Artificial Neural Network———----- Back-propagation learning

Supervised Learning

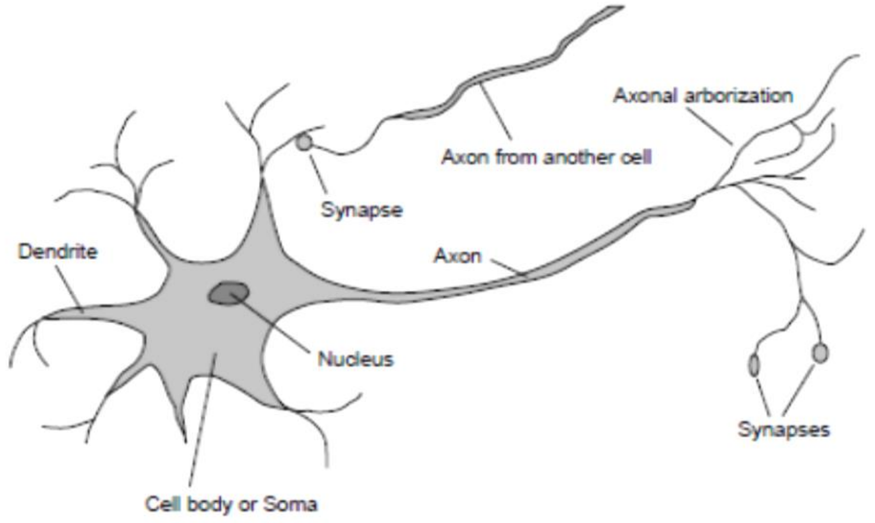
BY

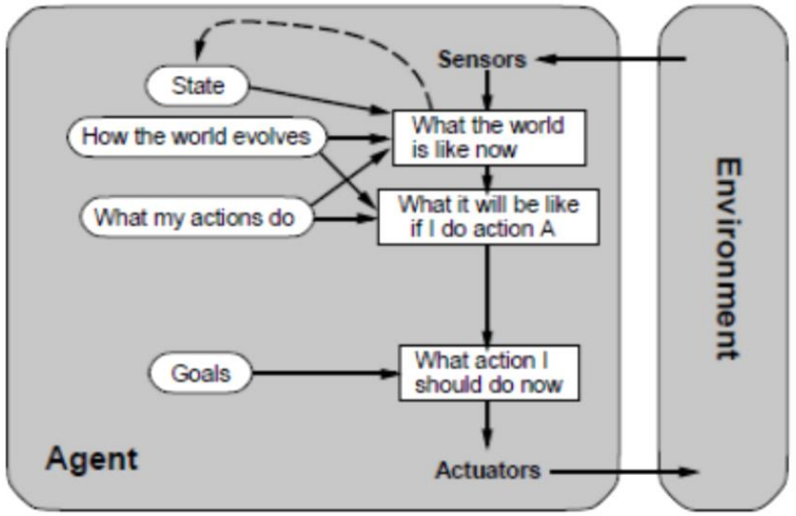
DR. ANUPAM GHOSH

21ST SEPT, 2024

History of AI

mid-80's




Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

- NN are back
- Statistical approaches rise
- AI is a science (formalization, specialization, complexity, ...)
- From mid 80's to mid 90's : **AI winter**
  - Over-optimistic promises?
  - Funding agencies (public and private) have expected too much
- Since mid 90's: unifying approach « intelligent agent »

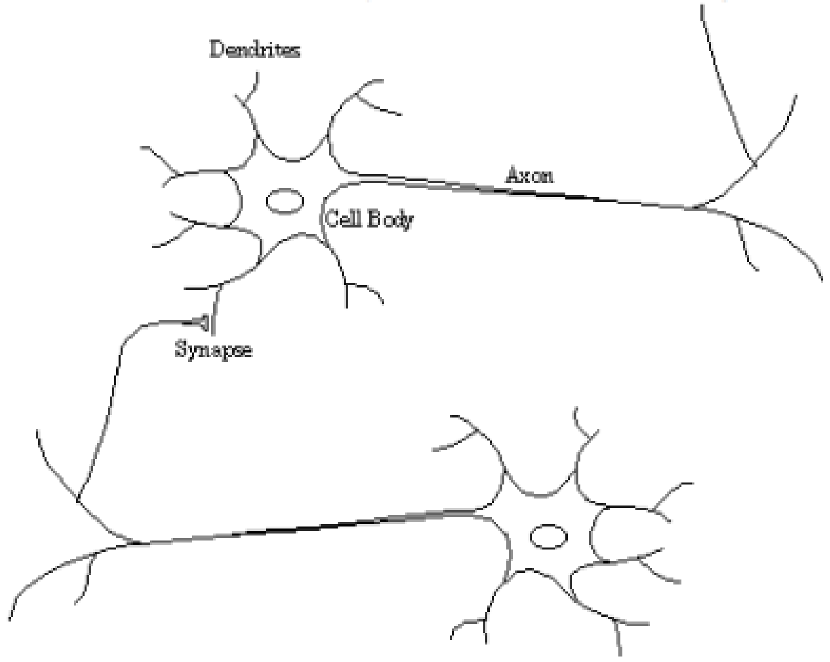
Biological Inspirations

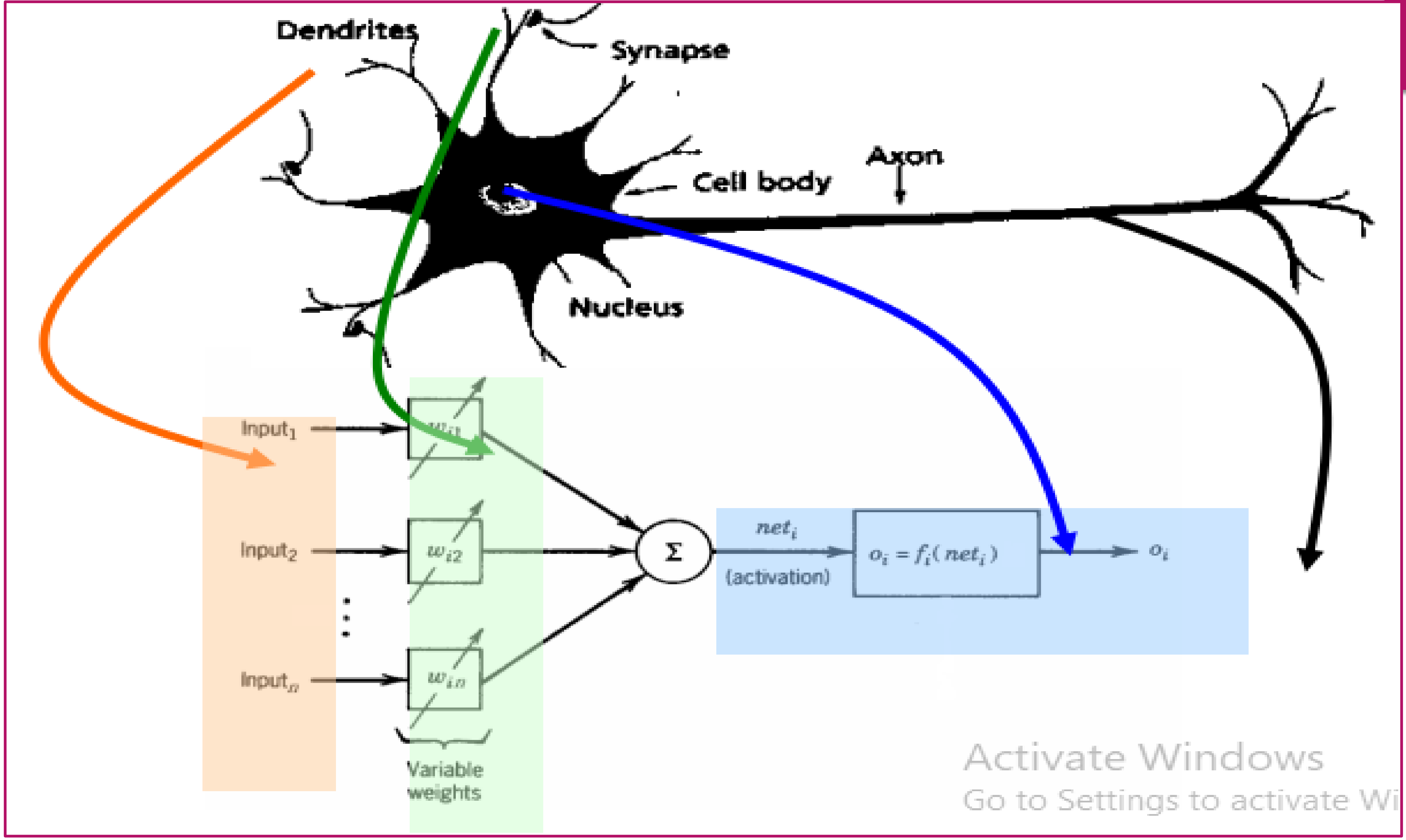
Humans perform complex tasks like vision, motor control, or language understanding very well.

One way to build intelligent machines is to try to imitate the (organizational principles of) human brain.

Biological Neuron

- dendrites: nerve fibres carrying electrical signals to the cell
- cell body: computes a non-linear function of its inputs
- axon: single long fiber that carries the electrical signal from the cell body to other neurons
- synapse: the point of contact between the axon of one cell and the dendrite of another, regulating a chemical connection whose strength affects the input to the cell.




Dendrites
Synapse
Axon
Cell body
Nucleus
Input$_1$
Input$_2$
Input$_n$
w_{i1}
w_{i2}
w_{in}
Variable weights
Σ
net_i
(activation)
$o_i = f_i(net_i)$
o_i

Artificial Neural Networks

Computational models inspired by the human brain:

- Massively parallel, distributed system, made up of simple processing units (neurons)
- Synaptic connection strengths among neurons are used to store the acquired knowledge.
- Knowledge is acquired by the network from its environment through a learning process

Properties of ANNs

Learning from examples

- labeled or unlabeled

Adaptivity

- changing the connection strengths to learn things

Non-linearity

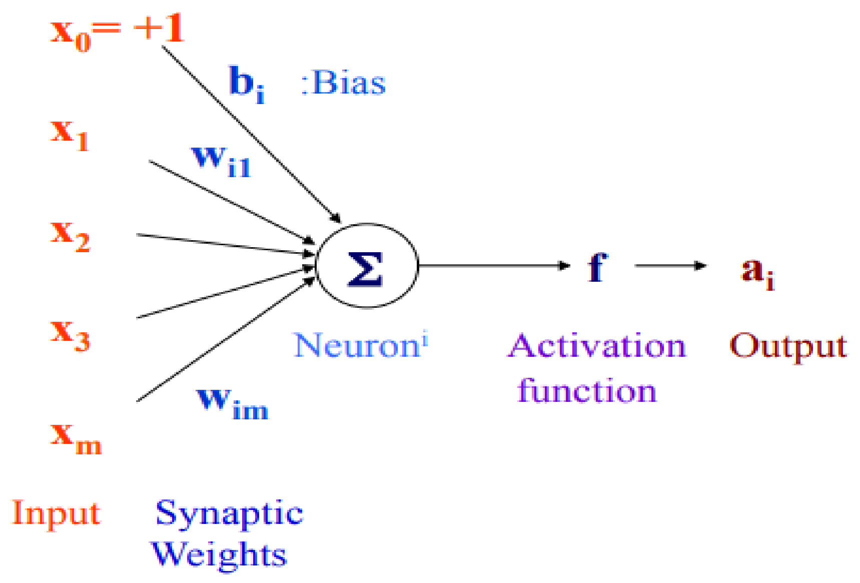
- the non-linear activation functions are essential

Fault tolerance

- if one of the neurons or connections is damaged, the whole network still works quite well

Thus, they might be better alternatives than classical solutions for problems characterised by:

- high dimensionality, noisy, imprecise or imperfect data; and
- a lack of a clearly stated mathematical solution or algorithm



Bias

$$a_i = f(n_i) = f\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right)$$

An artificial neuron:

- computes the weighted sum of its input (called its **net input**)
- adds its bias
- passes this value through an activation function

We say that the neuron "fires" (i.e. becomes active) if its output is above zero.

Activation Functions

- Identity

  $f(x) = x$

- Binary step

  $f(x) = 1$ if $x >= \theta$

  $f(x) = 0$ otherwise

- Binary sigmoid

  $f(x) = 1 / (1 + e^{-\sigma x})$

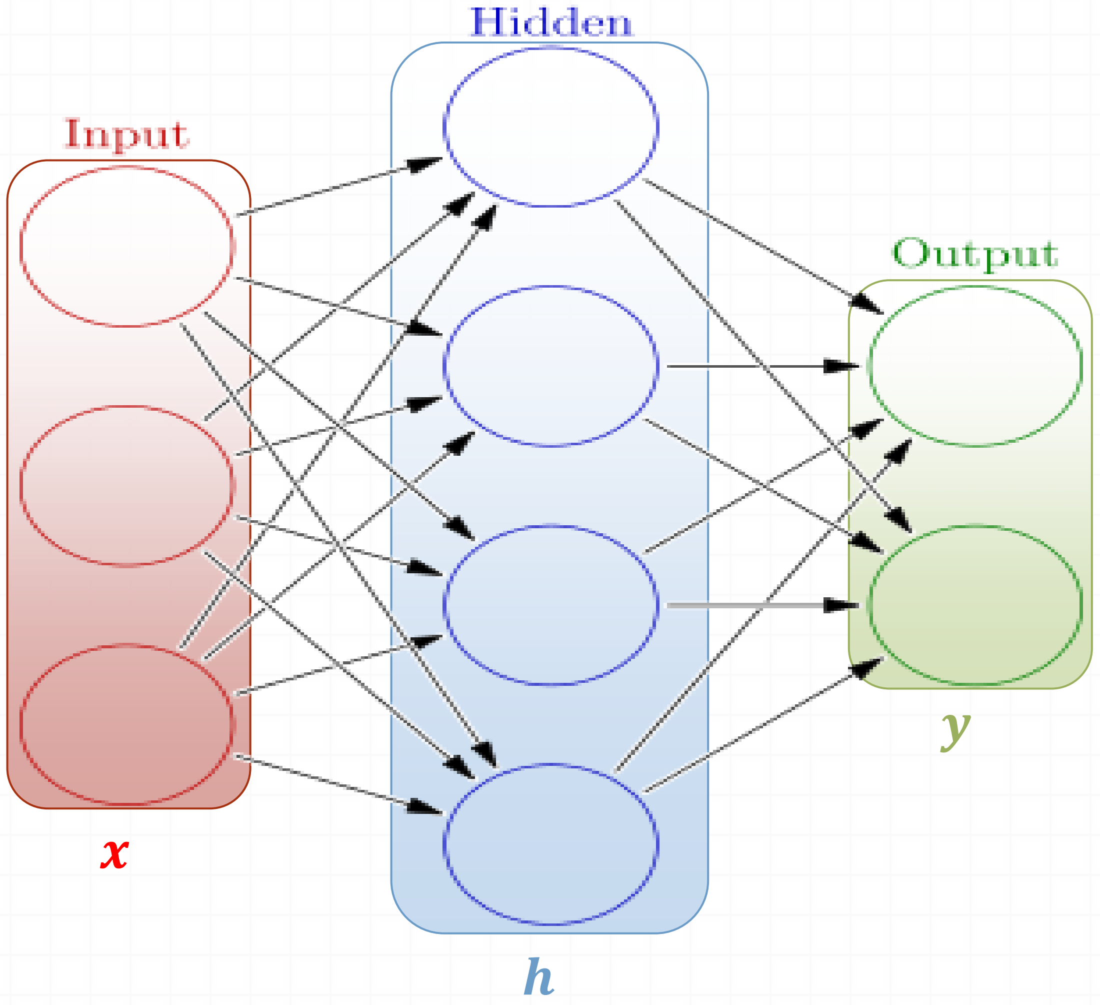
- Bipolar sigmoid

  $f(x) = -1 + 2 / (1 + e^{-\sigma x})$

- Hyperbolic tangent

  $f(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$

Neural Network Intro



Weights

$$h = \sigma(W_1 x + b_1)$$

$$y = \sigma(W_2 h + b_2)$$

Activation functions

How do we train?

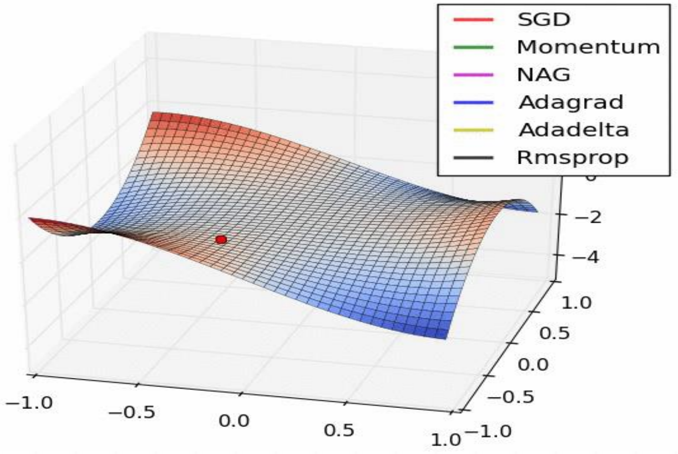
4 + 2 = 6 neurons (not counting inputs)
[3 x 4] + [4 x 2] = 20 weights
4 + 2 = 6 biases

26 learnable **parameters**

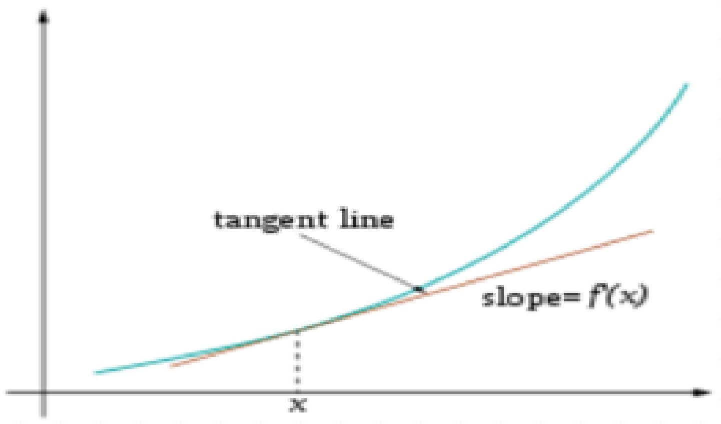
Demo

Training

$$\begin{tikzcd}
\text{Sample labeled data (\textbf{batch})} \arrow[r] & \text{\textbf{Forward} it through the network, get predictions} \arrow[r] & \text{\textbf{Back-propagate} the errors} \arrow[r] & \text{\textbf{Update} the network weights} \arrow[lll, bend left]
\end{tikzcd}$$

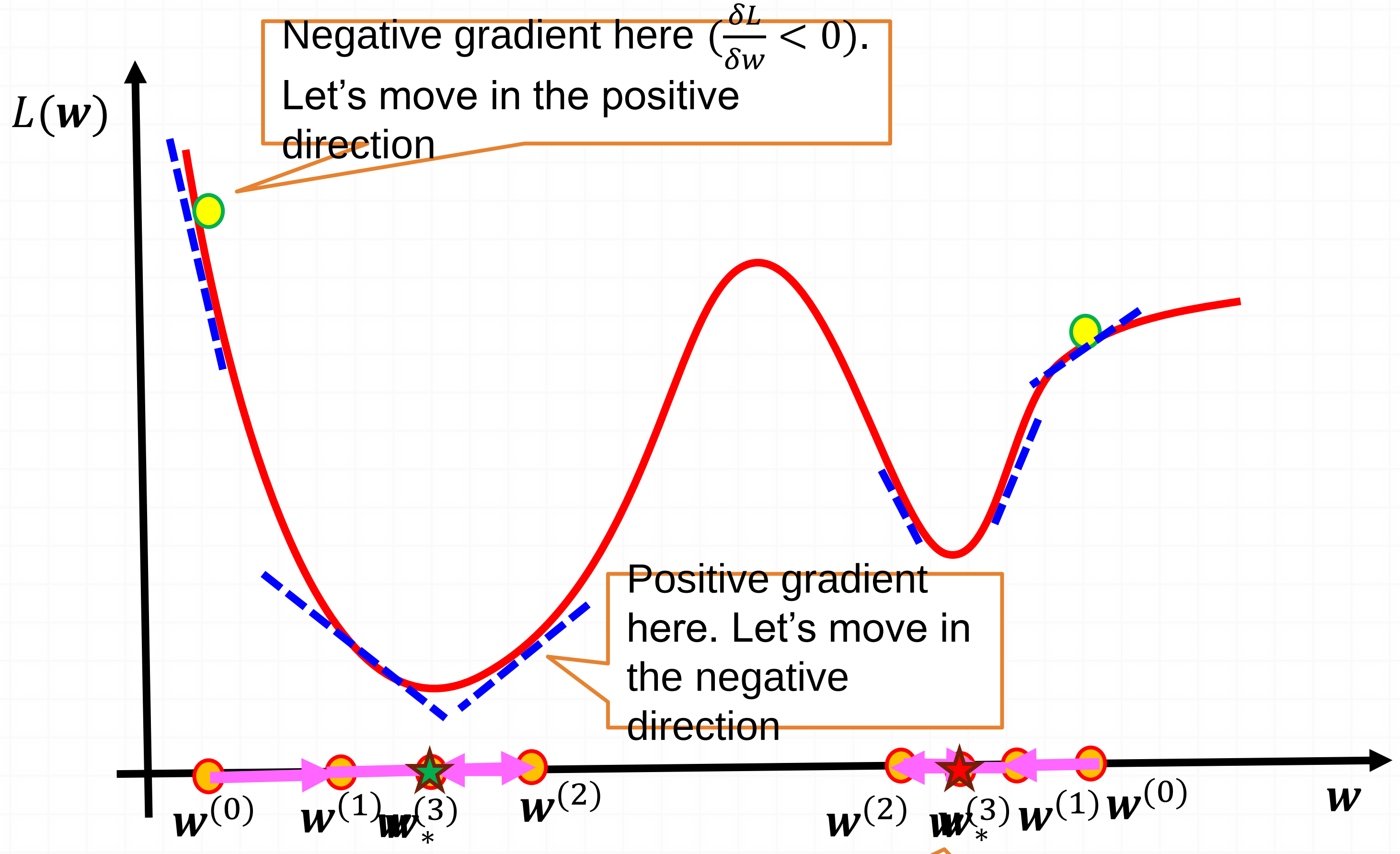


Optimize (min. or max.) **objective/cost function $J(\theta)$**
Generate **error signal** that measures difference between predictions and target values



Use error signal to change the **weights** and get more accurate predictions
Subtracting a fraction of the **gradient** moves you towards the **(local) minimum of the cost function**

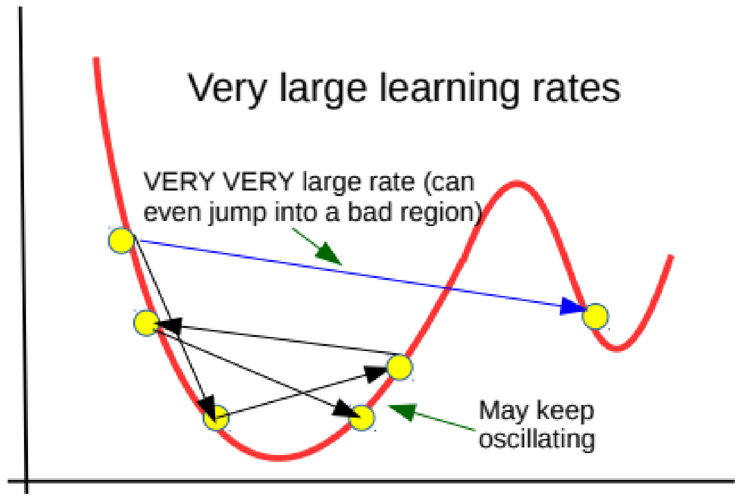
Gradient Descent: An Illustration

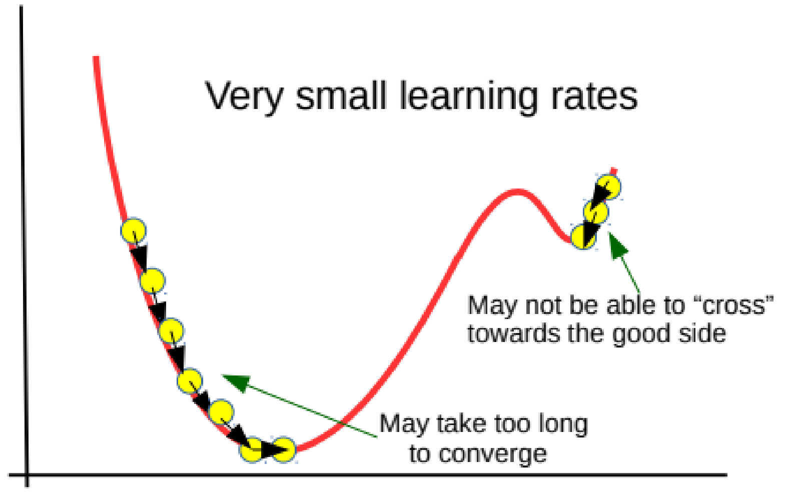


Stuck at a local minima

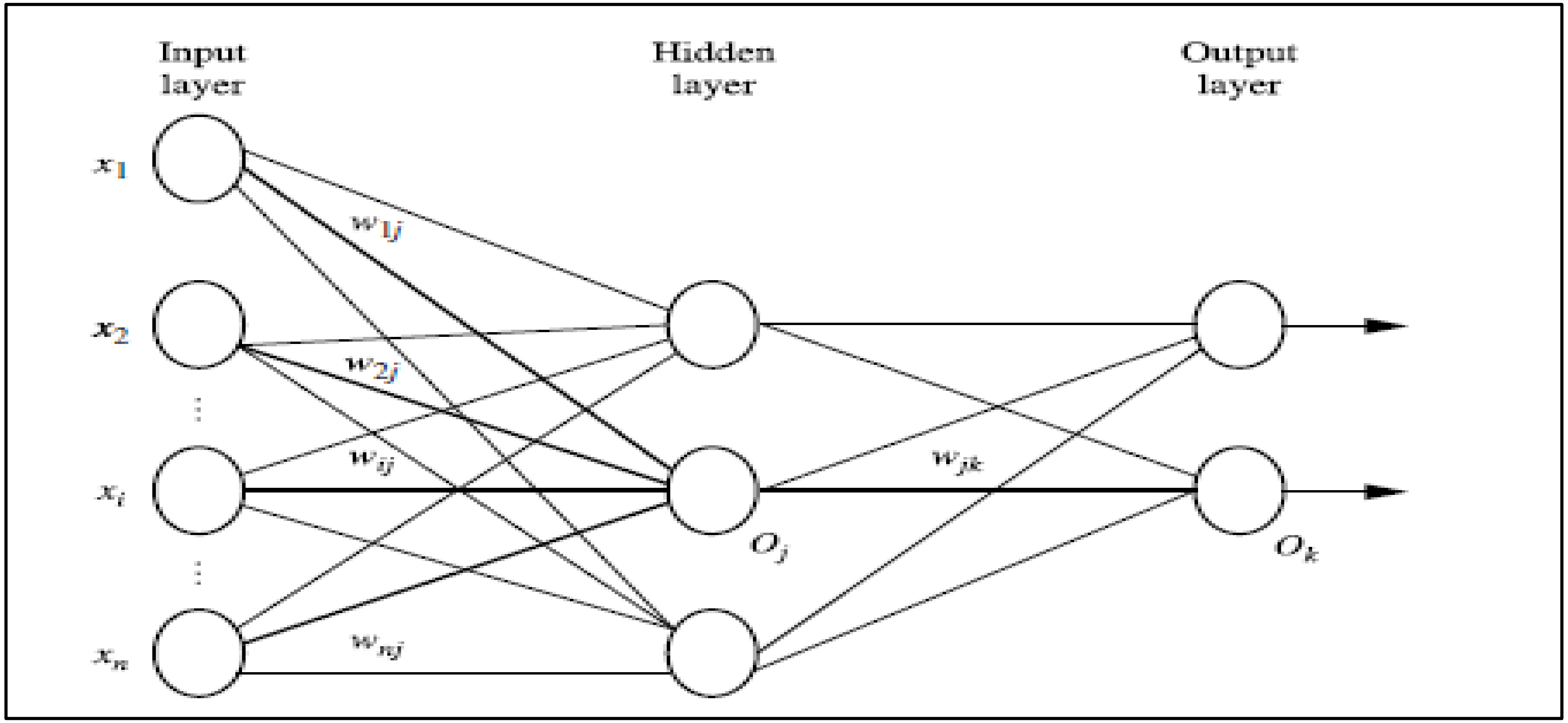
Good initialization is very important

Learning rate is very important

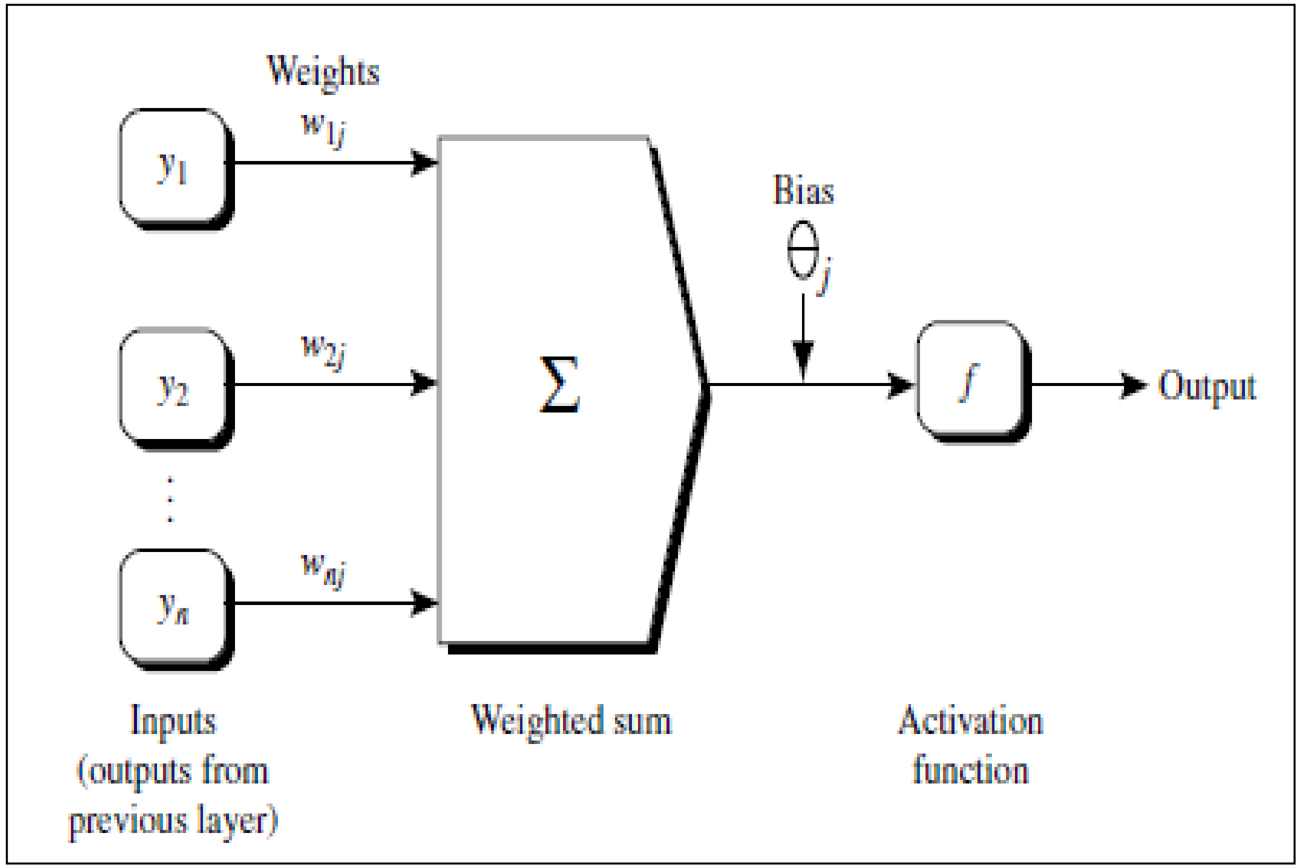

Very large learning rates
VERY VERY large rate (can even jump into a bad region)
May keep oscillating


Very small learning rates
May not be able to "cross" towards the good side
May take too long to converge

Multilayer Feed-Forward Neural Network



Back propagation Learning



- **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. The biases are similarly initialized to small random numbers
- Each training tuple, ***X***, is processed by the following steps.
- **Propagate the inputs forward:**

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

Calculate Output: Activation: Sigmoid

- Property of Sigmoid function: (Activation)
- Takes a real-valued number and "squashes" it into range between 0 and 1.
- Nice interpretation as the firing rate of a neuron

  0 = not firing at all

  1 = fully firing

- Sigmoid neurons saturate and kill gradients, thus NN will barely learn

  when the neuron's activation are 0 or 1

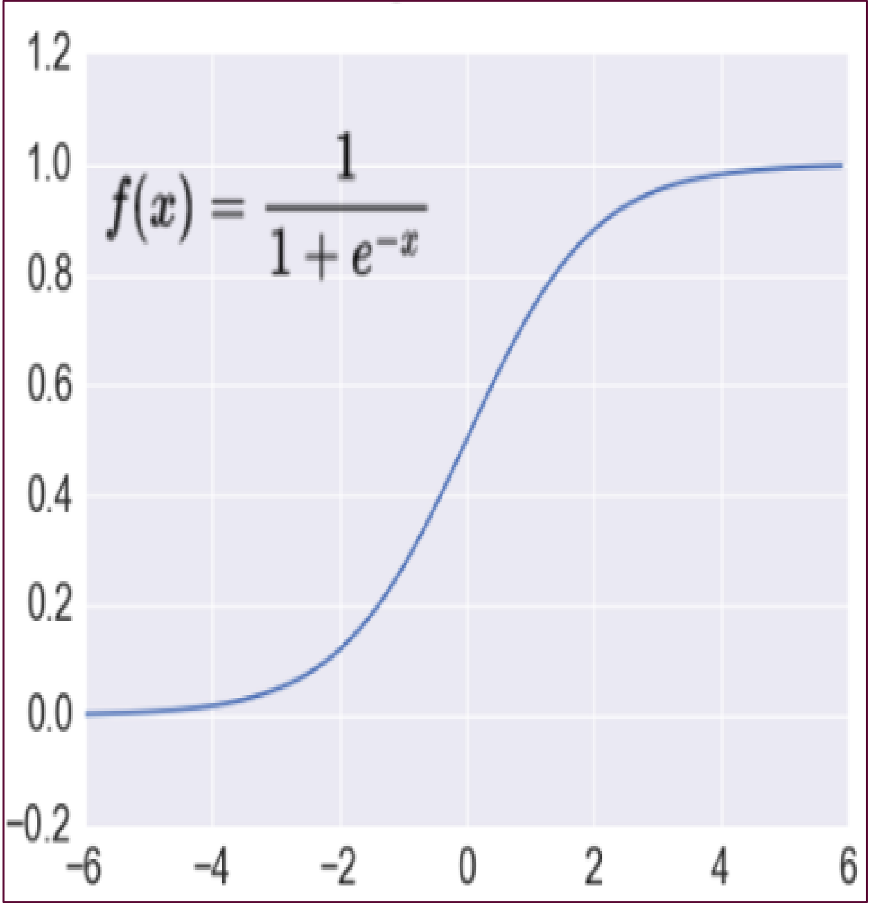
  (saturate)

  gradient at these regions almost zero

  almost no signal will flow to its weights

  if initial weights are too large then most

  neurons would saturate



- Calculate the output using Sigmoid function: (Activation)

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

Back propagate 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 is computed by

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

- 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

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

w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer, and Err_k is the error of unit k

Updating of weights and biases

The weights and biases are updated to reflect the propagated errors

The variable l is the learning rate, a constant typically having a value between 0.0 and 1.0

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

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

Back propagation learns using a gradient descent method to search for a set of weights that fits the training data so as to minimize the mean-squared distance between the network's class prediction and the known target value of the tuples

The learning rate helps avoid getting stuck at a local minimum in decision space (i.e., where the weights appear to converge, but are not the optimum solution) and encourages finding the global minimum

If the learning rate is too small, then learning will occur at a very slow pace. If the learning rate is too large, then oscillation between inadequate solutions may occur. A rule of thumb is to set the learning rate to 1=1/t , where t is the number of iterations through the training set so far.

Terminating condition

Training stops when

- All Δw in the previous epoch are so small as to be below some specified threshold,

  or

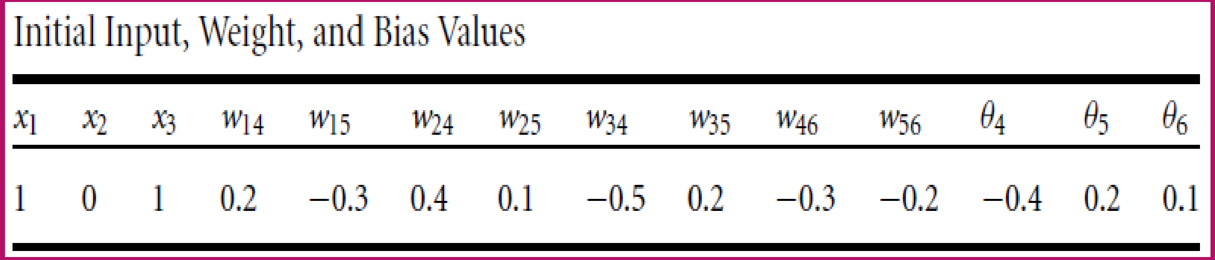
- The percentage of tuples misclassified in the previous epoch is below some threshold,

  or

- A pre-specified number of epochs has expired

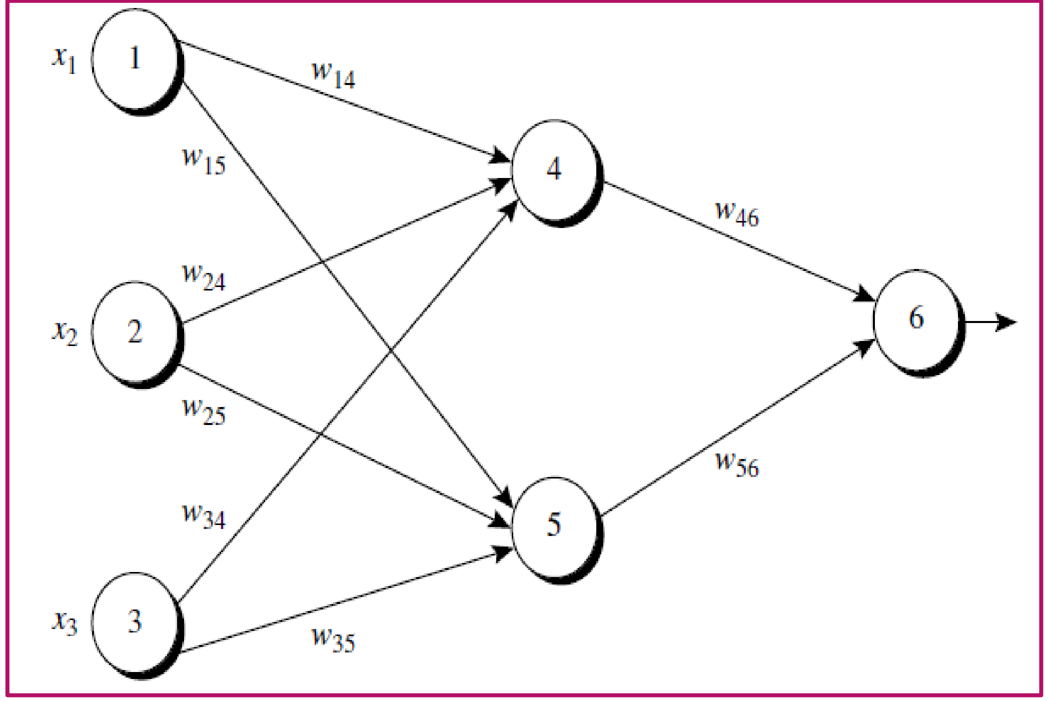
Demonstration:

Figure shows a multilayer feed-forward neural network. Let the learning rate be 0.9. The initial weight and bias values of the network are given in Table 9.1, along with the first training tuple, ***X*** = (1, 0, 1), with a class label of 1

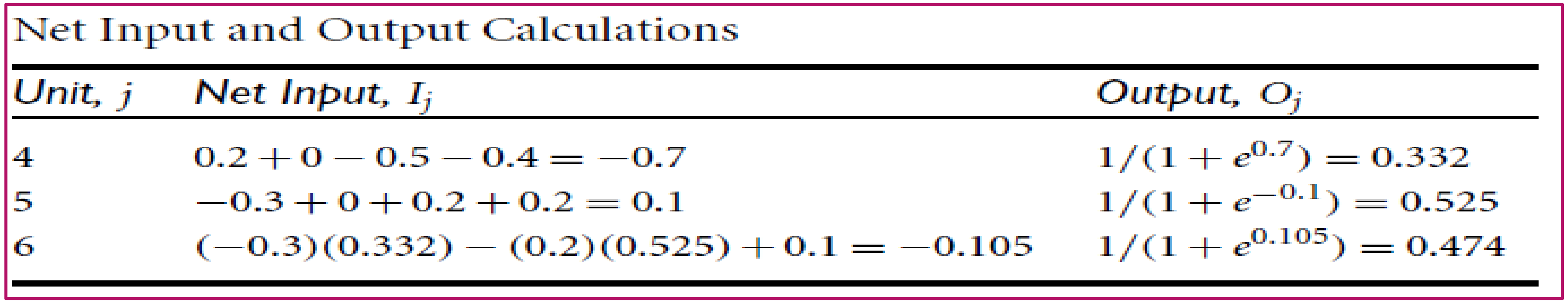


Initial Input, Weight, and Bias Values

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

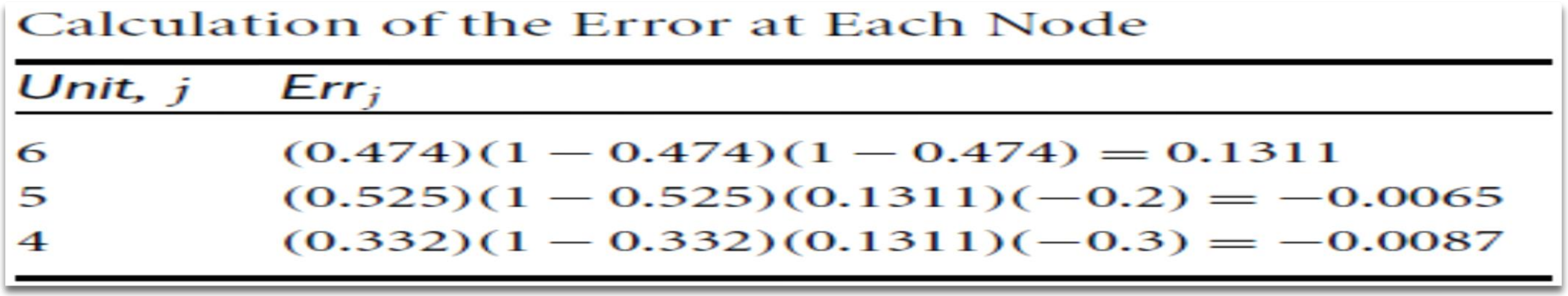


Solution:



Net Input and Output Calculations

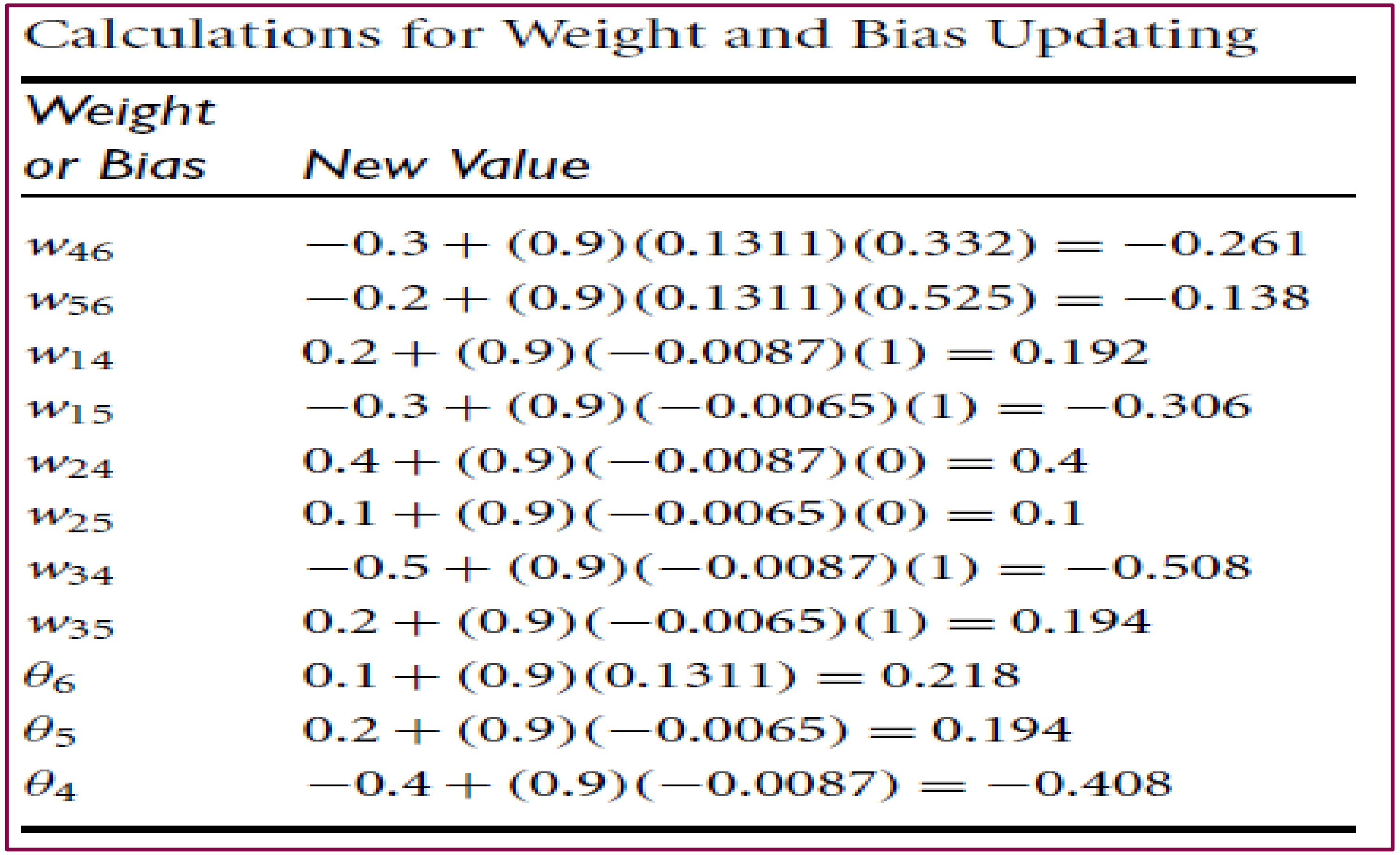
Unit, j	Net Input, I_j	Output, O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$



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	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

Weight and bias values after first iteration



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

Problem Statement

- Network topology:- 3-(2-2)-2
- Weight and bias values are initialized to 0
- For input set {1,0,1) the class label will be {1,0}

Q: What will be the updated weight and bias values after 2nd iteration if learning rate = 1 ?

It will be Continued....