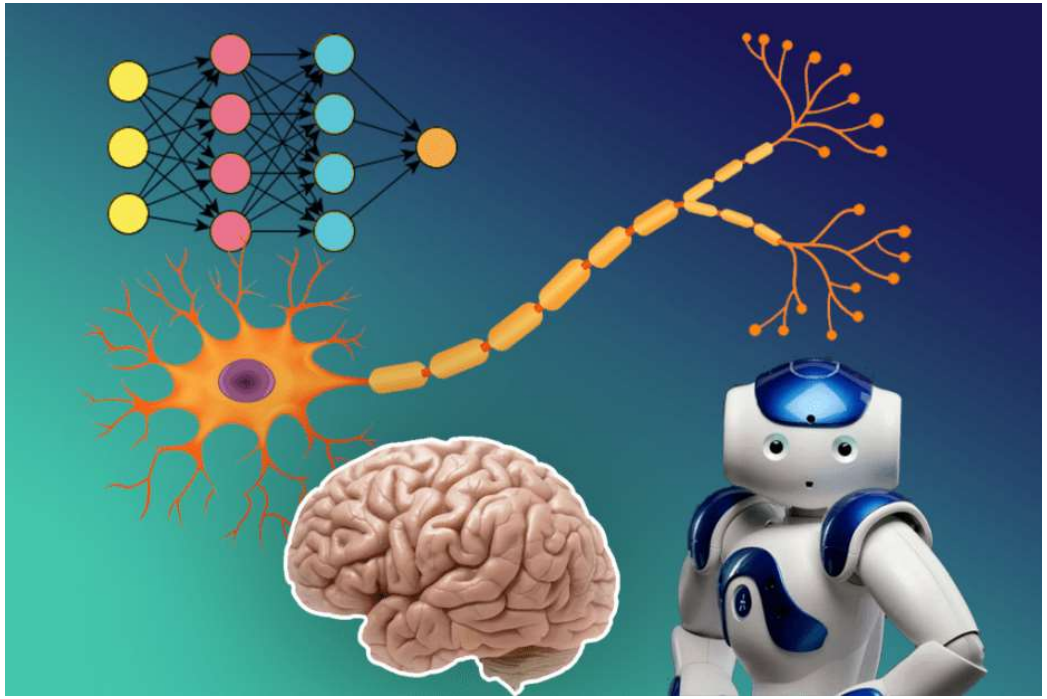
UNIT-II

(ARTIFICIAL NEURAL NETWORKS)



Artificial Neural Networks

Topics :

- Neural Network Representation
- Differences between Biological and Neural Network

ARTIFICIAL NEURAL NETWORK

- An **Artificial Neural Network(ANN)** may be defined as an **information – processing model** that is inspired by the **Biological Nervous System** , such as **Brain.**
- The model tries to **replicate only** the most **basic functions of the Brain.**
- An **ANN** is composed of a **large no. of highly inter connected processing units (Neurons)** working to solve the **specific Problem.**
- Like **Human Brain, ANN** is also a **Learn by Example**.

CONTD..

- Ie, Just as we **use our brains** to **identify patterns** and **classify different types of information**
- By **using the Neural Networks**, it allowing **computers to observe, learn, and react to complex situations faster than humans.**
- An **ANN is configured** as **different types of Applications**, such as **Face Recognition, Pattern Recognition etc**. through a **learning process.**

How ANN Works?

Working procedure of Human Brain and ANN:

- The term **"Artificial Neural Network"** is derived from **Biological neural networks** that develop the **structure of a human brain.**
- Similar to the **human brain** that has **neurons interconnected to one another.** A **neuron** is the fundamental unit which is used for to **is build the brain**
- A **typical brain** contains something like **100 billion microscopic cells** called **neurons. Each neuron** is made up of a **cell body (the central mass of the cell).** ie, the **sum of all inputs.**

CONTD..

- There are **over 100 billion neurons inside a human brain**. **Neurons are responsible for everything** that we do. Our **conscious & subconscious** is **controlled by these small power stations.**
- **The Structure of Biological Neural Network is shown Below:**

CONTD..

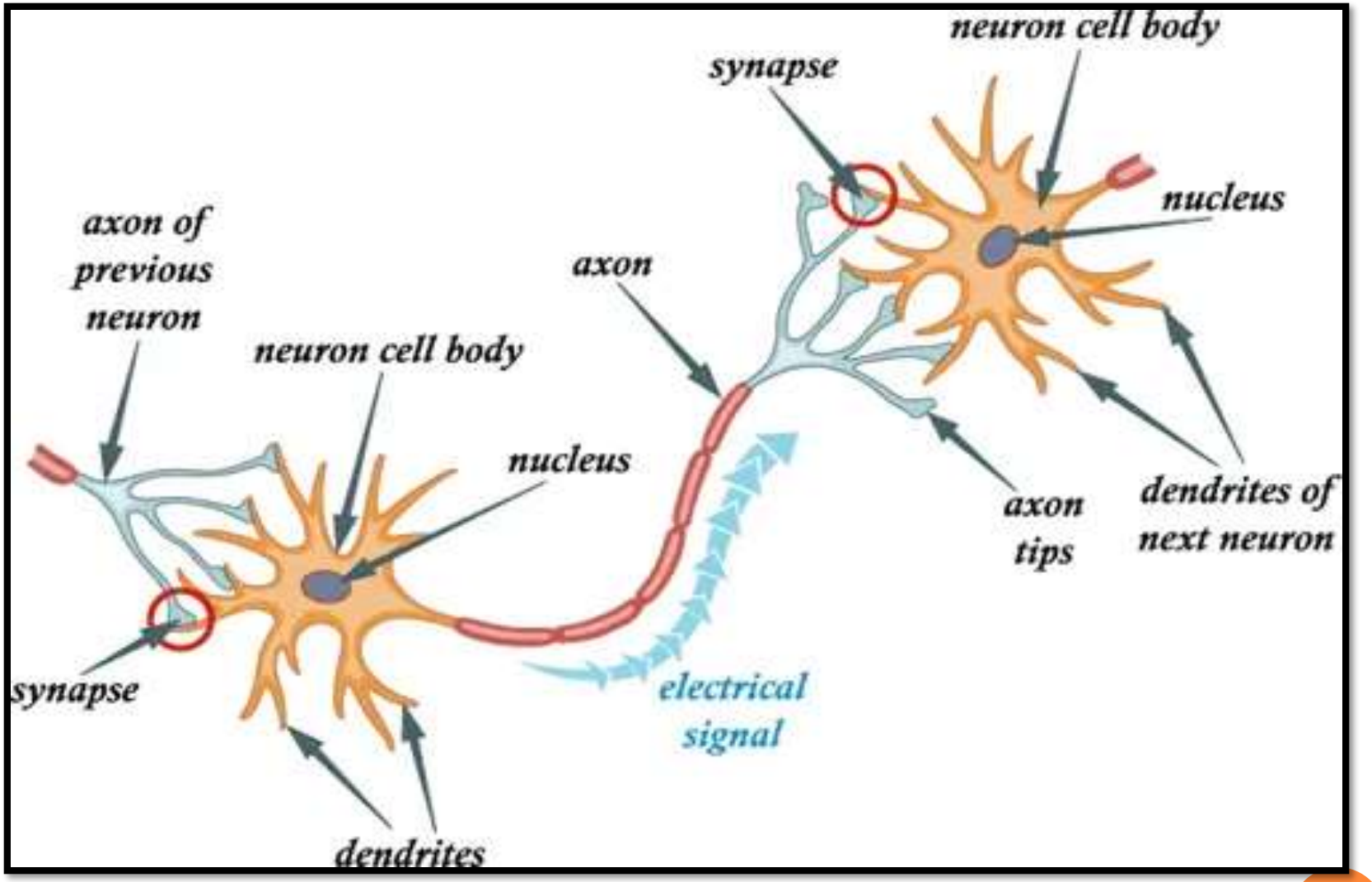
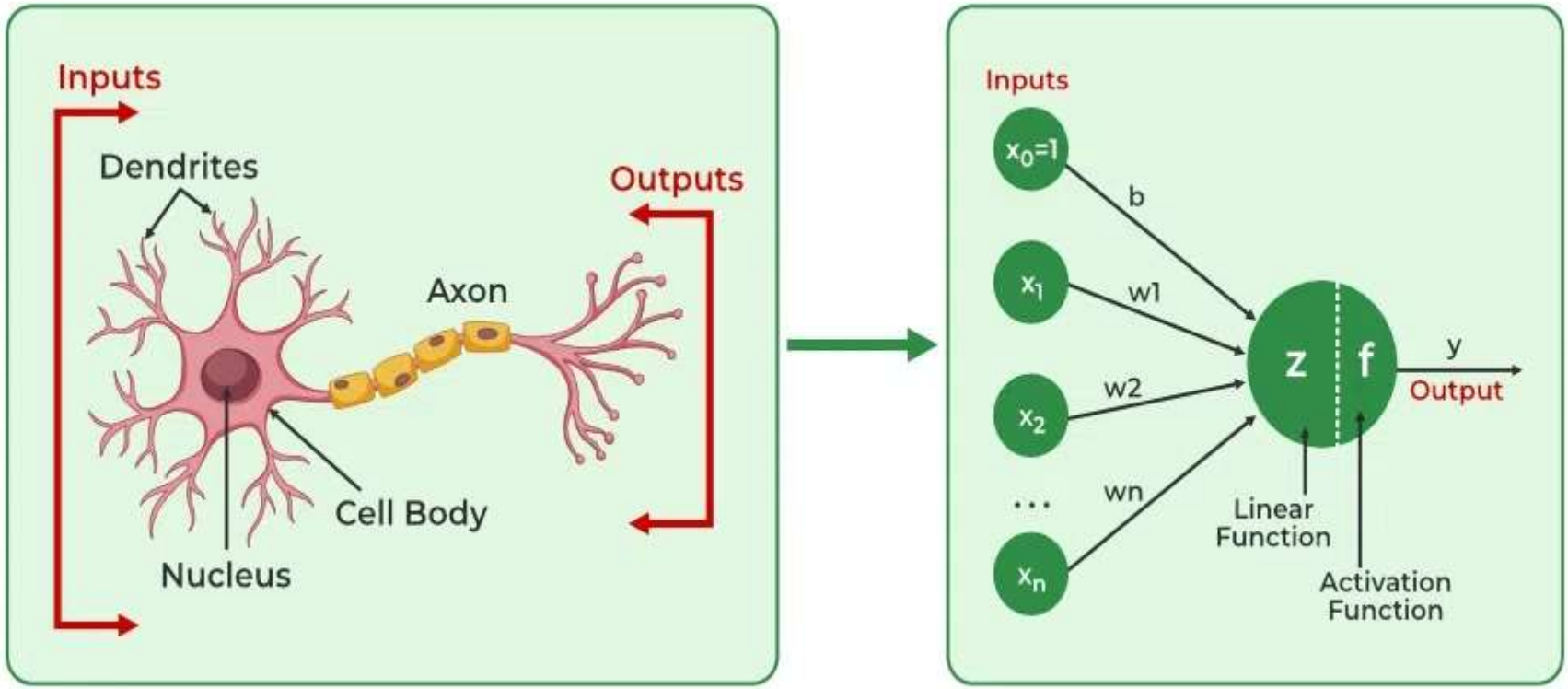


Figure: Biological Neural Network

CONTD..

- **Soma:** This is also called the **cell body**. It is where the **cell nucleus is located.**
- **Dendrites:** These are **tree-like networks** that are connected to the **cell body**. ie, it **receives** the **signal from other Neurons.(Input)**
- **Axon:** Axon carries the **signal from the cell body**. It splits into **strands (Chain or a single unit of a molecule such as DNA)** and each strand ends in a bulb-like structure called **synapse. (output)**
- **ie,** a **single axon** (the cell's output—carrying information away). **Synapse** is a junction between Two Neurons.

CONTD..


Inputs
Dendrites
Outputs
Axon
Cell Body
Nucleus
Inputs
$x_0=1$
b
x_1
w1
x_2
w2
...
wn
x_n
z
f
y
Output
Linear Function
Activation Function

CONTD..

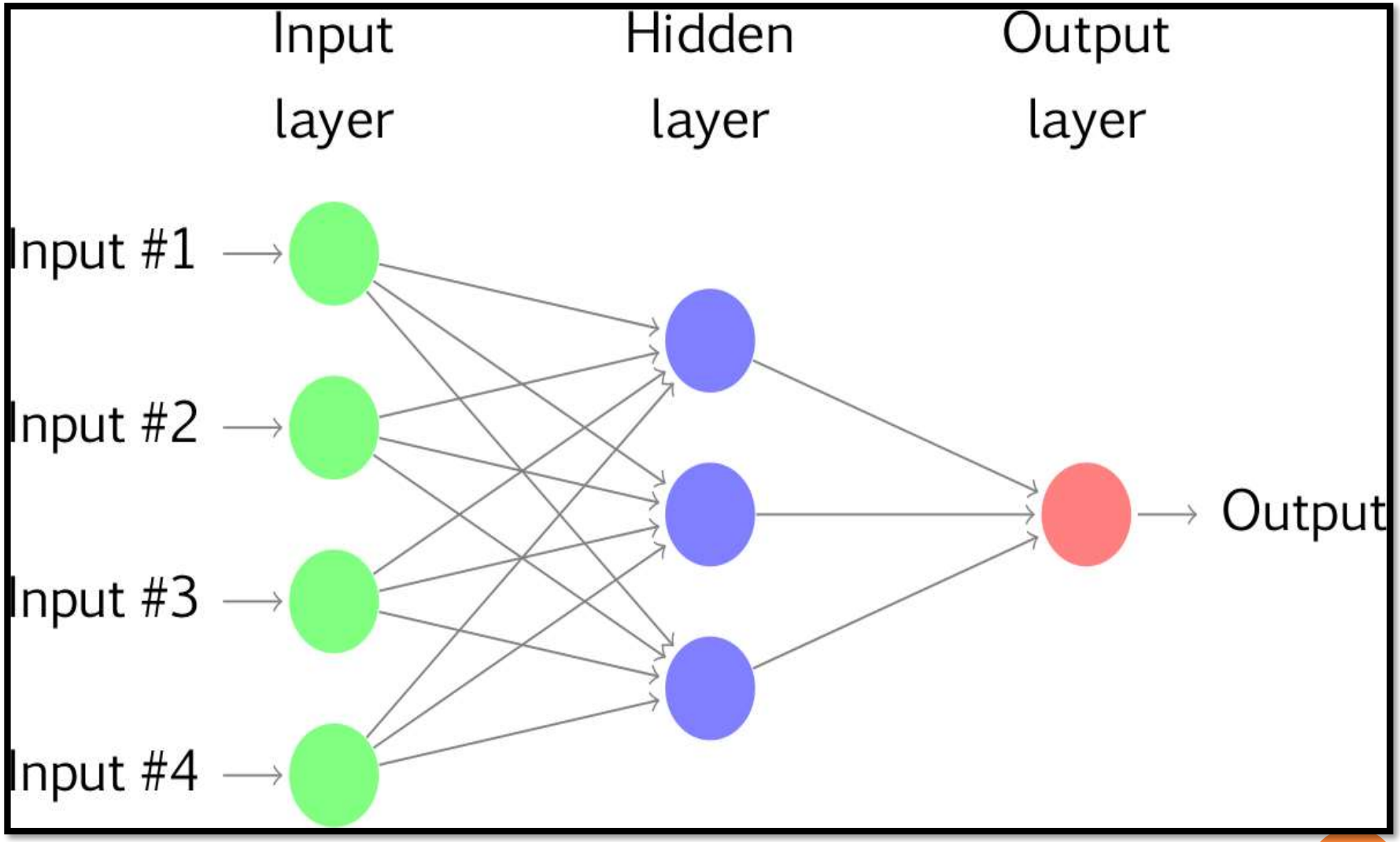
Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

Table: Terminology b/w Human Brain and ANN

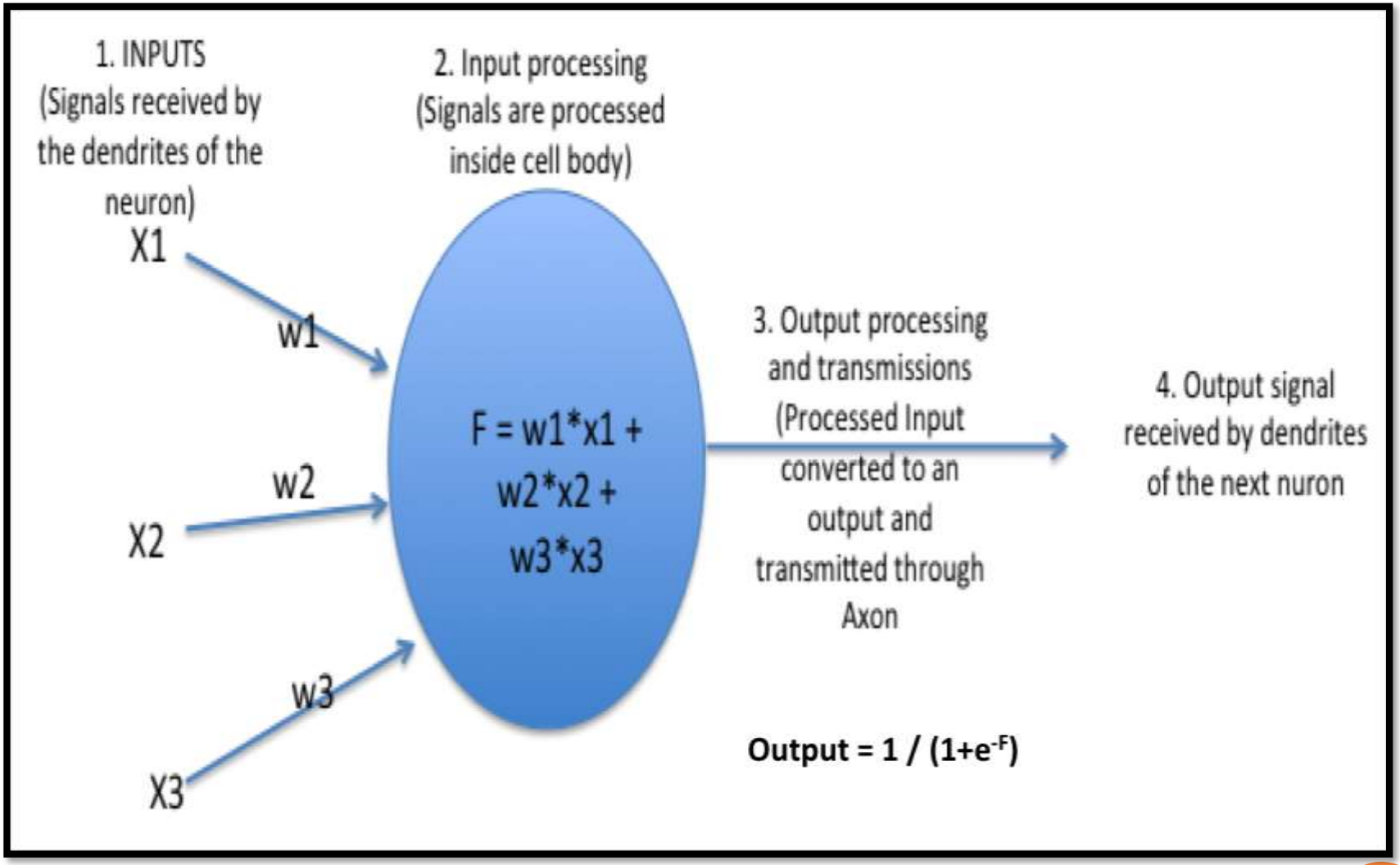
ARCHITECTURE OF ANN

- Artificial Neural Networks are **processing elements** either in the form of **algorithms or hardware devices modeled.**
- Artificial Neural Network primarily consists of **Three Layers:**
  1. **Input Layer**
  2. **Hidden Layer**
  3. **Output Layer**
- **Each Layer** consists of **more Neurons**
- The **nodes of the Input Layer** can take **input data and perform operations on it** and **send the results** of these operations to **other neurons of the Hidden Layer.**
- The **hidden layer** sends **data to the Output Layer.**

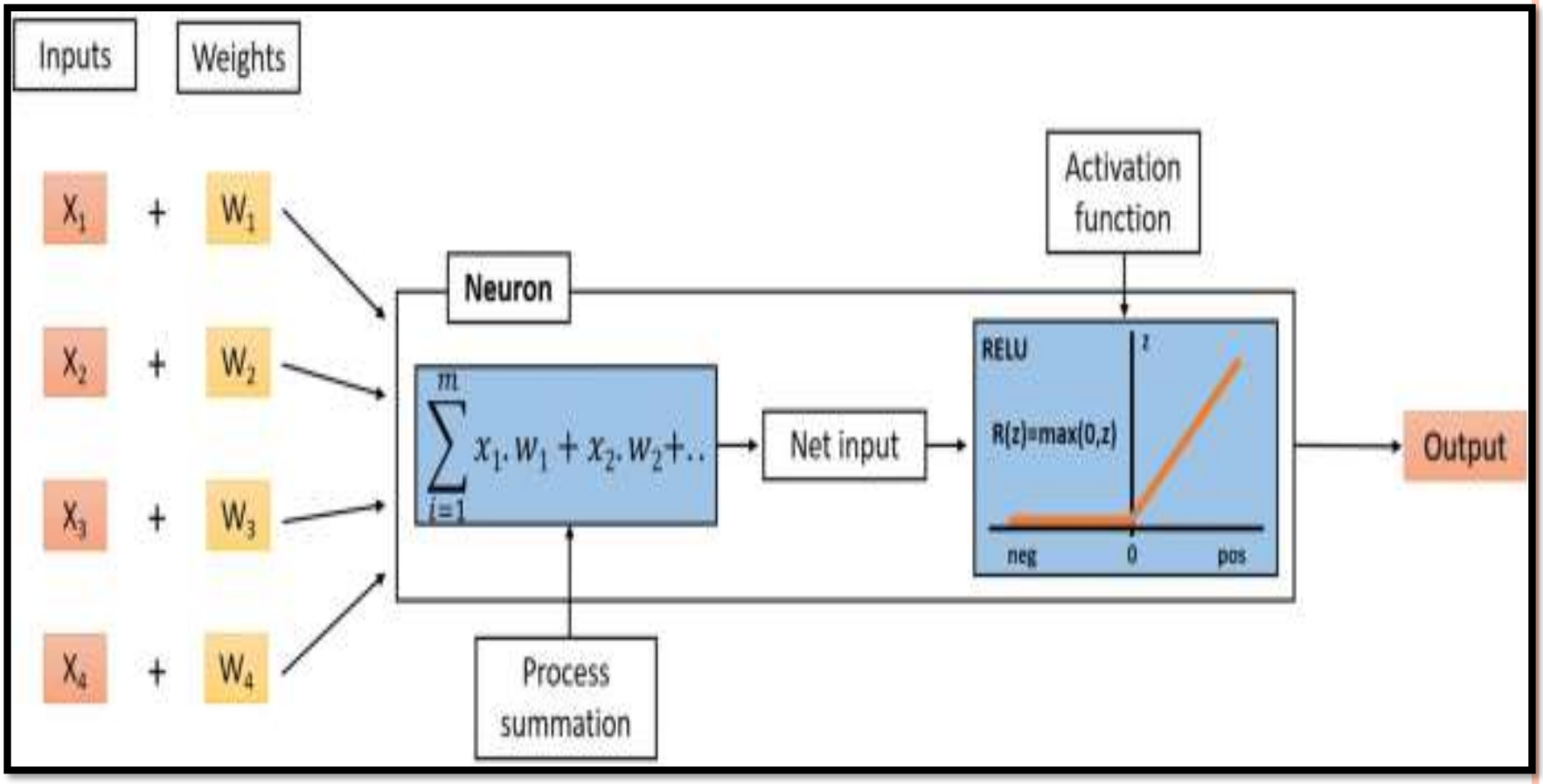
CONTD..


Input
layer
Hidden
layer
Output
layer
Input #1
Input #2
Input #3
Input #4
Output

WHAT HAPPENS INSIDE THE NEURON



WHAT HAPPENS INSIDE THE NEURON



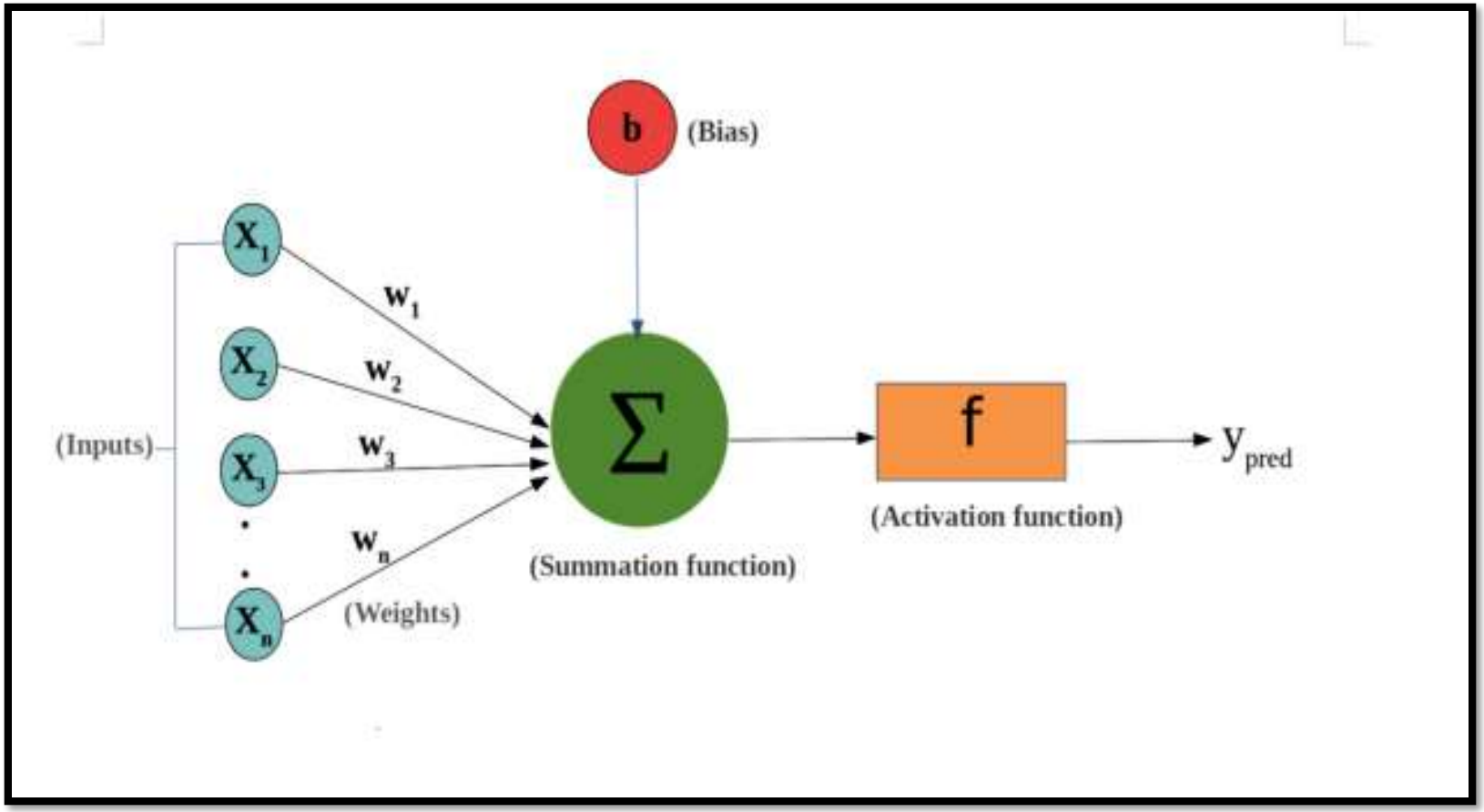
BIAS IN NEURAL NETWORK

- **Bias** is a **systematic error** that occurs due to **wrong assumptions** in the machine learning process.
- **Bias** is simply defined as the **inability of the model** because of that there is some **difference or error** occurring between the **model's predicted value** and the **actual value.**
- These differences between **actual or expected values** and the predicted values are known as **error or bias error or error due to bias.**
- **Biases** provide a **critical additional layer** of flexibility to **neural networks**. Biases are essentially constants associated with each neuron. Unlike weights, biases are not connected to

CONTD..

- **Biases** provide a **critical additional layer** of flexibility to **neural networks**.
- Biases are essentially **constants** associated with **each neuron**. Unlike weights, **biases are not connected to specific inputs** but are **added to the neuron's output.**
- It is used to **adjust the output** along the **weighted sum of the inputs of the neuron.**
- This **Bias Value** allows you to **shift the Activation Function** to **either Left or Right.**

CONTD..




```
output = sum (weights * inputs) + bias
```

CONTD..

- we can represent the **sigmoid activation function** with the **mathematical expression** as

$$sigmoid\ function = \frac{1}{1 + e^{-x}}$$

replacing x with the equation of a line

$$sigmoid\ function = \frac{1}{1 + e^{-(w*x+b)}}$$

CONTD..

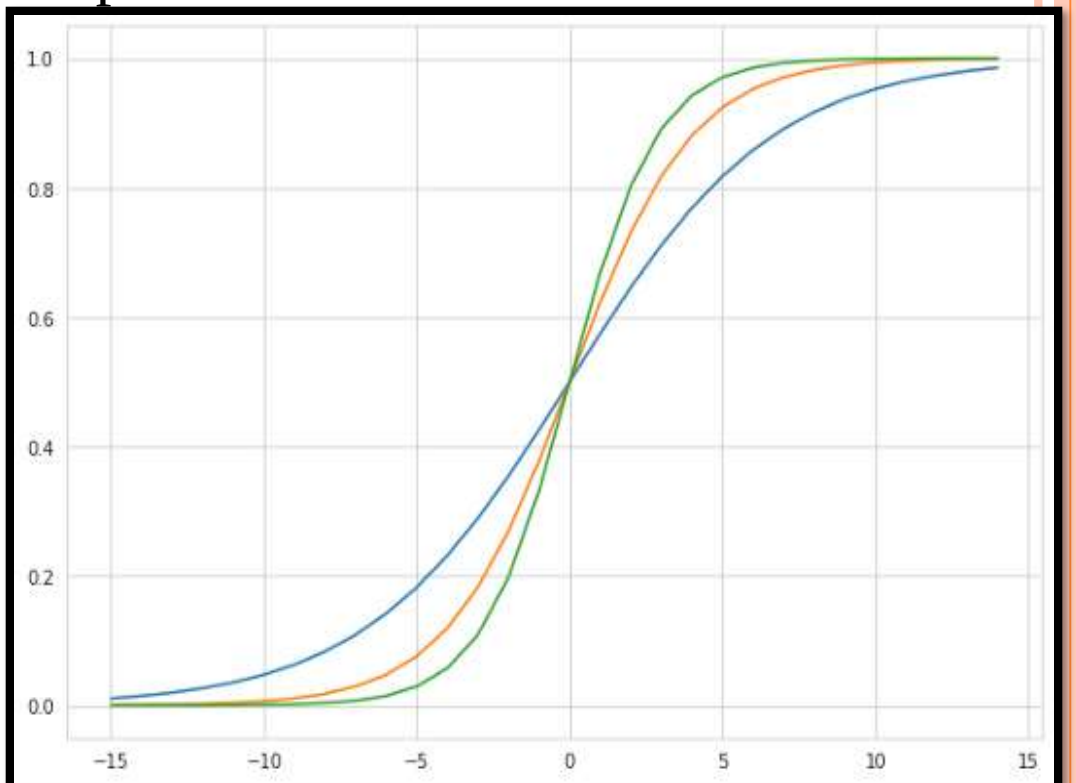
1. **Let us vary different values of w and fix the b value to 0**

when the **value of b =0** and

w = 0.3 — the blue line in the plot

w= 0.5 — the red line in the plot

w = 0.7 — the green line in the plot



CONTD..

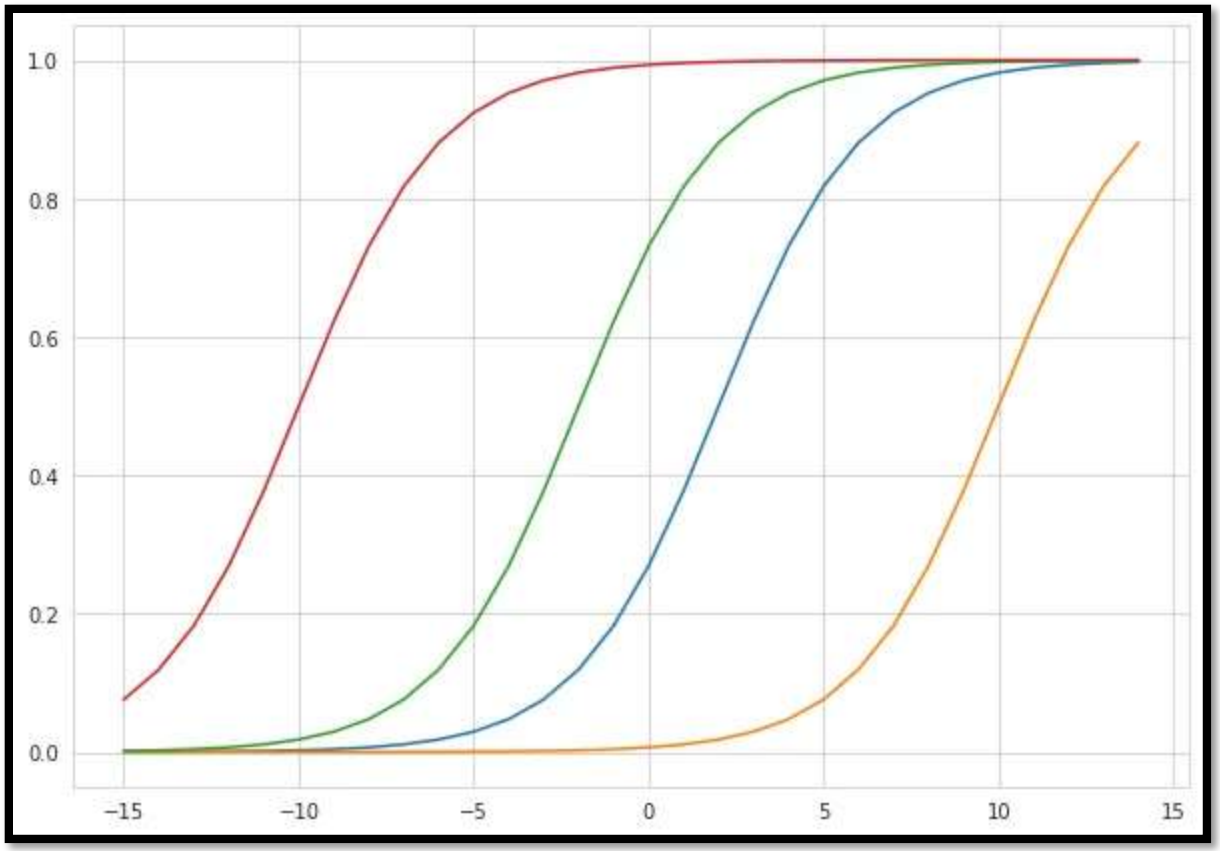
2. Let us vary different values of b and fix w value to 0.5

b = -1 — the red line in the plot

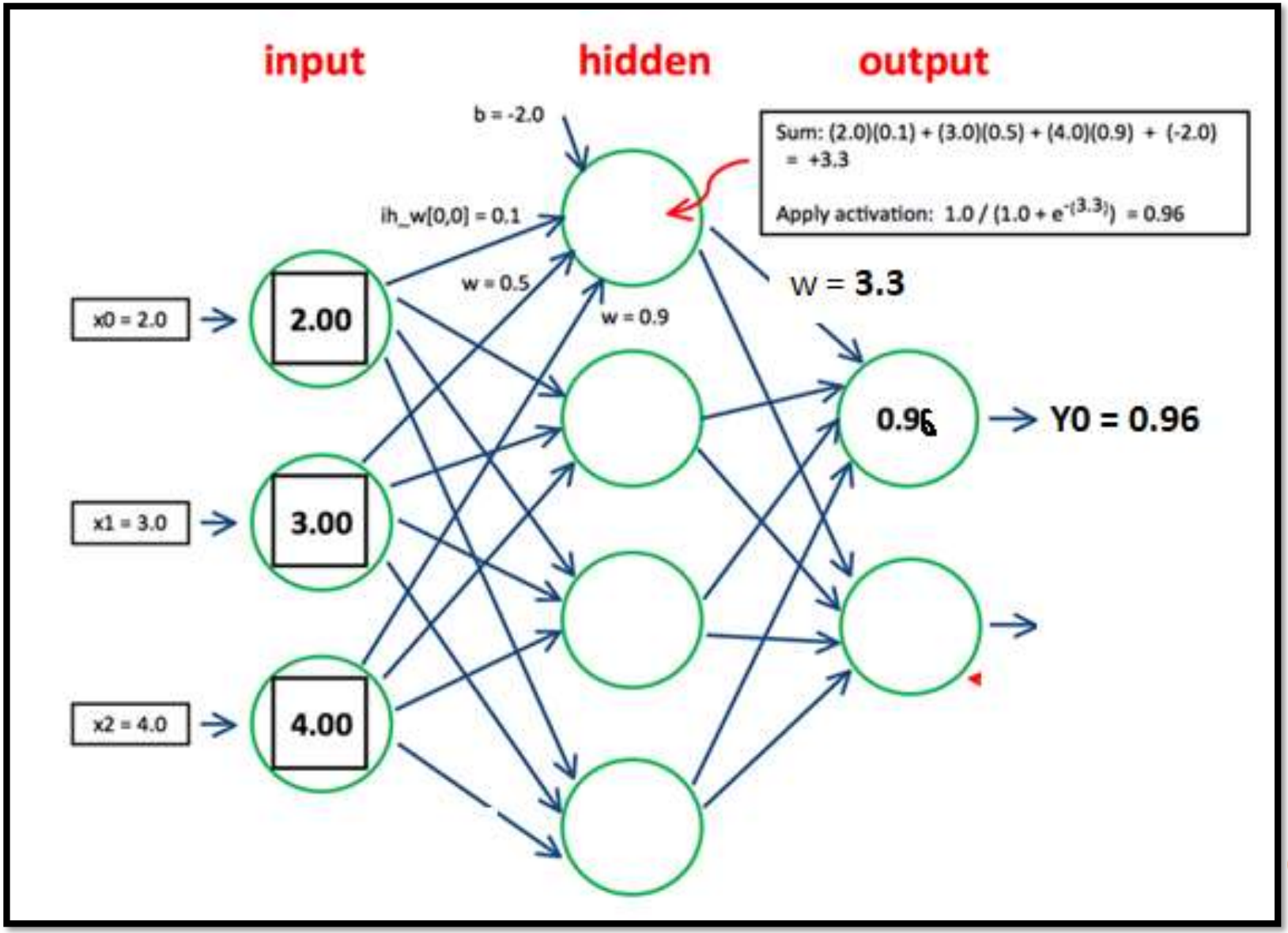
b= -5 — the green line in the plot

b = 1 — the blue line in the plot

b = 5 — the yellow line in the plot



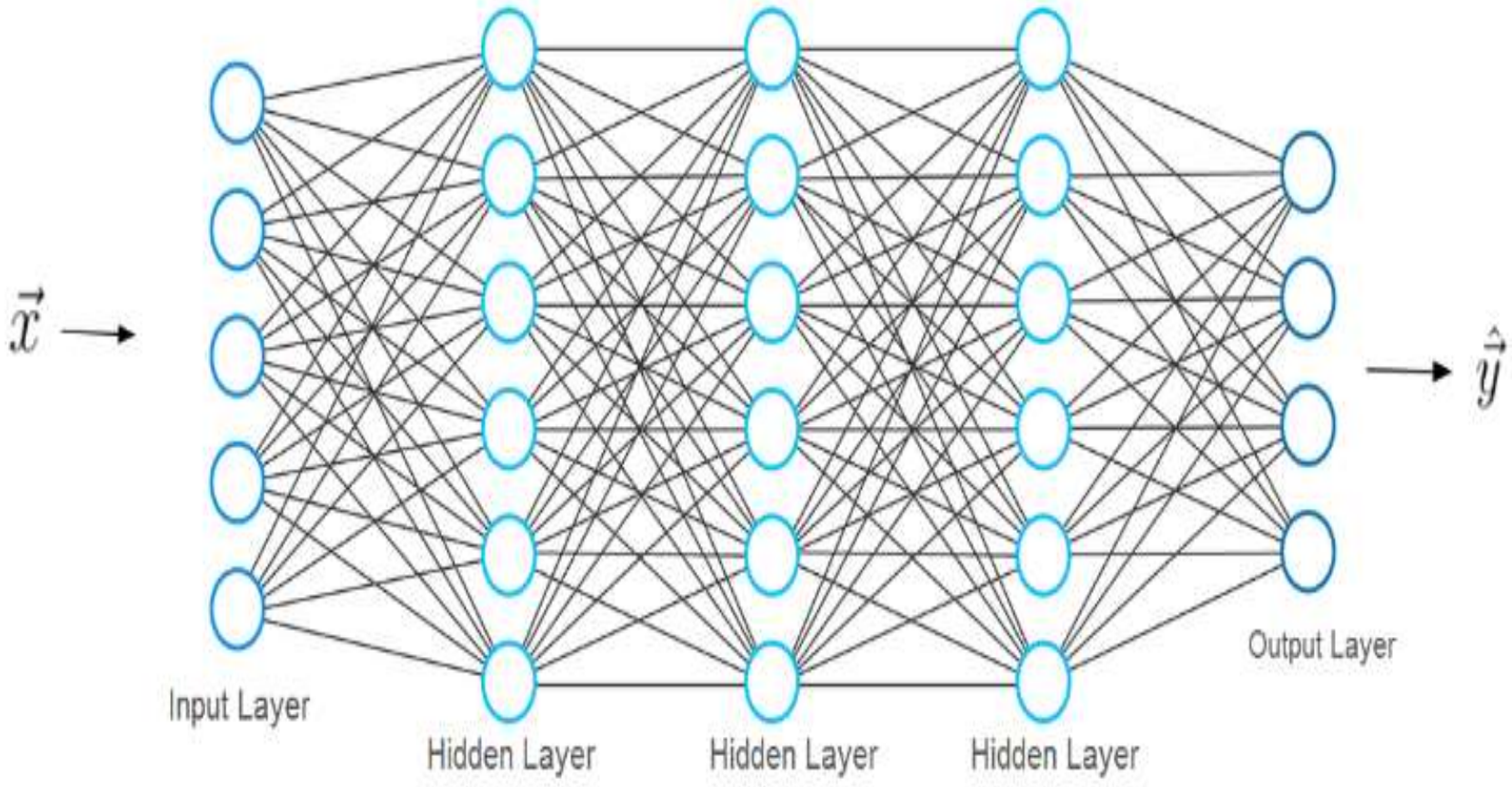
EXAMPLE


input
hidden
output
b = -2.0
Sum: (2.0)(0.1) + (3.0)(0.5) + (4.0)(0.9) + (-2.0)
= +3.3
Apply activation: 1.0 / (1.0 + e^-(3.3)) = 0.96
ih_w[0,0] = 0.1
w = 0.5
w = 0.9
W = 3.3
x0 = 2.0
2.00
x1 = 3.0
3.00
x2 = 4.0
4.00
0.96
Y0 = 0.96

LAYERED CONNECTION OF NEURAL NETWORK

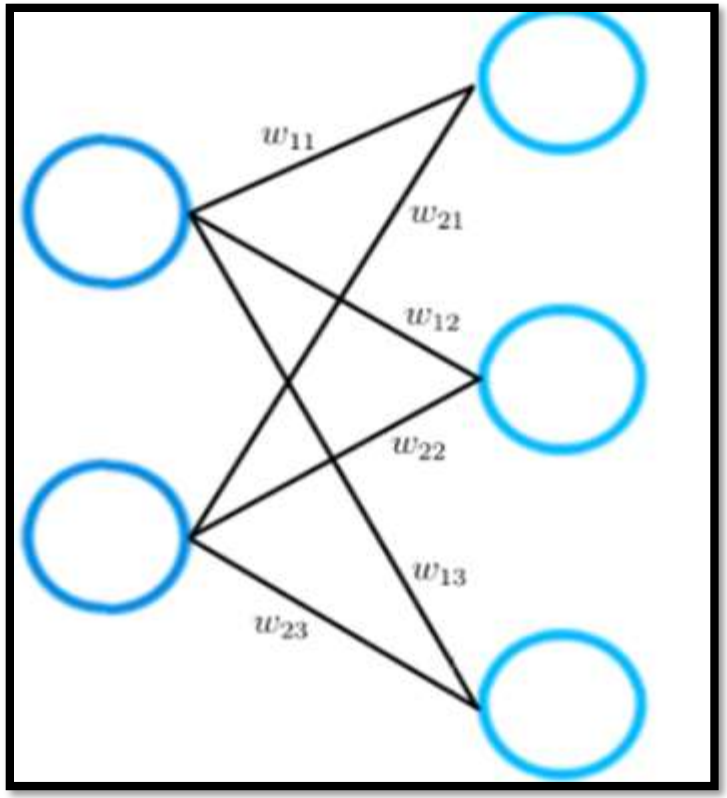
Three basic components are used in ANN are

1.Components **2. Learning** **3. Activation Function**



Explanation of Layers in ANN

- The **input layer** has **two input neurons**, while **the output layer** consists of **three neurons**.



CONTD..

- Each **connection** between **two neurons** is represented by a **different weight w**. Each of these **weight w has indices.**
- **The first value** of the indices stands for the **number of neurons in the layer** from **which the connections originate.**
- **The second value** for the number of the neurons in the layer to **which the connection leads.**
- All weights between **two neural network layers** can be represented **by a matrix** called the **weight matrix.**

CONTD..

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

A weight matrix.

In this particular example, the **number of rows** of the weight matrix corresponds to the **size of the input layer which is two** and the **number of columns** to **the size of the output layer which is three.**

LEARNING PROCESS OF NN

- For a given **input feature vector x**, the **neural network calculates** a **prediction vector**, which we call **here as h.**

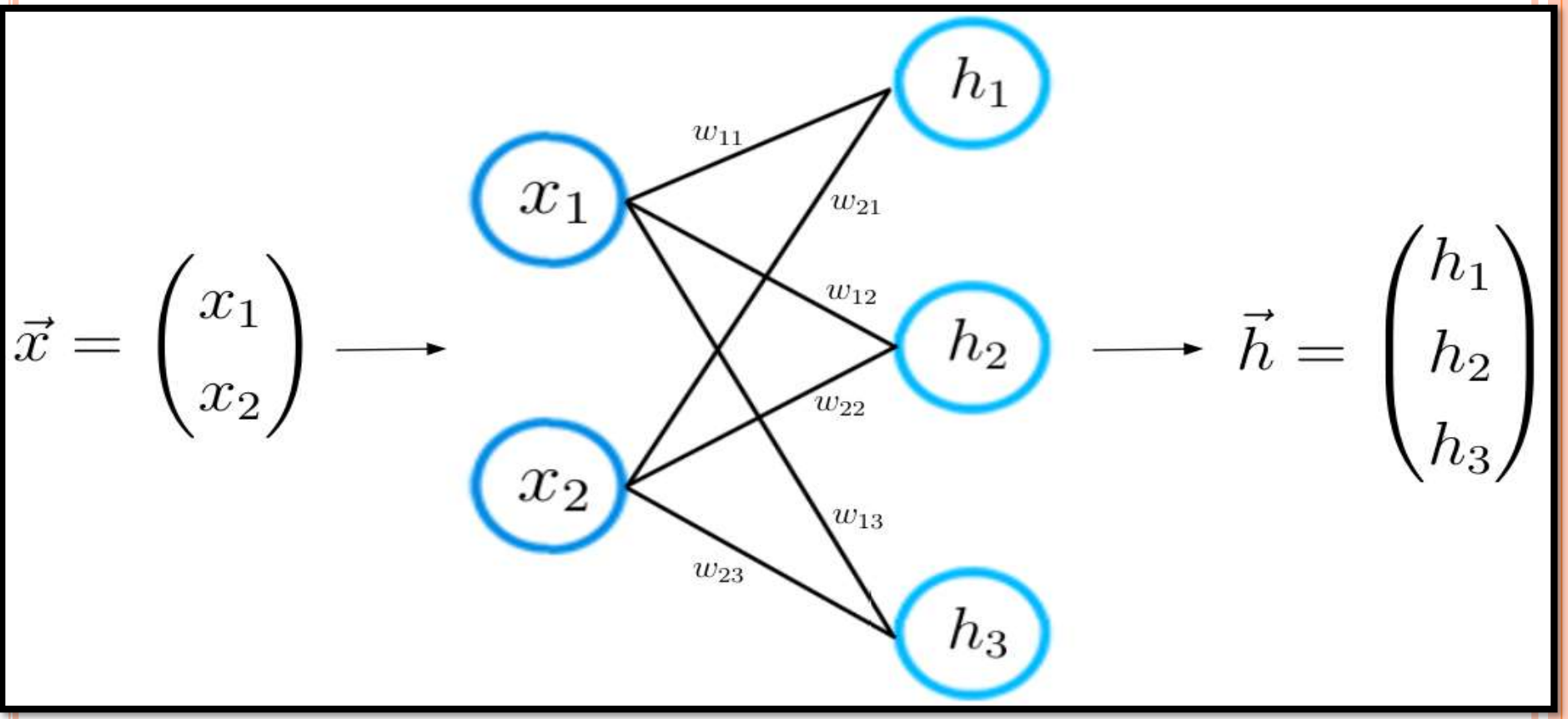


Figure: Forward Propagation

CONTD..

- This step is also referred to as the **forward propagation**. With the **input vector x** and the **weight matrix W** connecting the two neuron layers, **we compute the dot product** between the **vector x and the matrix W.** The result of this dot product is again a vector, which **we call it as Z**

$$\vec{x}^T \cdot W = \begin{pmatrix} x_1, & x_2 \end{pmatrix} \cdot \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$$= \begin{pmatrix} x_1 w_{11} + x_2 w_{21}, & x_1 w_{12} + x_2 w_{22}, & x_1 w_{13} + x_2 w_{23} \end{pmatrix}$$

$$= \begin{pmatrix} z_1, z_2, z_3 \end{pmatrix} = \vec{z}, \quad \vec{h} = \sigma(\vec{z})$$

CONTD..

- The **final prediction vector h** is obtained by applying a so-called **activation function** to the **vector z**. In this case, the activation function is represented by the **letter Sigma.**
- Here **neuron i**s simply a representation of a **numeric value.**

CONTD..

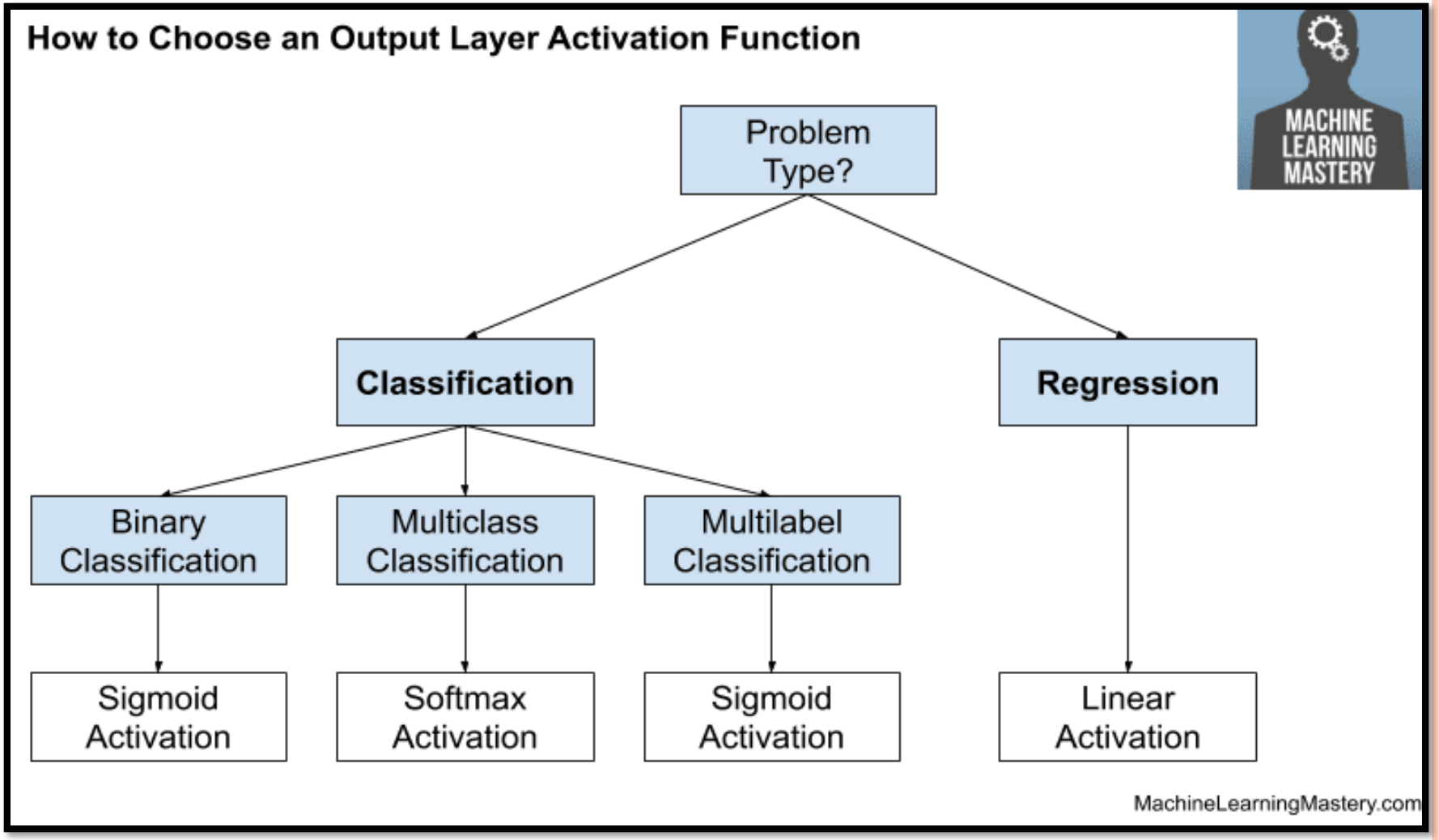
- **ACTIVATION FUNCTION:** It is also known as **Transfer Function.** It can also be attached with **2 neural networks**. It is also a **non linear Function.**
- Activation Functions are **Mathematical Functions** that determine the **output of a Neural Network**
- The **Activation Function** is **attached to each neuron** in a neural network and **determines whether neuron should be activated or not.**

CONTD..

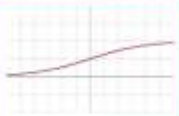
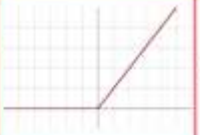
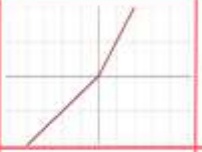
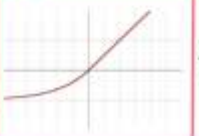
- The **main purpose** is to **convert** a **input signal of a node** in an ANN to an **Output Signal.**
- It translates the input signals to output signals. It maps the output values on a range like **0 to 1 or -1 to 1.**

CONTD..

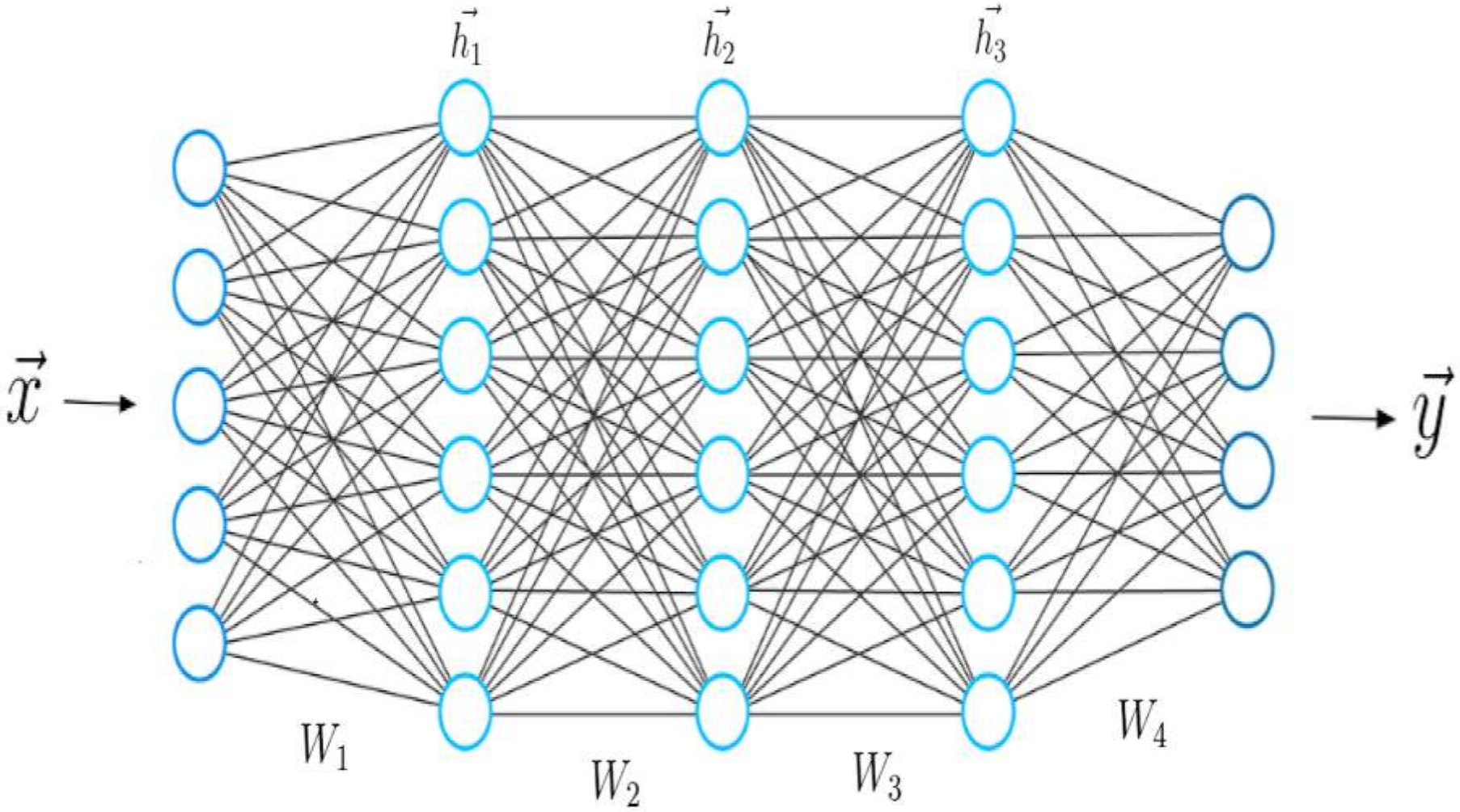
- **Some of Different Activation Functions used in Machine Learning are:**



CONTD..

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1 + e^{x})$	$f'(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^{x} - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^{x} & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

CONTD..



CONTD..

- we can extend our knowledge to a deeper architecture that **consists of 5 layers.**
- **h1 i**s now considered as the i**nput** for the upcoming **third layer.** The whole procedure from before is repeated until we obtain the **final output y:**

$$\vec{h_1} = \sigma(\vec{x}^T \cdot W_1)$$

$$\vec{h_2} = \sigma(\vec{h_1} \cdot W_2)$$

$$\vec{h_3} = \sigma(\vec{h_2} \cdot W_3)$$

$$\vec{y} = \sigma(\vec{h_3} \cdot W_4)$$

Criteria for Comparison

- 1. **Speed of Operation**
- 2. **Way of Processing Data**
- 3. **Size and Complexity of the Neural Network**
- 4. **Storage Capacity or Memory**
- 5. **Fault Tolerance**
- 6. **Control System**

CONTD..

- 1. **Speed of Operation:** The **Execution time of ANN** is of the order of **Nano Seconds**.
- But in case of **Biological Neural Network (BNN)** or **Human Brain,** execution time is of the order of **MilliSeconds.**
- **So ANN is faster than Brain.**
- **2. Data Processing:** In both **Brain and ANN processing** of **data** takes **place parallelly.**
- i.e, **Multiple Instructions** are **executed at the same time** , Simultaneously.
- **But ANN is faster than Brain in data processing.**

CONTD..

- 3. **Size and Complexity of the Network:** **Number of Neurons** in the **Brain** about **10^{11} and there are about 10^{15} inter connecting links.**
- But in **ANN** it can **10-1000 neurons** are used and the **complexity is reduced** when compared to Humans
- So, **Brain is More Complex as compared to ANN**

CONTD..

- 4. **Storage Capacity and Memory:** In **Human Brain**, **data is stored** in its **synaptic interconnections**.
- But in **ANN**, **data is stored** in specifically **allotted memory locations**
- **Storage Capacity of ANN is limited**, But that of **brain is Unlimited.**
- **Data once stored in ANN** can be **retrieved easily at any time.** But in **case of Brain** , **Data loss can occur due to memory Failure.**

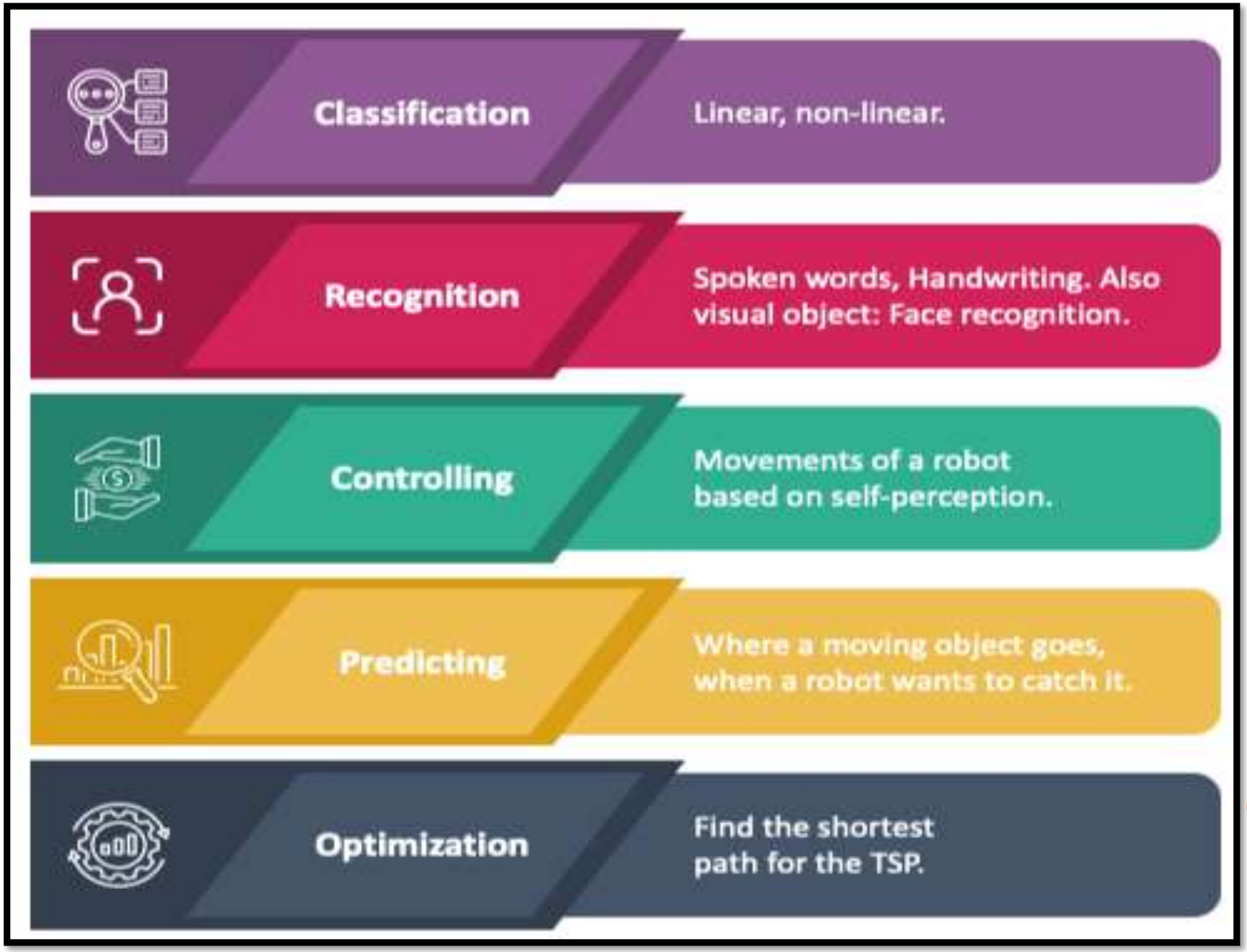
CONTD..

- 5. **Fault Tolerance:** In **ANN,** if **one of the neurons or connecting links get damaged or disconnected**. i.e, In the event of the **system failure**, **Corrupted Data cannot be recovered.**
- But in **Human Brain**, if **some of the Neuron Cells get damaged** or **Die,** it **doesn't cause much damage to the nervous system.**

CONTD..

- 6. **Control System :** In **ANN** are **modeled using computers** and the main controlling unit is the **Central Processing Unit(CPU)**
- But in **Human Brain**, control lies in the **active chemicals present** and the resultant **chemical reactions taking place.**

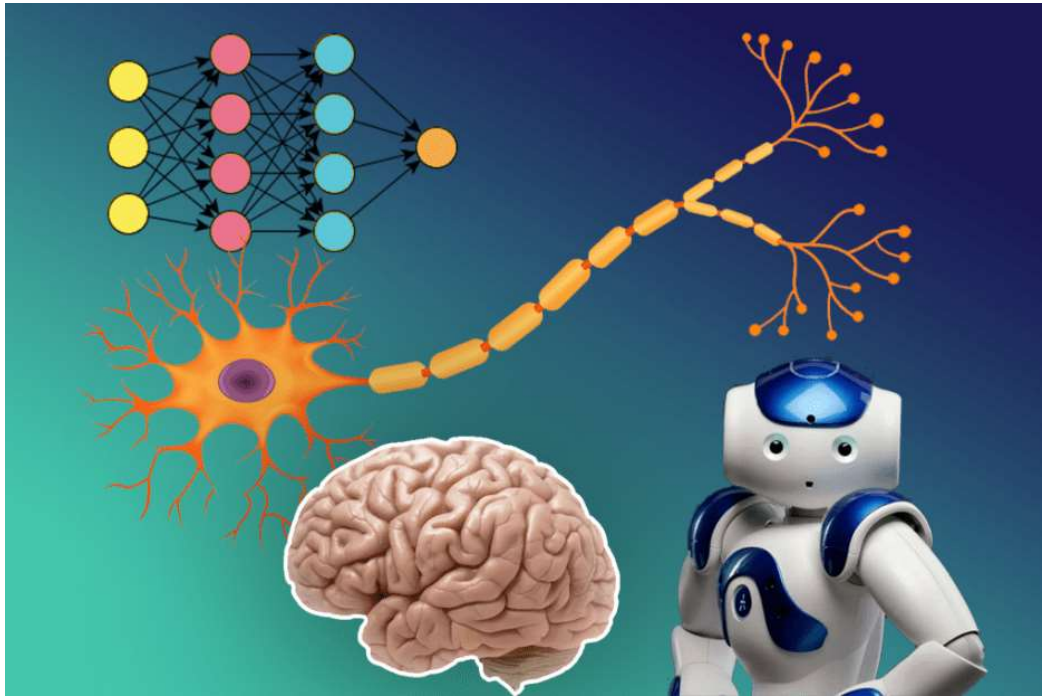
APPLICATIONS OF ARTIFICIAL NEURAL NETWORK



THANK YOU

UNIT-II

(APPROPRIATE PROBLEMS IN ANN)



TOPICS

Topics :

- Appropriate Problems for NN Learning

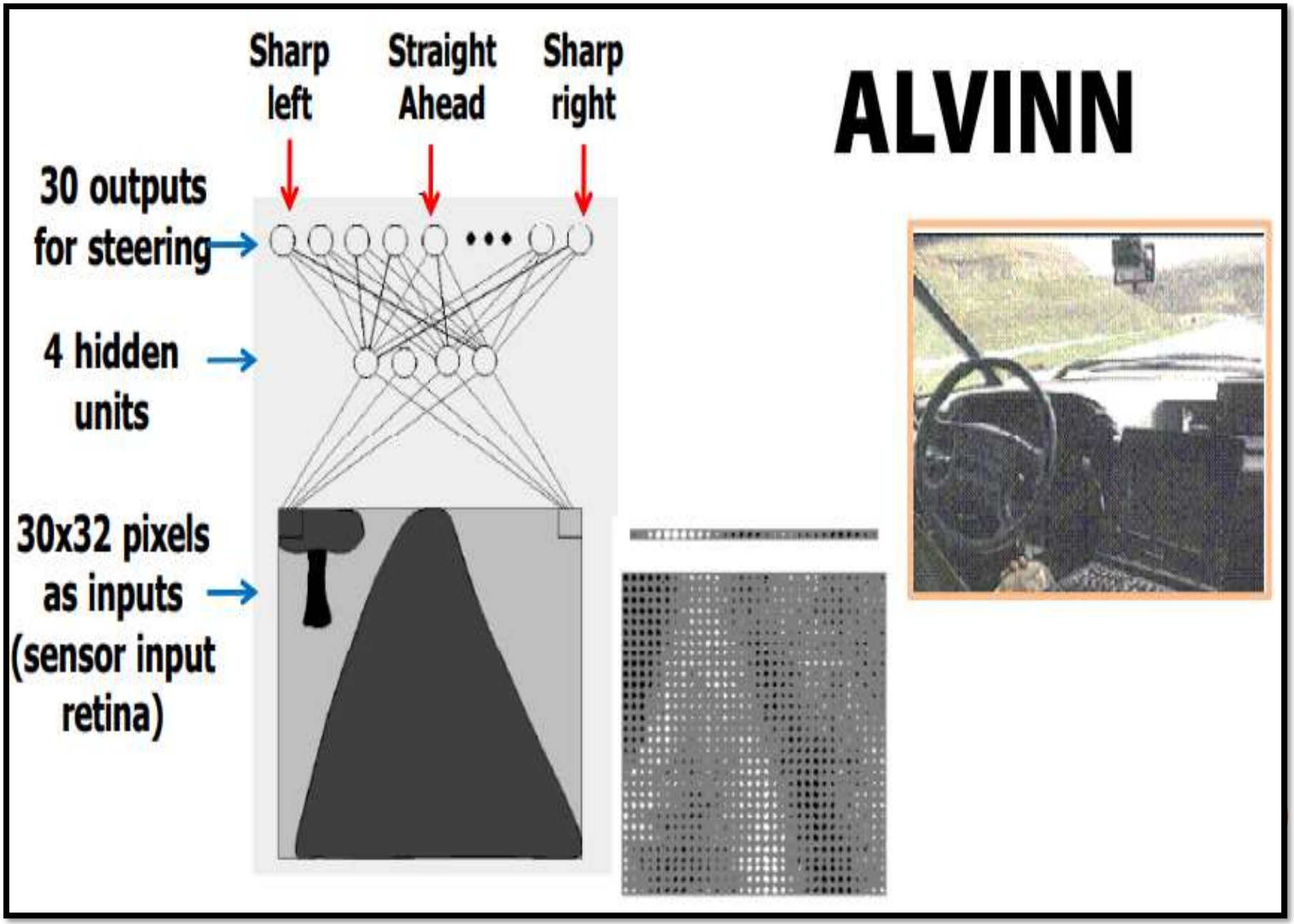
APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

- ANN learning is **well-suited to problems** in which the **training data** corresponds to **noisy, complex sensor data**, such as **inputs from cameras and microphones**.
- Artificial neural networks (ANNs) provide a **general, practical method for learning real-valued, discrete-valued,** and **vector-valued functions** from **examples.**
- Algorithms such as **BACKPROPAGATION gradient descent** to **tune network parameters** to **best fit** a **training set of input-output pairs.**

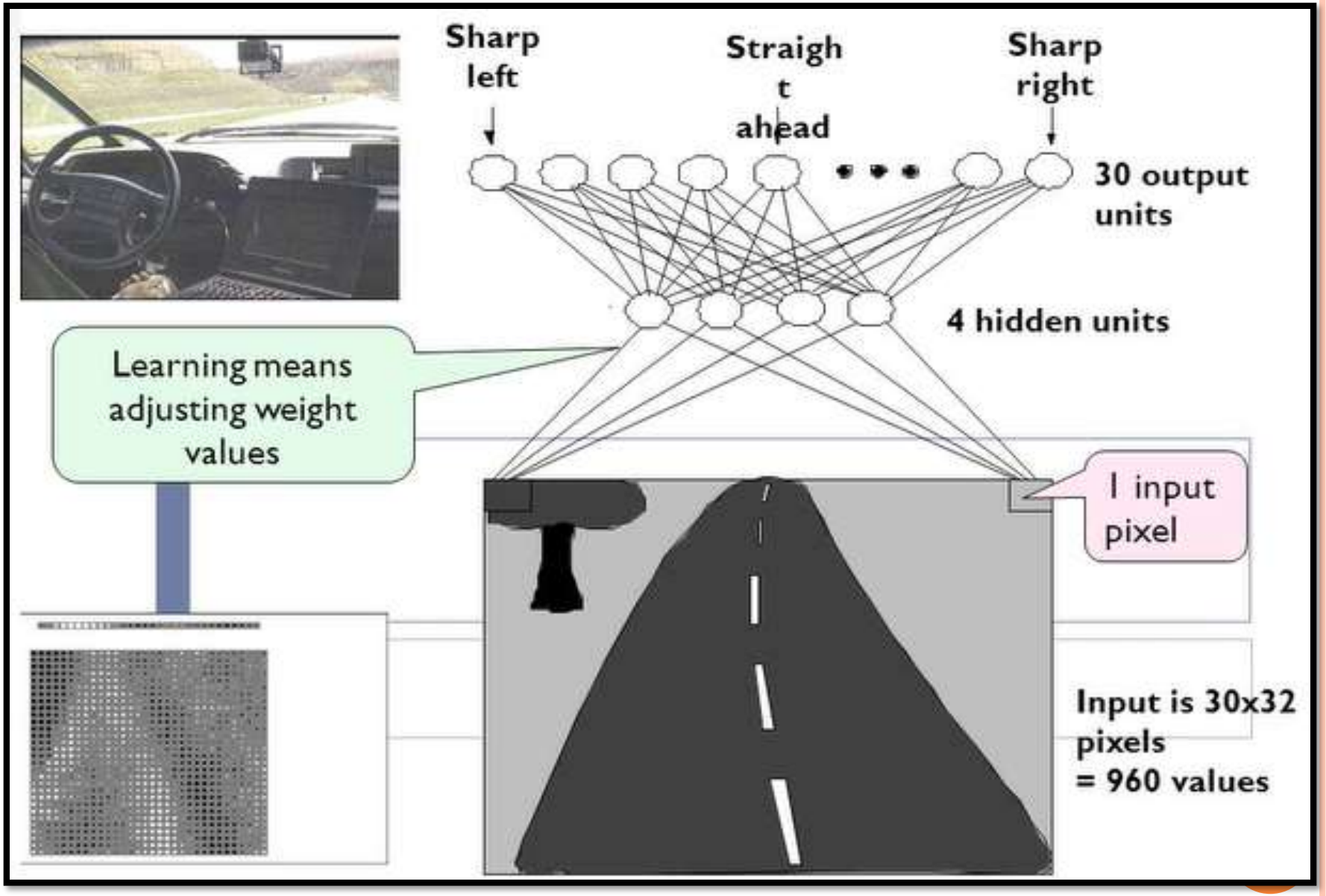
CONTD..

- **ANN learning is robust to errors** in the **training data** and has been **successfully applied to problems** such as **interpreting visual scenes**, **speech recognition**, and **learning robot control strategies**.
- One of the **Best Application in ANN** is **ALVINN(Autonomous Land Vehicle in a Neural Network)**
- It is Designed by **Pomerleau in the year 1993**, is a **Neural Network** that has performed well **in a Domain .**

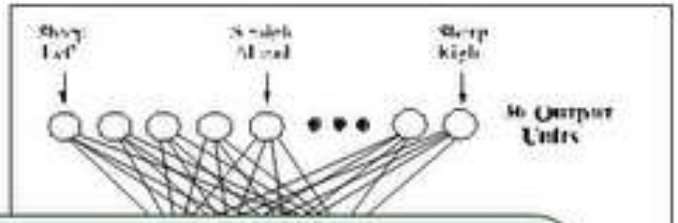
CONTD..


Sharp left
Straight Ahead
Sharp right
ALVINN
30 outputs for steering
4 hidden units
30x32 pixels as inputs (sensor input retina)

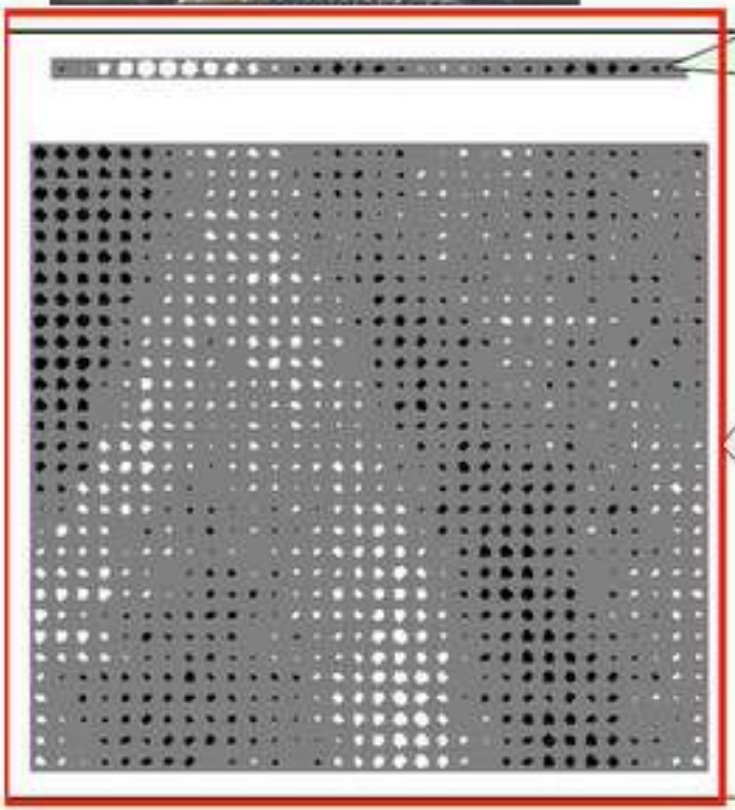
CONTD..


Sharp left
Straigh t ahead
Sharp right
30 output units
4 hidden units
Learning means adjusting weight values
1 input pixel
Input is 30x32 pixels = 960 values

CONTD..



- Output is array of 30 values
- This corresponds to steering instructions
- E.g. hard left, hard right



- This shows one hidden node
- Input is 30x32 array of pixel values
- = 960 values
- Note: no special visual processing
- Size/colour corresponds to weight on link

CONTD..

- Neural network **learning** to **steer an autonomous vehicle**.
- The **ALVINN system** uses **Back propagation** to **learn to steer an autonomous vehicle** (photo at top right) driving at **speed up to 70 miles per hour**.
- The diagram on the left shows **how the image of a forward forward-mounted mounted camera** is **mapped to 960 neural network inputs,** which are **fed forward to 4 hidden units, connected to 30 output units.**
- **Network output encoded** the **commanded steering direction.** The figure on the **right shows weight values** for one of the **hidden units in this network**.

CONTD..

- The **30 x 32 weights** into the **hidden unit** are **displayed** in the **large matrix** with **white blocks indicating positive** and **black indicating negative weights.**
- The **weights from this hidden unit** to the **30 output units** are depicted by the **smaller rectangular block directly above the large block.**
- The most **appropriate for problems** with the **following characteristics:**

CONTD..

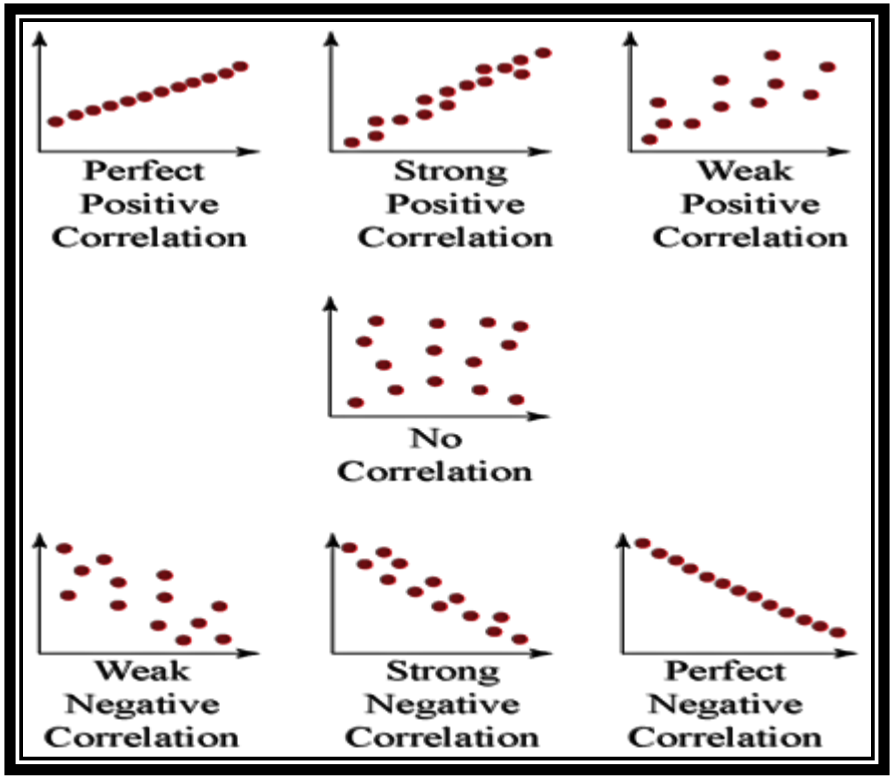
- 1. Instances are represented by many attribute-value pairs.
- 2. The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- 3. The training examples may contain errors.
- 4. Long training times are acceptable
- 5. Fast evaluation of the learned target function may be required.
- 6. The ability of humans to understand the learned Target Function is not important.

CONTD..

- **1. Instances are represented by many attribute-value pairs:**
- If the **particular attribute** has **many attribute value pairs** or the **Continuous Values** for the **particular attribute**, in such case the **Algorithms** i.e, **Decision Tree, Candidate Elimination Algorithms.**
- In that case we can use the ***Neural Networks.***
- In **ALVINN Example**, **input attributes** may be **highly correlated or independent of one another.**
- Currently ALVINN **takes images from a camera** and a **laser range finder as input**

CONTD..

- **Here the main use of Correlation is "Prediction".**
- If there is a **relationship between two variables,** we can make **predictions about one from another**.
- Some times **input values can be real values**.



CONTD..

2. The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.

- **Suppose the Target Function is the Discrete Values i.e, it means, each and every value is independent each other.**
- **Eg: Number of traffic accidents** that occur in a **specific city on a given day.**
- **Suppose the Target Function is the Continous Values i.e, it means, each and every value is dependent each other.**
- Eg: **interest rate** of loans in a certain country.

CONTD..

- **In the ALVINN System the output is a vector of 30 attributes , each corresponding to a recommendation regarding the Steering Direction.**
- The **value of each output** is some **real number** between **0 and 1,** which in this case corresponds to the **confidence in predicting** the corresponding **steering direction.**
- We can **also train a single network** to **output** of **both the steering command and suggested acceleration,** simply by **concatenating the vectors** that **encode these two output predictions**

CONTD..

- **3. The training examples may contain errors.**
- If the **Training Examples like**
  - **Missing Values**
  - **Outliers,** In such cases the ANN is the Best one to solve **these types of Errors.**

CONTD..

- **4. Long training times are acceptable :**
- **Network training algorithms** typically **require longer training times than**, say, **decision tree learning algorithms**.
- Whenever we use **Artificial Neural Networks** , we expect the **data set is very large**. i.e, the **attribute may contain large value pairs.** Then we have **to process all the attributes it takes some time** to **process all these values**
- In such situations , The **training time is also high**. So here we can **use ANN.**

CONTD..

- **Training times** can **range from a few seconds to many hours, depending on factors** such as the
  - **Number of weights in the network**,
  - **The number of training examples considered**, and
  - The **settings of various learning algorithm parameters**

CONTD..

- **5. Fast evaluation of the learned target function may be required.**
- In ANN, **Training it will take more time**. But **once learn, it evaluated is very fast.**
- **For eg:** Suppose we **learn a subject** , it **takes a time** . But when we are **revising it for second time** , i.e, **before the exam, it will not take so much time when compared to Learning Time.**
- Although ANN **learning times are relatively long**, **evaluating the learned network**, in order to **apply it to a subsequent instance, is typically very fast.**

CONTD..

- **For example, ALVINN applies** its **neural network several times per second to continually update its steering command** as the **vehicle drives forward.**

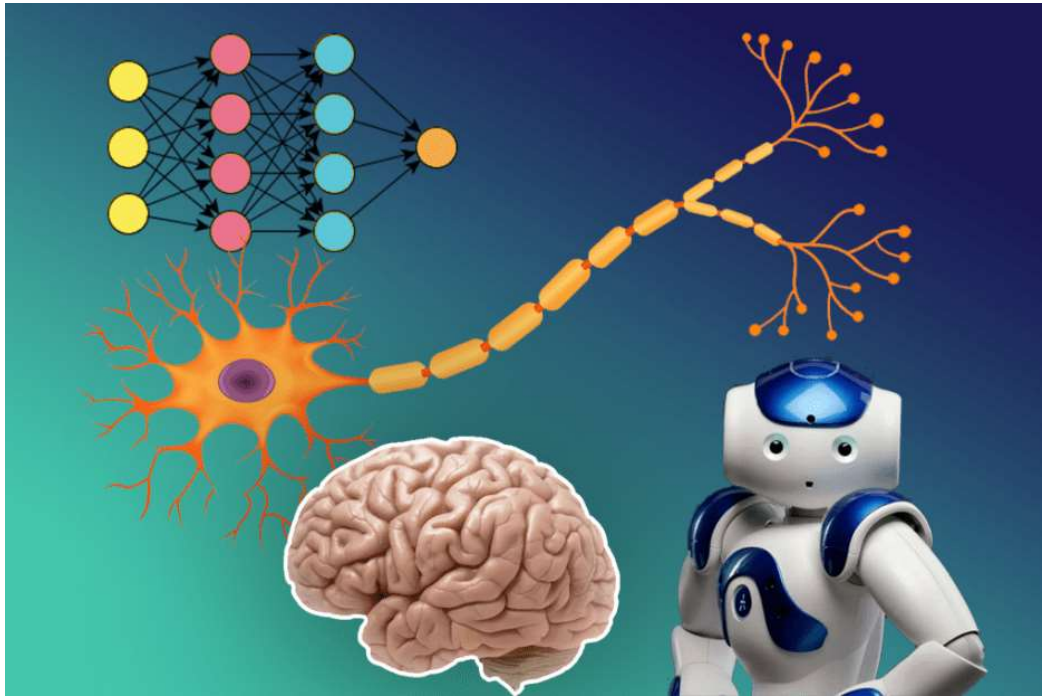
CONTD..

- **6. The ability of humans to understand the learned Target Function is not important.**
- What ever the **Target Function** that the **Machine is Learning, is very complicated**
- The **weights learned by neural networks** are often **difficult for humans to interpret**.
- **Learned neural networks** are **less easily communicated** to humans than **learned rules.**

THANK YOU

UNIT-II

(PERCEPTRON'S)



TOPICS

Topics :

- Perceptron's
- Representation of power of Perceptron
- Perceptron Training Rule
  - AND Gate using Perceptron Training Rule
  - OR Gate using Perceptron Training Rule

PERCEPTRON

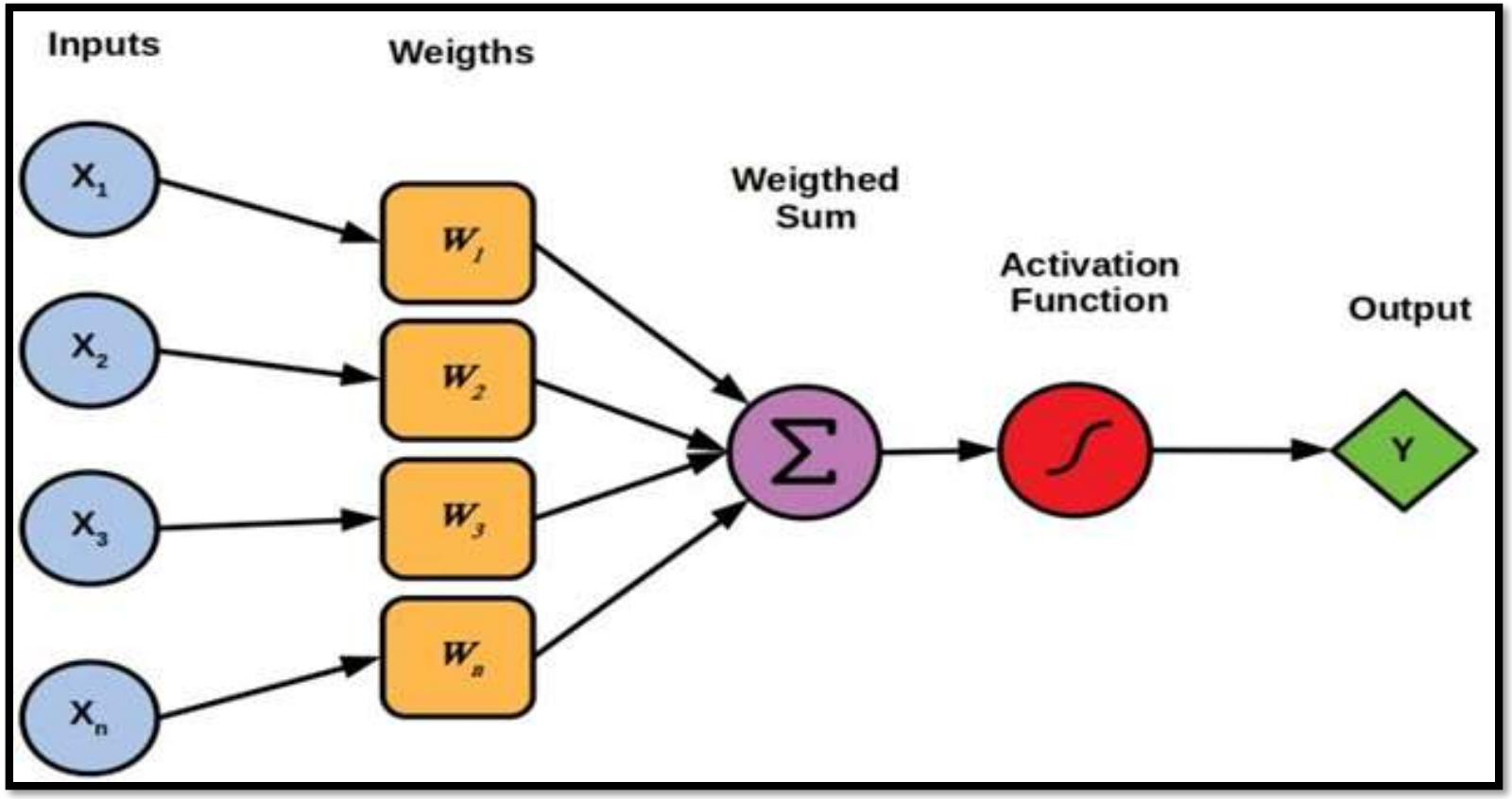
- A **Perceptron** is **an Artificial Neuron**
- **Frank Rosenblatt (1928 – 1971)** was an **American psychologist** notable in the field of **Artificial Intelligence**.
- In 1957 , he **"invented"** a **Perceptron program**, on **an IBM 704 computer** at **Cornell Aeronautical Laboratory**.
- **The Neurons receive input** from **our senses** by **electrical signals** and **store information** to **make decisions** based on **previous input.**
- So, A **Perceptron Unit** is **used to build** the **ANN System**

CONTD..

- Perceptron is a **Machine Learning algorithm** for **supervised learning** of various **binary classification tasks**.
- The **original Perceptron** was **designed** to **take a number of binary inputs,** and **produce one binary output (0 or 1).**

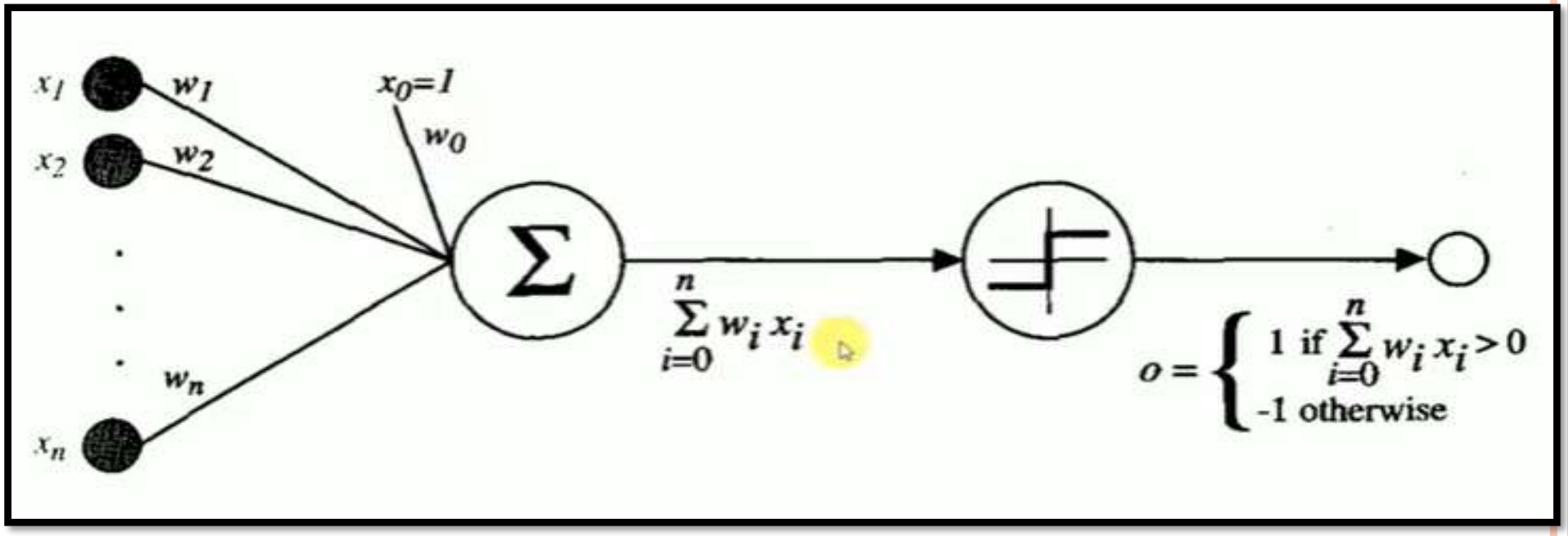
Architecture of the Perceptron

- **Mr. Frank Rosenblatt** invented the **perceptron model** as a **binary classifier.**

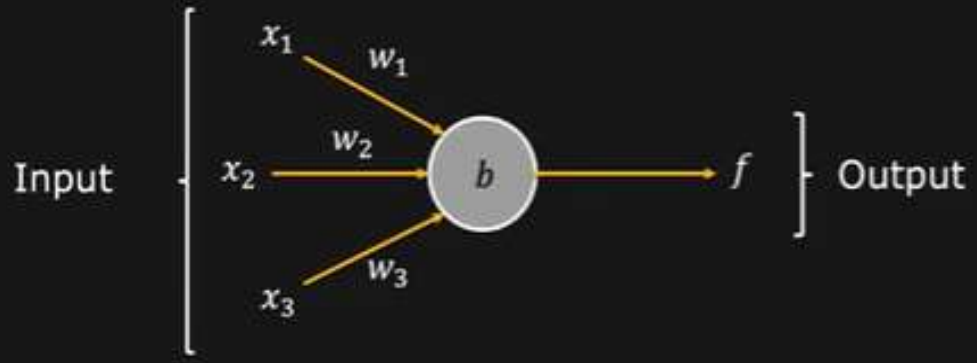


CONTD..

- The **idea** was to **use different weights** to represent the importance of **each input**, and **that the sum of the values should be greater than** a threshold value before making a **decision like true or false (0 or 1).**



Takes several inputs and gives a single binary output (Decision)



$$f = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq -b \\ 1 & \text{if } \sum_j w_j x_j > -b \end{cases} \qquad f = \begin{cases} 0 & \text{if } w.x + b \leq 0 \\ 1 & \text{if } w.x + b > 0 \end{cases}$$

where, $w_j = weights$

$b = bias\ (threshold)$

CONTD..

- More precisely, the **given inputs x1 through xn**, the **output o1(x1,-------xn)** computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

- The Perceptron receives multiple input signals, and if the sum of the input signals **exceeds a certain threshold**, it either **outputs a signal or does not return an output**.
- In the context of supervised learning and classification, this can then be used to **predict the class of a sample.**

Perceptron Learning Algorithm

Frank Rosenblatt suggested this algorithm:

- **Step-1: Set a threshold value**
- **Step-2: Multiply all inputs with its weights**
- **Step-3: Sum all the results**
- **Step-4: Activate the output**

WORKING PRINCIPLE

- Here, First, **multiply all input values** with **corresponding weight values** and **then add them** to determine the **weighted sum**. Mathematically, we can calculate the weighted sum as follows:

$$\sum wi*xi = x1*w1 + x2*w2 + \ldots wn*xn$$

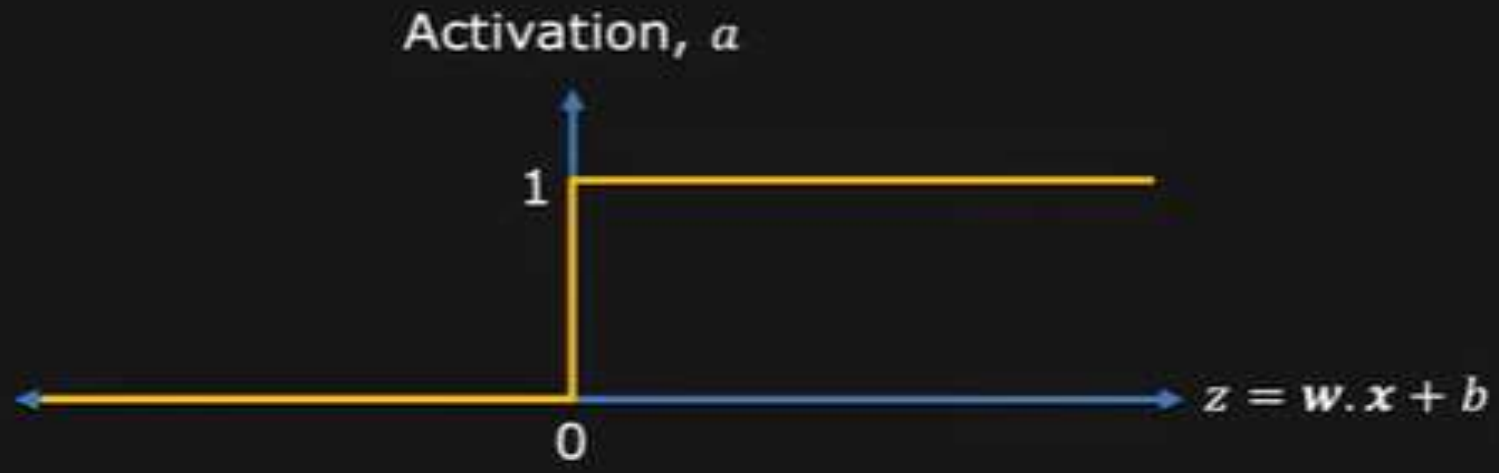
- and that **the sum of the values** should **be greater than a threshold value** before making **a decision like true or false (0 or 1).**
- **Add a special term** called **bias 'b'** to **this weighted sum** to **improve the model's performance**. $\sum wi*xi + b$

CONTD..

Activation Function of a Perceptron

Let: $z = w.x + b$

Activation: $a = f(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases}$



Activation Function for a Perceptron is a Step Function.

EXAMPLE

- Imagine a perceptron (in your brain).
- The perceptron tries to decide if you should go to a concert.
- Is the artist good?
- Is the weather good?
- What weights should these facts have?

Criteria	Input	Weight
Artists is Good	**x1** = 0 or 1	**w1** = 0.7
Weather is Good	**x2** = 0 or 1	**w2** = 0.6
Friend will Come	**x3** = 0 or 1	**w3** = 0.5
Food is Served	**x4** = 0 or 1	**w4** = 0.3
Alcohol is Served	**x5** = 0 or 1	**w5** = 0.4

CONTD..

1. Set a threshold value:

- Threshold = 1.5

2. Multiply all inputs with its weights:

- x1 * w1 = 1 * 0.7 = 0.7
- x2 * w2 = 0 * 0.6 = 0
- x3 * w3 = 1 * 0.5 = 0.5
- x4 * w4 = 0 * 0.3 = 0
- x5 * w5 = 1 * 0.4 = 0.4

3. Sum all the results:

- 0.7 + 0 + 0.5 + 0 + 0.4 = 1.6 (The Weighted Sum)

4. Activate the Output:

- Return true if the sum > 1.5 ("Yes I will go to the Concert")

CONTD..

- **Perceptron Terminology:**
- **1. Inputs :** Here Inputs called **Nodes**

  In the above example, the node values are: **1, 0, 1, 0, 1**

  The binary **input values** (0 or 1) can be interpreted as (no or yes) or (false or true).
- **2. Node Weights:** Weights shows the **strength of each node**.

  In the above example, the Weights are **0.7, 0.6, 0.5, 0.3, 0.4**

CONTD..

- **3. Activation Function:** The **activation functions maps the result (the weighted sum)** into a **required value like 0 or 1.**
- In the **above Example,** the **activation function** is simple: **(sum > 1.5)**
- The **binary output (1 or 0)** can be **interpreted as (yes or no) or (true or false).**
- **Note:** It is obvious that a ***decision is NOT made by one neuron alone.***
- **Threshold Value:** We have to define **a reliable threshold** for the **classifier** that **clearly indicates the split between the two classes.**

CONTD..

- **Bias:** Model bias refers to the presence of **systematic errors** in a model **that can cause it to consistently make incorrect predictions.**
- These errors can arise from **many sources**, including the **selection of the training data**, **the choice of features used to build the model**, or **the algorithm used to train the model.**

REPRESENTATION OF POWER OF PERCEPTRON

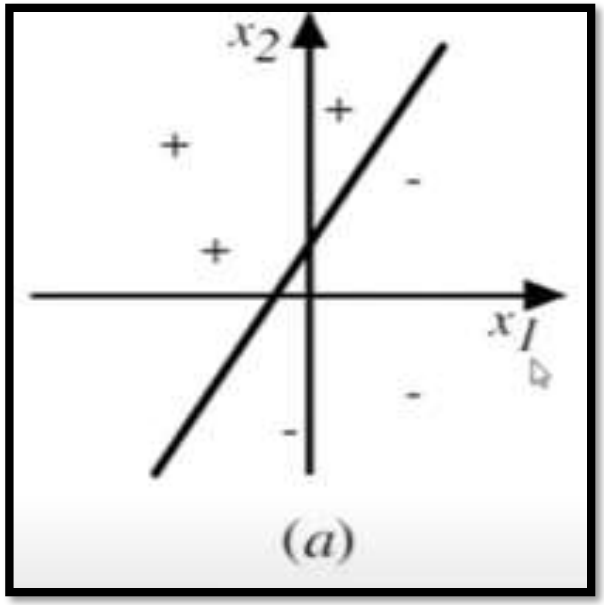
- As we can see, **it calculates a weighted sum of its inputs** and **thresholds it with a step function**.
- A **single perceptron** can be **used to represent many Boolean functions.**
- The Boolean Functions , **AND, OR, XOR** can be solved the Perceptron.
- **Geometrically,** this means **the perceptron can separate its input space** with a **Hyper Plane**.
- That's where the notion that a perceptron can **only separate linearly separable problems** .

Types of Perceptron's:

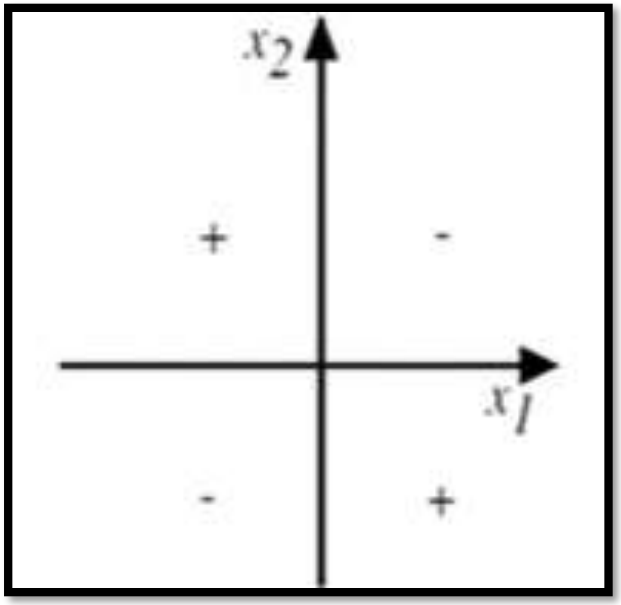
- **1. Single Layer Perceptron Model**
- **2. Multi Layer Perceptron Model**

- **1. Single Layer Perceptron Model:** The **main objective** of the single-layer perceptron model is **to analyze the linearly separable objects** with **binary outcomes**.
- Perceptrons can represent all of the Boolean Functions , **AND, OR ,NAND,NOR** can be solved the Perceptron
- Single Layer perceptron , **can only solve Linearly Separable Problems.**

- Some of the **Boolean Functions cannot represent the Single Perceptron,** Such as **XOR function** is **not linearly separable,** it really is impossible for **a single hyper plane to separate it**.



Linearly Separable



Not Linearly Separable

CONTD..

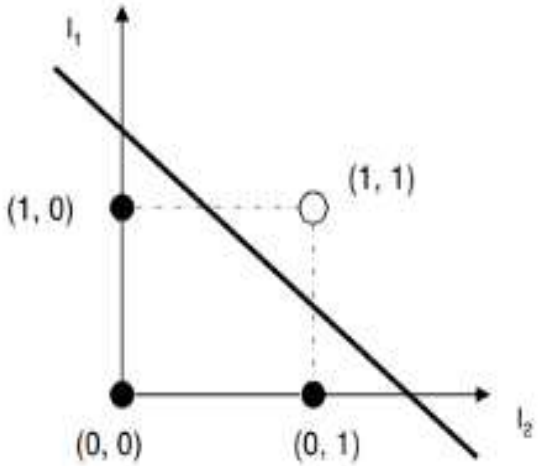
- **2. Multi Layer Perceptron Model:** Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a **greater number of hidden layers.**
- The multi-layer perceptron model is also known as the **Back Propagation Algorithm.**
- Hence, **a multi-layered perceptron model** has considered as **artificial neural networks** having **various layers** in which **activation function does not remain linear**, similar to a single layer perceptron model.

CONTD..

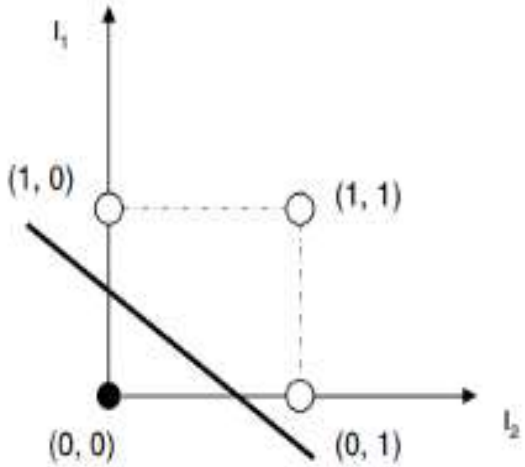
- **Instead of linear,** activation function can be executed as **sigmoid, TanH, ReLU, etc., for deployment.**
- A multi-layer perceptron model has **greater processing power** and can **process** **linear and non-linear patterns**.
- Further, it can also **implement logic gates** such as **AND, OR, XOR, NAND, NOT, XNOR, NOR.**

REPRESENTATION OF POWER OF PERCEPTRON

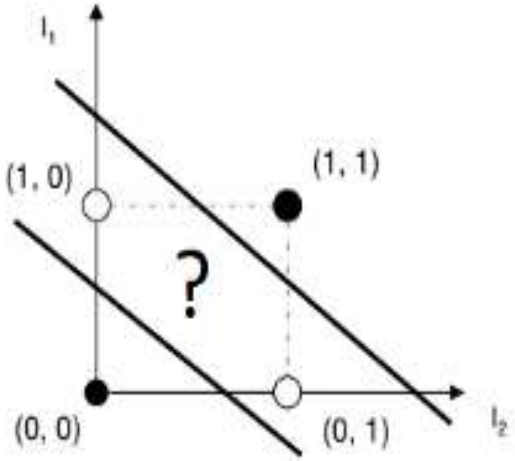
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



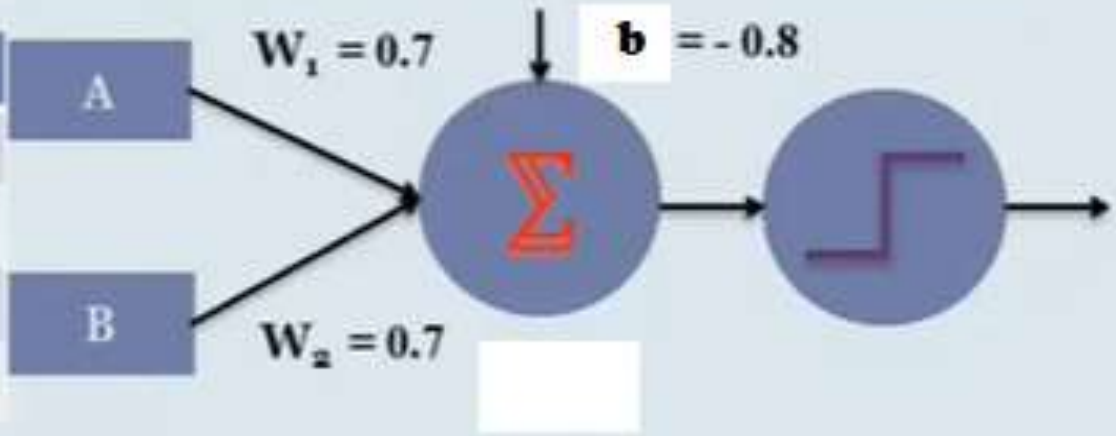
XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



EXAMPLE

- **Example: Representation of AND functions**

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1



If A=0 and B=0

$b + W_1 * A + W_2 * B = -0.8 + 0*0.7 + 0*0.7 = -0.8.$

This is not greater than the threshold of 0, so the output = -1.

CONTD..

- If A= 0 and B=1
- $\mathbf{b} + W_1 * A + W_2 * B = -0.8 \Rightarrow 0*0.7 + 1*0.7 = -0.1.$
- This is not greater than the threshold value of 0, so the output = -1.
- If A= 1 and B=0
- $\mathbf{b} + W_1 * A + W_2 * B = -0.8 + 1*0.7 + 0*0.7 = -0.1.$
- This is not greater than the threshold value of 0, so the output = -1.

CONTD..

- If A= 1 and B=1

$$b + W_1 * A + W_2 * B$$
$$= -0.8 + 1 * 0.7 + 1 * 0.7 = 0.6$$

- This is not greater than the threshold value of 0, so the output = +1.

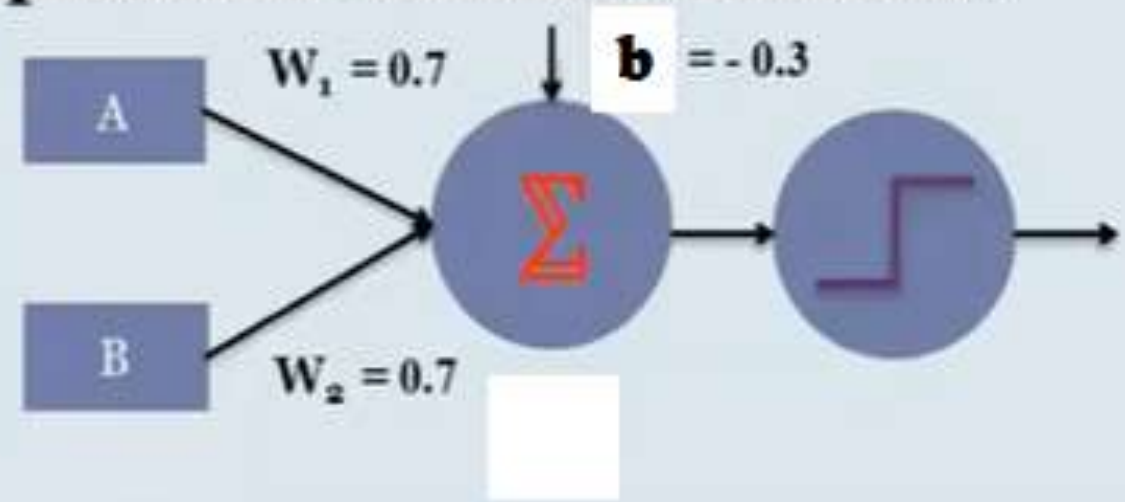
So , the Final weights, b= -0.8 , W1 = 0.7 , W2 = 0.7

By using these weights, All Classified Correctly.

EXAMPLE

- **Example: Representation of OR functions**

A	B	AvB
0	0	0
0	1	1
1	0	1
1	1	1



If A=0 OR B=0

$b + W_1 * A + W_2 * B = -0.3 + 0*0.7 + 0*0.7 = -0.3.$

This is not greater than the threshold of 0, so the output = -1.

CONTD..

- If A= 0 OR B=1
- $b + W_1 * A + W_2 * B = -0.3 + 0*0.7 + 1*0.7 = 0.4$
- This is not greater than the threshold value of 0, so the output = +1.
- If A= 1 OR B=0
- $b + W_1 * A + W_2 * B = -0.3 + 1*0.7 + 0*0.7 = 0.4$
- This is not greater than the threshold value of 0, so the output = +1.

CONTD..

- If A= 1 & B=1

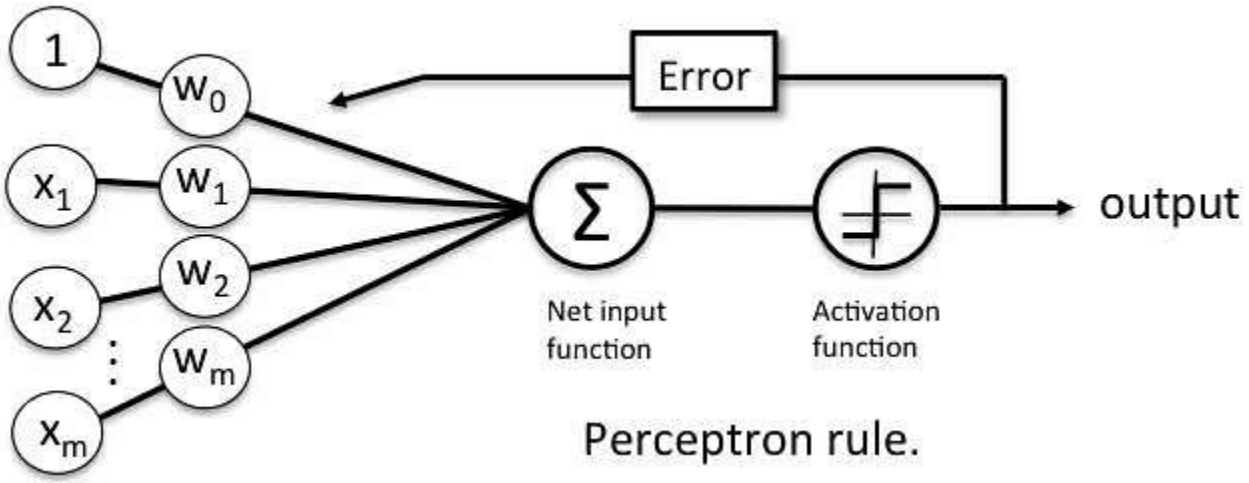
$$b + W_1 * A + W_2 * B$$

$$= -0.3 + 1 * 0.7 + 1 * 0.7 = 1.1$$

- This is not greater than the threshold value of 0, so the output = +1.
- Drawback of perceptron
- The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable

PERCEPTRON LEARNING RULE

- In this Rule, **First begin with random weights**, then **iteratively apply the perceptron** to **each training example**, **modify the perceptron weights** **whenever it misclassifies an example.**



Perceptron rule.

CONTD..

- This **process is repeated**, **iterating through the training examples as many times as needed** until **the perceptron classifies all training examples correctly**
- Here, **Weights are modified** at **each step** according to the **Perceptron Training Rule.**

PERCEPTRON LEARNING RULE

- This **rule revises** the **weight, wi** associated with the **input xi** according to the rule.

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

- Here, t = Targeted Output
- o = Actual Output
- **Learning Rate:** The learning rate is a **hyperparameter** that **controls how much to change the model in response to the estimated error each time** the **model weights** are **updated.**

ALGORITHM

Perceptron_training_rule *(X, η)*
initialize ***w*** *(wi ← an initial (small) random value)*
repeat
  for each training instance ***(x, tx)*** $\in X$
    compute the real output ***ox*** *= Activation(Summation(****w.x****))*
    if ***(tx ≠ ox)***
      for each wi
        $wi \leftarrow wi + \Delta wi$
        $\Delta wi \leftarrow \eta\ (tx - ox)xi$
      end for
    end if
  end for
until all the training instances in X are correctly classified
return ***w***

AND GATE PERCEPTRON TRAINING RULE

- The Truth Table of AND Gate is :

A	B	A^B
0	0	0
0	1	0
1	0	0
1	1	1

- To design the Perceptron Training Rule , we need to have Two Weights.

w1 = 1.2, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

1. A=0, B=0 and Target = 0
   - $wi.xi = 0*1.2 + 0*0.6 = 0$
   - This is not greater than the threshold of 1, so the output = 0

CONTD..

- Here, **Target Output** is **"0"** and **Actual Output** is **"0"**. So **Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

2. A=0, B=1 and Target = 0
   - $wi.xi = 0*1.2 + 1*0.6 = 0.6$
   - This is not greater than the threshold of 1, so the output = 0

- Here, **Target Output** is **"0"** and **Actual Output** is **"0"**. So **Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

CONTD..

3. A=1, B=0 and Target = 0
   - $wi.xi = 1*1.2 + 0*0.6 = 1.2$
   - This is greater than the threshold of 1, so the output = 1

- Here, **Target Output is "0"** and **Actual Output is "1". So Both of them are not same.** So, we **need to modify** the **Weights.**
- So the **Weights are modified** using the **Perceptron Rule.**

$$wi = wi + n(t - o)xi$$

- Where, **η = learning Rate = 0.5** and

xi =(x1,x2) = (1,0)

CONTD..

- So, we have to **apply these values** into **Perceptron Rule.**

$$wi = wi + n(t - o)xi$$
$$w1 = 1.2 + 0.5(0 - 1)1 = 0.7$$
$$w2 = 0.6 + 0.5(0 - 1)0 = 0.6$$

- These are the **two modified Weights**.
- Now **w1= 0.7** and **w2= 0.6**
- **So, again we have to apply this updated weights to all AND operations.**
- So, that here we **can check whether** these **two weights classify** these **examples(AND Operations) correctly or Not.**

CONTD..

- So, the **modified weights** are

w1 = 0.7, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

1. A=0, B=0 and Target = 0
   - *wi.xi = 0*0.7 + 0*0.6 = 0*
   - This is not greater than the threshold of 1, so the output = 0

- Here, **Target Output is "0"** and **Actual Output is "0". So Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

2. A=0, B=1 and Target = 0
   - *wi.xi = 0*0.7 + 1*0.6 = 0.6*
   - This is not greater than the threshold of 1, so the output = 0

CONTD..

- Here, **Target Output** is **"0"** and **Actual Output** is **"0.6"**. This is not **Greater than 1.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

3. A=1, B=0 and Target = 0
   - $wi.xi = 1*0.7 + 0*0.6 = 0.7$
   - This is not greater than the threshold of 1, so the output = 0

- Here, **Target Output** is **"0"** and **Actual Output** is **"0.7"**. This is not **Greater than 1.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

CONTD..

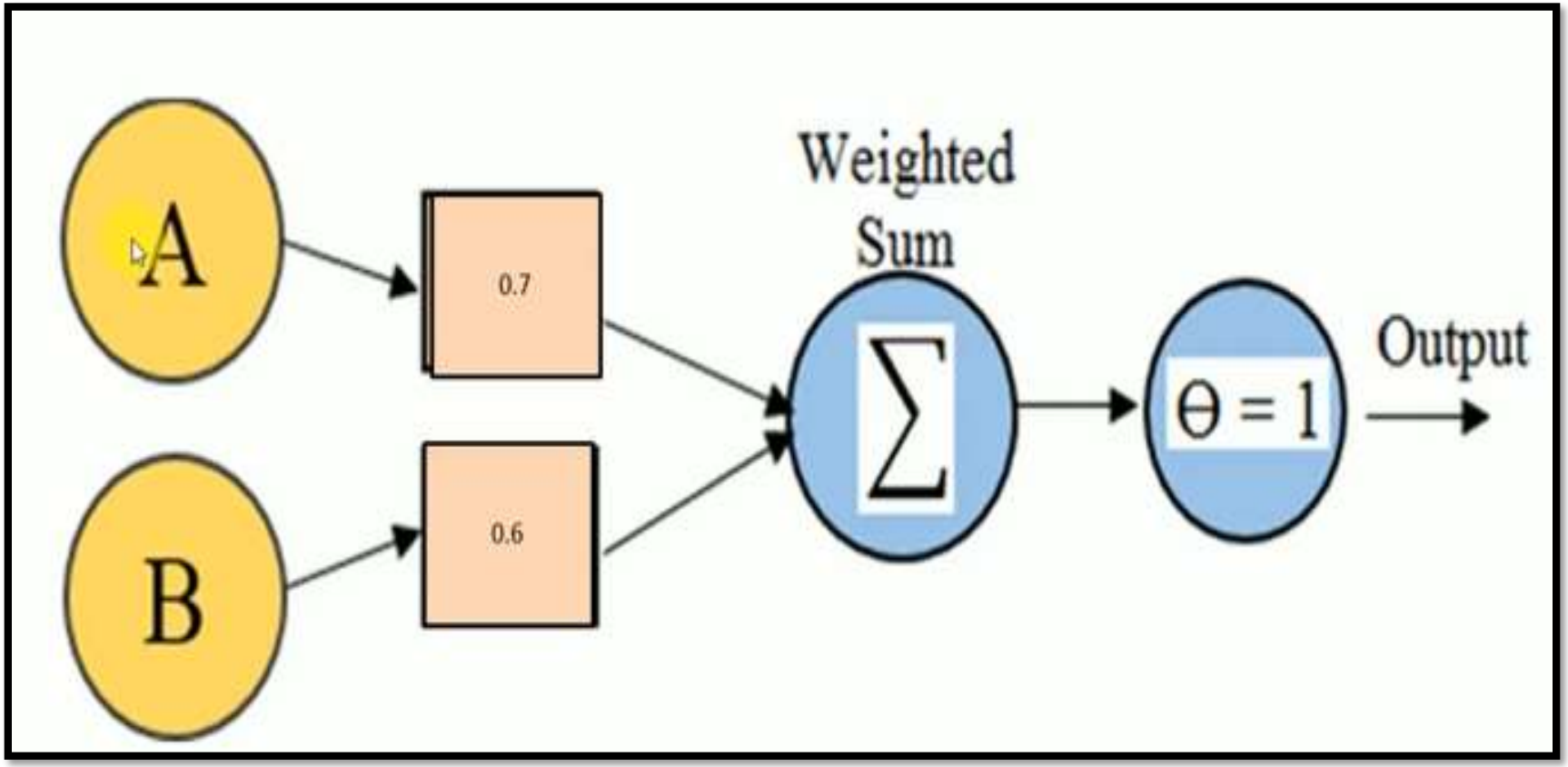
4. A=1, B=1 and Target = 1
   - $wi.xi = 1*0.7 + 1*0.6 = 1.3$
   - This is greater than the threshold of 1, so the output = 1

- Here, **Target Output** is **"1"** and **Actual Output** is **"1.3"**. **Both are Same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**
- **So, classified all training examples correctly with**

w1 = 0.7, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

- Hence, these are the **final Learned Parameters** w.r.to **AND Gate Perceptron Training Rule.**

CONTD..



OR GATE PERCEPTRON TRAINING RULE

- The Truth Table of OR Gate is :

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

- To design the Perceptron Training Rule , we need to have Two Weights. w1 = 0.6, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

1. A=0, B=0 and Target = 0
   - $wi.xi = 0*0.6 + 0*0.6 = 0$
   - This is not greater than the threshold of 1, so the output = 0

CONTD..

- Here, **Target Output** is **"0"** and **Actual Output** is **"0"**. So **Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

2. A=0, B=1 and Target = 1
   - $wi.xi = 0*0.6 + 1*0.6 = 0.6$
   - This is not greater than the threshold of 1, so the output = 0

- Here, **Target Output** is **"1"** and **Actual Output** is **"0.6"**. So **Both of them are not same.** So, we **need to modify** the **Weights.**

CONTD..

So the **Weights are modified** using the **Perceptron Rule.**

$$wi = wi + n(t - o)xi$$

- Where, **n = learning Rate = 0.5** and

  xi =(x1,x2) = (0,1)

- So, we have to **apply these values** into **Perceptron Rule.**

$$wi = wi + n(t - o)xi$$

$$w1 = 0.6 + 0.5(1 - 0)0 = 0.6$$

$$w2 = 0.6 + 0.5(1 - 0)1 = 1.1$$

- These are the **two modified Weights**.
- Now **w1= 0.6** and **w2= 1.1**

CONTD..

So, again we have to apply this updated weights to all OR operations.

- So, that here we **can check whether** these **two weights classify** these **examples(OR Operations) correctly or Not.**
- **So, the Updated Weights are**

w1 = 0.6, w2 = 1.1 Threshold = 1 and Learning Rate n = 0.5

1. A=0, B=0 and Target = 0
   - *wi.xi = 0*0.6 + 0*1.1 = 0*
   - This is not greater than the threshold of 1, so the output = 0

CONTD..

- Here, **Target Output** is **"0"** and **Actual Output** is **"0"**. So **Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

2. A=0, B=1 and Target = 1
   - $wi.xi = 0*0.6 + 1*1.1 = 1.1$
   - This is greater than the threshold of 1, so the output = 1

- Here, **Target Output** is **"1"** and **Actual Output** is **"1"**. So **Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

CONTD..

3. A=1, B=0 and Target = 1
   - $wi.xi = 1*0.6 + 0*1.1 = 0.6$
   - This is not greater than the threshold of 1, so the output = 0

➢ Here, **Target Output is "1"** and **Actual Output is "0". So Both of them are not same.** So, we **need to modify** the **Weights.**

CONTD..

So the **Weights are modified** using the **Perceptron Rule.**

$$wi = wi + n(t - o)xi$$

- Where, **n = learning Rate = 0.5** and

  xi =(x1,x2) = (1,0)

- So, we have to **apply these values** into **Perceptron Rule.**

$$wi = wi + n(t - o)xi$$

$$w1 = 0.6 + 0.5(1 - 0)1 = 1.1$$

$$w2 = 1.1 + 0.5(1 - 0)0 = 1.1$$

- These are the **two modified Weights**.
- Now **w1= 1.1** and **w2= 1.1**

CONTD..

So, again we have to apply this updated weights to all OR operations.

- So, that here we **can check whether** these **two weights classify** these **examples(OR Operations) correctly or Not.**
- **So, the Updated Weights are**

w1 = 1.1, w2 = 1.1 Threshold = 1 and Learning Rate n = 0.5

1. A=0, B=0 and Target = 0
   - *wi.xi = 0*1.1 + 0*1.1 = 0*
   - This is not greater than the threshold of 1, so the output = 0

CONTD..

- Here, **Target Output** is **"0"** and **Actual Output** is **"0"**. So **Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

2. A=0, B=1 and Target = 1
   - *wi.xi = 0*1.1 + 1*1.1 = 1.1*
   - This is greater than the threshold of 1, so the output = 1

- Here, **Target Output** is **"1"** and **Actual Output** is **"1"**. So **Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

CONTD..

3. A=1, B=0 and Target = 1
   - *wi.xi = 1*1.1 + 0*1.1 = 1.1*
   - This is greater than the threshold of 1, so the output = 1

➢ Here, **Target Output is "1"** and **Actual Output is "1". So Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**

4. A=1, B=1 and Target = 1
   - *wi.xi = 1*1.1 + 1*1.1 = 2.2*
   - This is greater than the threshold of 1, so the output = 1

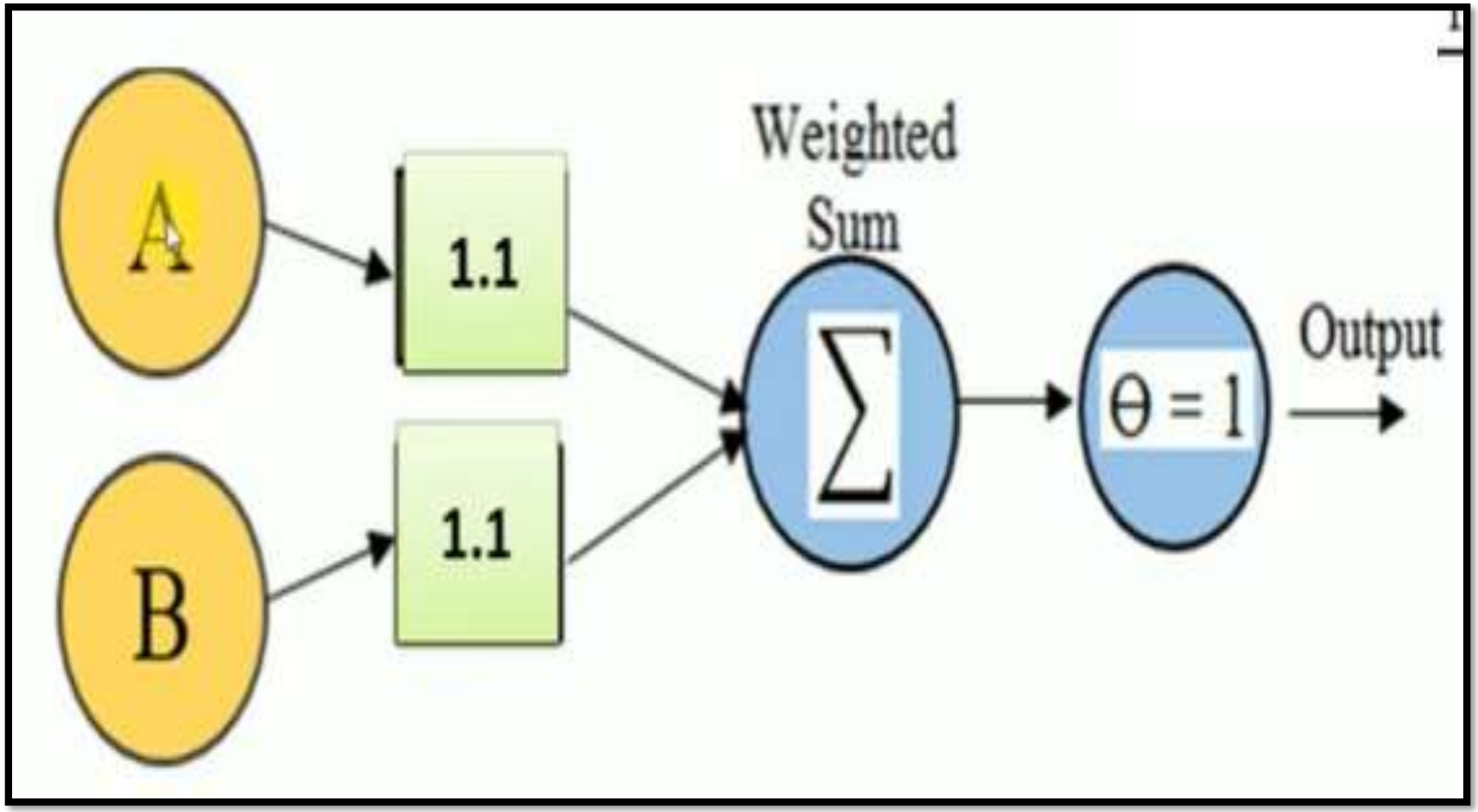
CONTD..

- Here, **Target Output** is **"1"** and **Actual Output** is **"1"**. So **Both of them are same.** So, there is **no need of change** the **Weights.(no need of Weight Updation)**
- **So, classified all training examples correctly with**

$w1 = 1.1, w2 = 1.1$ Threshold = 1 and Learning Rate $n = 0.5$

- Hence, these are the **final Learned Parameters** w.r.to **OR Gate Perceptron Training Rule.**

CONTD..



Real time Applications of Perceptron's

Any Digital Logic Circuit can be implemented by using Perceptron.

THANK YOU

UNIT-II

(Multi Layer Networks and the Back Propagation Algorithm)

TOPICS

Topics :

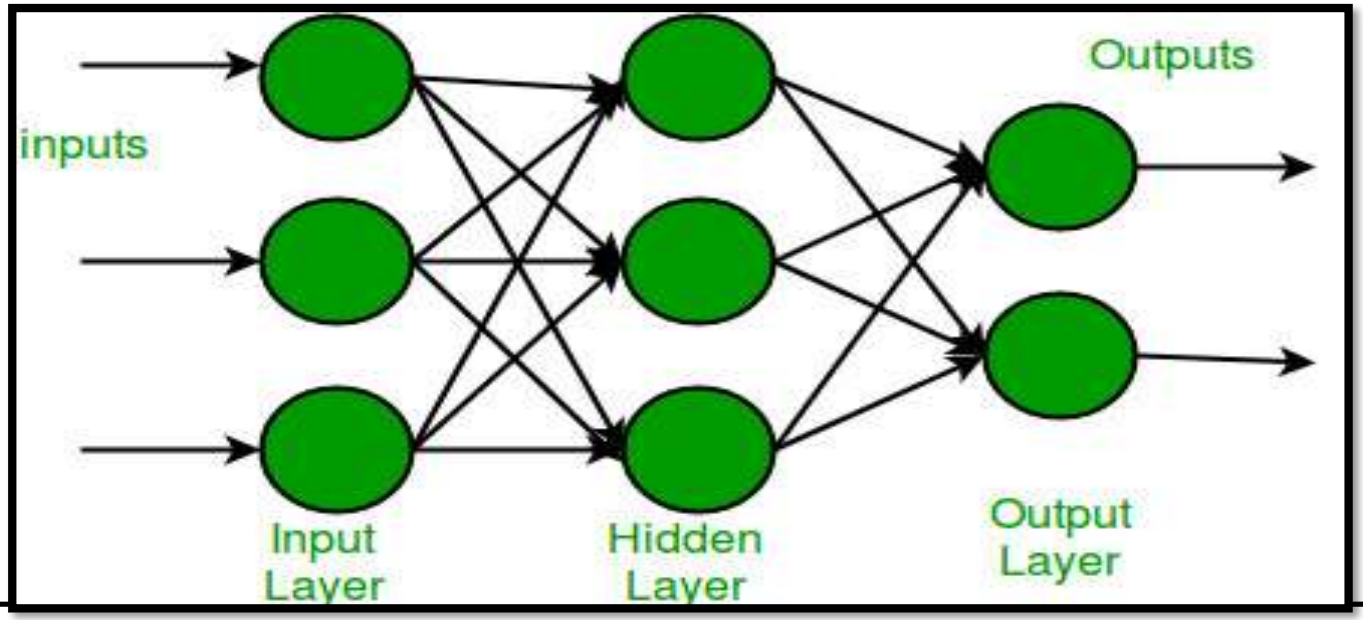
- Multi Layer Neural Networks
- Back Propagation Algorithm
- Example Problem

MULTI LAYER NETWORKS

- A **fully connected Multi-Layer Neural Network** is called a **Multilayer Perceptron (MLP).**
- The **Multi-Layer-Perceptron** was **first introduced** by **M. Minsky and S. Papert in 1969.** It is an **extended Perceptron** and has **one ore more hidden neuron layers between its input and output layers**
- Due to its extended structure, a **Multi-Layer-Perceptron** is **able to solve every logical operation, including the XOR problem.**

CONTD..

- Multilayer perceptron's (MLPs) are **feed forward neural networks** **trained with** the standard **back propagation algorithm.**
- By using the **Back propagation**, is a procedure to **repeatedly adjust the weights** so as to **minimize the difference between actual output and desired output**



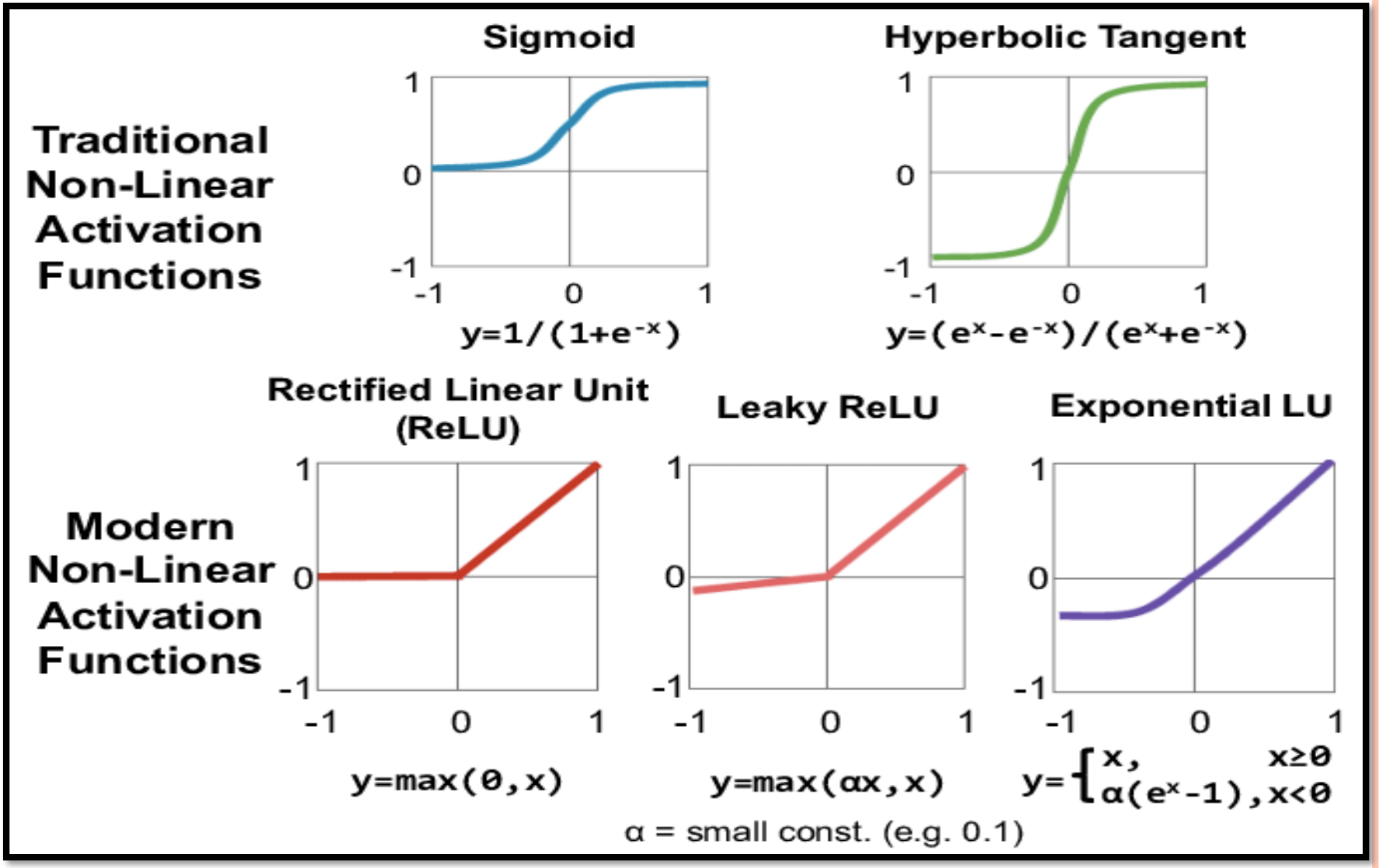
CONTD..

- In the multi-layer perceptron diagram above, we can see that there are **three input nodes** and the **hidden layer** has **three nodes**. The **output layer** gives **two outputs**, therefore there are **two output nodes.**
- The nodes in the **input layer take input** and **forward it for further process,** in the diagram above the nodes in the **input layer forwards their output to each of the three nodes**
- In the **hidden layer**, and in the same way, **the hidden layer processes the information and passes it to the output layer**.

CONTD..

- Hence, **a multi-layered perceptron model** has considered as **artificial neural networks** having **various layers** in which **activation function does not remain linear**, similar to a **single layer perceptron model**.
- In this Networks, we used the **Non Linear Activation Functions used.ie, Sigmoid, Relu, Tanh etc.** Since the probability of any event lies **between 0 and 1.**
- In Multi Layer Networks, We can follow the **Supervised Learning.**

Different Non Linear Activation Functions



BACK PROPAGATION ALGORITHM

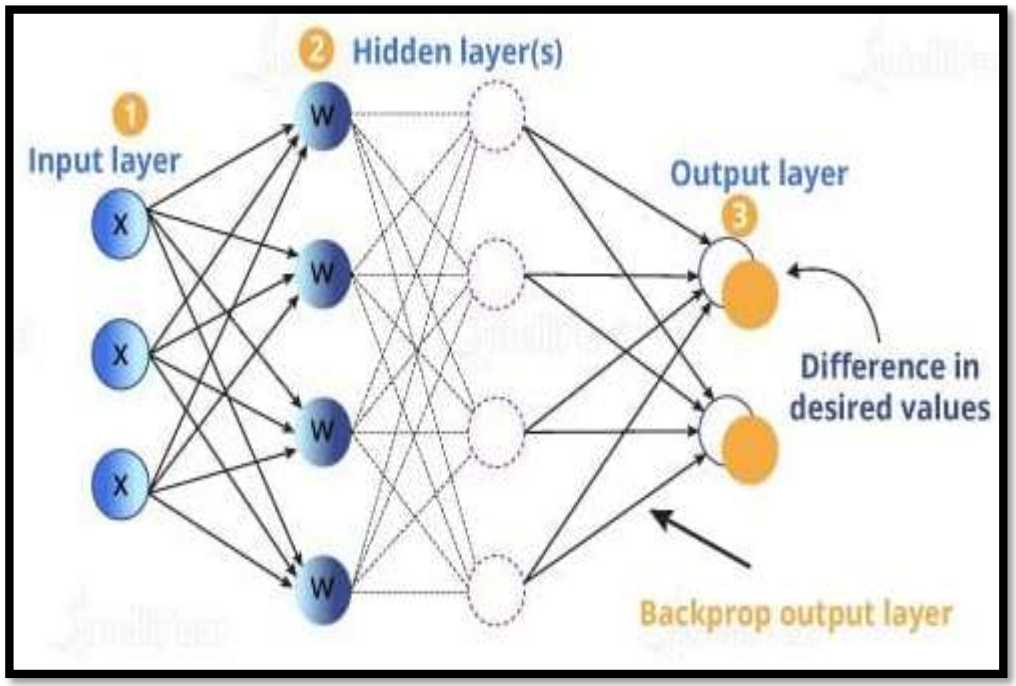
- In an **Artificial Neural Network**, the **values of weights and biases** are **randomly initialized**.
- Due to **random initialization**, the **neural network probably has errors for the given inputs.**
- So, We need to **reduce error values** as much as possible.
- So, for **reducing these error values,** we need a mechanism that can **compare the desired output of the neural network** with the **Target network's output**

CONTD..

- Suppose, that **consists of errors** and **adjusts its weights and biases** such that it **gets closer to the desired output after each iteration**.
- For this, we **train the network** such that it **back propagates and updates the weights and biases**. This is the concept of the **back propagation algorithm.**
- Below are the steps that an artificial neural network follows to **gain maximum accuracy** and **minimize error values**

CONTD..

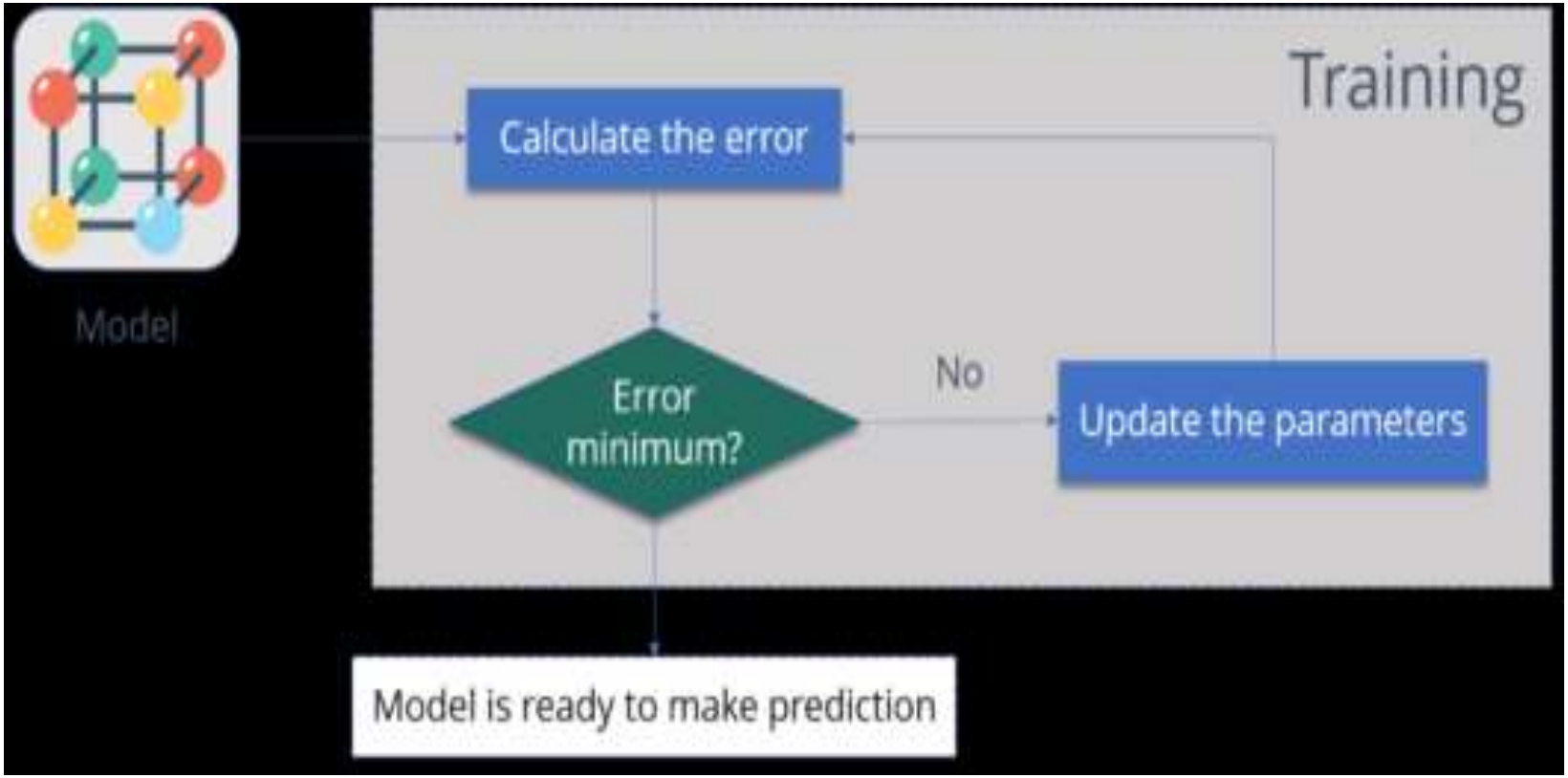
- **Steps:**
  - **1. Parameter Initialization**
  - **2. Feed Forward Propagation**
  - **3. Back Propagation**



CONTD..

One way to train our model is called as Back propagation.

Consider the diagram below:



CONTD..

Let me summarize the steps for you:

Calculate the error – How far is your model output from the actual output.

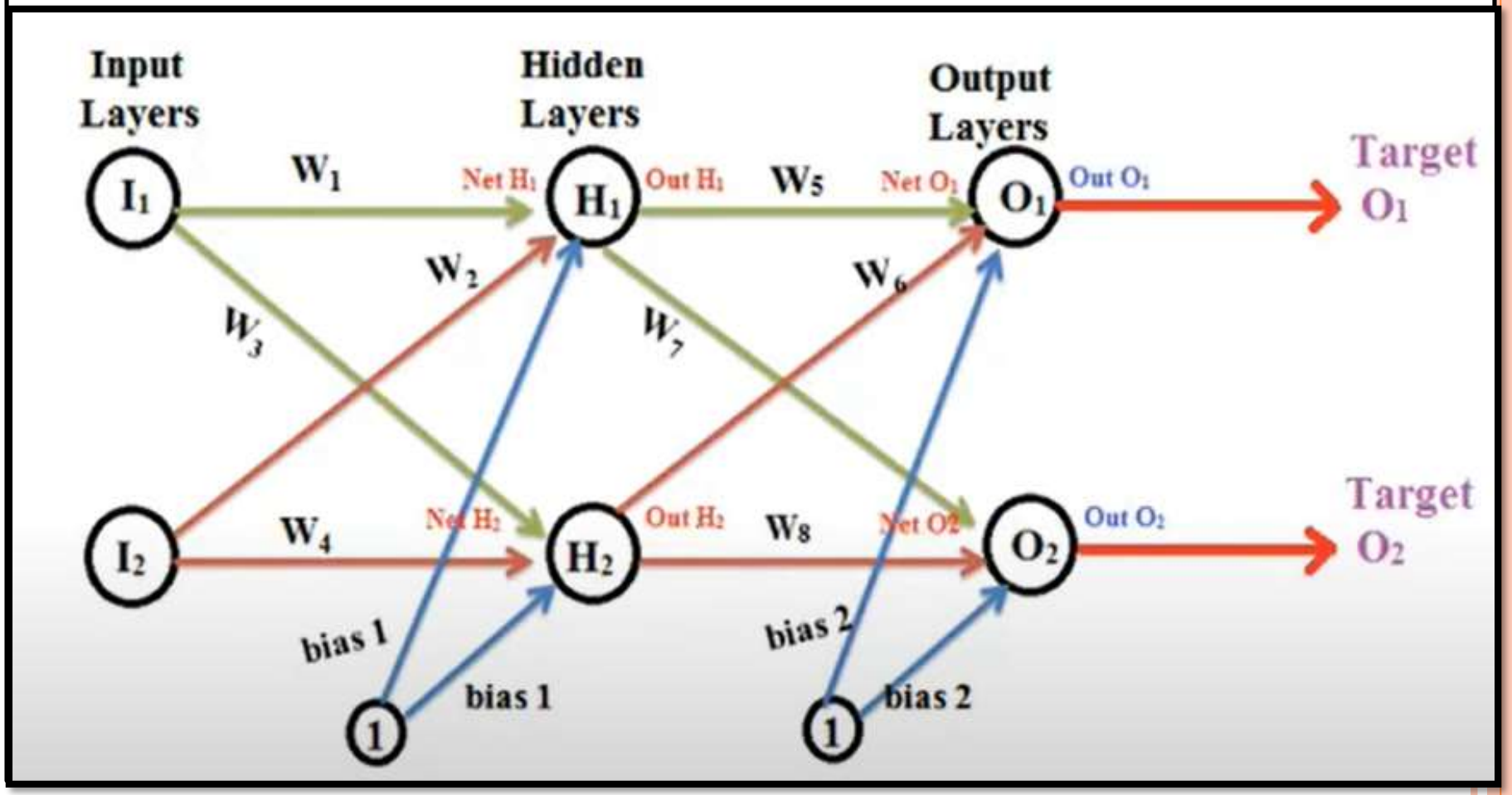
Minimum Error – Check whether the error is minimized or not.

Update the parameters – If the **error is huge** then, **update the parameters (weights and biases).** After that again check the error. Repeat the process until the error becomes minimum.

Model is ready to make a prediction – Once the **error becomes minimum**, you can **feed some inputs to your model and it will produce the output.**

BACK PROPAGATION ALGORITHM

- The below Figure show you the Network: i.e, Feed Forward Network.



CONTD..

- In the above Figure,
- Input layer with two inputs neurons
  - I1, I2 are the Input Layers
- One hidden layer with two neurons
  - H1, H2 are the Hidden Layers
- Output layer with two neurons
  - O1,O2 are the Output Layers
- Target O1, Target O2 are the Desired Output (or) the Expected Output

CONTD..

- Here I1, I2 are the Input Layers and W1, W2, W3, W4 are the corresponding Weights.
- From the above Network, First we can Calculate the Net H1 and Net H2.

$$net\ H_1 = I_1.W_1 + I_2.W_2 + b_1 \quad .....(1)$$

$$net\ H_2 = I_1.W_3 + I_2.W_4 + b_1 \quad .....(2)$$

CONTD..

- After Calculating **net H1 and net H2.** compute **Out H1 and Out H2** as follows:

$$Out\ H_1 = \frac{1}{1+e^{-net H_1}} \quad .....(3)$$

$$Out\ H_2 = \frac{1}{1+e^{-net H_2}} \quad .....(4)$$

CONTD..

- Here we can use the **Activation Function** is the **sigmoid function** ,the values for which it is used in the range, **0 to 1.**
- It is used for models where we have to **predict the probability.** Since the probability of any event lies **between 0 and 1**, the **sigmoid function** is the **right choice.**

CONTD..

- **Similarly, we can Calculating net O1 and net O2.**

$$net\ O_1 = W_5 . out\ H_1 + W_6 . out\ H_2 + b_2 \quad .....(5)$$

$$net\ O_2 = W_7 . out\ H_1 + W_8 . out\ H_2 + b_2 \quad .....(6)$$

CONTD..

- From the above Equations, we can calculate Out O1 and Out O2 as follows:

$$Out\ O_1 = \frac{1}{1+e^{-netO_1}} \quad .....(7)$$

$$Out\ O_2 = \frac{1}{1+e^{-netO2}} \quad .....(8)$$

CONTD..

- Now , Set some **Target Values** for **both Outputs** and **Calculate Error Outputs** based on **Target and Calculated Outputs.**

$$E_{O_1} = \frac{1}{2}\{Target\ O_1 - Output\ O_1\}^2$$

Here, E_{o1} is the **Error at Output O1**

$$E_{O_2} = \frac{1}{2}\{Target\ O_2 - Output\ O_2\}^2$$

Here, E_{o2} is the **Error at Output O2**

- Next, we can Calculating the **Total Error**

$$E_T = E_{O_1} + E_{O_2} \quad .....(9)$$

Here, E_T is the **Total Error**

CONTD..

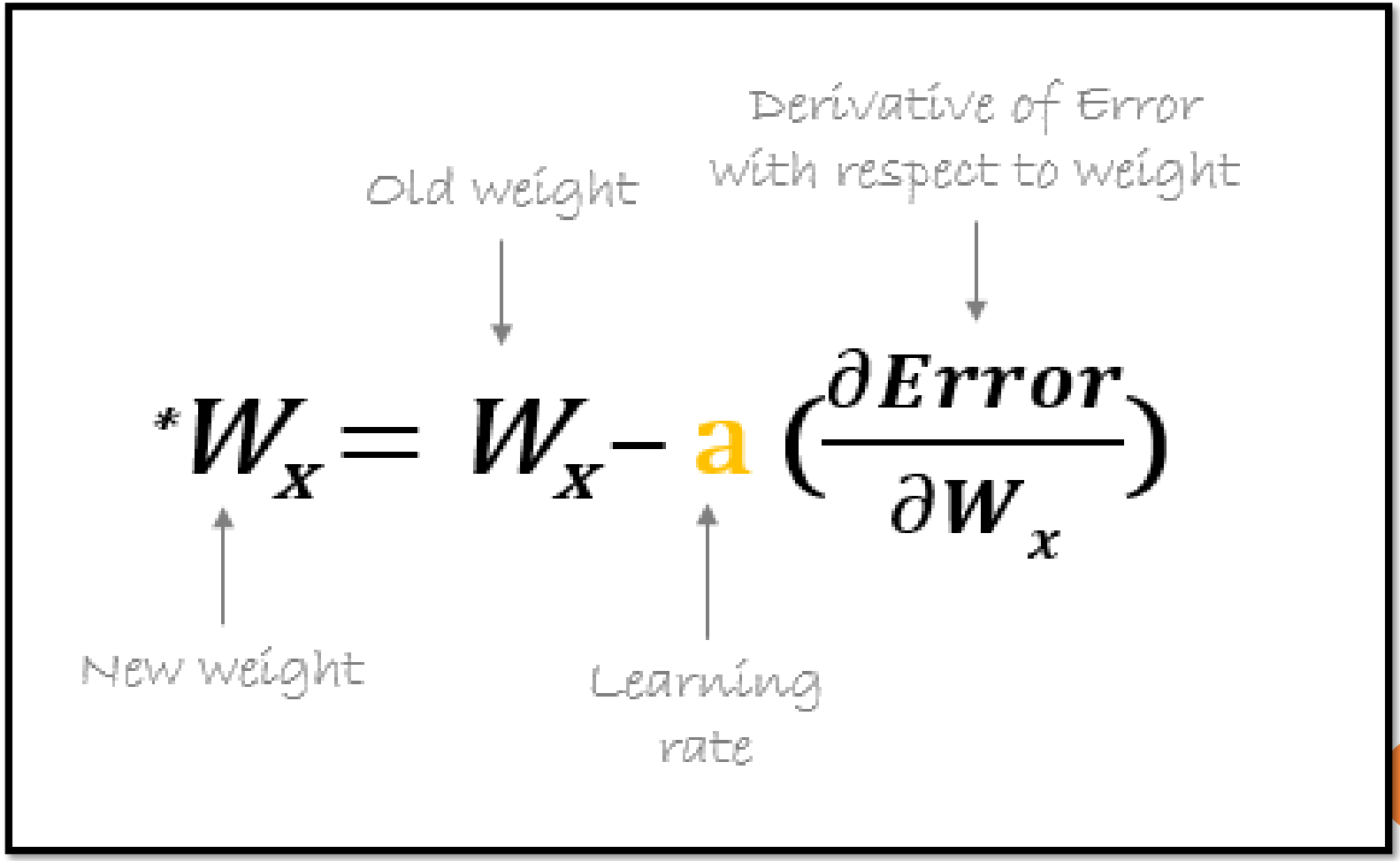
- Our **main goal** of the **training** is to **reduce the error** or the difference between **prediction and actual output.**
- Suppose there is an **Error occurred** , we will use **" Back Propagation"**

BACK PROPAGATION

- **Back propagation,** short for **"backward propagation of errors",** is a mechanism used to **update the weights** using gradient descent.
- **Gradient descent** is an **iterative optimization algorithm** for **finding the minimum of a function**.
- In our case we want to **minimize the error function**.
- It calculates the **gradient of the error function** with respect to the **neural network's weights**.
- The calculation proceeds **backwards through the network**.

CONTD..

- **The Formula for the Gradient Descent is:**



CONTD..

- **2. Back ward Pass:**
- When we **got the Error**. i.e, **Target Output O1 and Actual Output O1.**
- Similarly, **Target Output O2 and Actual Output O2.**
- So that we have to **move Backward** and we have to **adjust the weights** of **W5, W6, W7, W8.**
- Now we have to **Calculate** the **Output Values.**
- If we get the **Expected Output and Actual Outputs are Same**, **We will stop**.

CONTD..

- Otherwise again **we will back and Update** the **weights** of **W1, W2, W3, W4** also.
- This process continues unless until the **Actual Output and Target Output are the Same.**
- **Update the Weights: Here we can use the Gradient Descent.**
- **i.e, for suppose we have to update the weights : W5**
- **1. For W5:**

$$W_5^+ = W_5 - a\left(\frac{\partial Error}{\partial W_5}\right)$$

CONTD..

- Here, **W5** has to be updated w.r.to the **Total Error.**
- So here **first calculate** the

$$\left(\frac{\partial Error}{\partial W_5}\right)$$

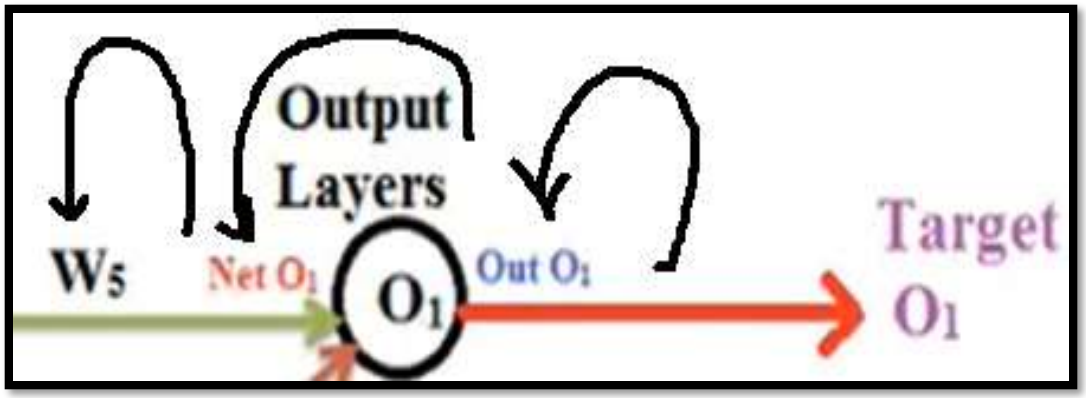
- **This can be written as**

$$\frac{\partial E_{Total}}{\partial_{W_5}} = \frac{\partial E_{Total}}{\partial_{OutO_1}} \cdot \frac{\partial_{OutO_1}}{\partial_{netO_1}} \cdot \frac{\partial_{netO_1}}{\partial_{W_5}} \quad \ldots\ldots(10)$$

- So, here will calculate **each and every term separately**.

CONTD..

- Here we can apply the **Chain Rule**.



- **First term,**

$$\frac{\partial E_{Total}}{\partial_{OutO_1}} = \frac{\partial(E_{O_1} + E_{O_2})}{\partial_{OutO_1}} \rightarrow (11)$$

CONTD..

- we will written as,

$$\frac{\partial E_{Total}}{\partial_{OutO_1}} = \frac{\frac{1}{2}\{Target\ O_1 - Output\ O_1\}^2 + \frac{1}{2}\{Target\ O_2 - Output\ O_2\}^2}{\partial_{OutO_1}}$$

- **After apply the partial Derivation, we will get :**

$$\frac{\partial E_{Total}}{\partial_{OutO_1}} = \frac{1}{2}.2\{Target\ O_1 - Output\ O_1\}.(-1) + 0$$

CONTD..

- **After Simplifying,**

$$\frac{\partial E_{Total}}{\partial_{OutO_1}} = -\{Target\ O_1 - Output\ O_1\} \ldots\ldots\ldots (12)$$

CONTD..

- **Second term,**

$$\frac{\partial (\text{out } o_1)}{\partial \text{net } o_1} = \frac{1}{\partial \text{net } o_1}\left(\frac{1}{1+e^{-\text{net } o_1}}\right)$$

- Assume net $O_1 = x$

$$= \frac{1}{\partial x}\left[\frac{1}{1+e^{-x}}\right]$$

CONTD..

- **Apply the partial Derivation, we will get :**

$$= \frac{1}{\partial x}\left[\frac{1}{1+e^{-x}}\right]$$

$$= \frac{1}{\partial x}\left[(1+e^{-x})^{-1}\right]$$

$$= -1 \cdot (1+e^{-x})^{-1-1} \cdot (-1) \cdot e^{-x}$$

$$= (1+e^{-x})^{-2} \cdot e^{-x} = \frac{e^{-x}}{(1+e^{-x})^{2}}$$

CONTD..

- **After apply the partial Derivation, we will get :**

$$= \frac{1}{(1+e^{-x})} \cdot \frac{1}{(1+e^{-x})} \times e^{-x}$$

$$= \frac{1}{(1+e^{-x})} \cdot \left[1 - \frac{1}{1+e^{-x}}\right]$$

$$= out_{o_1} * (1 - out_{o_1}) \quad \left[\because out_{o_1} = \frac{1}{1+e^{-x}}\right]$$

$\longrightarrow$ (13)

CONTD..

- **Third term,**

$$\frac{\partial (net\,o_1)}{\partial w_5} = \frac{\partial}{\partial w_5}\left[w_5 * out\,H_1 + w_6 * out\,H_2 + b_2 * 1\right]$$

$$= out\,H_1 \longrightarrow (14)$$

- **By combining, all these terms**

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial out\,o_1} * \frac{\partial out\,o_1}{\partial net\,o_1} * \frac{\partial net\,o_1}{\partial w_5}$$

$$= -\left[Target\,o_1 - output\,o_1\right] * out\,o_1\left[1 - out\,o_1\right] * out\,H_1$$

CONTD..

- After Calculating the $\left(\frac{\partial Error}{\partial W_5}\right)$

- **Update the Weight , W5 as**

$$W_5^+ = W_5 - a\left(\frac{\partial Error}{\partial W_5}\right)$$

- Here,
- **W5** is the **Old Value**
- **a** – is the **Learning Rate or Step Rate.**

CONTD..

Similarly Update all the Weights W_6 as W_6^+

W_1 as W_1^+

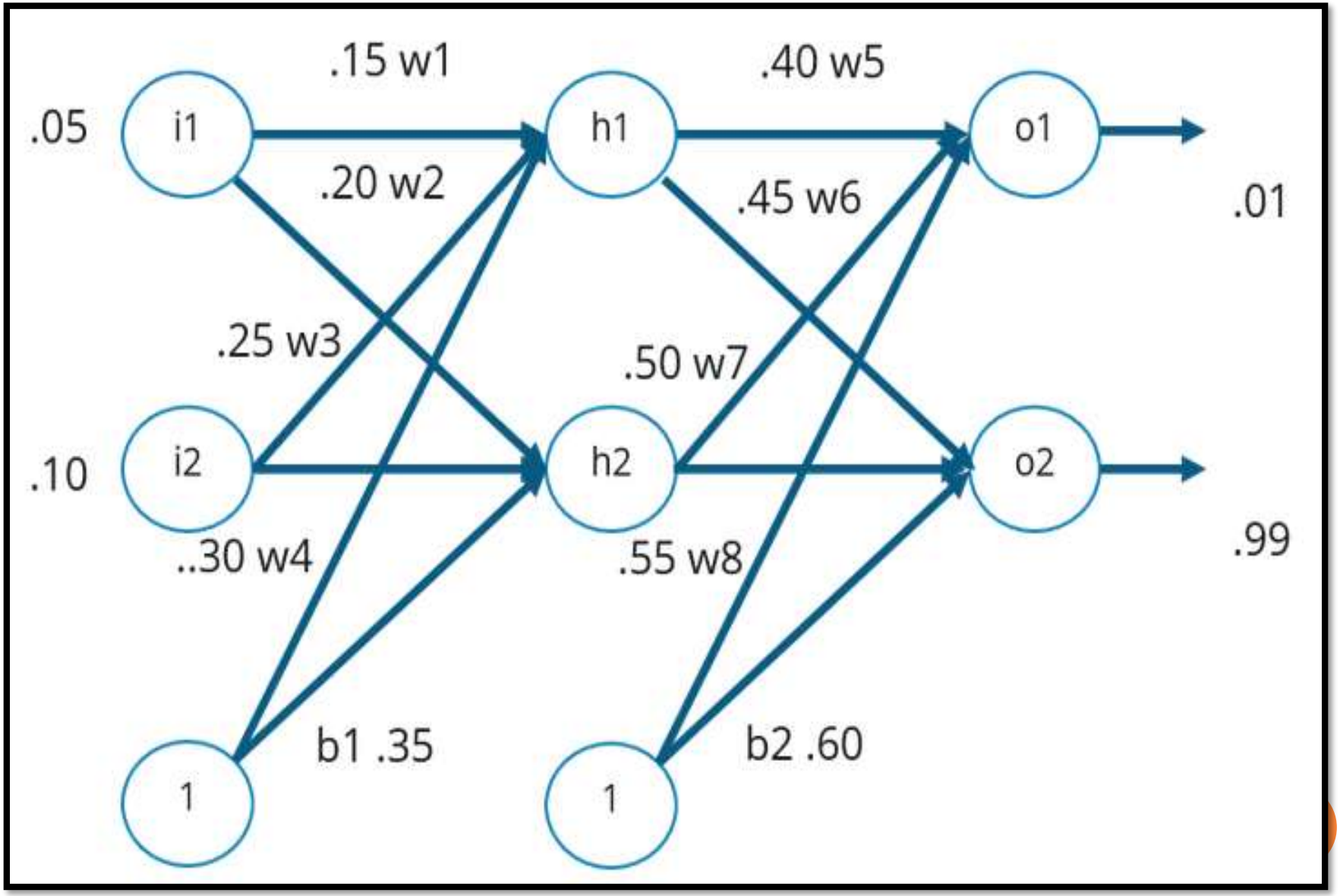
W_2 as W_2^+ etc

Now, with new Updated weights

Calculate new Total Error (E_T) & Repeat the

Procedure until E_T is equal to zero.

Example Problem



CONTD..

The above **network contains the following:**

1. **two inputs : X1 and X2**
2. **two hidden neurons : h1 and h2**
3. **two output neurons : o1 and o2**
4. **two biases : b1 and b2**

Below are the **steps involved in Back propagation:**

Step – 1: Forward Propagation

Step – 2: Backward Propagation

Step – 3: Putting all the values together and calculating the updated weight value

CONTD..

Assume the following values:

$I_1 = 0.05$	$W_1 = 0.15$	$W_5 = 0.4$	$b_1 = 0.35$
$I_2 = 0.10$	$W_2 = 0.2$	$W_6 = 0.45$	$b_2 = 0.6$
	$W_3 = 0.25$	$W_7 = 0.5$	
	$W_4 = 0.3$	$W_8 = 0.55$	

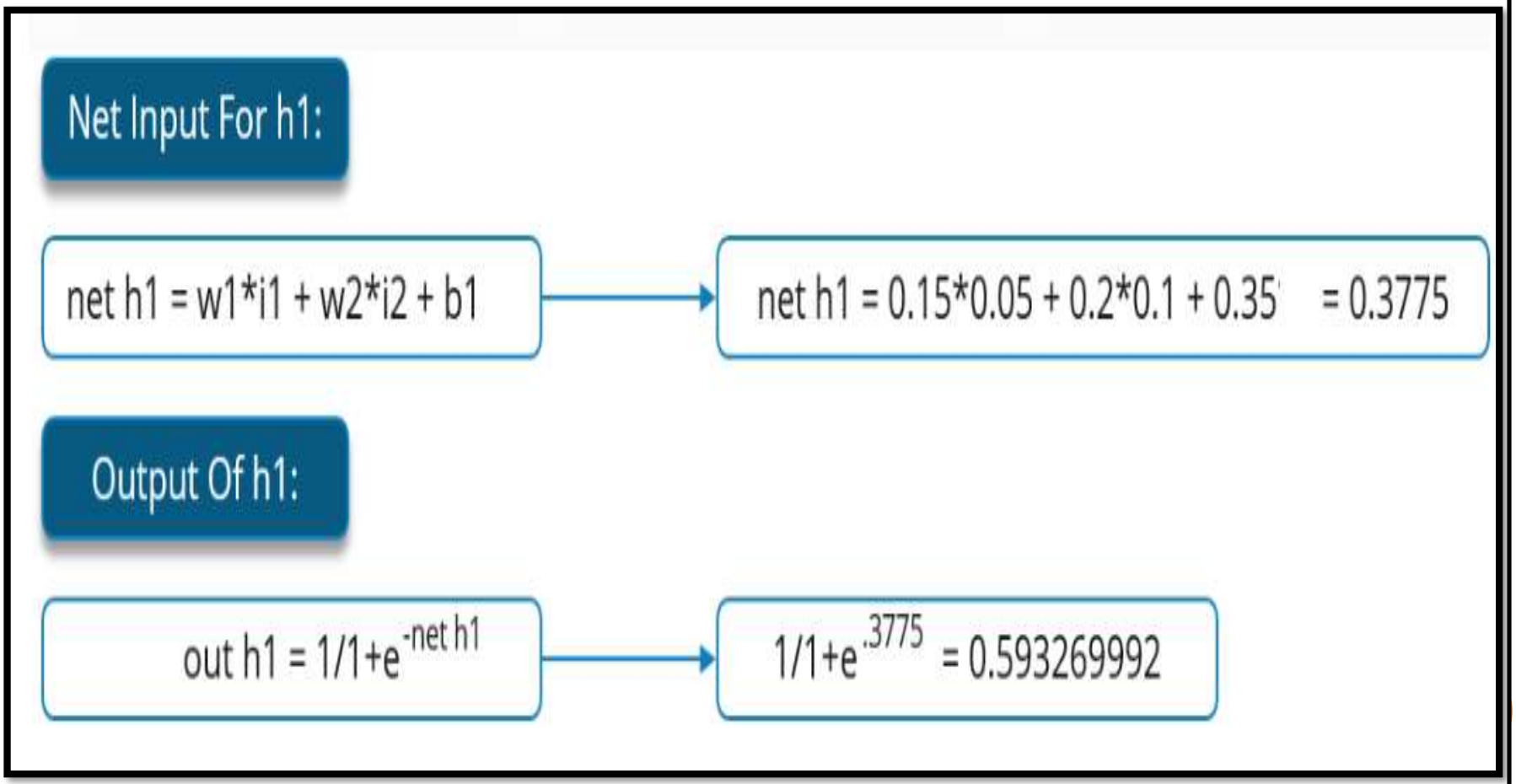
Target $O_1 = 0.01$

Target $O_2 = 0.99$

CONTD..

Step – 1: Forward Propagation :

We will start by **propagating forward**.



CONTD..

Net Input For h2:

Net h2 = i1 * W3 + i2 * W4 + B1

= 0.05 * 0.25 + 0.10 * 0.30 + 0.35 = 0.3925

$$\text{Out } h2 = 1/1+e^{-\text{net } h2}$$

$$= 1/1+e^{-0.3925}$$

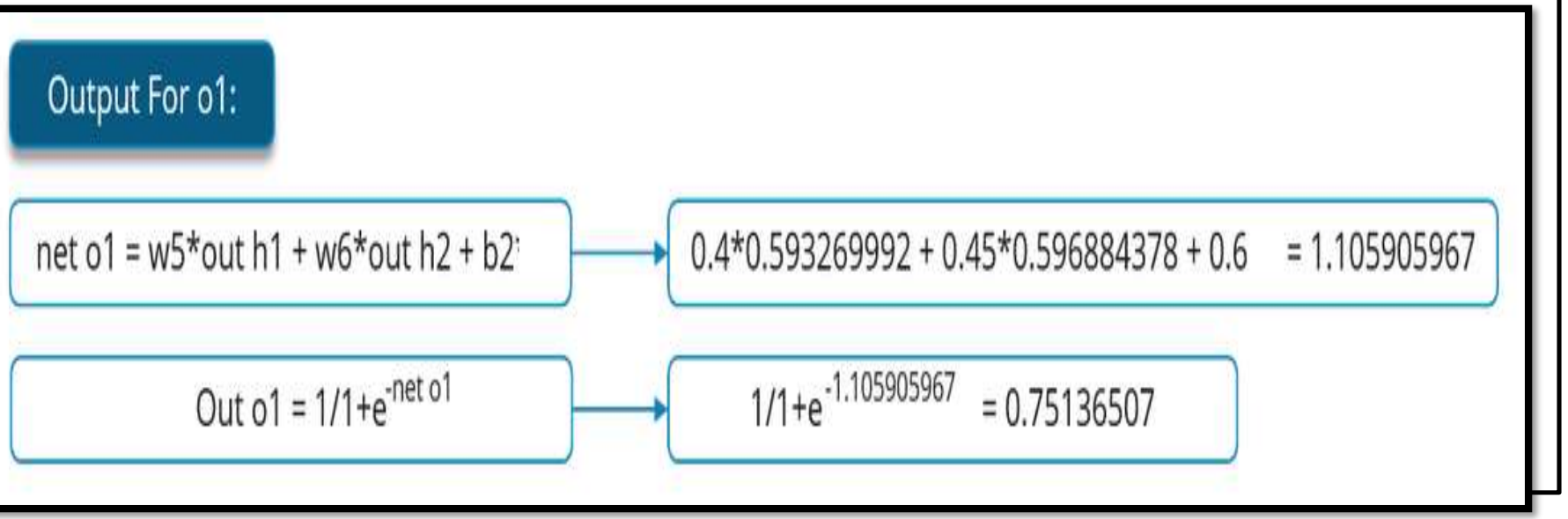
= 0.59688

Output Of h2:

out h2 = 0.596884378

CONTD..

We will **repeat this process** for the **output layer neurons**, using the **output from the hidden layer neurons as inputs**.



CONTD..

$$\text{Net o2} = w7 * \text{out h1} + w8 * \text{out h2} + b2$$

$$= 0.50 * 0.5932 + 0.55 * 0.59688 + 0.60$$

$$= 1.22286$$

$$\text{Out o2} = 1/1 + e^{-\text{net o2}}$$

$$= 1/\ 1+ e^{-1.22286}$$

Output For o2:

Out o2 = 0.772928465

CONTD..

Calculating the Total error:

Error For o1:

$E_{o1} = \Sigma 1/2(target - output)^2$ → $\frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$

Error For o2:

$E_{o2} = 0.023560026$

Total Error:

$E_{total} = E_{o1} + E_{o2}$ → $0.274811083 + 0.023560026 = 0.298371109$

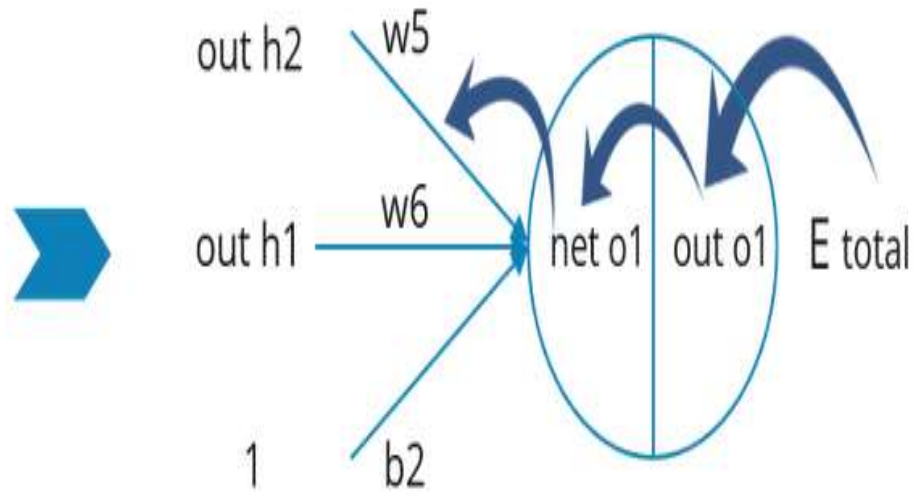
CONTD..

Step – 2: Backward Propagation

Now, we will **propagate backwards**. This way we will try to **reduce the error** by **changing the values of weights and biases.**

Consider **W5,** we will calculate the rate of change of error w.r.t

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$



CONTD..

Calculate Each and Every Term:

First Term:

$$E_{total} = 1/2(\text{target o1} - \text{out o1})^2 + 1/2(\text{target o2} - \text{out o2})^2$$

$$\frac{\delta Etotal}{\delta out\ o1} = -(\text{target o1} - \text{out o1}) = -(0.01 - 0.75136507) = 0.74136507$$

CONTD..

Calculate Second Term:

$$\text{out o1} = 1/1+e^{-neto1}$$

$$\frac{\delta out\ o1}{\delta net\ o1} = \text{out o1 (1 - out o1)} = 0.75136507\ (1 - 0.75136507) = 0.186815602$$

Calculate Third Term:

$$\frac{\delta net\ o1}{\delta w5} = \text{out h1} = 0.593269992$$

CONTD..

Step – 3: Putting all the values together and **calculating the updated weight value**

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$

0.082167041

$$= -[0.01 - 0.7513] * 0.7513[1-0.7513] * 0.5932$$

$$= 0.082$$

CONTD..

Let's calculate the updated value of W5:

$$w5^{+} = w5 - n\frac{\delta Etotal}{\delta w5}$$

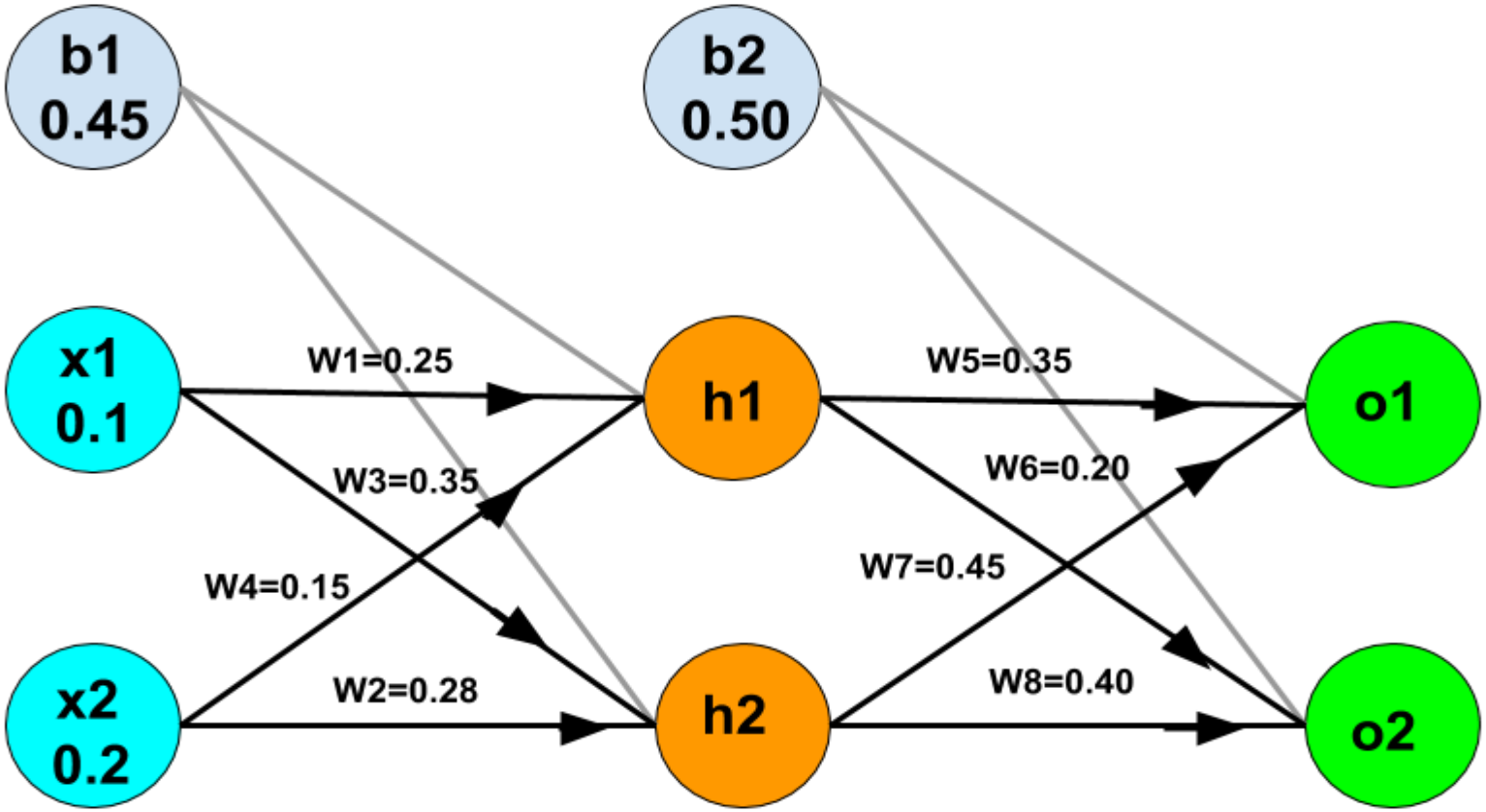
$$w5^{+} = 0.4 - 0.5 * 0.082167041$$

Updated w5 → 0.35891648

CONTD..

- Similarly, we can **calculate the other weight values as well.**
- After that we will **again propagate forward** and **calculate the output.** Again, **we will calculate the error.**
- If the **error is minimum** we **will stop right there**, else we will **again propagate backwards** and **update the weight values.**
- This **process will keep on repeating until error becomes minimum.**

EXAMPLE PROBLEM - 2



The correct output from output node *o1* and *o2* be *y1* and *y2* respectively. Let's assume the value of *y1 = 0.05* and the value of *y2 = 0.95* which are the correct outputs labeled for the given inputs.

THANK YOU