Advanced Classifiers Backpropagation

P.Mirunalini

Department of cse

SSNCE

Classification by Backprogration

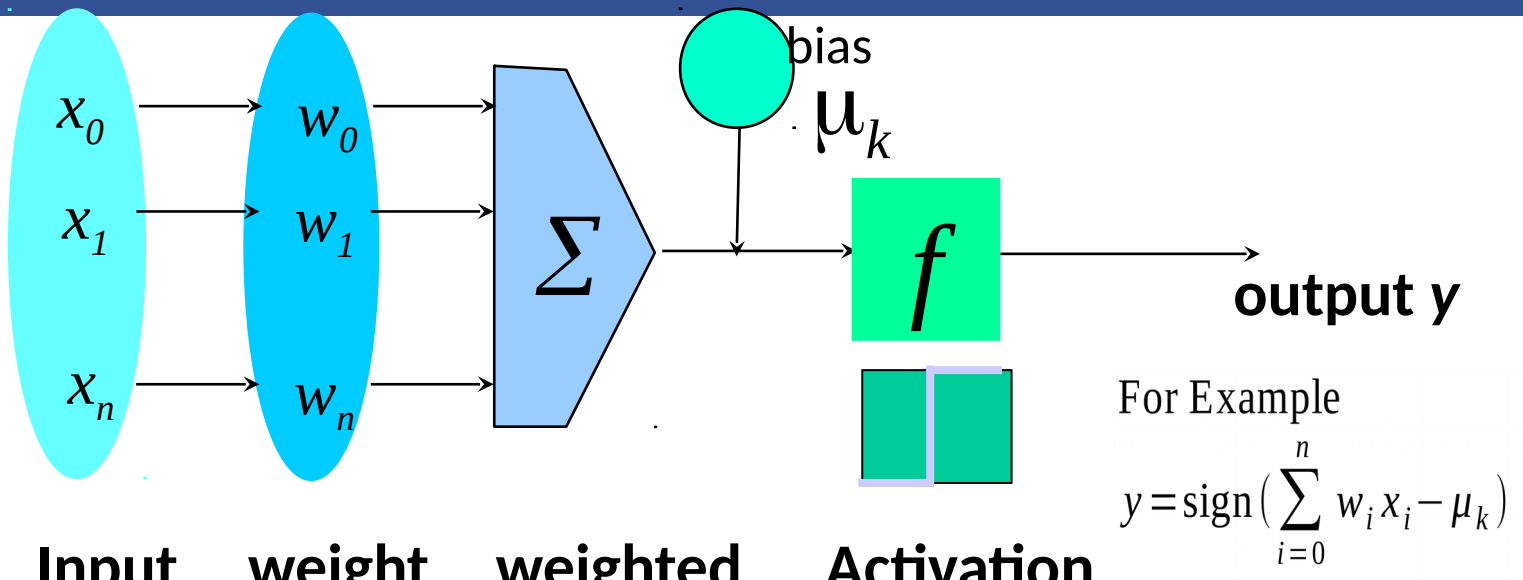
- Backpropagation: A neural network learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogs of neurons
- A neural network:
  - A set of connected input/output units where each connection has a weight associated with it
  - During the learning phase, 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

Advantages of Neural Networks

- High tolerance of noisy data
- Used when little knowledge of relationships between attributes and classes
- Suited for continuous-valued inputs and outputs
- **Applications:** handwritten character recognition, pathology and laboratory medicine, training a computer to pronounce English text.
- Parallelization techniques can be used to speed up the computation process.

SSN

Neuron: A Hidden/Output Layer Unit



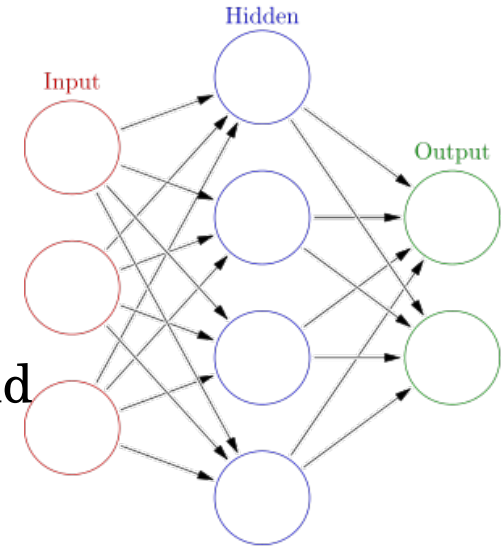
- An n-dimensional input vector x is mapped into variable y by means of the scalar product and a nonlinear function mapping
- The inputs to unit are outputs from the previous layer.
- They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit.
- Then a nonlinear activation function is applied to it.

Multilayer Feed-Forward Neural Network

- Backpropogation performs learning on multilayer feed forward network
- It iteratively learns a set of weights for prediction of the class label of tuples.
- **A multilayer feed-forward neural network consists of an input layer, one or more hidden layers, and an output layer.**
- Each layer is made up of units
- The inputs to the network correspond to the attributes measurement for each training tuple and are fed simultaneously into the units making up the input layer
- The inputs passed through the input layer and are then weighted and fed simultaneously to a hidden layer

How a Multi-Layer Neural Network Works

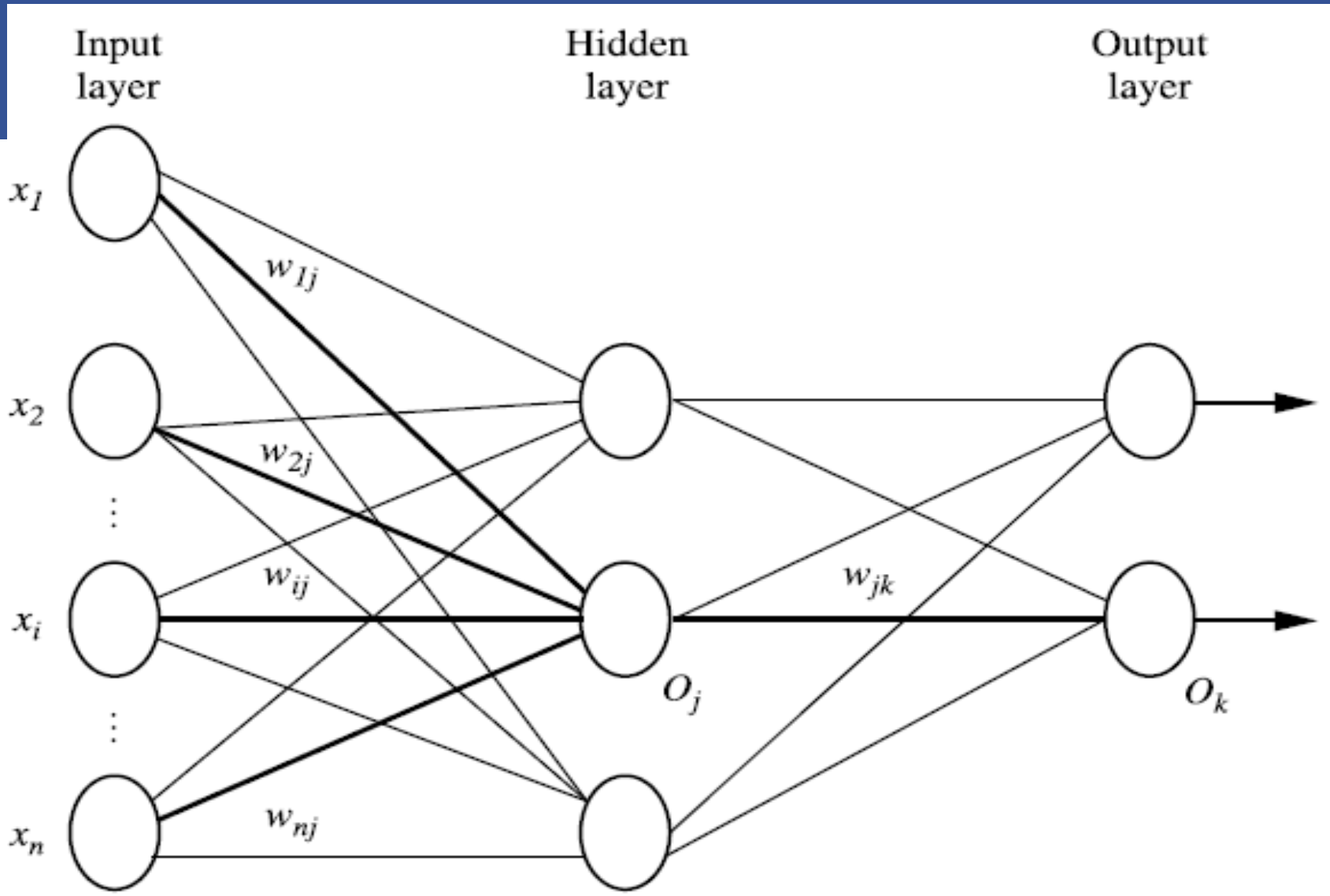
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction for the tuples.
- The units in the input layer called as input units and output and hidden layers units are as neurodes.
- **The network is feed-forward:** None of the weights cycles back to an input unit or to a previous layer
- Fully connected : Each unit provides input to each unit in the next forward layer



SSN

How a Multi-Layer Neural Network Works

- Multilayer neural network has n-layer neural network based on the count of hidden layers and output layer.
- Each output unit
  - takes a input as weighted sum of the outputs from previous layer
  - Apply nonlinear (activation) function to the weighted input
- Multilayer feed forward neural networks are able to model the class prediction as a nonlinear combination of inputs.
- From a statistical point of view, networks perform **nonlinear regression**
  - Given enough hidden units and enough training samples, they can closely approximate any function



A multilayer feed-forward neural network.

Defining a Network Topology

- Decide the network topology
  - Specify # of units in the input layer, # of hidden layers (if > 1), # of units in each hidden layer, and # of units in the output layer
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- Discrete-valued attributes has been encoded as one input unit per domain value.
- Output:if binary classification one output unit used to represent two classes for more than two classes, one output unit per class is used
- The network design is a trail and error process and may affect accuracy of resulting trained network
- Repeat the training process with a different network topology or a different set of initial weights when accuracy is unacceptable after training the n/w

Back Propagation

- Back propagation:
  - Learns Iteratively by processing a set of training tuples
  - compares the network's prediction with the actual known target value
  - For each training tuple, the weights are modified to minimize the mean squared error between the network's prediction and the actual target value
  - Modifications are made in the “backwards” direction: from the output layer, through each hidden layer down to the first hidden layer, hence “back propagation ”

Back Propagation Algorithm

- Steps
  - Initialize weights to small random numbers, associated with biases.
  - Propagate the inputs forward (by applying activation function)
  - Back propagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Initialize the Weights

- The weights in the network are initialized to small random numbers
- e.g., ranging from -1.0 to 1.0, or -0.5 to 0.5
- Each unit has a bias associated with it and are initialized to small random numbers.
- Each training tuple, X, is processed as follows:

Propagate the Inputs : Forward

- The training tuple is fed to the network's input layer.
- The inputs pass through the input units remains unchanged.
- For an input unit, j its output, Oj , is equal to its input value, Ij
- The net input and output of each unit in the hidden or output layers is computed as a linear combination of its inputs.
- To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed with biased.
- Given a unit, j in a hidden or output layer, the net input, I_j , to unit j is

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

Propagate the Inputs : Forward

- Each unit in the hidden and output layers takes its net input and then applies an activation function.
- The function symbolizes the activation of the neuron represented by the unit. (**logistic, or sigmoid, function is used.**)
- Given the net input I_j to unit j, then Oj , the output of unit j, is computed as
- Function is referred as squashing function

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

- It maps larger input domain onto smaller range of 0 to 1.

Backpropogate the Error

- The error is propagated backward by updating the weights and biases to reflect the error of the network's prediction.
- For a unit j in the output layer, the error Err_j is computed by

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

  - Oj(1-Oj) is the derivative of logistic function
- To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit j in the next layer are considered.
- The error of a hidden layer unit j is

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

Backpropogate the Error

- The weights and biases are updated to reflect the propagated errors.
- Weights are updated by the following equations,

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

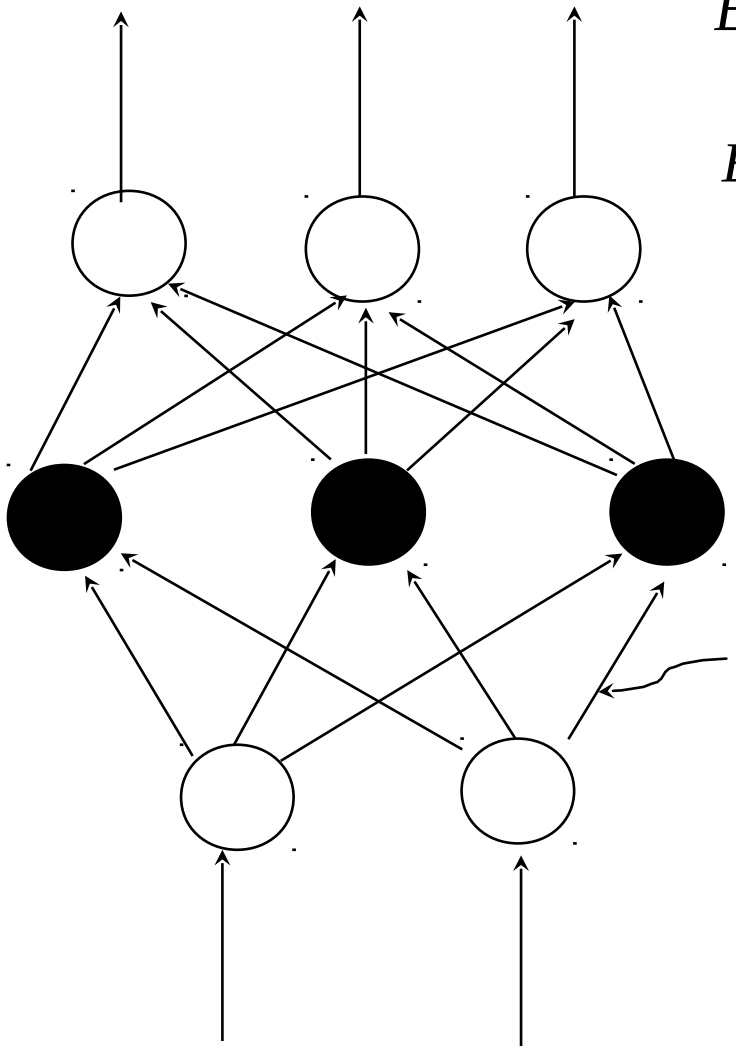
l is the learning rate and learns from gradient descent method to minimize the mean squared distance between the prediction and target value.

- Biases are updated by the following equations
  - the updates are called as Echo updating

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

SSN

A Multi-Layer Feed-Forward Neural Network



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

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

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

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

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

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

Terminating condition

- Training stops when
  - All Δw_{ij} in the previous epoch are so small as to be below some specified threshold.
  - The percentage of tuples misclassified in the previous epoch is below some threshold,
  - A prespecified number of epochs has expired.

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:

(1) Initialize all weights and biases in *network*;
(2) while terminating condition is not satisfied {
(3) for each training tuple X in D {
(4) // Propagate the inputs forward:
(5) for each input layer unit j {
(6) $O_j = I_j$; // output of an input unit is its actual input value
(7) for each hidden or output layer unit j {
(8) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
(9) $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
(10) // Backpropagate the errors:
(11) for each unit j in the output layer
(12) $Err_j = O_j(1-O_j)(T_j - O_j)$; // compute the error
(13) for each unit j in the hidden layers, from the last to the first hidden layer
(14) $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
(15) for each weight w_{ij} in *network* {
(16) $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
(17) $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
(18) for each bias θ_j in *network* {
(19) $\Delta \theta_j = (l) Err_j$; // bias increment
(20) $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
(21) } }

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

From Neural Networks to Deep Learning

- Train networks with many layers (vs. shallow nets with just a couple of layers)
- Multiple layers work to build an improved feature space
  - First layer learns 1st order features (e.g., edges, …)
  - 2nd layer learns higher order features (combinations of first layer features, combinations of edges, etc.)
  - In current models, layers often learn in an unsupervised mode and discover general features of the input space—serving multiple tasks related to the unsupervised instances (image recognition, etc.)
  - Then final layer features are fed into supervised layer(s)
    - And entire network is often subsequently tuned using supervised training of the entire net, using the initial

Problem

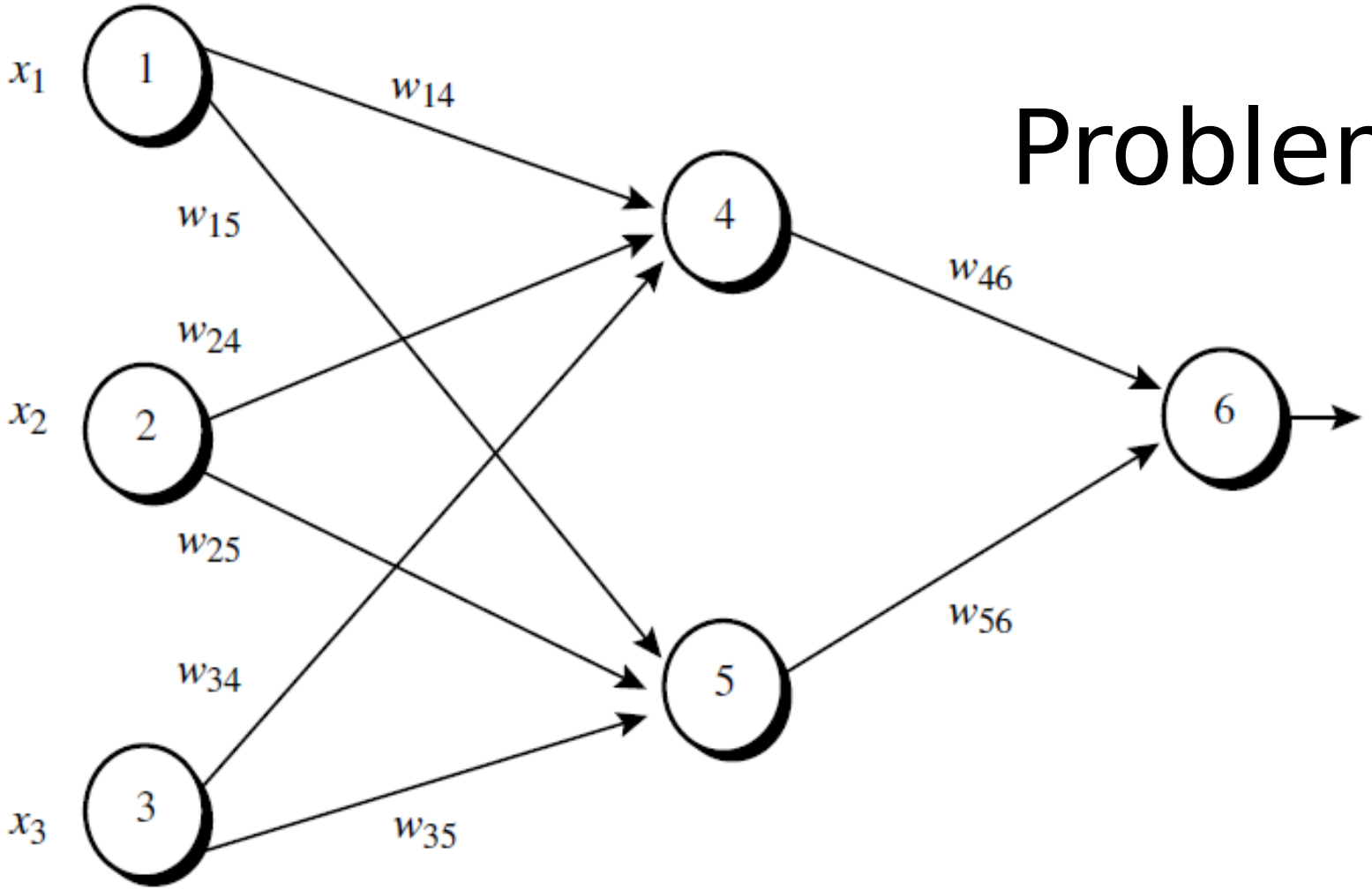


Table 6.3 Initial input, weight, and bias values.

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.

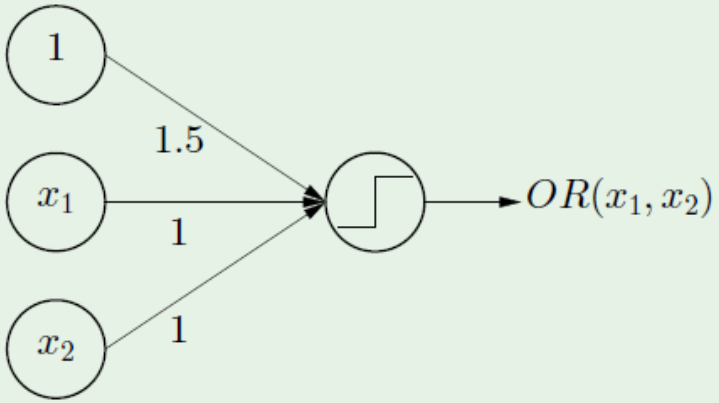
Unit j	*Err_j*
6	$(0.474)(1-0.474)(1-0.474)=0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

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	$-0.4+(0.9)(-0.0087)=-0.408$

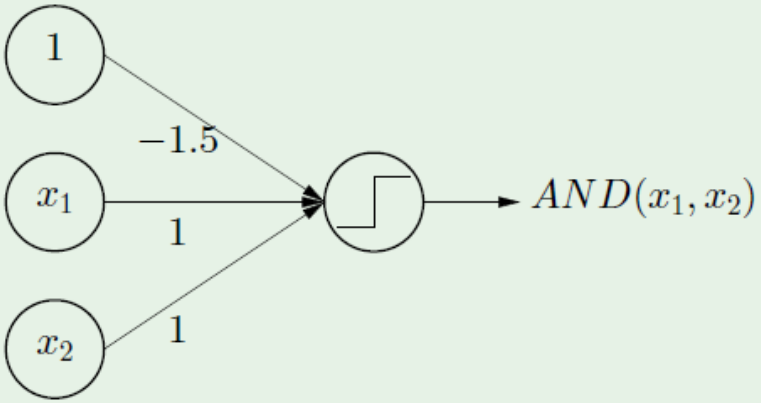
Perceptron - Example

Threshold >=1.5



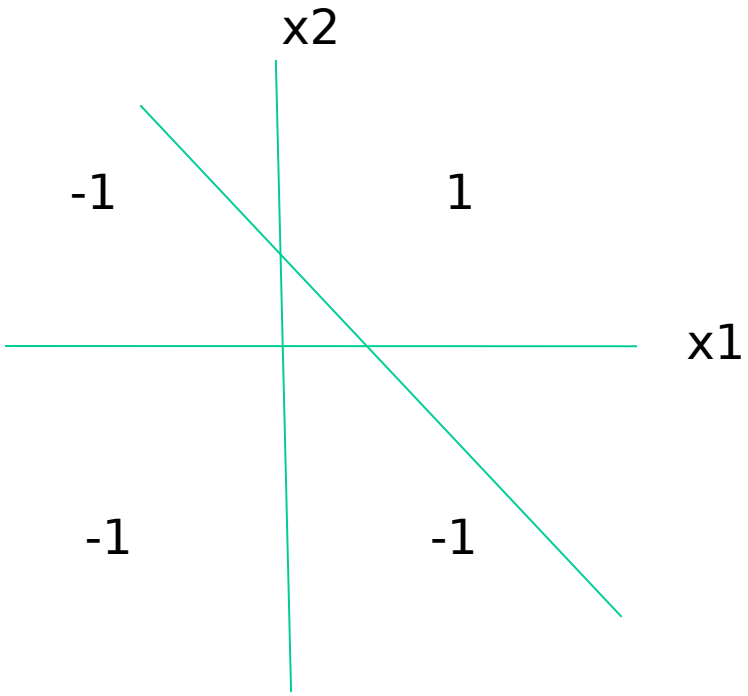
X1	x2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

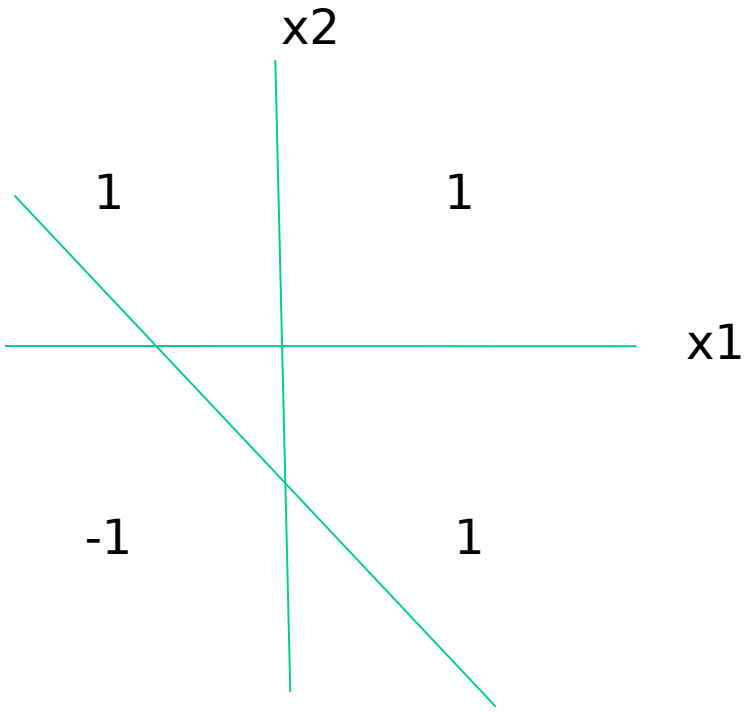
Threshold > -1.5



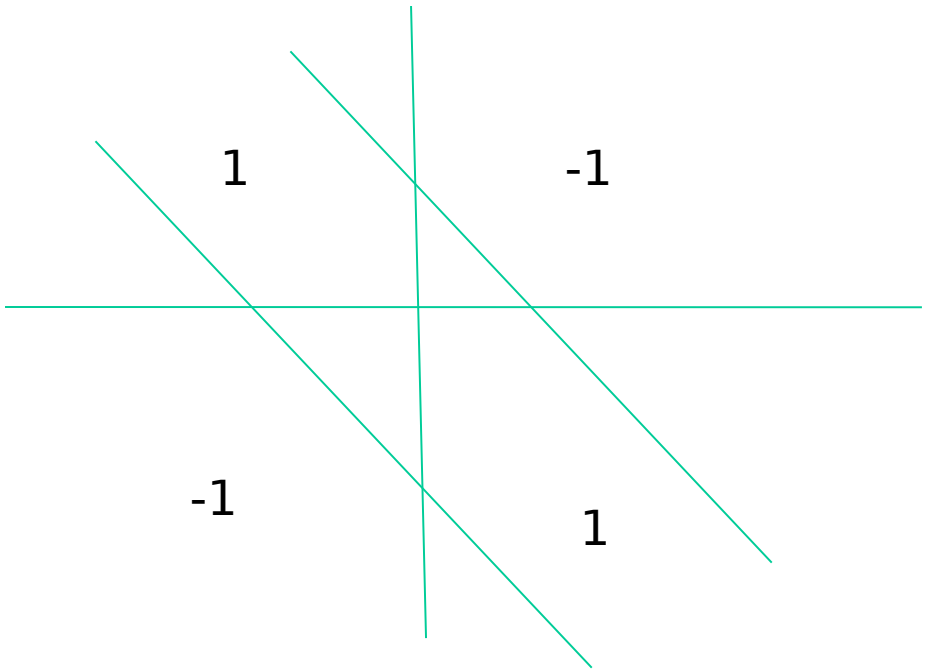
X1	x2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

AND ---OR

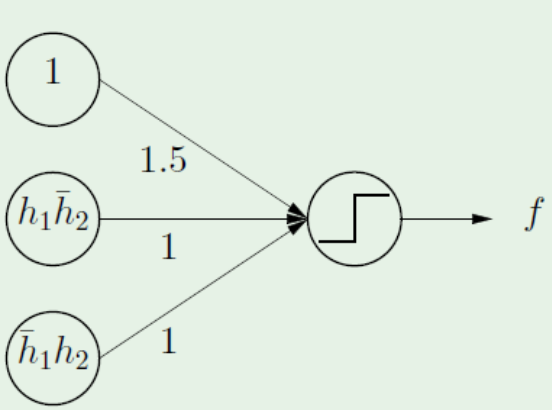


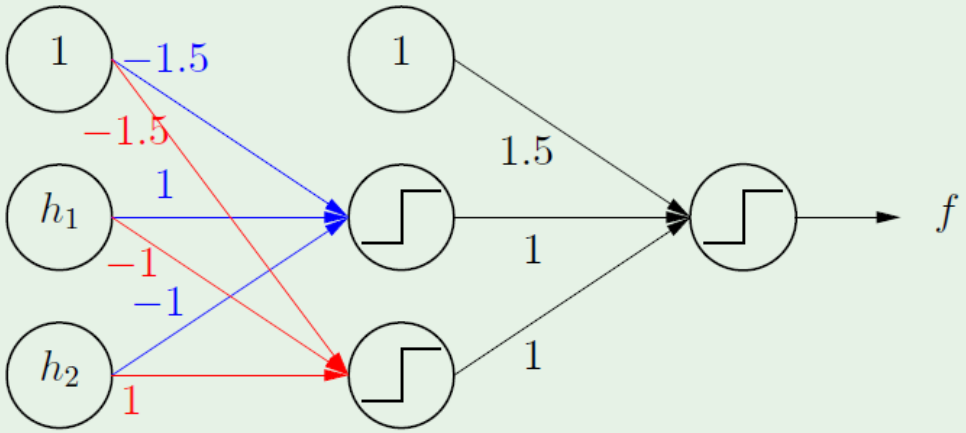


XOR



XOR Problem





A Powerful model