# Deep Learning Techniques Course Code:20AIM61A

Module 1

# About the Course

**Course Outcome**

- Understand the concepts of Neural networks, its main functions, operations and the execution pipeline
- Apply deep learning algorithms, neural networks and traverse the layers of data abstraction which will empower the student to understand data more precisely.
- Analyze deep learning models in Tensor Flow and interpret the results
- Design convolutional neural networks, training deep networks and high-level interfaces
- Use the language and fundamental concepts of artificial neural networks to solve real world problems.

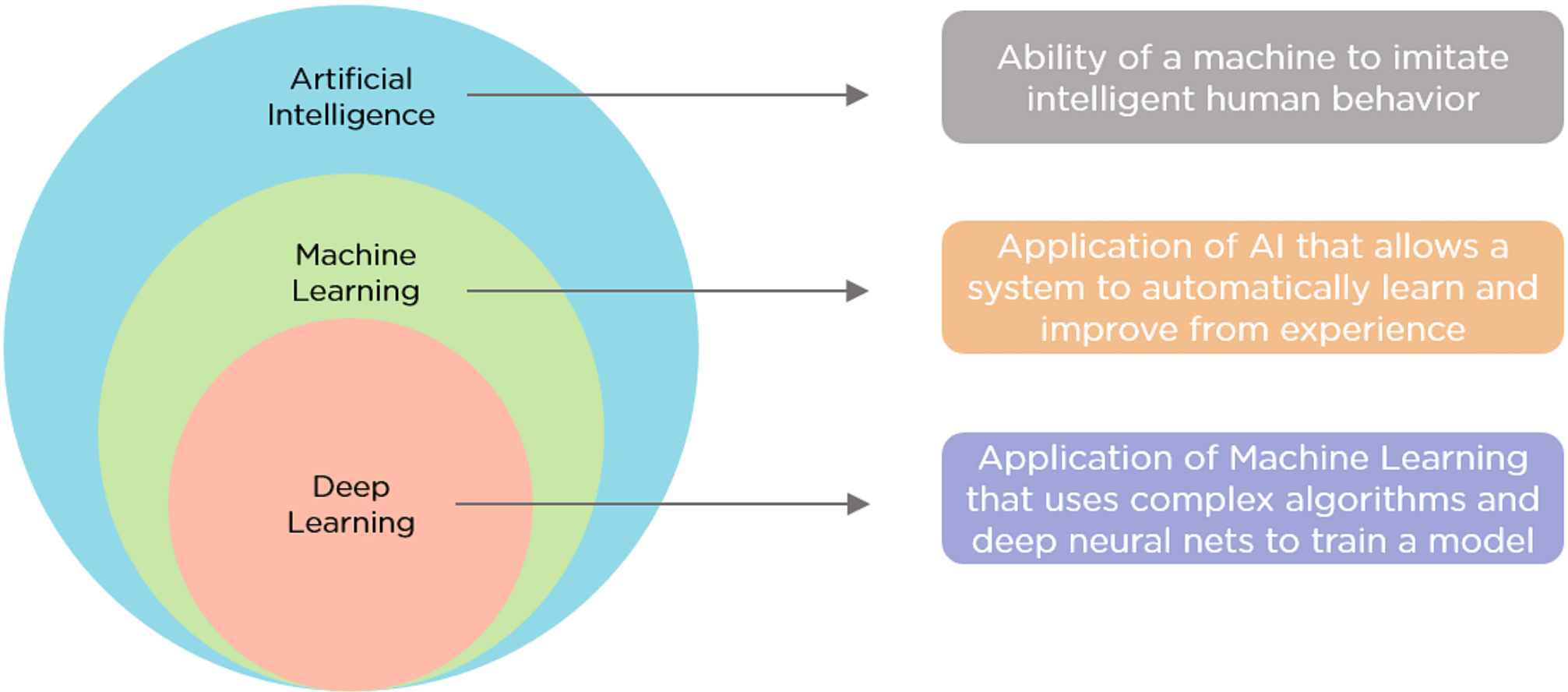
**Text Books**


1. Ian Good Fellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, MIT Press, 2017
2. Satish Kumar, Neural Networks: A Classroom Approach, Tata McGraw-Hill Education, 2004
3. Francois Chollet, “Deep Learning with Python”, Manning Publications, 2018


**Reference Book**


1. Yegnanarayana, B., Artificial Neural Networks PHI Learning Pvt. Ltd, 2009
2. Golub, G.,H., and Van Loan,C.,F., Matrix Computations, JHU Press,2013

# Artificial Intelligence, Machine Learning and Deep Learning



# What is Artificial Intelligence?

- Artificial intelligence, commonly referred to as AI, is the process of imparting data, information, and human intelligence to machines.
- The main goal of Artificial Intelligence is to develop self-reliant machines that can think and act like humans.
- These machines can mimic human behavior and perform tasks by learning and problem-solving.

# Application

# Applications

- Machine Translation such as Google Translate
- Self Driving Vehicles such as Google’s Waymo
- AI Robots such as Sophia and Aibo
- Speech Recognition applications like Apple’s Siri or OK Google

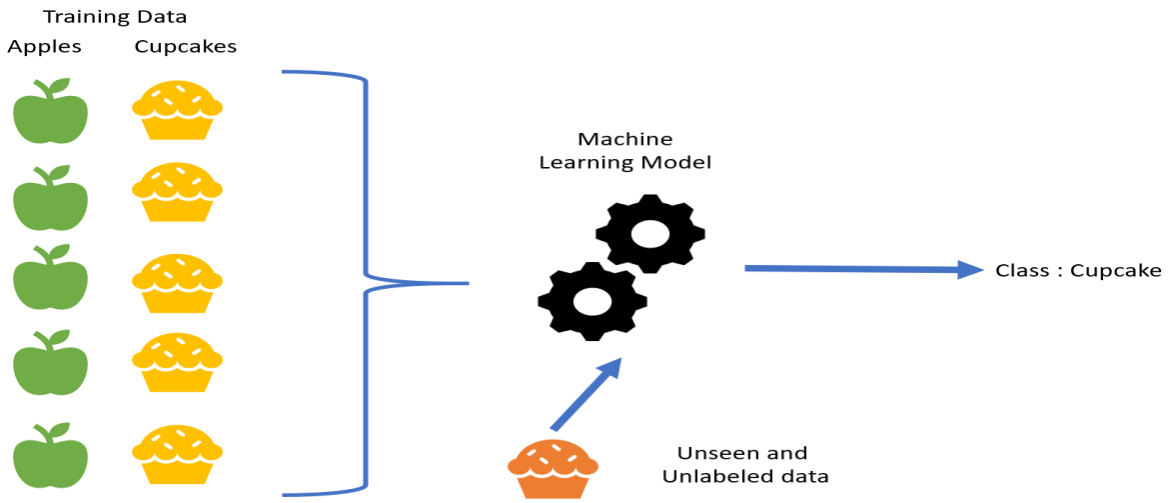
# What is Machine Learning?

- Machine learning is a growing technology which enables computers to learn automatically from past data. Machine learning uses various algorithms for **building mathematical models and making predictions using historical data or information**.
- How Does Machine Learning Work?

# Types of Machine Learning

- Machine learning algorithms are classified into three main categories:
- 1. Supervised Learning
- In supervised learning, the data is already labeled, which means you know the target variable.
- Using this method of learning, systems can predict future outcomes based on past data.
- It requires that at least an input and output variable be given to the model for it to be trained.

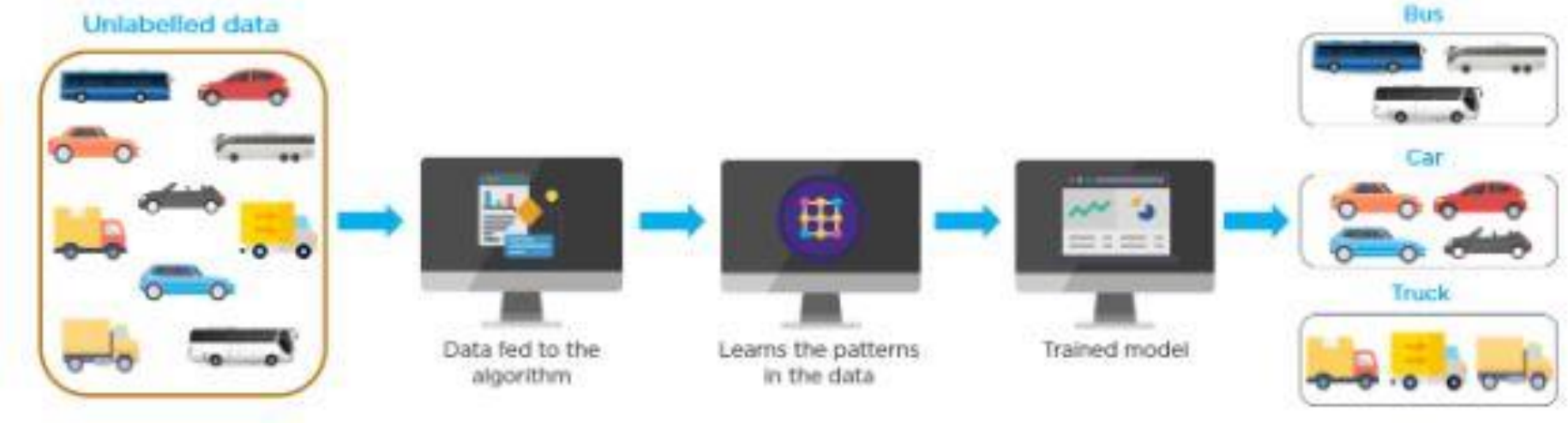
- The algorithm is trained using labeled data of dogs and cats. The trained model predicts whether the new image is that of a cat or a dog.




- Some examples of supervised learning include linear regression, logistic regression, support vector machines, Naive Bayes, and decision tree.

# Unsupervised Learning

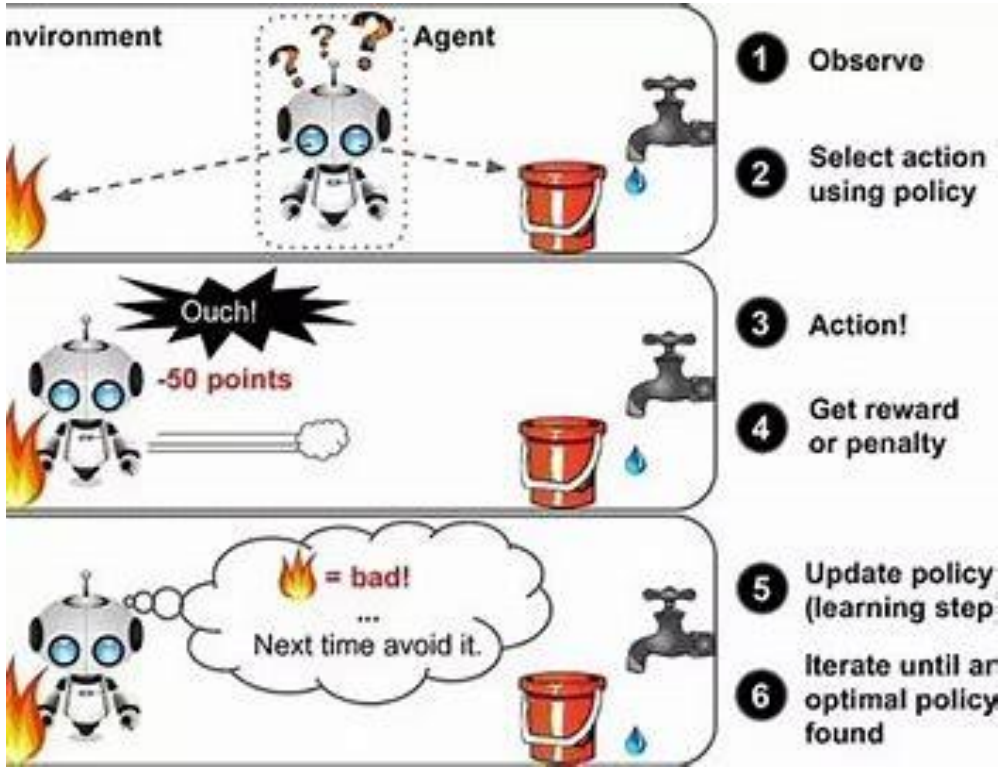
- Unsupervised learning algorithms employ unlabeled data to discover patterns from the data on their own.
- The systems are able to identify hidden features from the input data provided.




- Some examples of unsupervised learning include k-means clustering, hierarchical clustering, and anomaly detection.

# Reinforcement Learning

- The goal of reinforcement learning is to train an agent to complete a task within an uncertain environment.
- The agent receives observations and a reward from the environment and sends actions to the environment.
- The reward measures how successful action is with respect to completing the task goal.
- Examples of reinforcement learning algorithms include Q-learning and Deep Q-learning Neural Networks



# Machine Learning Applications

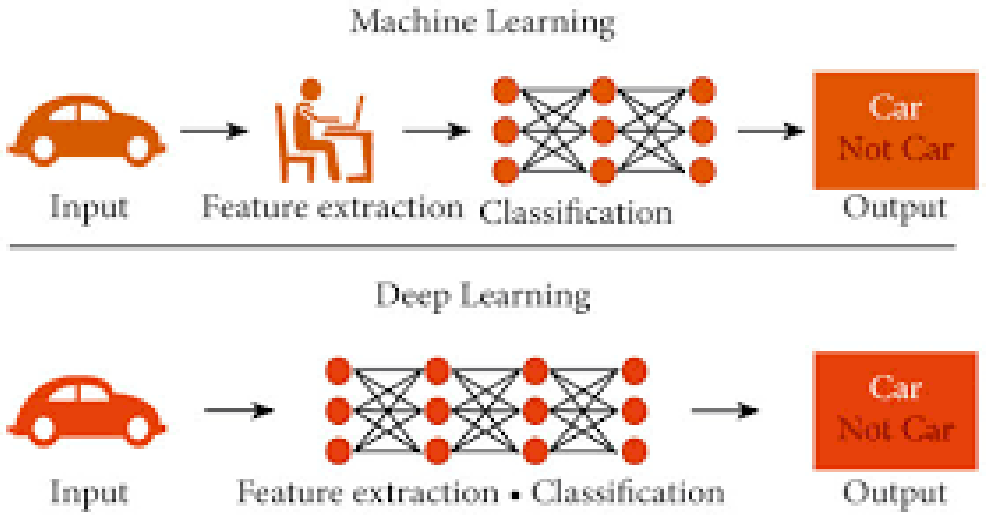
- Sales forecasting for different products
- Fraud analysis in banking
- Product recommendations
- Stock price prediction

# Module 1:Basics of Neural Networks

- Basic concept of Neurons
- Perceptron Algorithm
- Feed Forward and Back Propagation Networks

# What is Deep Learning?

- Deep learning is a subtype of machine learning. With machine learning, you manually extract the relevant features of an image.
- With deep learning, you feed the raw images directly into a deep neural network that learns the features automatically.
- Deep learning often requires hundreds of thousands or millions of images for the best results. It's also computationally intensive and requires a high-performance GPU.




| Machine Learning | Deep Learning |
|---|---|
| + Good results with small data sets | – Requires very large data sets |
| + Quick to train a model | – Computationally intensive |
| – Need to try different features and classifiers to achieve best results | + Learns features and classifiers automatically |
| – Accuracy plateaus | + Accuracy is unlimited |

# Difference between Machine Learning and Deep Learning

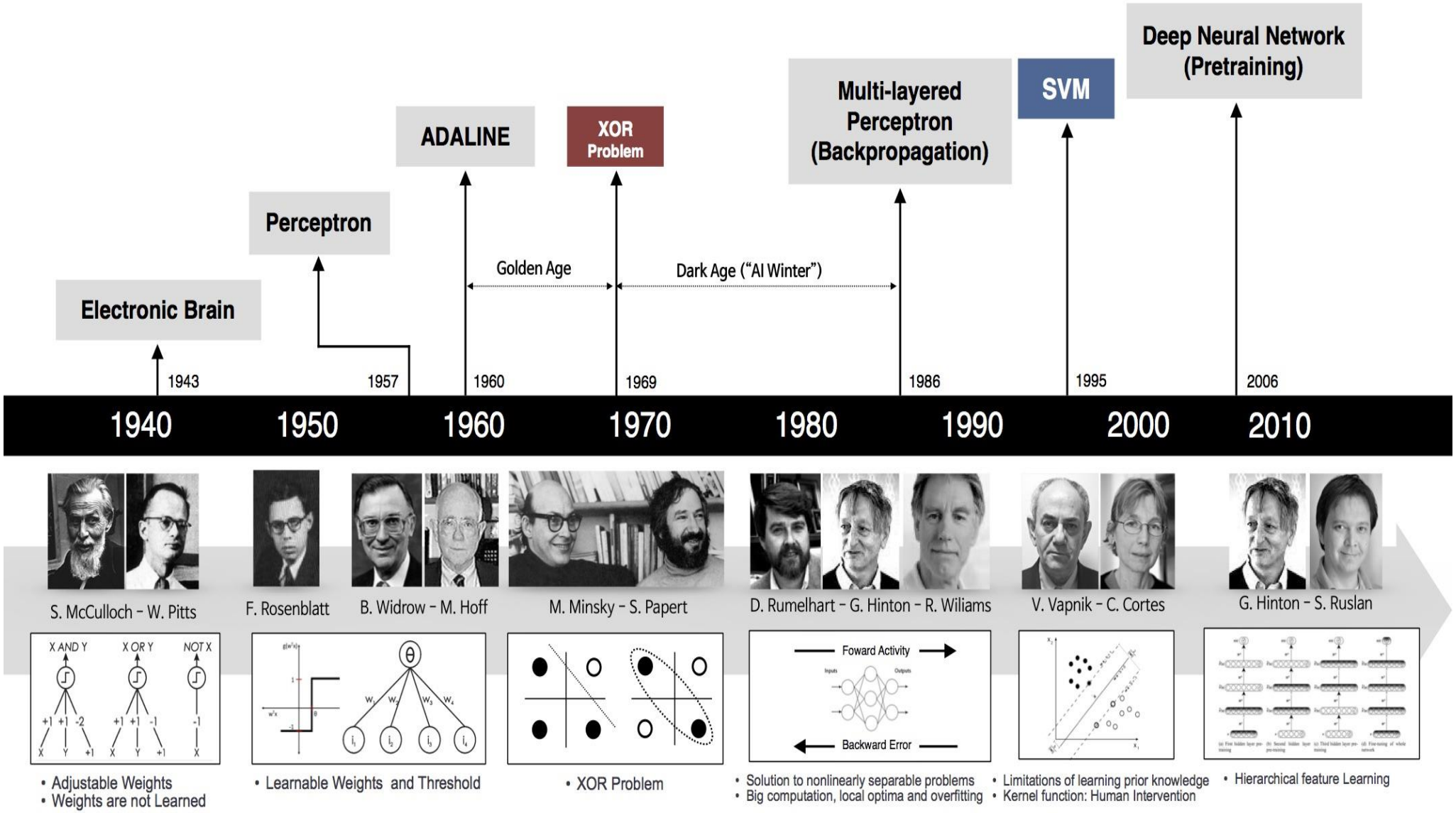
| Machine Learning | Deep Learning |
| --- | --- |
| Works on small amount of Dataset for accuracy. | Works on Large amount of Dataset. |
| Dependent on Low-end Machine. | Heavily dependent on High-end Machine. |
| Divides the tasks into sub-tasks, solves them individually and finally combine the results. | Solves problem end to end. |
| Takes less time to train. | Takes longer time to train. |
| Testing time may increase. | Less time to test the data. |

# Difference between Machine Learning and Deep Learning

| Machine Learning | Deep Learning |
| --- | --- |
| A model is created by relevant features which are manually extracted from images to detect an object in the image. | Relevant features are automatically extracted from images. It is an end-to-end learning process. |
| Machine learning is less complex as compare to deep learning. | Deep learning is generally more complex. |
| Without a high-performance GPU and lots of labelled data, machine learning techniques can be used based on applications. | To get reliable results in less time one should have a high-performance GPU and lots of labelled data. |

# History of Deep Learning

- The history of deep learning dates back to 1943 when Warren McCulloch and Walter Pitts created a ***computer model based on the neural networks*** of the human brain.
- Warren McCulloch and Walter Pitts used a combination of *mathematics and algorithms* they ***called threshold logic*** to mimic the thought process.
- The development of the basics of a continuous ***Back Propagation Model*** is credited to Henry J. Kelley in 1960.
- The concept of back propagation existed in the early 1960s but only became useful until 1985.


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

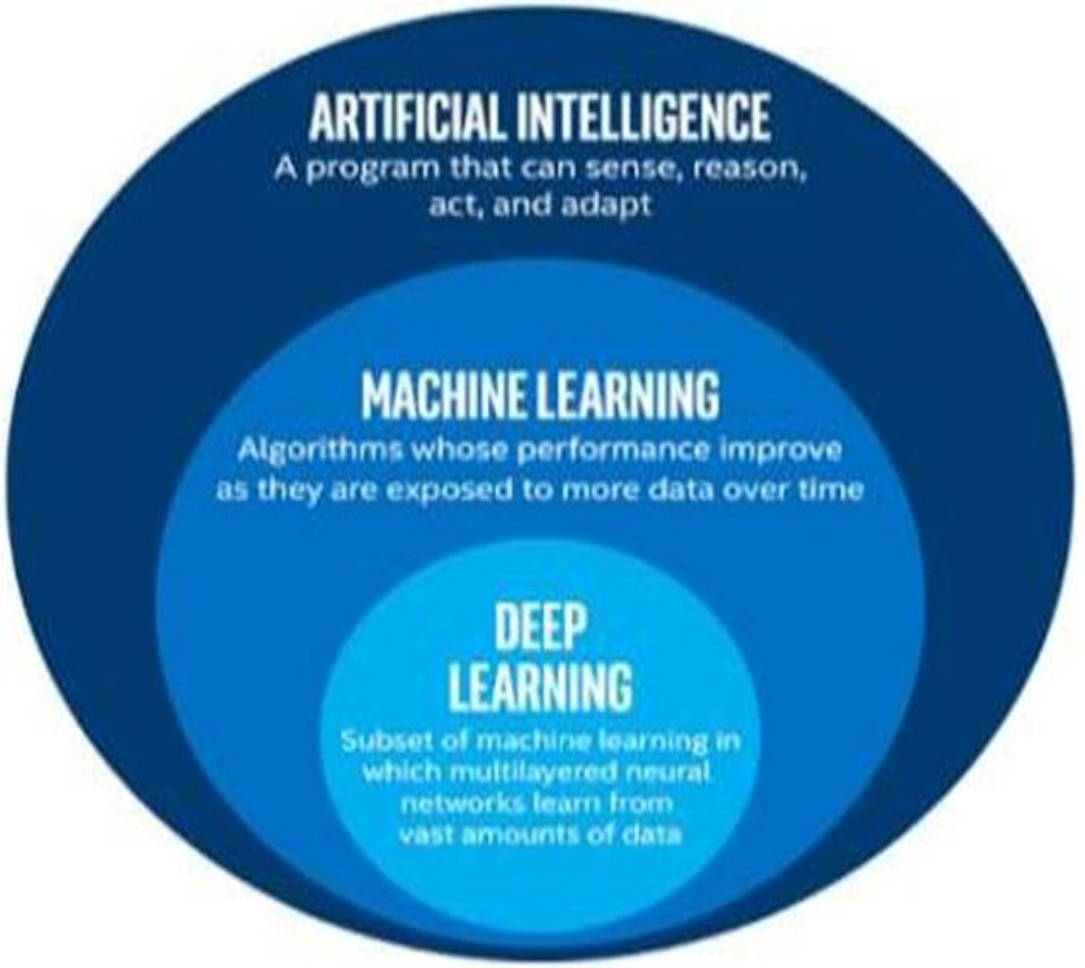
# Deep Learning

- Deep learning is a subfield of machine learning that uses artificial neural networks with multiple layers to model and solve complex problems.
- It involves training these neural networks on large amounts of data, allowing the network to learn and improve its accuracy over time.

# Deep learning

- Deep learning is a type of machine learning in which a model learns to perform classification tasks directly from images, text, or sound.
- Deep learning is usually implemented using a neural network architecture.
- The term "**deep**" refers to the **number of layers in the network**—the more layers, the deeper the network. Traditional neural networks contain only 2 or 3 layers, while deep networks can have hundreds.

- **Deep learning** is a subset of machine learning that uses an algorithm inspired by the human brain. It learns from experiences. Its main motive is to simulate human-like decision making.



# Deep learning applications

- Deep learning has a wide range of applications in various fields, including:

1. Computer Vision: Image and video recognition, object detection, face recognition, image segmentation, and autonomous vehicles.
2. Image Recognition – Recognizes and identifies peoples and objects in images as well as to understand content and context. This area is already being used in Gaming, Retail, Tourism, etc.
3. Predicting Earthquakes – Teaches a computer to perform viscoelastic computations which are used in predicting earthquakes.
4. Natural Language Processing: Deep learning models are used to understand, generate and translate human languages. Applications include machine translation, text summarization, and sentiment analysis.
5. Healthcare: Medical image analysis, disease diagnosis, personalized treatment, drug discovery, and patient monitoring.
    - Deep learning models are used in medical imaging to detect diseases, in drug discovery to identify new treatments, and in genomics to understand the underlying causes of diseases.
6. Finance: Deep learning models are used to detect fraud, predict stock prices, and analyze financial data.
7. E-commerce: Personalized recommendations, customer segmentation, and price optimization.

5. Gaming: Game AI, opponent modeling, and content generation.

6. Robotics: Deep learning models are used to control robots and drones, and to improve their ability to perceive and interact with the environment.

7. Energy: Predictive maintenance, fault detection, and demand forecasting.

8. Agriculture: Crop yield prediction, soil analysis, and pest control.

# Deep Learning

- Deep Learning models are trained using large amounts of labeled data and require significant computational resources.
- With the **increasing availability of large amounts of data and computational resources**, deep learning has been able to achieve state-of-the-art performance in a wide range of applications such as image and speech recognition, natural language processing, and more.

# Deep Learning

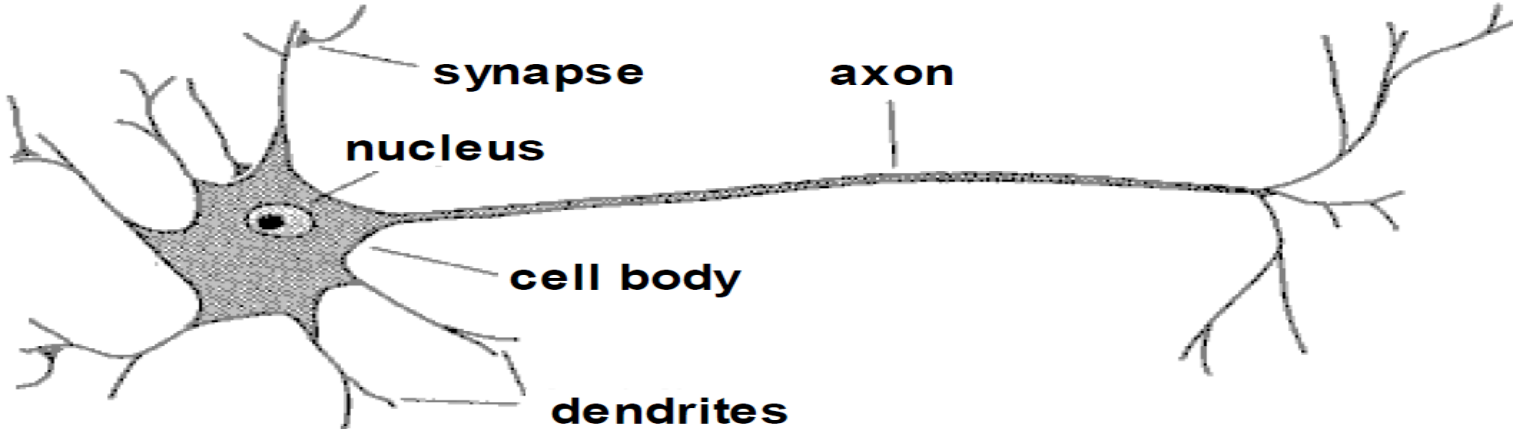
- In human brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousand of their neighbors.
- The question here is how do we recreate these neurons in a computer?
- So, we create an artificial structure called an artificial neural net where we have nodes or neurons.
- We have some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

- The building block of an ANN is called the perceptron, which is an algorithm inspired by the biological neuron .
- Although the perceptron was invented in 1957, ANNs remained in obscurity because they require extensive training, and the amount of training to get useful results exceeded the computer power and data sizes available.
- To appreciate the recent increase in computing power, consider that in 2012 the Google Brain project had to use a custom-made computer that consumed 600 kW of electricity and cost around $5,000,000.

- By 2014, Stanford AI Lab was getting more computing power by using three off-the-shelf graphics processing unit (GPU)-accelerated servers that each cost around $33,000 and consumed just 4 kW of electricity.
- Today, you can buy a specialized Neural Compute Stick(Vision Processing Unit (VPU) enables deep neural network testing, tuning and prototyping) that delivers more than 100 gigaflops of computing performance for $80.

# Basic Concepts of Neurons

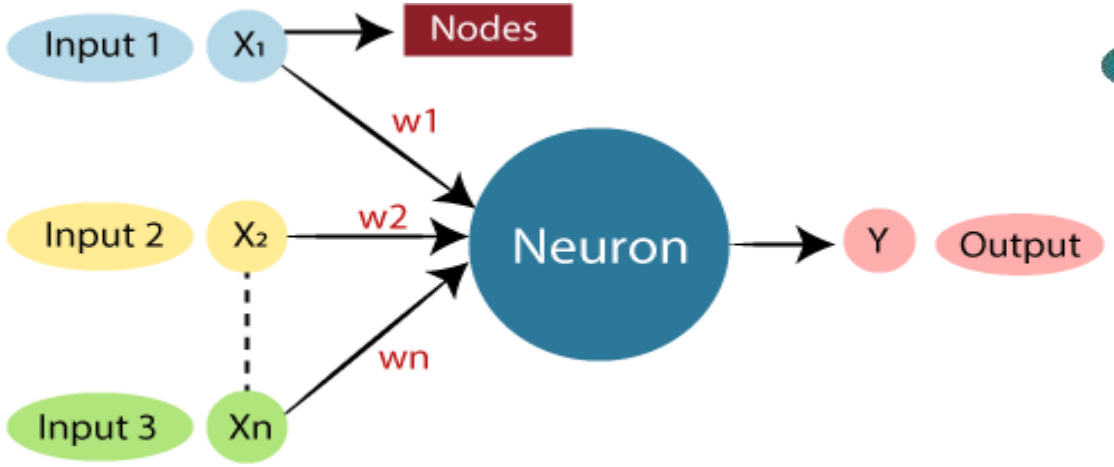
- The study of artificial neural networks (ANNs) has been inspired by the observation that biological learning systems are built of very complex webs of interconnected ***Neurons***
- Human information processing system consists of brain neuron: basic building block cell that communicates information to and from various parts of body
- Simplest model of a neuron: considered as a threshold unit –a processing element(PE)
- Collects inputs & produces output if the sum of the input exceeds an internal threshold value

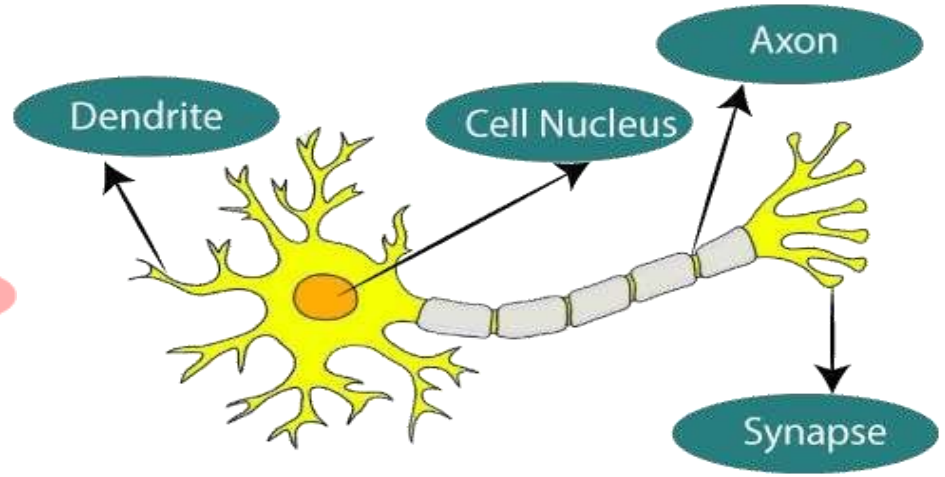


# Working of a Biological Neuron

- **Dendrites** − They are tree-like branches, responsible for receiving the information from other neurons it is connected to. In other sense, we can say that they are like the ears of neuron.
- **Soma** − It is the cell body of the neuron and is responsible for processing of information, they have received from dendrites.
- **Axon** − It is just like a cable through which neurons send the information.
- **Synapses** − It is the connection between the axon and other neuron dendrites.

# Basic Concepts of Neuron



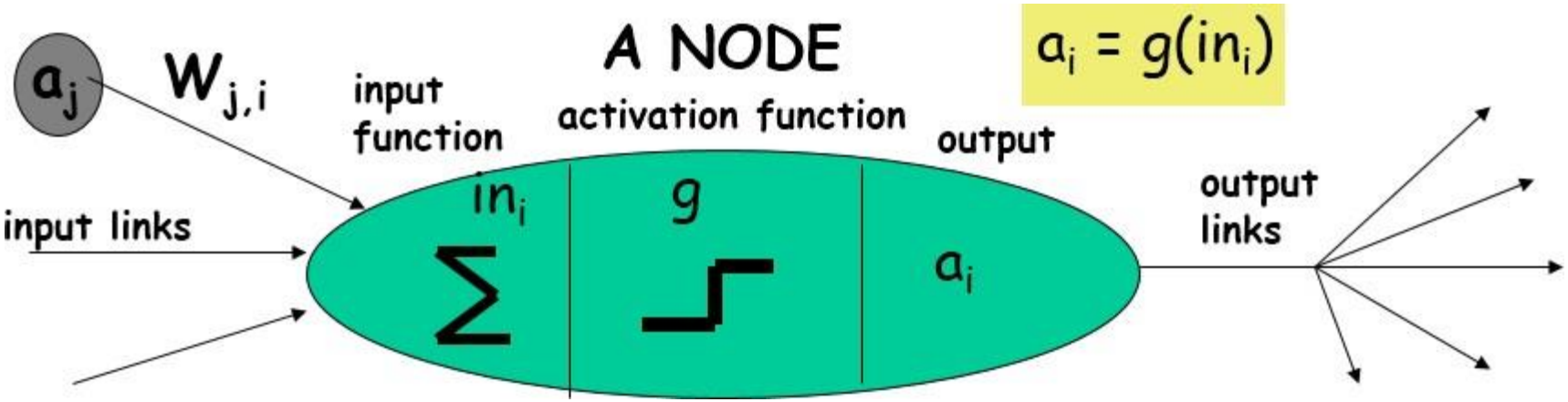




| Biological Neural Network | Artificial Neural Network |
|---|---|
| Dendrites | Inputs |
| Cell nucleus | Nodes |
| Synapse | Weights |
| Axon | Output |

## Architectures of Artificial Neural Networks

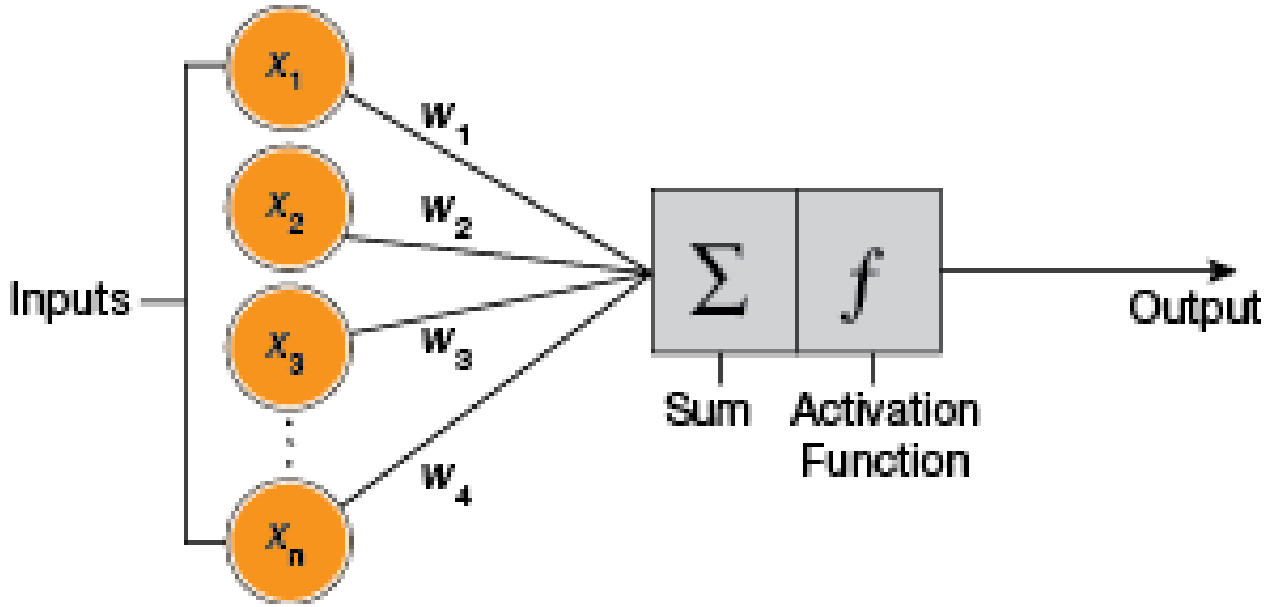
- An artificial neural network can be divided into three parts (layers), which areknown as:

- ***Input layer:*** This layer is responsible for receiving information (data), signals, features, or measurements from the external environment. These inputs are usually normalized within the limit values produced by activation functions
- ***Hidden, intermediate, or invisible layers:*** These layers are composed of neurons which are responsible for extracting patterns associated with the process or system being analyzed. These layers perform most of the internal processing from anetwork.
- ***Output layer :*** This layer is also composed of neurons, and thus is responsible for producing and presenting the final network outputs, which result from the processing performed by the neurons in the previous layers.



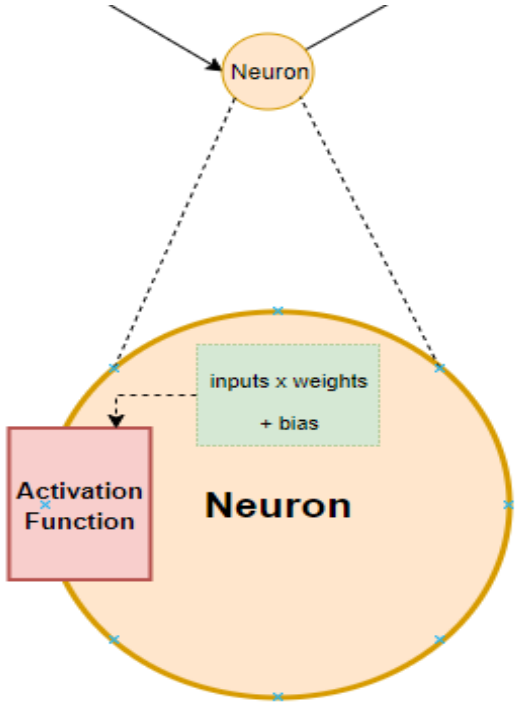
# Perceptron

- A perceptron is a mathematical model of a neuron.
- It receives weighted inputs, which are added together and passed to an activation function.
- The activation function decides whether it should produce an output.



- Perceptron receives inputs and computes an output.
- Each input has an associated weight.
- All the inputs are individually multiplied by their weights, added together, and passed into an activation function that determines whether the neuron should fire and produce an output .

- When a neural network is trained on the training set, it is initialized with a set of weights.
- These weights are then optimized during the training period and the optimum weights are produced.



- A neuron first computes the weighted sum of the inputs.

$$Y = \sum (weight * input) + bias$$

- As an instance, if the inputs are: $x_1, x_2, \ldots, x_n$

- And the weights are:

$$w_1, w_2, \ldots, w_n$$

- Then a weighted sum is computed as: $x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$

- Subsequently, a bias (constant) is added to the weighted sum

$$x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + \ bias$$

- Finally, the computed value is fed into the activation function, which then prepares an output.

$$activation\ function(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + bias)$$

- The **weights** are essentially reflecting how important an input is.
- The **bias** is used to shift the result of activation function towards the positive or negative side.

# SCENARIO 1

- Assume you are predicting the price of a car in dollars.
- Your understanding is that the price of the car is dependent on the year it was made and the number of miles it has driven.
- Let’s assume that your hypothesis is that the higher the year of the car, the pricey the car. And subsequently, the more the car is driven, the cheaper the car.
- There is a positive relationship between the price of the car and the year it was made and a negative relationship between the price of the car and the miles it has been driven.

- As a result, we expect to see **positive weight** for the feature that represents **year** and **negative weight** for the feature that represents **miles**.

$$price\ of\ car = (w_1 x\ Year + w_2 x\ Miles)$$

- *w1 is going to be positive and w2 is expected to be negative*

# Scenario 2

- Let’s assume you want your neural network to return 2 when the input is 0.
- As the sum of product of weight and input is going to be 0, how will you ensure the neuron of the network returns 2?
- *You can add a bias of 2.*
- If we do not include the bias then the neural network is simply performing a matrix multiplication on the inputs and weights. This can easily end up over-fitting the data set.

- There are many different types of activation functions with different properties, but one of the simplest is the step function.
- A step function outputs a 1 if the input is higher than a certain threshold, otherwise it outputs a 0.
- For example, if a perceptron has two inputs ($x_1$ and $x_2$):
- $x_1 = 0.9$
- $x_2 = 0.7$
- which have weightings ($w_1$ and $w_2$) of:
- $w_1 = 0.2$
- $w_2 = 0.9$
- and the activation function threshold is equal to 0.75, then weighing the inputs and adding them together yields:

- and the activation function threshold is equal to 0.75, then weighing the inputs and adding them together yields:
- $x_1w_1 + x_2w_2 = (0.9\times0.7) + (0.2\times0.9) = 0.81$
- Because the total input is higher than the threshold (0.75), the neuron will fire. Since we chose a simple step function, the output would be 1.

# Perceptron

- A perceptron is a simple binary classification algorithm, proposed by Cornell scientist Frank Rosenblatt.
- It helps to divide a set of input signals into two parts—“yes” and “no”.
-

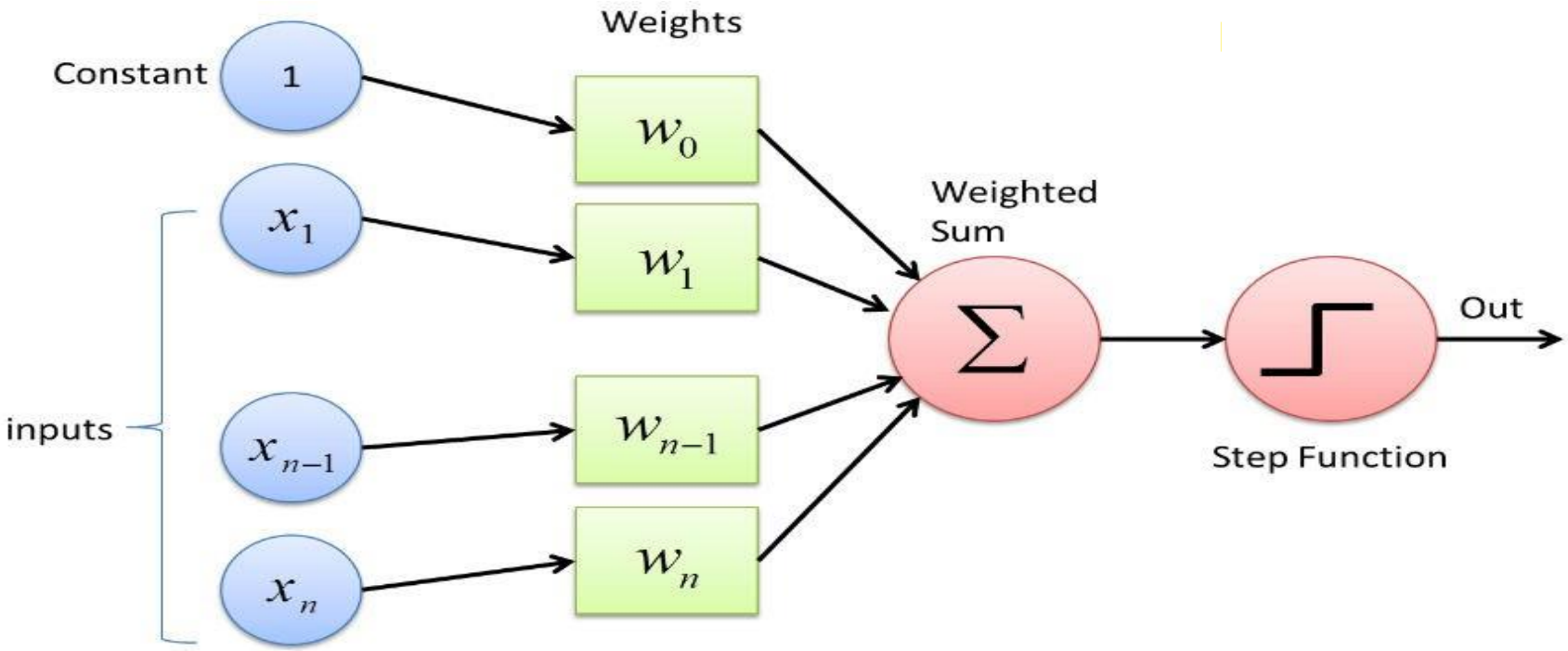



Fig : Perceptron

- Perceptron is a single layer neural network
- A perceptron takes a vector of real-valued inputs, calculates a linear combinationof these inputs, then outputs a 1 if the result is greater than some threshold and -1otherwise
- Given inputs ***x1*** through ***xn,*** the output **O*(x1,*** . . . , ***xn)*** computed by the perceptronis

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

- where each ***wi*** is a real-valued constant, or weight, that determines the contribution of input ***xi*** to the perceptron output
- ***-w0*** is a threshold that the weighted combination of inputs ***w1x1*** + . . . + ***wnxn*** must surpass in order for the perceptron to output a 1.

# Steps in Perceptron

- The perceptron model begins with multiplying all input values and their weights, then adds these values to create the weighted sum.

$\sum wi*xi = x1*w1 + x2*w2 + x3*w3+……..x4*w4$

- Further, this weighted sum is applied to the activation function ‘f’ to obtain the desired output. This activation function is also known as the step function.

$Y=f(\sum wi*xi + b)$

bias ,b is used to improve the model performance

# Activation Function

- An activation function is a non-linear function applied to the output of a neural network's layer.
- It is used to introduce non-linearity into the model, which allows it to learn complex patterns and relationships in the data.
- There are several types of activation functions used in artificial neural networks (ANNs), including:

  - Sigmoid function: This function maps any input value to a value between 0 and 1. It is commonly used in the output layer of a binary classification problem.

  - ReLU (Rectified Linear Unit): This function returns the input if it is positive and zero otherwise. It is one of the most widely used activation functions in deep learning because of its simplicity and effectiveness.

- Tanh (Hyperbolic Tangent): This function maps any input value to a value between -1 and 1. It is commonly used in the hidden layers of a neural network.

- Softmax: This function is used in the output layer of a multi-class classification problem. It normalizes the outputs so that they sum up to one, which can be interpreted as the probabilities of the input belonging to each class.

- Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it.
- There are several activation functions you may encounter in practice:
- Sigmoid:takes real-valued input and squashes it to range between 0 and 1.

$$\sigma(x) = \frac{1}{(1+exp(-x))}$$

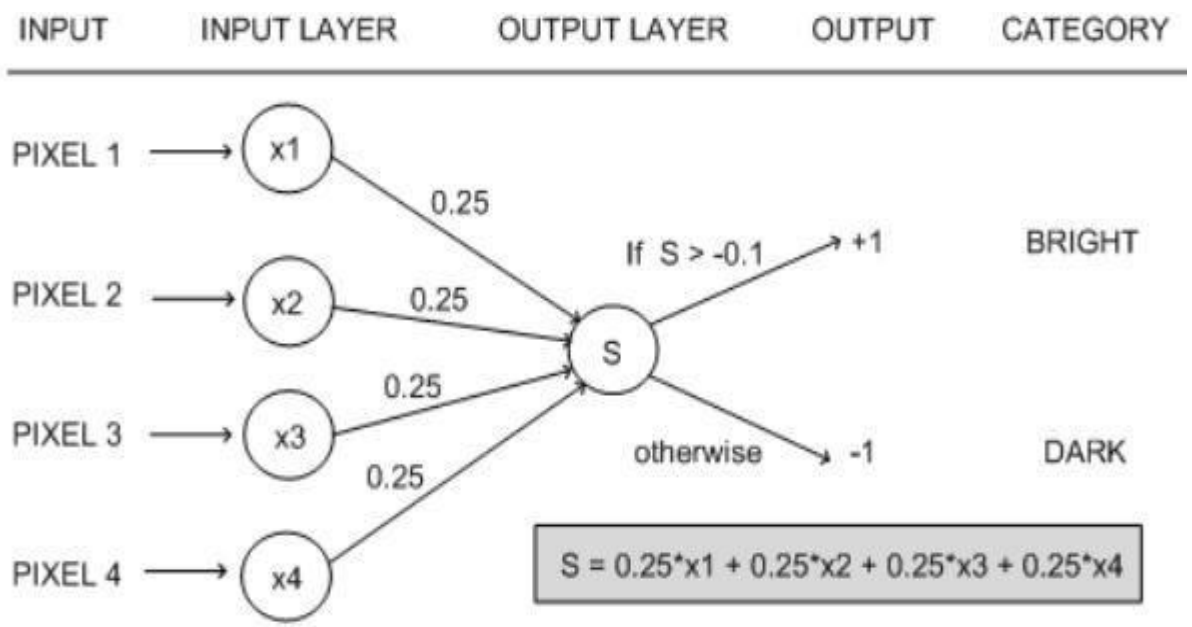
- 
- tanh:takes real-valued input and squashes it to the range [-1, 1 ].

$$tanh(x) = 2\sigma(2x) - 1$$

- ReLu:ReLu stands for Rectified Linear Units. It takes real-valued input and thresholds it to 0 (replaces negative values to 0 ).

$$f(x) = max(0, x)$$

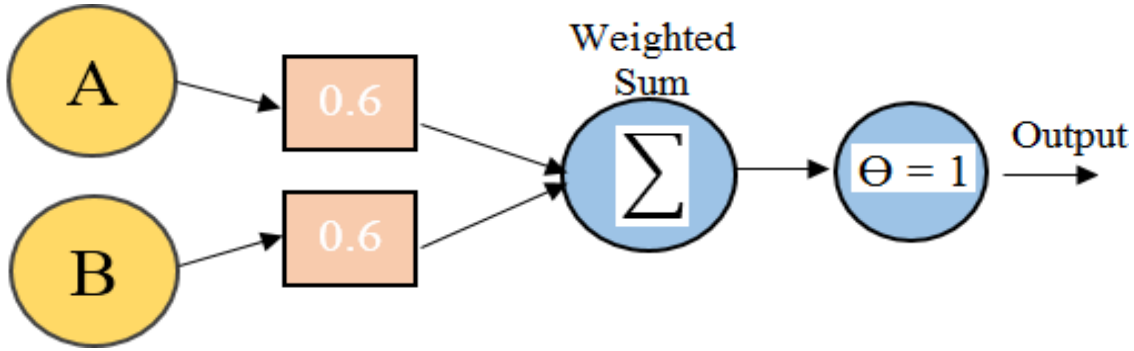
# Example for Perceptron



# Example For Perceptron

- A single perceptron can be used to represent many Boolean functions
- **AND function**

| A | B | A ^ B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




- If A=0 & B=0 → 0*0.6 + 0*0.6 = 0.

  This is not greater than the threshold of 1, so the output = 0.
- If A=0 & B=1 → 0*0.6 + 1*0.6 = 0.6.

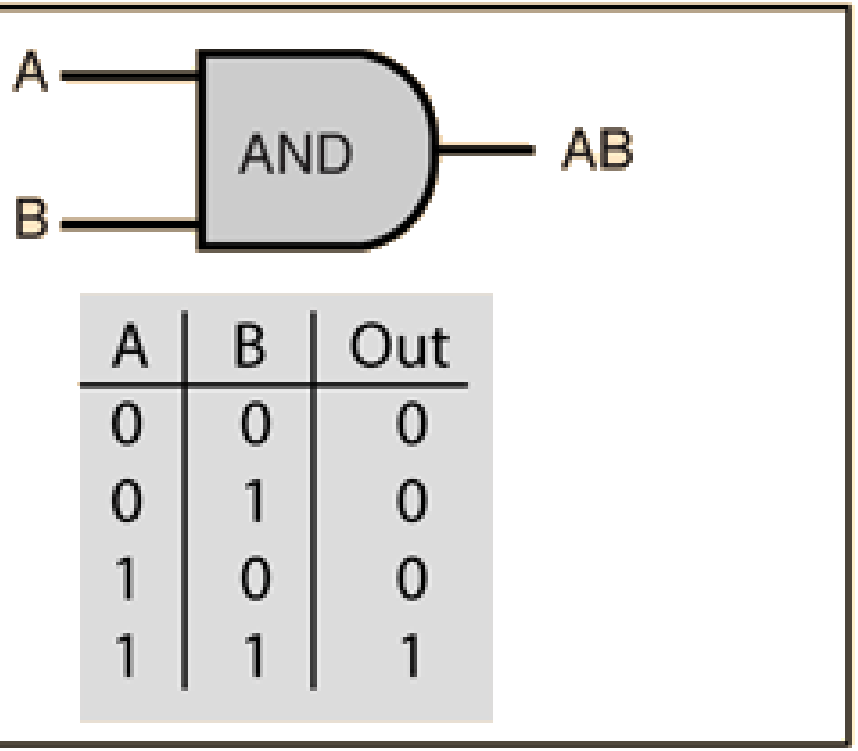
  This is not greater than the threshold, so the output = 0.
- If A=1 & B=0 → 1*0.6 + 0*0.6 = 0.6.

  This is not greater than the threshold, so the output = 0.
- If A=1 & B=1 → 1*0.6 + 1*0.6 = 1.2.

  This exceeds the threshold, so the output = 1.

- The Perceptron algorithm states that:
- Prediction (y`) = 1 if $Wx+b > 0$ and 0 if $Wx+b \leq 0$
- The steps in this method are very similar to how Neural Networks learn, which is as follows;
  - Initialize weight values and bias
  - Forward Propagate
  - Check the error
  - Backpropagate and Adjust weights and bias
  - Repeat for all training examples

# AND Gate




| A | B | Out |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

What are the weights and bias for the AND perceptron?

We need to understand that the output of an AND gate is 1 only if both inputs (in this case, x1 and x2) are 1

**Row 1**

•From w1*x1+w2*x2+b, initializing w1, w2, as 1 and b as –1, we get;

*x1(1)+x2(1)–1*

•Passing the first row of the AND logic table (x1=0, x2=0), we get;

*0+0–1 = –1*

•From the Perceptron rule, if Wx+b≤0, then y`=0. Therefore, this row is correct, and no need for Backpropagation.

**Row 2**

•Passing (x1=0 and x2=1), we get;

*0+1–1 = 0*

•From the Perceptron rule, if Wx+b≤0, then y`=0. This row is correct, as the output is 0 for the AND gate.

•From the Perceptron rule, this works (for both row 1, row 2 and 3).

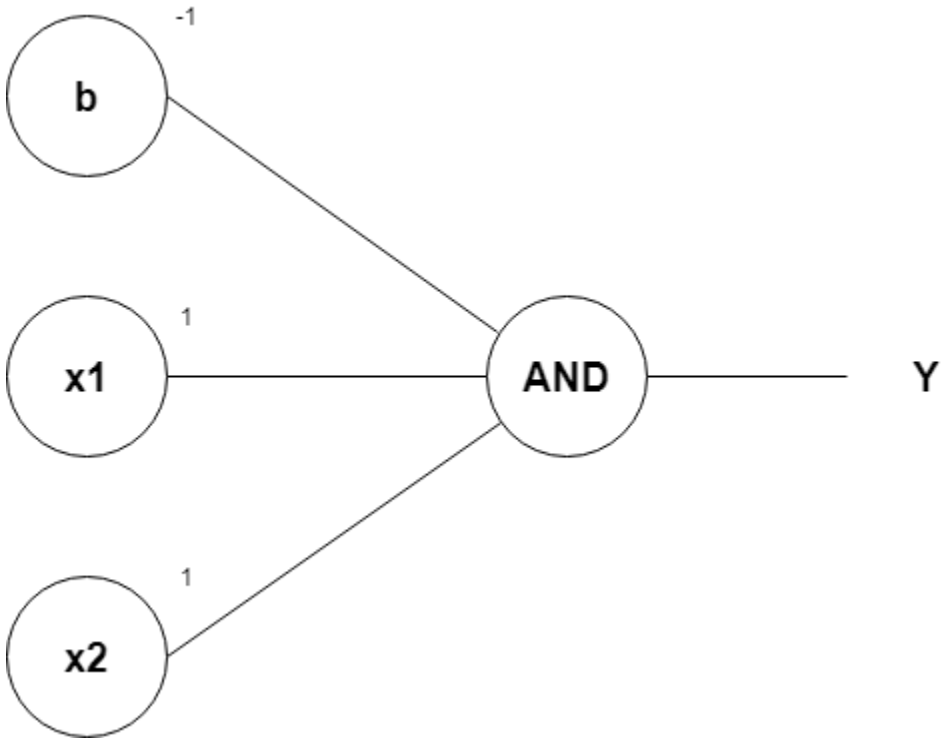
**Row 4**

•Passing (x1=1 and x2=1), we get;

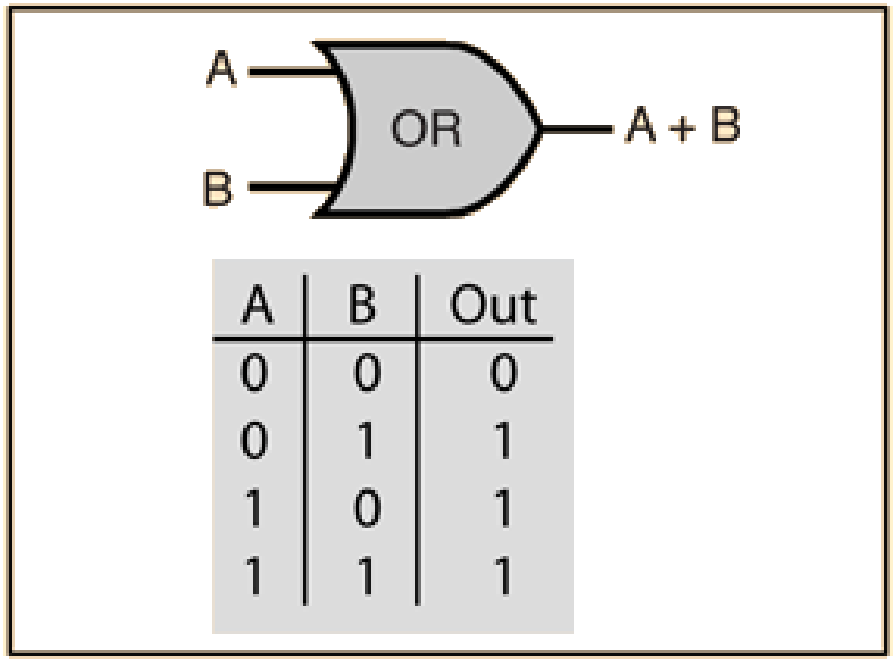
*1+1–1 = 1*

•Again, from the perceptron rule, this is still valid.

- Therefore, we can conclude that the model to achieve an AND gate, using the Perceptron algorithm is;
- x1+x2–1



# OR Gate




| A | B | Out |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

From the diagram, the OR gate is 0 only if both inputs are 0.

**Row 1**

•From w1x1+w2x2+b, initializing w1, w2, as 1 and b as –1, we get;

*x1(1)+x2(1)–1*

•Passing the first row of the OR logic table (x1=0, x2=0), we get;

*0+0–1 = –1*

•From the Perceptron rule, if Wx+b≤0, then y`=0. Therefore, this row is correct.

**Row 2**

•Passing (x1=0 and x2=1), we get;

*0+1–1 = 0*

•From the Perceptron rule, if Wx+b <= 0, then y`=0. Therefore, this row is incorrect.

•So we want values that will make inputs x1=0 and x2=1 give y` a value of 1. If we change w2 to 2, we have;

*0+2–1 = 1*

•From the Perceptron rule, this is correct for both the row 1 and 2.

**Row 3**

- Passing (x1=1 and x2=0), we get;

*1+0–1 = 0*

- From the Perceptron rule, if Wx+b <= 0, then y`=0. Therefore, this row is incorrect.
- Since it is similar to that of row 2, we can just change w1 to 2, we have;

*2+0–1 = 1*

- From the Perceptron rule, this is correct for both the row 1, 2 and 3.

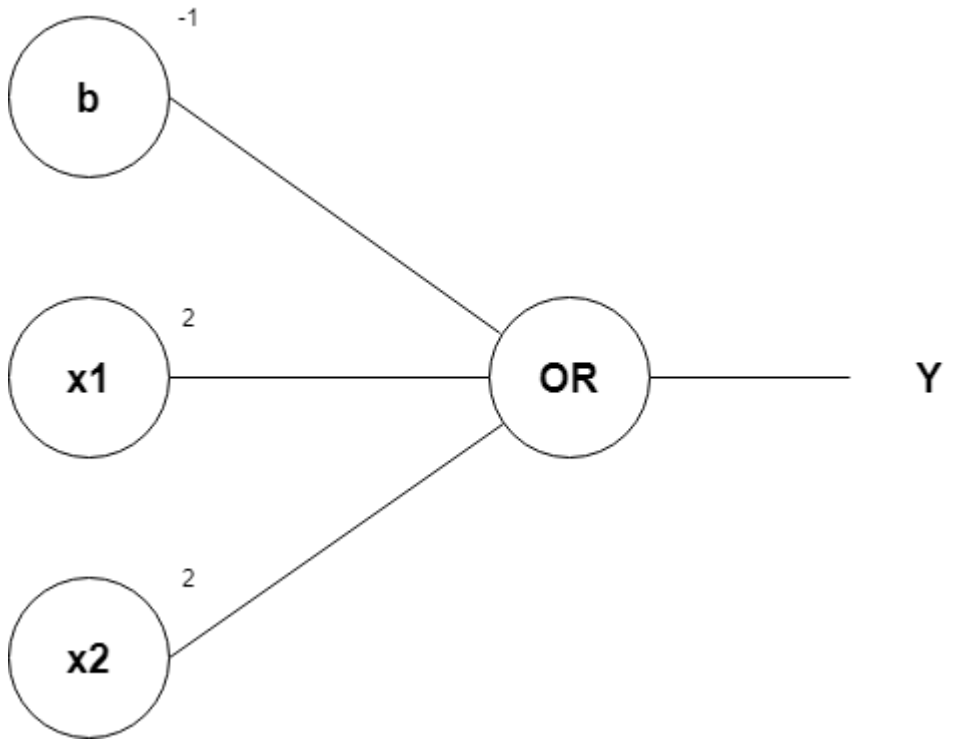
**Row 4**

- Passing (x1=1 and x2=1), we get;

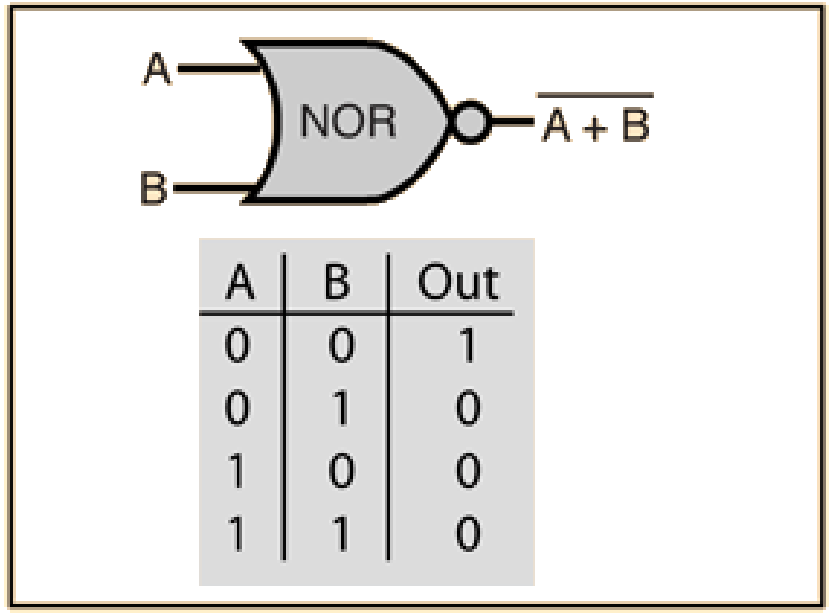
*2+2–1 = 3*

•Again, from the perceptron rule, this is still valid. Therefore, we can conclude that the model to achieve an OR gate, using the Perceptron algorithm is;

*2x1+2x2–1*



# NOR Gate




| A | B | Out |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

From the diagram, the NOR gate is 1 only if both inputs are 0.

**Row 1**

•From w1x1+w2x2+b, initializing w1 and w2 as 1, and b as –1, we get;

*x1(1)+x2(1)–1*

•Passing the first row of the NOR logic table (x1=0, x2=0), we get;

*0+0–1 = –1*

•From the Perceptron rule, if Wx+b≤0, then y`=0. This row is incorrect, as the output is 1 for the NOR gate.

•So we want values that will make input x1=0 and x2 = 0 to give y` a value of 1. If we change b to 1, we have;

*0+0+1 = 1*

•From the Perceptron rule, this works.

**Row 2**

•Passing (x1=0, x2=1), we get;

*0+1+1 = 2*

•From the Perceptron rule, if Wx+b > 0, then y`=1. This row is incorrect, as the output is 0 for the NOR gate.

•So we want values that will make input x1=0 and x2 = 1 to give y` a value of 0. If we change w2 to –1, we have;

*0–1+1 = 0*

•From the Perceptron rule, this is valid for both row 1 and row 2.

**Row 3**

- Passing (x1=1, x2=0), we get;

*1+0+1 = 2*

- From the Perceptron rule, if Wx+b > 0, then y`=1. This row is incorrect, as the output is 0 for the NOR gate.
- So we want values that will make input x1=0 and x2 = 1 to give y` a value of 0. If we change w1 to –1, we have;

*–1+0+1 = 0*

- From the Perceptron rule, this is valid for both row 1, 2 and 3.

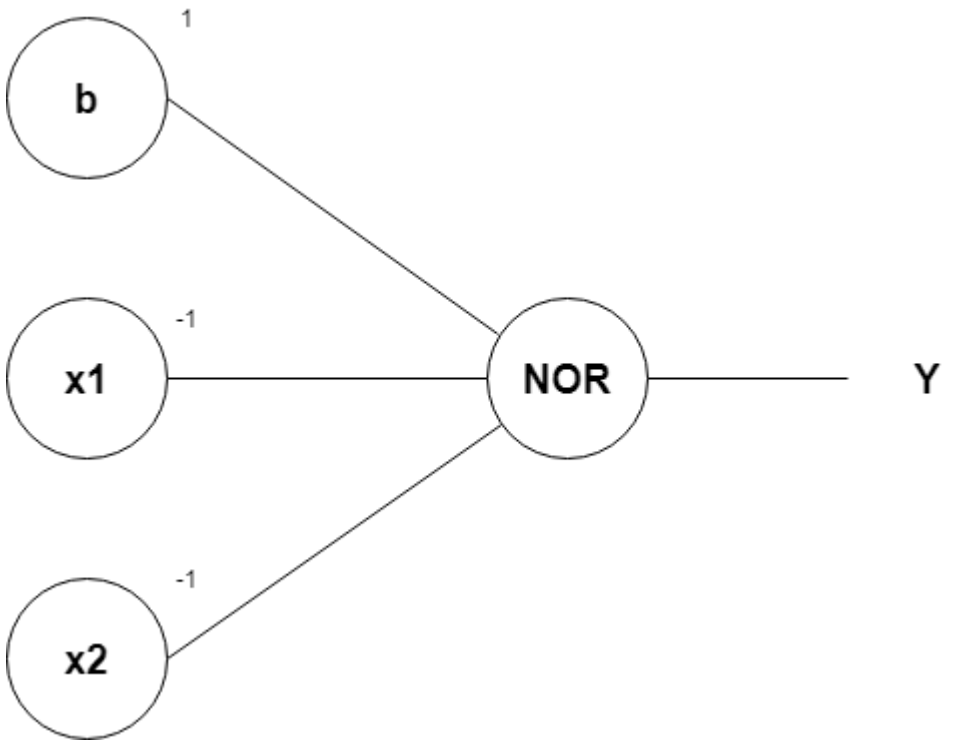
**Row 4**

- Passing (x1=1, x2=1), we get;

*-1-1+1 = -1*

- Therefore, we can conclude that the model to achieve a NOR gate, using the Perceptron algorithm is;

- -x1-x2+1



## The Perceptron Training Rule

The learning problem is to determine a weight vector that causes the perceptron toproduce the correct + 1 or - 1 output for each of the given training examples.

To learn an acceptable weight vector

- Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times asneeded until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the perceptron training rule,which revises the weight *wi* associated with 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*** is the target output for the current training example
***o*** is the output generated by the perceptron
η is a positive constant called the ***learning rate***

# Perceptron Learning Algorithm

- Our goal is to find the **w** vector that can perfectly classify positive inputs and negative inputs in our data.

- **Case 1**: When x belongs to P and its dot product w.x < 0
  **Case 2**: When x belongs to N and its dot product w.x ≥ 0

**Algorithm:** Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize **w** randomly;
**while** !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$ ;
  **if** $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$ ;
  **end**
  **if** $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$ ;
  **end**
**end**
//the algorithm converges when all the inputs are classified correctly

# Types of Perceptron

- **Single Layer Perceptron model:** The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes. A Single-layer perceptron can learn only linearly separable patterns.
- **Multi-Layered Perceptron model**: It is mainly similar to a single-layer perceptron model but has more hidden layers.

# Singe-layer Perceptron

- The simplest type of feedforward neural network is the perceptron, a feedforward neural network with no hidden units. Thus, a perceptron has only an input layer and an output layer.
- The output units are computed directly from the sum of the product of their weights with the corresponding input units, plus some bias.
- Historically, the perceptron's output has been binary, meaning it outputs a value of 0 or 1. This is achieved by passing the product sum into the step function $H(x)$. This is defined as
-

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0. \end{cases}$$

# Singe-layer Perceptron

- For a binary perceptron with *n*-dimensional input $\vec{x}$, *n*-dimensional weight vector $\vec{w}$, and bias *b*, the 1-dimensional output *o* is

$$o = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} + b \geq 0 \\ 0 & \text{if } \vec{w} \cdot \vec{x} + b < 0. \end{cases}$$

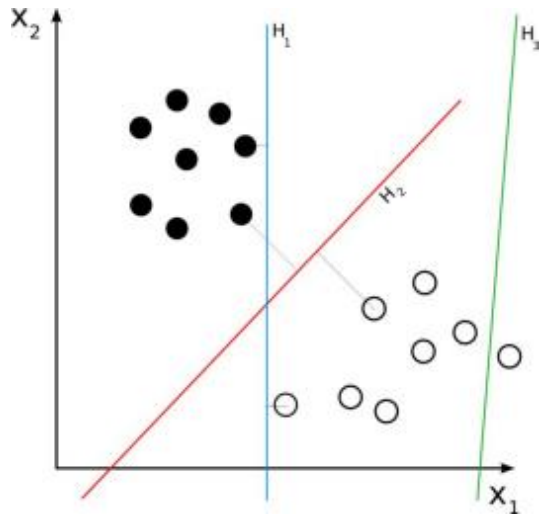
- Since the perceptron divides the input space into two classes, 0 and 1, depending on the values of $\vec{w}$ and ,*b*, it is known as a linear classifier.
- The line separating the two classes is known as the **classification boundary** or decision boundary.

- In the case of a two-dimensional input (as in the diagram) it is a line, while in higher dimensions this boundary is a hyperplane. The weight vector $\vec{w}$ defines the slope of the classification boundary while the bias *b* defines the intercept.
- More general single-layer peceptrons can use activation functions other than the step function *H*(*x*).
- Typical choices are the identity function *f*(*x*)=*x*, the sigmoid function.

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

- and the hyperbolic tangent tanh(*x*): $\tanh(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}}$

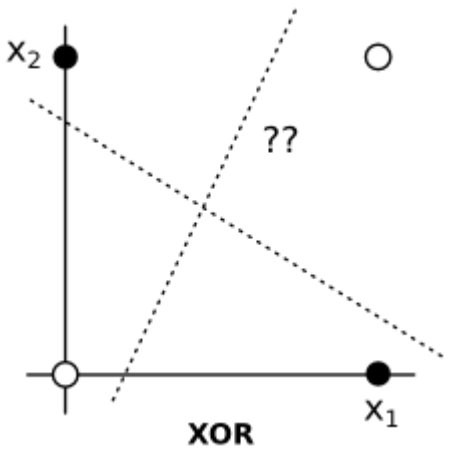
- Use of any of these functions ensures the output is a continuous number (as opposed to binary), and thus ***not every activation function yields a linear classifier.***
- In order for a perceptron to learn to correctly classify a set of input-output pairs ($\vec{x}$,$y$), it has to adjust the weights $\vec{w}$ and bias $b$ in order to learn a good classification boundary.
- The figure below shows many possible classification boundaries, the best of which is the boundary labeled $H2$.
- If the perceptron uses an activation function other than the step function (e.g. the sigmoid function), then the weights and bias should be adjusted so that the output $o$ is close to the true label $y$.


X2
H1
H3
H2
X1

- where $\vec{w}i$ and *bi* are the values of $\vec{w}$ and *b* after the ith iteration of gradient descent,and $\frac{\partial f}{\partial x}$ s is the partial derivative of *f* with respect to *x*. $\alpha$ is known as the **learning rate**, which controls the step size gradient descent takes each iteration, typically chosen to be a small value, e.g. $\alpha$=0.01.
- Values of $\alpha$ that are too large cause learning to be suboptimal (by failing to converge), while values of $\alpha$ that are too small make learning slow (by taking too long to converge).

# Limitation

- It was mentioned earlier that single-layer perceptrons are linear classifiers.
- That is, they can only learn **linearly separable** patterns. Linearly separable patterns are datasets or functions that can be separated by a linear boundary (a line or hyperplane).
- The XOR, or "exclusive or", function is a simple function on two binary inputs . A plot of and truth table for XOR is below.



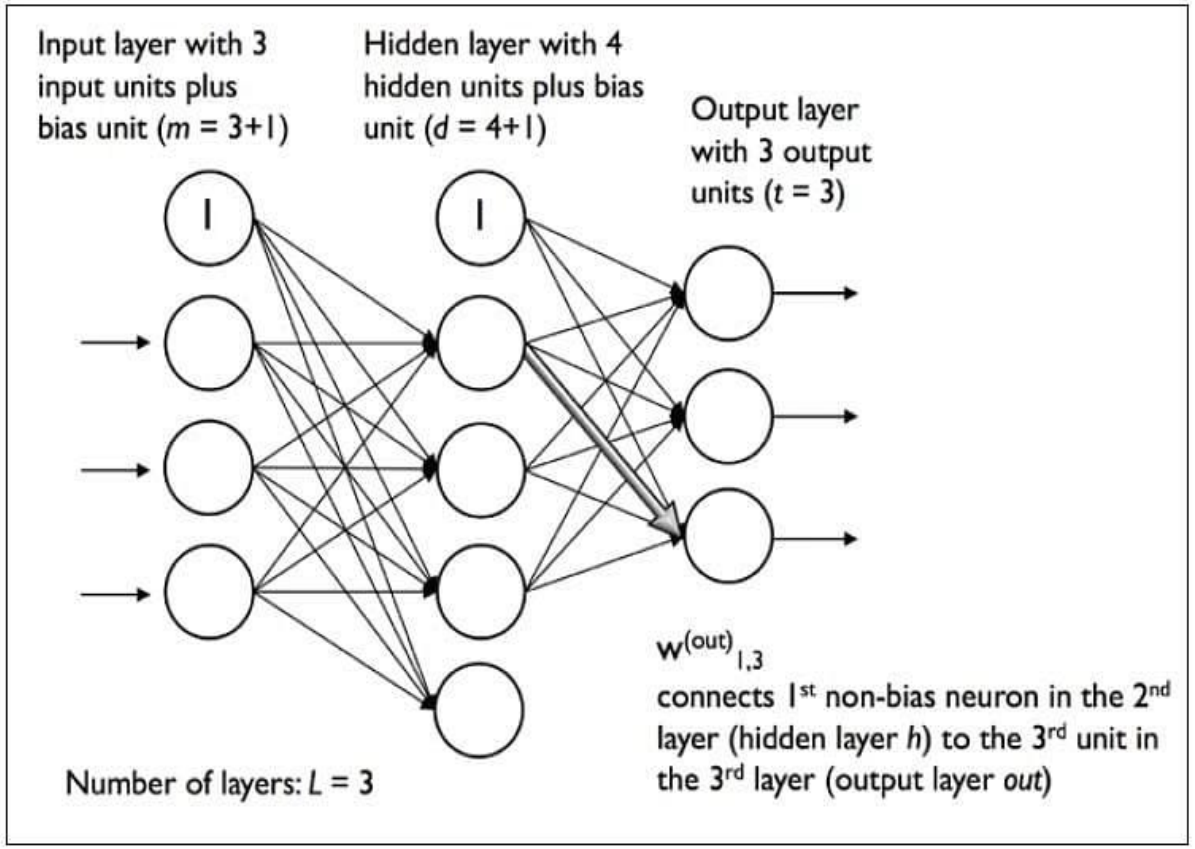

| $X_1$ | $X_2$ | $XOR$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- For the plot of the XOR function, it is impossible to find a linear boundary that separates the black and white inputs from one another.
- This is because XOR is not a linearly separable function, and by extension, perceptrons cannot learn the XOR function.
- Similar analogs exist in higher dimensions, i.e. more than two inputs.
- The obvious extension is to add more layers of units so that there are nonlinear computations in between the input and output.

# Multi-layer Perceptron

- The **multi-layer perceptron** (MLP) is an artificial neural network composed of many perceptron.
- Unlike single-layer perceptrons, MLPs are capable of learning to compute non-linearly separable functions.
- Because they can learn nonlinear functions, they are one of the primary machine learning techniques for both regression and classification in supervised learning.
- Multilayer neural networks contain multiple computational layers; the additional intermediate layers between input and output are referred to as hidden layers

- The specific architecture of multilayer neural networks is referred to as **feed-forward networks**




- Feedforward neural networks have the property that information (i.e. computation) flows forward through the network, i.e. there are no loops in the computation graph

- The number of layers is known as the depth, and the number of units in a layer is known as the width

- Feed-forward networks assumes that all nodes in one layer are connected to those of the next layer.

# Multi Layer Perceptron

- The following defines a prototypical m-layer (here m=2 hidden layers) MLP that computes a one-dimensional output o on an n-dimensional input $\vec{x}=\{x1,\ldots,xn\}$.

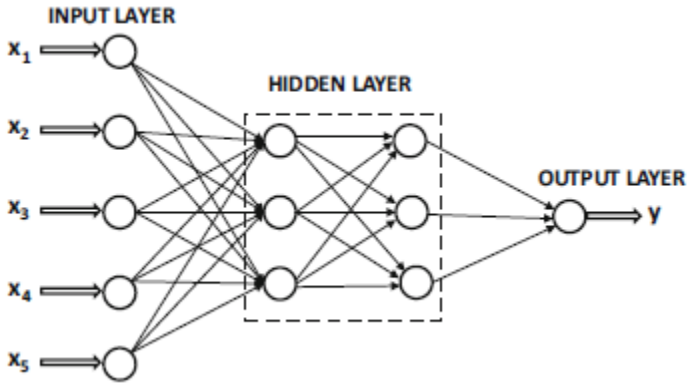
1. The output perceptron has an activation function g(o) and hidden layer perceptrons have activation functions g(h).
2. Every perceptron in layer li is connected to every perceptron in layer li−1; layers are "fully connected." Thus, every perceptron depends on the outputs of all the perceptrons in the previous layer (this is without loss of generality since the weight connecting two perceptrons can still be zero, which is the same as no connection being present).
3. There are no connections between perceptrons in the same layer.

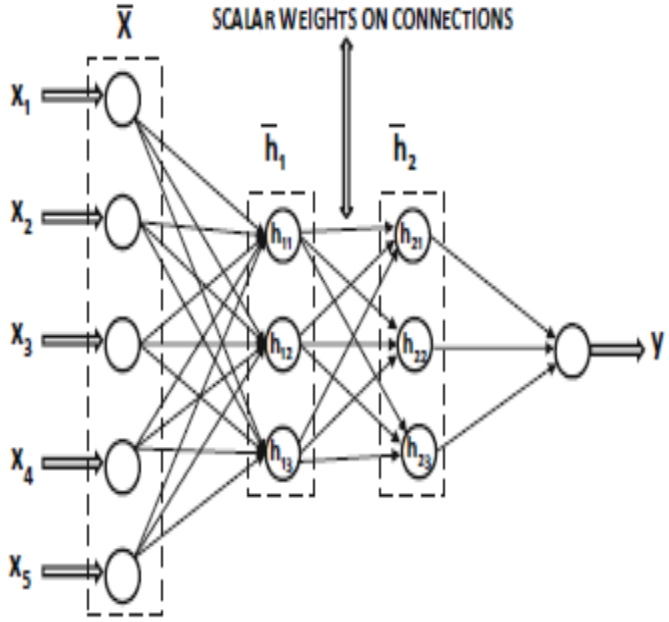
# Feedforward Neural Networks

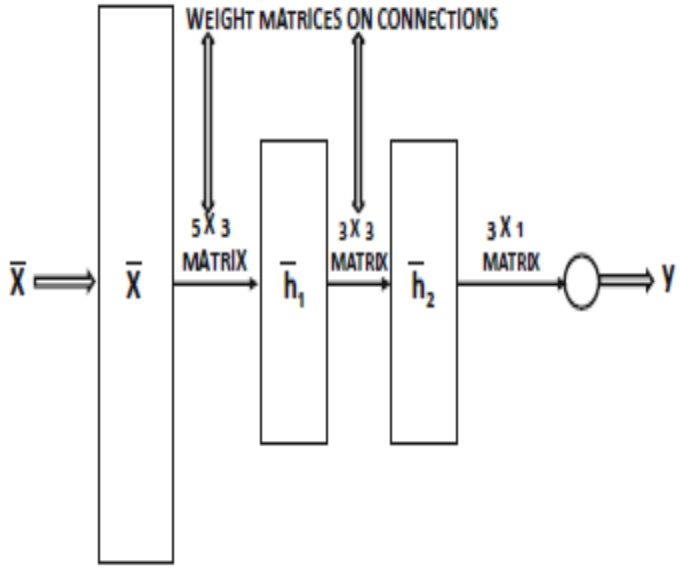
- **Feedforward neural networks** are artificial neural networks where the connections between units do not form a cycle.
- They are called feedforward because information only travels forward in the network (no loops), first through the input nodes, then through the hidden nodes (if present), and finally through the output nodes.
- Two types of FFN
    1. Single-layer feedforward network
    2. Multi-layer feedforward networks

# Feed Forward Network

- The basic architecture of a feed-forward network with two hidden layers and a single output layer.



- **Error Function or Loss Function**
- The learning process requires the definition of an error function $E$ that quantifies the difference between the computed output of the perceptron $o$ and the true value $y$ for an input $\vec{x}$ over a set of multiple input-output pairs ( $\vec{x}$, $y$).
- Historically, this error function is the mean squared error (MSE), defined for a set of N input-output.

; $X = \{(\vec{x_1}, y_1), \ldots, (\vec{x_N}, y_N)\}$ as

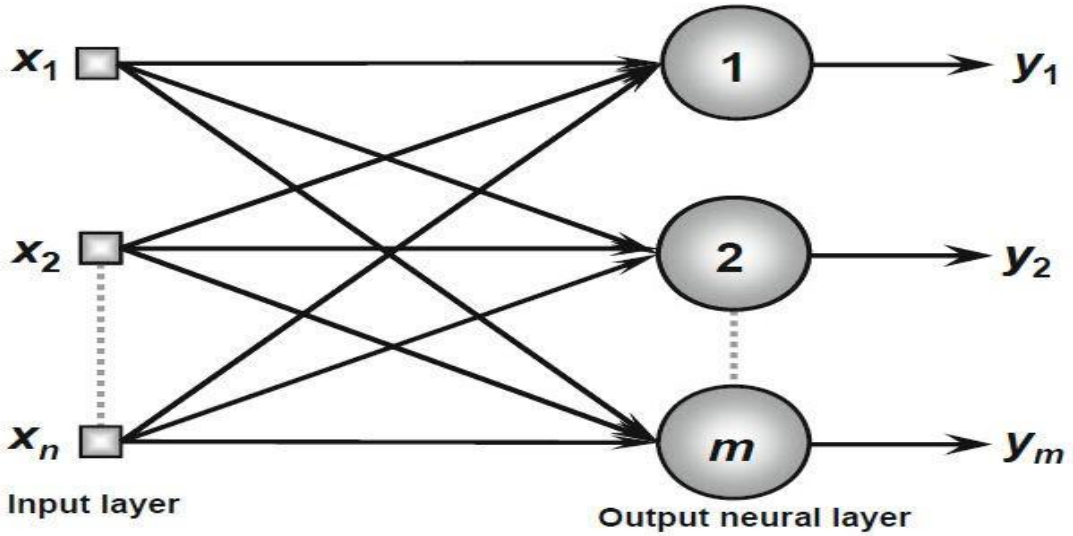
$$E_{total} = \sum \frac{1}{2}(\text{target} - \text{output})^2$$

- where $oi$ denotes the output of the perceptron on input $\vec{x}i$ with activation function $g$.
- The factor of 1/2 is included in order to simplify the calculation of the derivative later.
- Thus, $E(X)=0$ when $oi=yi$ for all input-output pairs $(\vec{x}i,yi)$ in $X$, so a natural objective is to attempt to change $\vec{w}$ and $b$ such that $E(X)$ is as close to zero as possible.
- Thus, minimizing $E(X)$ with respect to $\vec{w}$ and $b$ should yield a good classification boundary.

$$\vec{w_{i+1}} = \vec{w}_i - \alpha \frac{\partial E(X)}{\partial \vec{w}_i}$$
$$b_{i+1} = b_i - \alpha \frac{\partial E(X)}{\partial b_i},$$

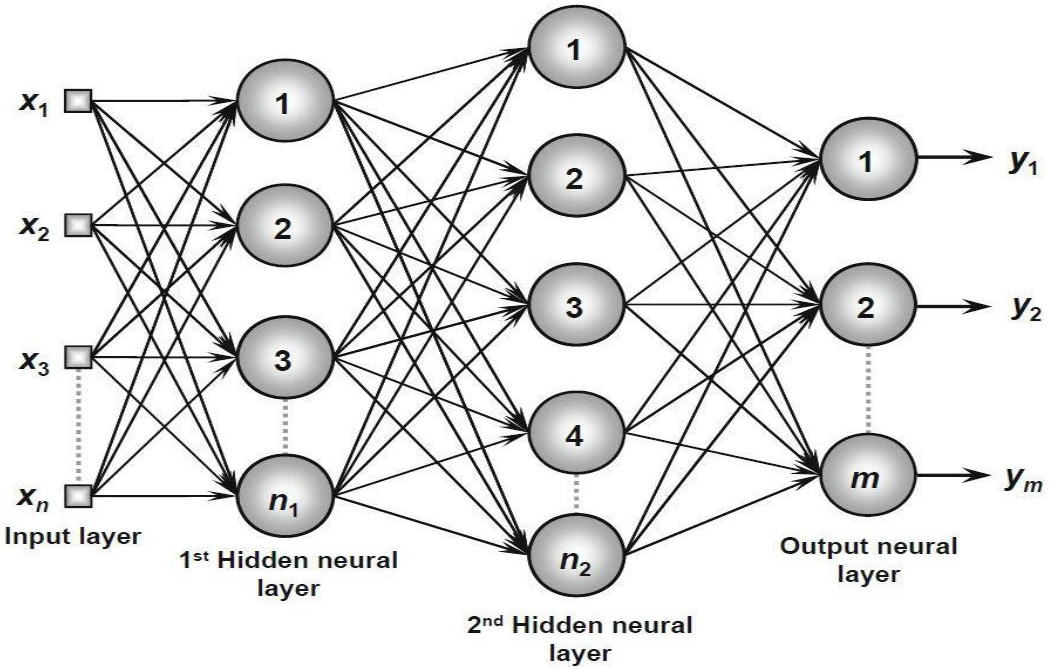
## Single-Layer Feedforward Architecture

- This artificial neural network has just one input layer and a single neural layer, which is also the output layer.
- Figure illustrates a simple-layer feedforward network composed of n inputs and m outputs.
- The information always flows in a single direction (thus, unidirectional), which is from the input layer to the output layer



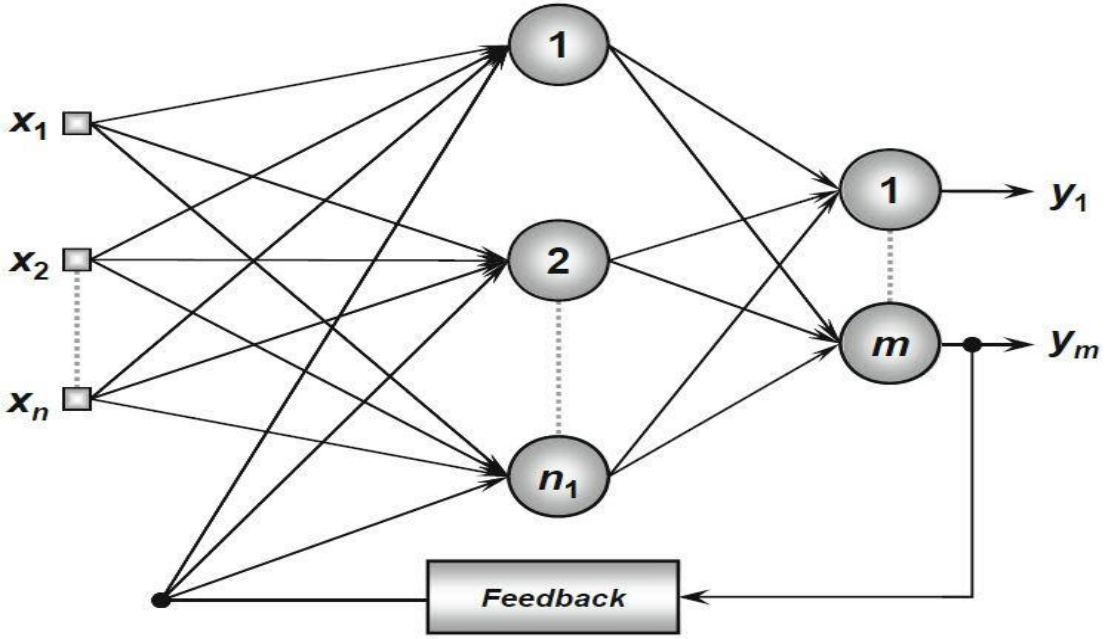
**Multi-Layer Feedforward Architecture**

- This artificial neural feedforward networks with multiple layers are composed of one or more hidden neural layers.
- Figure shows a feedforward network with multiple layers composed of one input layer with n sample signals, two hidden neural layers consisting of $n_1$ and $n_2$ neurons respectively, and, finally, one output neural layer composed of m neurons representing the respective output values of the problem being analyzed



# Neural Network with Back Propagation

- Feedback Architecture
- In these networks, the outputs of the neurons are used as feedback inputs for other neurons.
- Figure illustrates an example of a Perceptron network with feedback, where one of its output signals is fed back to the middle layer.



# Backpropagation

- Backpropagation is a supervised learning algorithm, which means it requires a set of training data with known inputs and outputs.

- The algorithm works by propagating the error back through the network from the output layer to the input layer, adjusting the weights of each connection along the way.

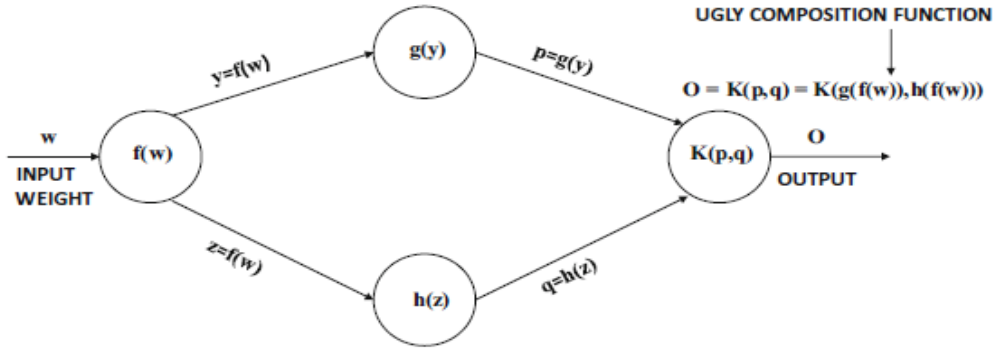
- The backpropagation algorithm can be broken down into four steps:
- **Forward propagation**: The input values are passed through the network, and each neuron's output is calculated using the current weights.
- **Error calculation**: The difference between the predicted output and the actual output is calculated, and the error is propagated backwards through the network.
- **Backward propagation**: The error is used to update the weights of each neuron in the network. The weights are adjusted in the opposite direction of the gradient of the error function, which is calculated using the chain rule of calculus.
- **Repeat:** Steps 1-3 are repeated for each training example until the error is minimized.

# Back Propagation

- The backpropagation algorithm is a powerful tool for training neural networks, but it can be slow and computationally expensive for large datasets or complex networks.
- There are also several variations of backpropagation, such as **stochastic gradient descent** and **mini-batch gradient descent**, which can improve the efficiency and speed of the algorithm.
- Gradient descent is an optimization algorithm used to update the weights of the neurons in a neural network during the training process.
- The goal is to minimize the error between the predicted output and the actual output of the network.

# Chain Rule

- The main goal of the backward phase is to learn the gradient of the loss function with respect to the different weights by using the chain rule of differential calculus.
- These gradients are used to update the weights.
- these gradients are learned in the backward direction, starting from the output node, this learningprocess is referred to as the backward phase.

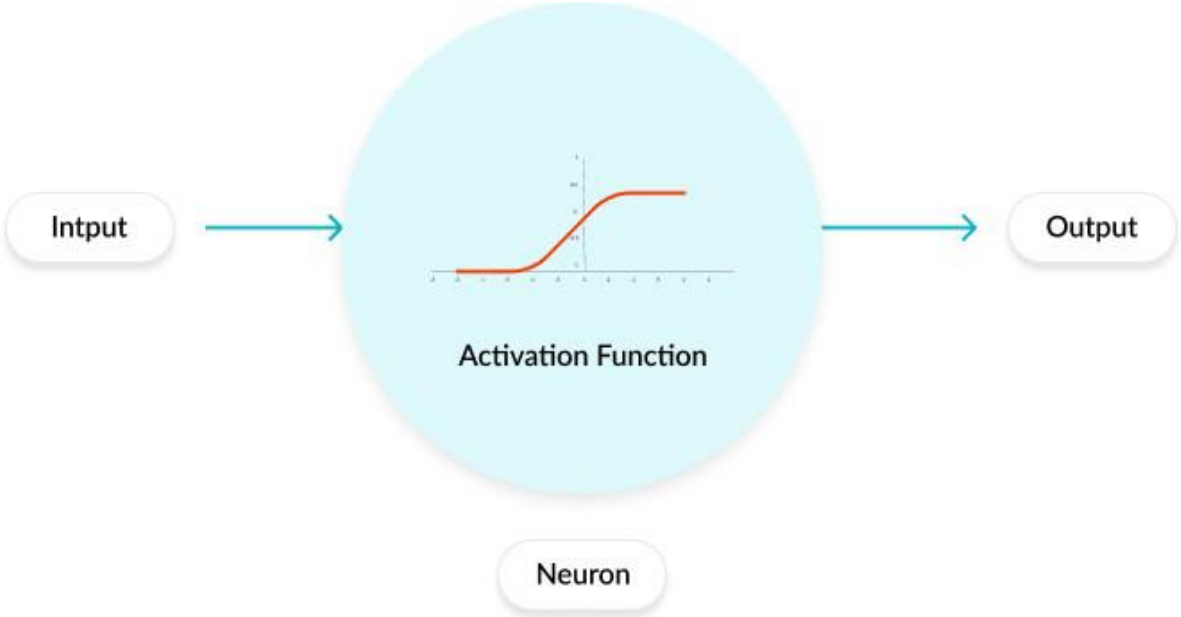



$$\frac{\partial o}{\partial w} = \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial w} \quad \text{[Multivariable Chain Rule]}$$

$$= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial y} \cdot \frac{\partial y}{\partial w} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial w} \quad \text{[Univariate Chain Rule]}$$

# Activation Functions

- The activation function is a mathematical “gate” in between the input feeding the current neuron and its output going to the next layer.
- It can be as simple as a step function that turns the neuron output on and off, depending on a rule or threshold. Or it can be a transformation that maps the input signals into output signals that are needed for the neural network to function.

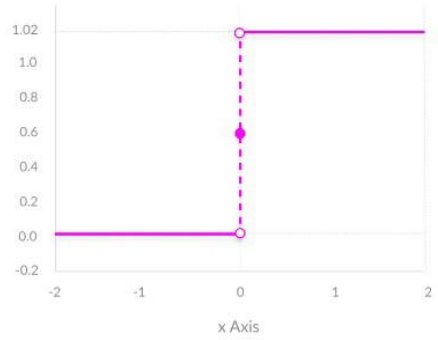


# Types of Activation Function

- Different Types of Activation Function are
- Binary Step Function
- Linear Activation Function
- Non Linear Activation Function

# Binary Step Activation Function

- A binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.



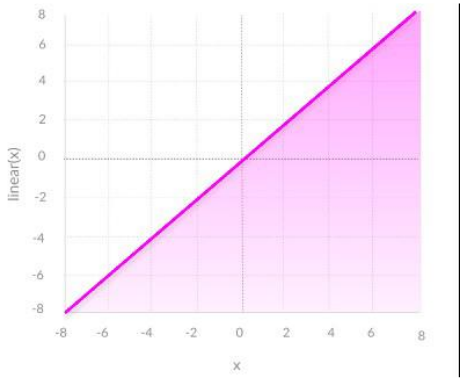

*Binary step*

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

The problem with a step function is that
it does not allow multi-value outputs—for example, it cannot support classifying the inputs into one of several categories.
Sharp boundaries eg: buying car problem

# Linear Activation Function

- It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input.
- Linear function is better than a step function because it allows multiple outputs, not just yes and no.




*Linear*

$$f(x) = x$$

# Problems with Linear Activation Function

- Not possible to use backpropagation (gradient descent) to train the model—
- – The derivative of the function is a constant, and has no relation to the input, X. So it's not possible to go back and understand which weights in the input neurons can provide a better prediction.
- All layers of the neural network collapse into one—
- – With linear activation functions, no matter how many layers in the neural network, the last layer will be a linear function of the first layer (because a linear combination of linear functions is still a linear function). So a linear activation function turns the neural network into just one layer.

# Non-Linear Activation Function

- Modern neural network models use non-linear activation functions.
- They allow the model to create complex mappings between the network's inputs and outputs, which are essential for learning and modeling complex data, such as images, video, audio, and data sets which are nonlinear or have high dimensionality.
- Almost any process imaginable can be represented as a functional computation in a neural network, provided that the activation function is non-linear.
- Non-linear functions address the problems of a linear activation function:
- – They allow backpropagation because they have a derivative function which is related to the inputs.
- – They allow "stacking" of multiple layers of neurons to create a deep neural network.
- Multiple hidden layers of neurons are needed to learn complex data sets with high levels of accuracy.

# Non-Linear Activation Function

- Sigmoid / Logistic
- Tanh / Hyperbolic Tangent
- ReLU (Rectified Linear Unit)
- Leaky ReLU
- Parametric ReLU
- Softmax
- Swish

# Sigmoid / Logistic


*Sigmoid / Logistic*

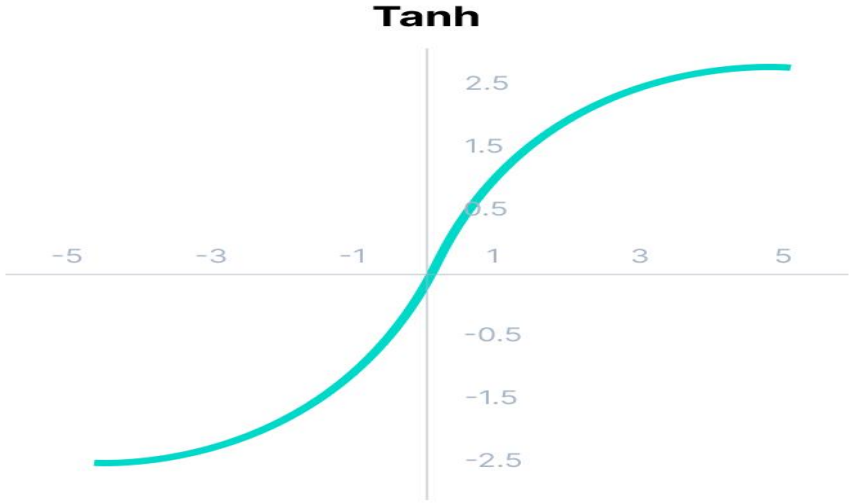
$$f(x) = \frac{1}{1 + e^{-x}}$$

•It is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.

•The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function.

# Tanh

- Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1.
- In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.
- Problem with Sigmoid and tanh is Vanishing Gradient Problem

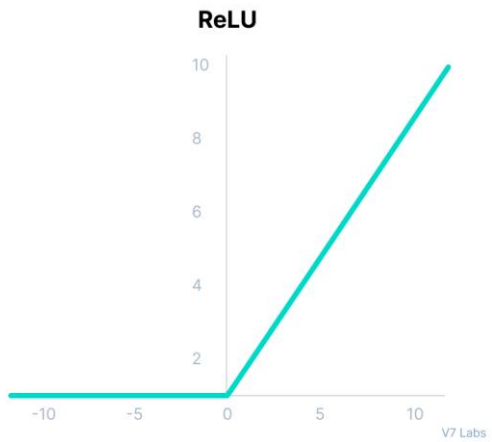



$$Tanh$$

$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)}$$

# ReLU

- ReLU stands for Rectified Linear Unit.
- Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.
- ReLU function does not activate all the neurons at the same time.
- The neurons will only be deactivated if the output of the linear transformation is less than 0.




$$ReLU$$

$$f(x) = max\,(0, x)$$

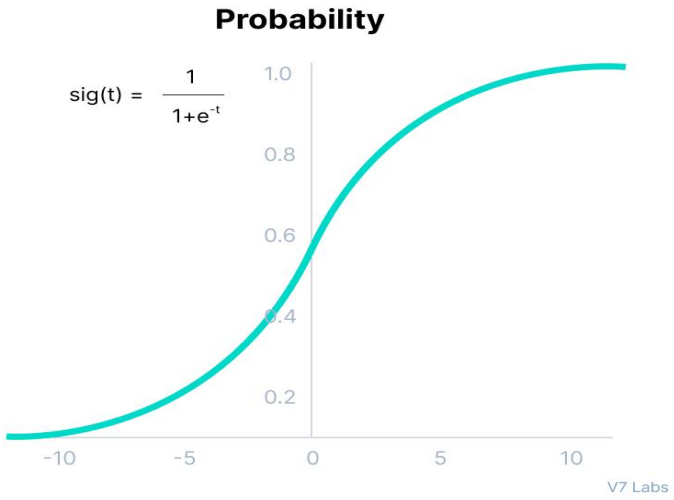
If you are not sure which activation function to use in hidden layer, use ReLU

# Softmax Function

- The Softmax function is described as a combination of multiple sigmoids.

- It calculates the relative probabilities. Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class.

- It is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification.




**Softmax**

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$