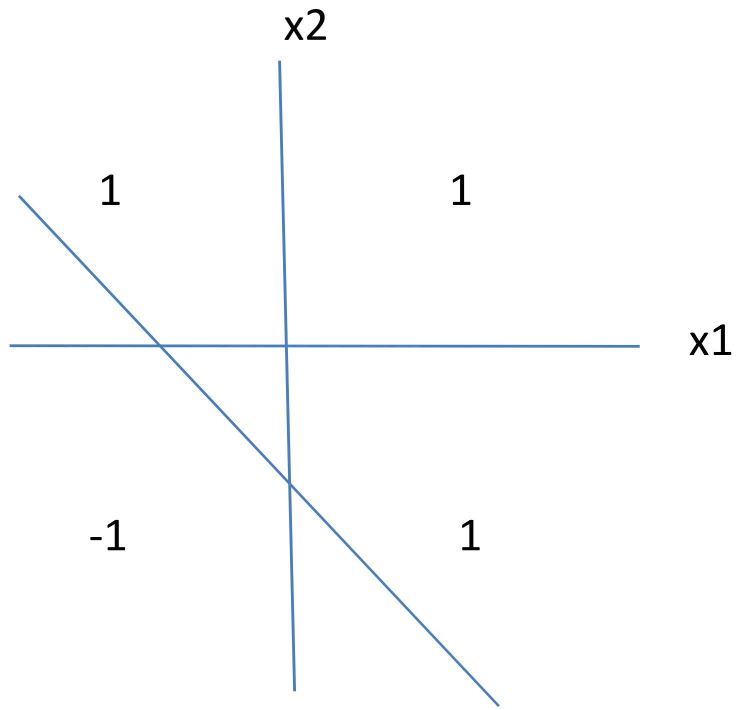


X1	x2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

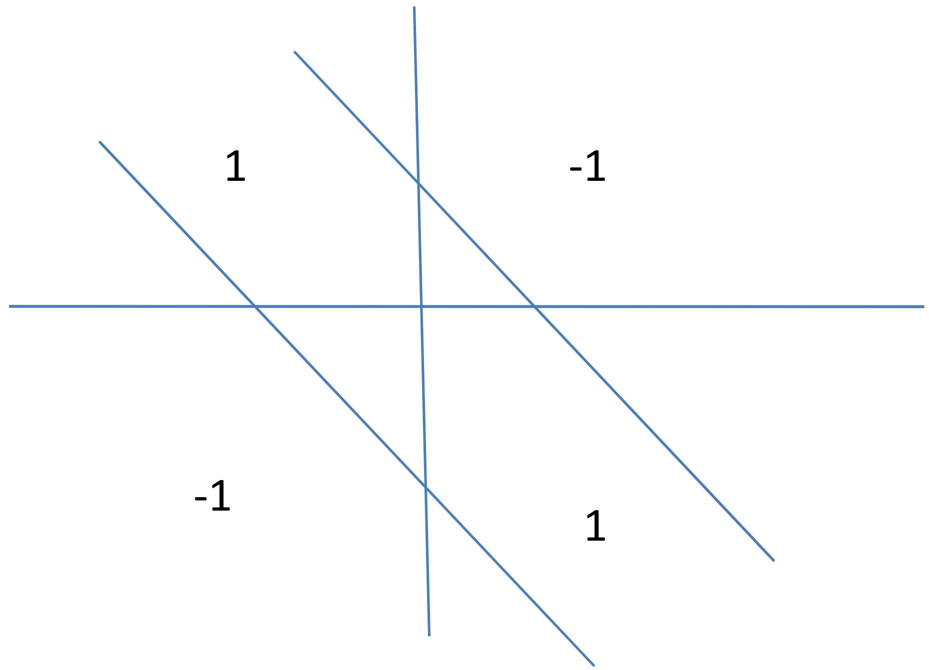
X1	x2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

AND ---OR



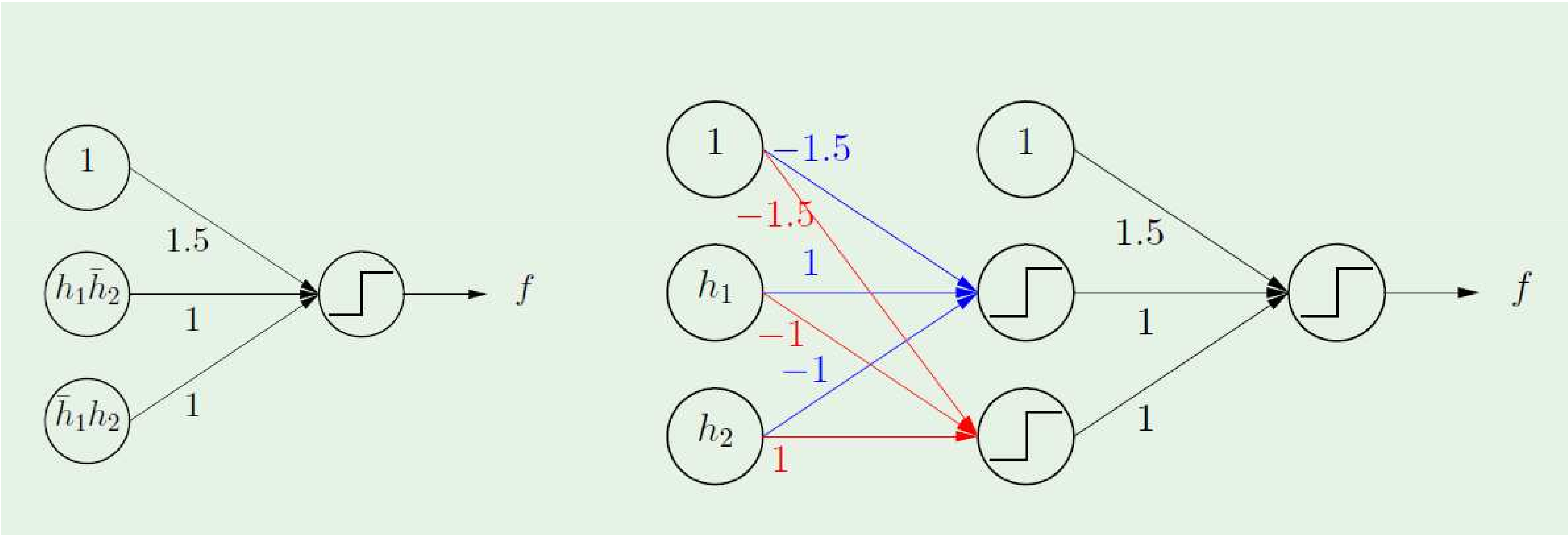


XOR

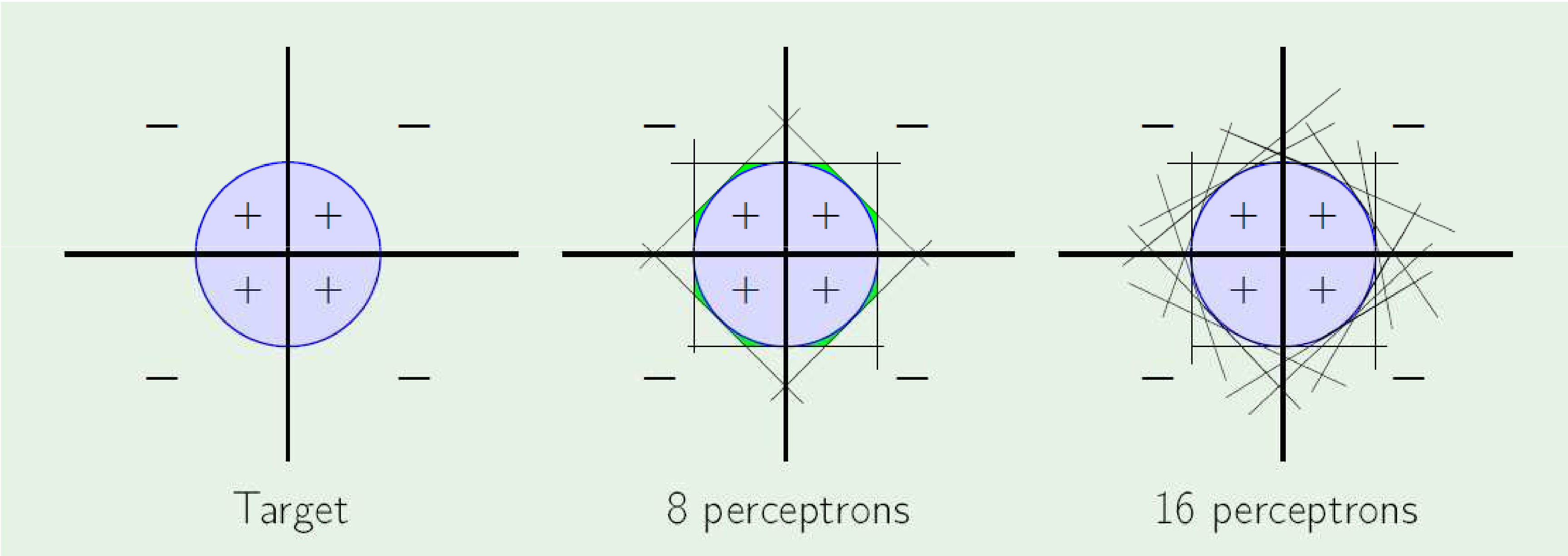


XOR Problem

f=h1h2’+h1’h2

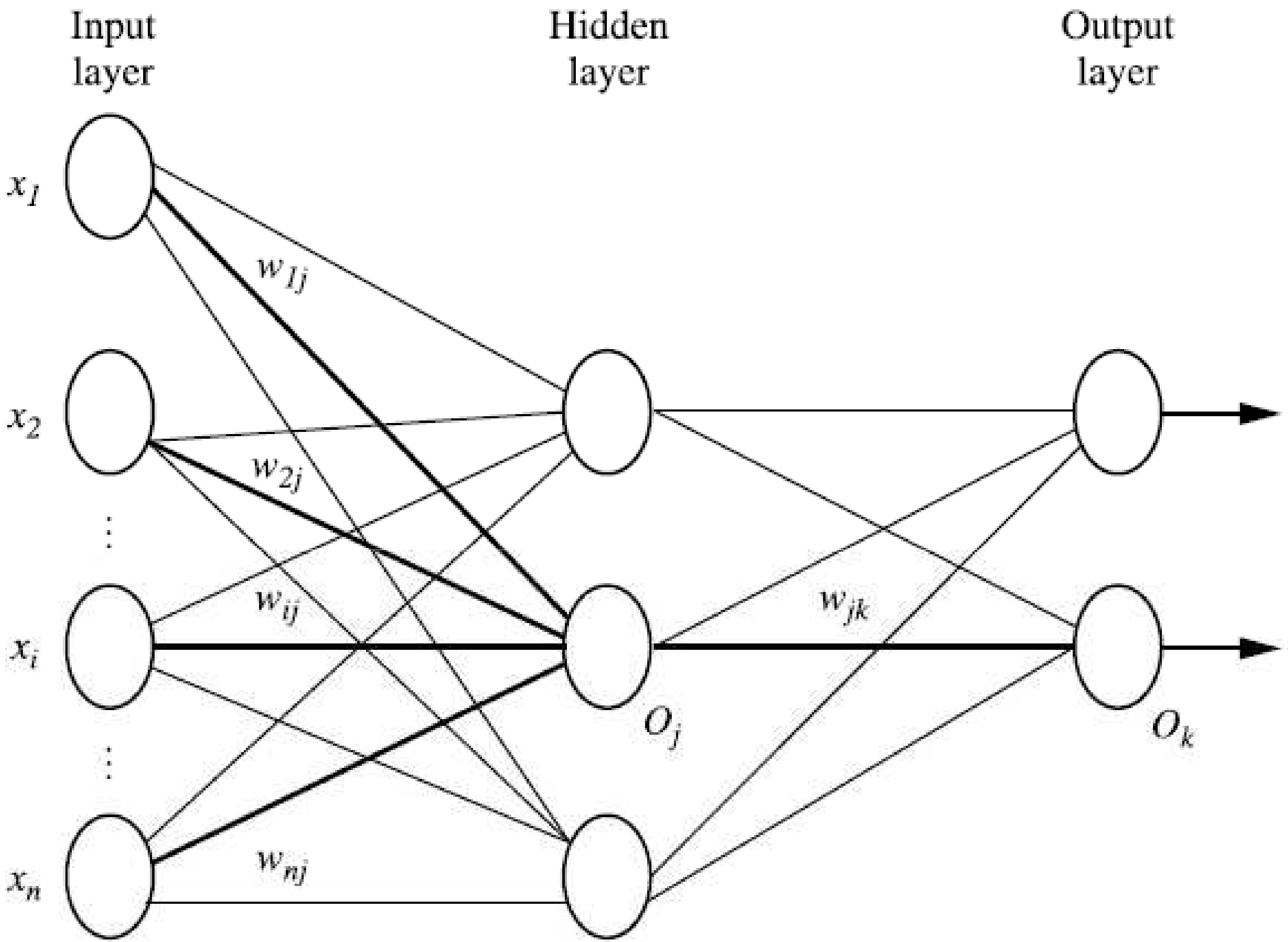


A Powerful model



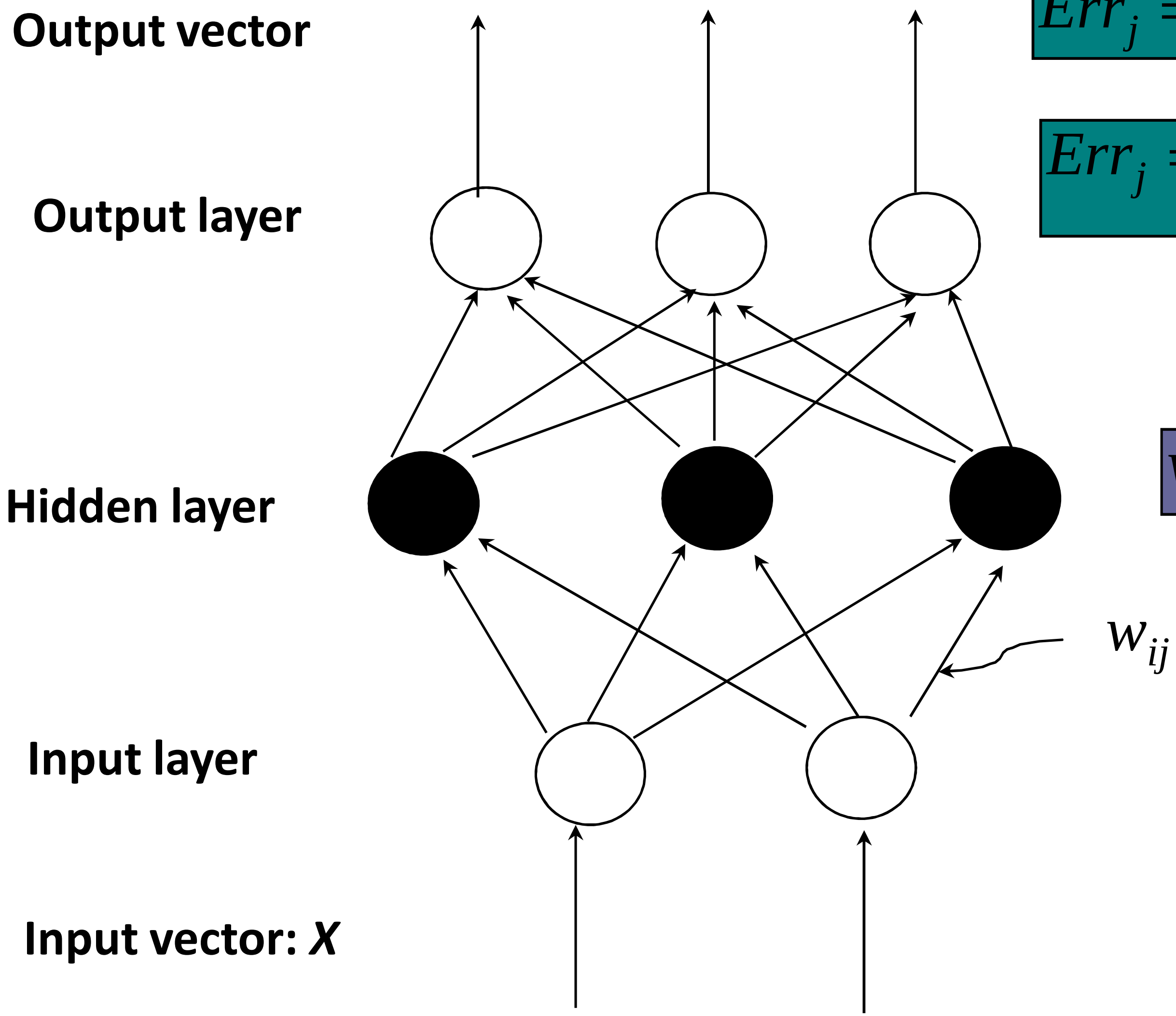
Classification by Backpropagation

- Backpropagation: **A neural network learning algorithm**
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- **A neural network:** A set of connected input/output units where each connection has a weight associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units.



A multilayer feed-forward neural network.

A Multi-Layer Feed-Forward Neural Network



$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

$$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$$

$$\theta_j = \theta_j + (l)Err_j$$

$$w_{ij} = w_{ij} + (l)Err_j O_i$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

Backpropagation Algorithm

Algorithm: Backpropagation. Neural network learning for classification or prediction, using the backpropagation algorithm.

Input:

- *D*, a data set consisting of the training tuples and their associated target values;
- *l*, the learning rate;
- *network*, a multilayer feed-forward network.

Output: A trained neural network.

Method:

Initialize all weights and biases in *network*;
while terminating condition is not satisfied {
for each training tuple X in D {
// Propagate the inputs forward:
for each input layer unit j {
$O_j = I_j$; // output of an input unit is its actual input value
for each hidden or output layer unit j {
$I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
$O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
// Backpropagate the errors:
for each unit j in the output layer
$Err_j = O_j(1-O_j)(T_j - O_j)$; // compute the error
for each unit j in the hidden layers, from the last to the first hidden layer
$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
for each weight w_{ij} in *network* {
$\Delta w_{ij} = (l) Err_j O_i$; // weight increment
$w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
for each bias θ_j in *network* {
$\Delta \theta_j = (l) Err_j$; // bias increment
$\theta_j = \theta_j + \Delta \theta_j$; } // bias update
} }

How A Multi-Layer Neural Network Works?

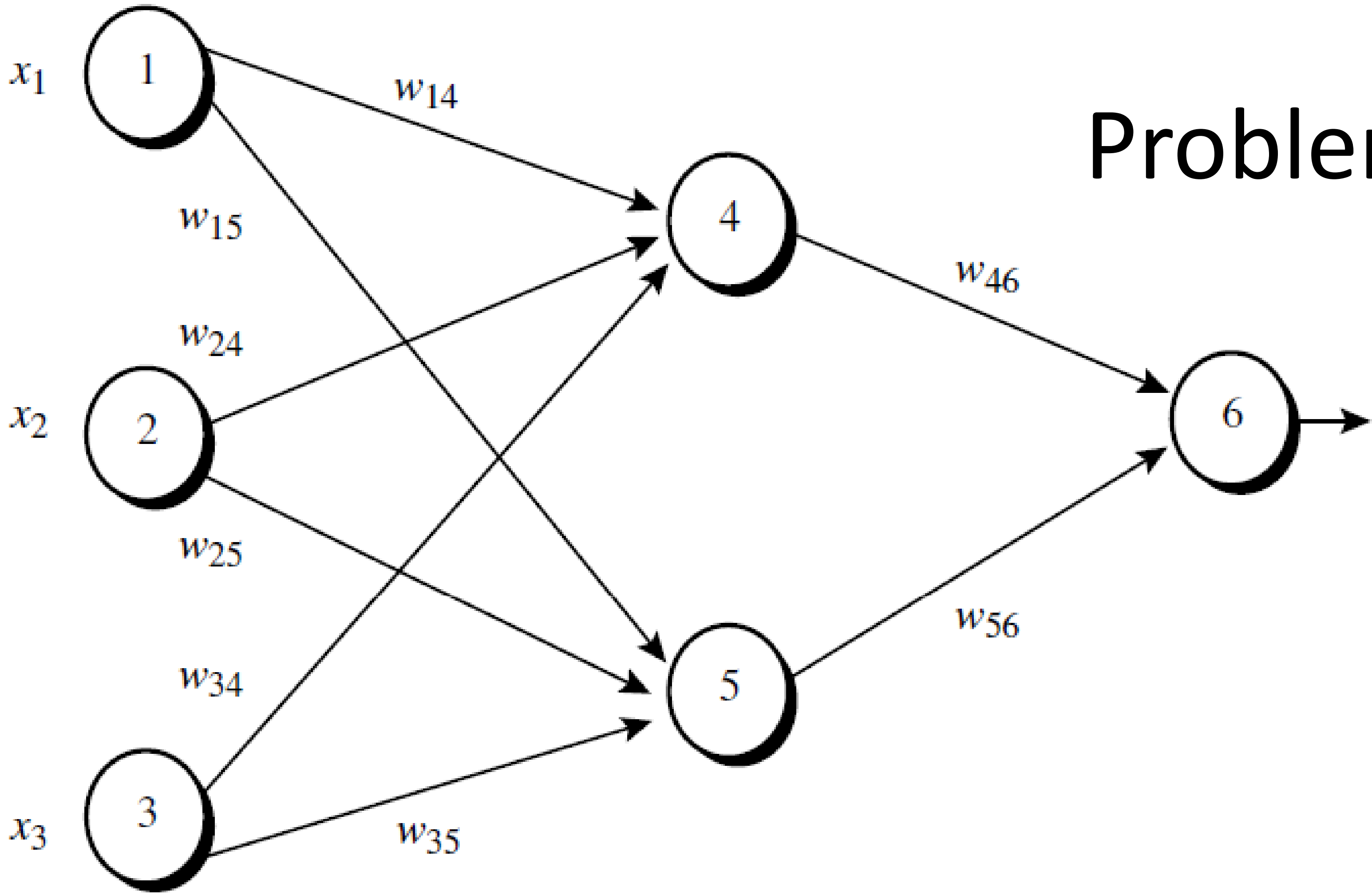
- The **inputs to the network correspond to the attributes measured for each training tuple**
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the **output layer, which emits the network's prediction**
- The network is **feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer**
- From a statistical point of view, networks perform **nonlinear regression: Given enough hidden units and enough training samples, they can closely approximate any function**

Neural Networks - Weakness & Strength

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or ``structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of ``hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs and outputs
  - Successful on a wide array of real-world data
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks

Defining a Network Topology

- First decide the **network topology: # of units in the input layer, # of hidden layers(if > 1), # of units in each hidden layer, and # of units in the output layer**
- Normalizing the input values for each attribute measured in the training tuples to [0.0—1.0]
- One **input unit per domain value.**
- **Output: if for classification and more than two classes, one output unit per class is used**
- Once a network has been trained and its accuracy is **unacceptable, repeat the training process with a different network topology or a different set of initial weights**



Problem

Table 6.3 Initial input, weight, and bias values.

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Solution

Table 6.4 The net input and output calculations.

Unit j	*Net input*, I_j	*Output*, O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

Table 6.5 Calculation of the error at each node.

Unit j	*Err_j*
6	$(0.474)(1-0.474)(1-0.474)=0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

Table 6.6 Calculations for weight and bias updating.

Weight or bias	*New value*
w_{46}	$-0.3+(0.9)(0.1311)(0.332)=-0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525)=-0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1)=0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1)=-0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0)=0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0)=0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1)=-0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1)=0.194$
θ_6	$0.1+(0.9)(0.1311)=0.218$
θ_5	$0.2+(0.9)(-0.0065)=0.194$
θ_4	$-0.4+(0.9)(-0.0087)=-0.408$