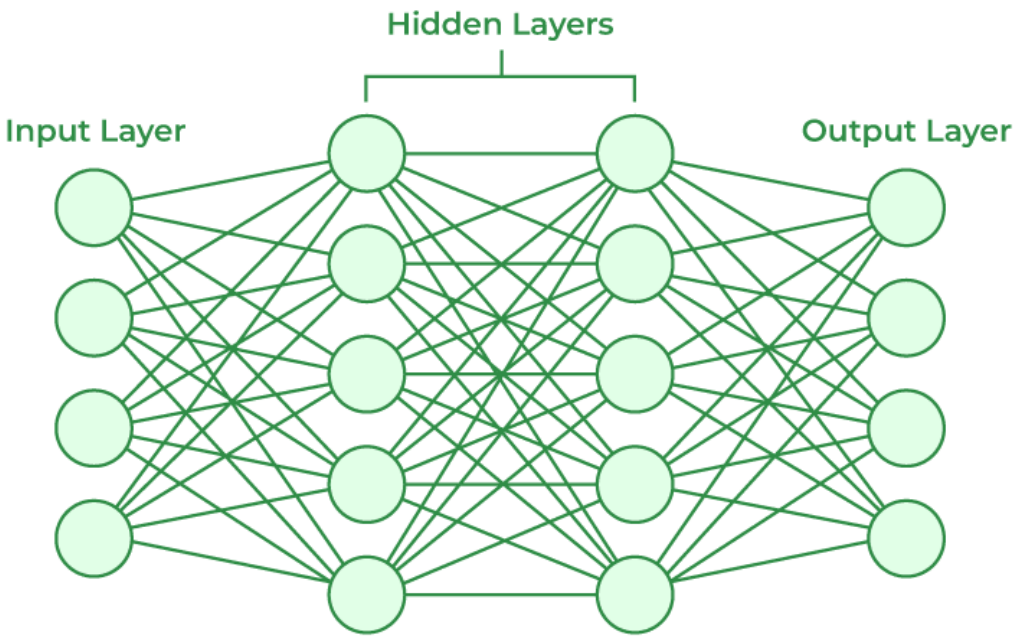
Module 3 – Neural Network

Artificial Neural Networks

Artificial Neural Networks contain artificial neurons which are called units . These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.



The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets. The input

layer of an artificial neural network is the first layer, and it receives input from external sources and releases it to the hidden layer, which is the second layer. In the hidden layer, each neuron receives input from the previous layer neurons, computes the weighted sum, and sends it to the neurons in the next layer. These connections are weighted means effects of the inputs from the previous layer are optimized more or less by assigning different-different weights to each input and it is adjusted during the training process by optimizing these weights for improved model performance.

Artificial neurons vs Biological neurons

The concept of artificial neural networks comes from biological neurons found in animal brains So they share a lot of similarities in structure and function wise.

- Structure : The structure of artificial neural networks is inspired by biological neurons. A biological neuron has a cell body or soma to process the impulses, dendrites to receive them, and an axon that transfers them to other neurons. The input nodes of artificial neural networks receive input signals, the hidden layer nodes compute these input signals, and the output layer nodes compute the final output by processing the hidden layer's results using activation functions.

Biological Neuron	Artificial Neuron
Dendrite	**Inputs**
Cell nucleus or Soma	**Nodes**
Synapses	**Weights**
Axon	**Output**

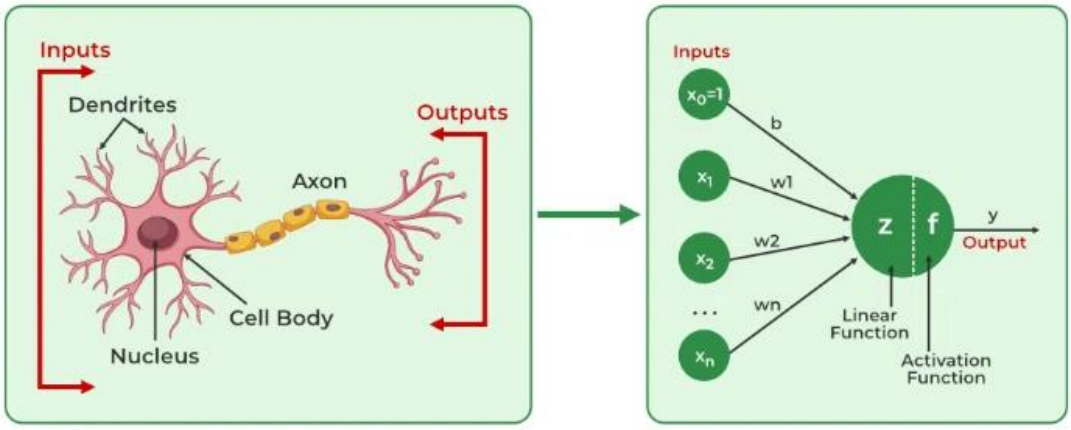
- **Synapses :** Synapses are the links between biological neurons that enable the transmission of impulses from dendrites to the cell body.

Synapses are the weights that join the one-layer nodes to the next-layer nodes in artificial neurons. The strength of the links is determined by the weight value.

- **Learning :** In biological neurons, learning happens in the cell body nucleus or soma, which has a nucleus that helps to process the impulses. An action potential is produced and travels through the axons if the impulses are powerful enough to reach the threshold. This becomes possible by synaptic plasticity, which represents the ability of synapses to become stronger or weaker over time in reaction to changes in their activity. In artificial neural networks, backpropagation is a technique used for learning, which adjusts the weights between nodes according to the error or differences between predicted and actual outcomes.

Biological Neuron	Artificial Neuron
Synaptic plasticity	**Backpropagations**

- **Activation :** In biological neurons, activation is the firing rate of the neuron which happens when the impulses are strong enough to reach the threshold. In artificial neural networks, A mathematical function known as an activation function maps the input to the output, and executes activations**.**



Biological neurons to Artificial neurons

How do Artificial Neural Networks learn?

Artificial neural networks are trained using a training set. For example, suppose you want to teach an ANN to recognize a cat. Then it is shown thousands of different images of cats so that the network can learn to identify a cat. Once the neural network has been trained enough using images of cats, then you need to check if it can identify cat images correctly. This is done by making the ANN classify the images it is provided by deciding whether they are cat images or not. The output obtained by the ANN is corroborated by a human-provided description of whether the image is a cat image or not. If the ANN identifies incorrectly then back-propagation is used to adjust whatever it has learned during training. Backpropagation is done by fine-tuning the weights of the connections in ANN units based on the error rate obtained. This process continues until the artificial neural network can correctly recognize a cat in an image with minimal possible error rates.

Types of Neural Networks

There are *seven* types of neural networks that can be used.

- Feedforward Networks: A feedforward neural network is a simple artificial neural network architecture in which data moves from input to output in a single direction. It has input, hidden, and output layers; feedback loops are absent. Its straightforward architecture makes it appropriate for a number of applications, such as regression and pattern recognition.

- Multilayer Perceptron (MLP): MLP is a type of feedforward neural network with three or more layers, including an input layer, one or more hidden layers, and an output layer. It uses nonlinear activation functions.
- Convolutional Neural Network (CNN): A Convolutional Neural Network (CNN) is a specialized artificial neural network designed for image processing. It employs convolutional layers to automatically learn hierarchical features from input images, enabling effective image recognition and classification. CNNs have revolutionized computer vision and are pivotal in tasks like object detection and image analysis.
- Recurrent Neural Network (RNN): An artificial neural network type intended for sequential data processing is called a Recurrent Neural Network (RNN). It is appropriate for applications where contextual dependencies are critical, such as time series prediction and natural language processing, since it makes use of feedback loops, which enable information to survive within the network.
- Long Short-Term Memory (LSTM): LSTM is a type of RNN that is designed to overcome the vanishing gradient problem in training RNNs. It uses memory cells and gates to selectively read, write, and erase information.

Applications of Artificial Neural Networks

1. Social Media: Artificial Neural Networks are used heavily in Social Media. For example, let's take the 'People you may know' feature on Facebook that suggests people that you might know in real life so that you can send them friend requests. Well, this magical effect is achieved by using Artificial Neural Networks that analyze your profile, your interests, your current friends, and also their friends and various other factors to calculate the people you might potentially know. Another common application of Machine Learning in social media is facial recognition . This is done by finding around 100 reference points on the person's face and then matching them with those already available in the database using convolutional neural networks.
2. Marketing and Sales: When you log onto E-commerce sites like Amazon and Flipkart, they will recommend your products to buy based on your previous browsing history. Similarly, suppose you love Pasta, then Zomato, Swiggy, etc. will show you restaurant recommendations based on your tastes and previous order history. This is true across all new-age marketing segments like Book sites, Movie services, Hospitality sites, etc.

and it is done by implementing personalized marketing . This uses Artificial Neural Networks to identify the customer likes, dislikes, previous shopping history, etc., and then tailor the marketing campaigns accordingly.

3. Healthcare : Artificial Neural Networks are used in Oncology to train algorithms that can identify cancerous tissue at the microscopic level at the same accuracy as trained physicians. Various rare diseases may manifest in physical characteristics and can be identified in their premature stages by using Facial Analysis on the patient photos. So the full-scale implementation of Artificial Neural Networks in the healthcare environment can only enhance the diagnostic abilities of medical experts and ultimately lead to the overall improvement in the quality of medical care all over the world.

4. Personal Assistants: I am sure you all have heard of Siri, Alexa, Cortana, etc., and also heard them based on the phones you have!!! These are personal assistants and an example of speech recognition that uses Natural Language Processing to interact with the users and formulate a response accordingly. Natural Language Processing uses artificial neural networks that are made to handle many tasks of these personal assistants such as managing the language syntax, semantics, correct speech, the conversation that is going on, etc.

Advantages of Neural Networks

Neural networks are widely used in many different applications because of their many benefits:

- Adaptability: Neural networks are useful for activities where the link between inputs and outputs is complex or not well defined because they can adapt to new situations and learn from data.
- Pattern Recognition: Their proficiency in pattern recognition renders them efficacious in tasks like as audio and image identification, natural language processing, and other intricate data patterns.
- Parallel Processing: Because neural networks are capable of parallel processing by nature, they can process numerous jobs at once, which speeds up and improves the efficiency of computations.
- Non-Linearity: Neural networks are able to model and comprehend complicated relationships in data by virtue of the non-linear **activation**

functions found in neurons, which overcome the drawbacks of linear models.

Disadvantages of Neural Networks

Neural networks, while powerful, are not without drawbacks and difficulties:

- Computational Intensity: Large neural network training can be a laborious and computationally demanding process that demands a lot of computing power.
- Black box Nature: As “black box” models, neural networks pose a problem in important applications since it is difficult to understand how they make decisions.
- Overfitting: Overfitting is a phenomenon in which neural networks commit training material to memory rather than identifying patterns in the data. Although regularization approaches help to alleviate this, the problem still exists.
- Need for Large datasets: For efficient training, neural networks frequently need sizable, labeled datasets; otherwise, their performance may suffer from incomplete or skewed data.

Frequently asked questions

1. What is a neural network?

A neural network is an artificial system made of interconnected nodes (neurons) that process information, modeled after the structure of the human brain. It is employed in machine learning jobs where patterns are extracted from data.

2. How does a neural network work?

Layers of connected neurons process data in neural networks. The network processes input data, modifies weights during training, and produces an output depending on patterns that it has discovered.

3. What are the common types of neural network architectures?

Feedforward neural networks, recurrent neural networks (RNNs), convolutional neural networks (CNNs), and long short-term memory networks (LSTMs) are examples of common architectures that are each designed for a certain task.

McCulloch and Pitts Network

The **McCulloch and Pitts Network** is considered one of the first models of a **neural network**, introduced by **Warren McCulloch** and **Walter Pitts** in 1943. Their model laid the foundation for artificial neural networks and computational neuroscience by introducing the concept of a neuron as a binary threshold unit that simulates the firing behavior of biological neurons.

Key Concepts of McCulloch and Pitts Network:

1. Neuron Model:

The McCulloch-Pitts neuron is a **simplified binary model** of a biological neuron. It receives inputs from other neurons or sources, processes them, and produces an output (either 0 or 1). The neuron "fires" (outputs 1) if a certain threshold of input is reached.

2. Binary Inputs and Outputs:

- The inputs to the neuron are binary (0 or 1), representing whether the input signal is present or absent.
- The output is also binary (0 or 1), representing whether the neuron fires or not.

3. Weighted Sum of Inputs:

Each input is associated with a weight (similar to the concept used in modern neural networks). The neuron calculates a weighted sum of its inputs:

$$\text{sum} = \sum_{i=1}^{n} w_i x_i$$

where:

- w_i is the weight of the i-th input.
- x_i is the i-th input (either 0 or 1).

4. Threshold Function:

The neuron applies a threshold to the weighted sum of inputs. If the weighted sum exceeds or meets a predefined threshold θ (theta), the neuron outputs 1 (fires); otherwise, it outputs 0 (does not fire).

The mathematical model is:

The mathematical model is:

$$y = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0, & \text{otherwise} \end{cases}$$

where y is the output of the neuron.

5. Logical Operations:

McCulloch and Pitts demonstrated that their model could perform basic **logical operations** (such as AND, OR, and NOT), showing that a network of such neurons could simulate complex computations.

- **AND Gate**: The neuron outputs 1 only if all inputs are 1, simulating the behavior of the logical AND function.
- **OR Gate**: The neuron outputs 1 if at least one input is 1, simulating the behavior of the logical OR function.
- **NOT Gate**: The output is 1 when the input is 0, simulating the logical NOT function.

6. Network of Neurons:

McCulloch and Pitts suggested that a network of interconnected neurons could simulate any computable function, given enough neurons and proper configuration of weights and thresholds. This was a key insight that contributed to the later development of neural networks and **artificial intelligence**.

Applications and Contributions:

- **Foundations of Neural Networks**: The McCulloch and Pitts neuron is a precursor to modern artificial neural networks. It introduced the idea that neurons could be represented mathematically and interconnected to perform computations.
- **Turing Completeness**: McCulloch and Pitts showed that a sufficiently large network of these simple neurons could simulate a Turing machine, meaning that it could perform any computation that is computable.
- **Threshold Logic**: The concept of threshold logic, where outputs are activated when the weighted sum of inputs exceeds a certain threshold, is still used in modern neural networks, especially in perceptrons and multilayer networks.

Limitations of the Model:

1. **Binary Activation**: The McCulloch-Pitts neuron only outputs binary values (0 or 1), which limits its ability to model more complex real-world functions compared to modern neural networks that use continuous activation functions like the sigmoid or ReLU.
2. **No Learning Mechanism**: The McCulloch-Pitts network does not include a learning rule. The weights and thresholds must be manually specified, unlike modern neural networks that use learning algorithms like **backpropagation** to adjust weights automatically.

Relation to Modern Neural Networks:

- The **McCulloch-Pitts neuron** is a precursor to the **perceptron**, introduced later by Frank Rosenblatt. The perceptron is a more sophisticated version of the McCulloch-Pitts model, with adjustable weights and a learning algorithm.
- The fundamental idea of summing inputs and applying a threshold or activation function remains central to modern deep learning models, where the concept has been expanded and refined with different

architectures (e.g., convolutional neural networks, recurrent neural networks).

Summary:

The **McCulloch-Pitts network** is a foundational concept in neural networks and AI, representing neurons as binary threshold units capable of performing logical operations. Although simplistic by today's standards, it introduced key ideas that shaped the development of neural networks and laid the groundwork for future advancements in artificial intelligence.

Perceptron

The **Perceptron** is one of the simplest types of artificial neural networks, and it forms the foundation for more complex neural network architectures. It was introduced by **Frank Rosenblatt** in 1958 and is a **binary classifier** that determines whether an input belongs to one of two classes. The perceptron can be considered as the building block of neural networks, used to model simple linear relationships between inputs and outputs.

Key Concepts of the Perceptron:

1. Perceptron Structure:

A **perceptron** is a model of a single neuron in a neural network. It takes several inputs, applies weights to them, sums them up, and passes the result through an activation function to produce an output. The output is binary (either 0 or 1), making it a linear classifier.

The perceptron model can be mathematically represented as:

$$f(x) = \begin{cases} 1, & \text{if } w^T x + b \geq 0 \\ 0, & \text{if } w^T x + b < 0 \end{cases}$$

where:

- x is the input vector.
- w is the weight vector (the model parameters).
- b is the bias term (which shifts the decision boundary).
- $f(x)$ is the output (either 0 or 1, depending on whether the condition is met).

2. Mathematics Behind the Perceptron:

- **Input Vector x**: A vector of feature values that describes an example from the dataset.
- **Weight Vector w**: The parameters that determine the contribution of each input to the output.
- **Bias b**: An additional parameter that allows the decision boundary to shift, making it more flexible.
- **Weighted Sum**: The perceptron computes the dot product of the input vector and the weight vector, plus the bias:

$$w^T x + b = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

- **Activation Function**: The perceptron uses a **step function** (also called the Heaviside function) as its activation function. If the weighted sum is greater than or equal to 0, the perceptron outputs 1; otherwise, it outputs 0.

3. Perceptron Learning Algorithm:

The perceptron learning algorithm is an iterative process used to adjust the weights based on the errors in prediction. The goal is to find a set of weights that correctly classify all the training examples.

- **Initialization**: Randomly initialize the weights and bias.
- **Training Process**:

1. For each training example, compute the predicted output using the current weights.
2. Compare the predicted output with the actual label (target).
3. Update the weights and bias if the prediction is incorrect.
4. Repeat the process for all training examples and continue iterating until the model converges or reaches a certain number of iterations.

- **Weight Update Rule**: If the prediction is incorrect, the weights are updated as follows:

$$w_i = w_i + \eta(y - \hat{y})x_i$$

$$b = b + \eta(y - \hat{y})$$

Where:

- y is the true label.
- $\hat{y}$ is the predicted label.
- η is the learning rate (a small positive number that controls the step size of updates).

(Note : Y hat predicted label, eta learning rate)

The idea is that if the perceptron makes a wrong prediction, it adjusts its weights in the direction that would make the correct prediction more likely.

4. Linearly Separable Data:

- The perceptron can only solve problems where the data is **linearly separable**, meaning the data points can be separated by a straight line (or hyperplane in higher dimensions).
- For example, the perceptron can successfully solve an AND or OR problem, as these are linearly separable, but it cannot solve the **XOR problem** because XOR is **not linearly separable**.

5. Decision Boundary:

- The perceptron defines a **linear decision boundary** (a line in 2D, a plane in 3D, or a hyperplane in higher dimensions) to separate the two classes. The weights and bias determine the orientation and position of this boundary.
- The decision boundary is given by the equation

$$w^T x + b = 0.$$

Advantages and Limitations of the Perceptron:

Advantages:

1. **Simplicity**: The perceptron is easy to implement and understand.
2. **Computational Efficiency**: It has low computational complexity, making it efficient for linearly separable data.
3. **Foundation for Neural Networks**: The perceptron is the building block for more complex neural networks, such as multilayer perceptrons (MLPs).

Limitations:

1. **Linear Separability**: The perceptron can only solve problems that are linearly separable. It cannot solve problems like XOR.
2. **Convergence**: The perceptron learning algorithm will only converge if the training data is linearly separable. If it's not, the algorithm will continue to iterate without finding a solution.
3. **Binary Output**: The perceptron only provides binary outputs (0 or 1), which limits its use in more complex classification or regression tasks that require continuous outputs.

Multilayer Perceptron (MLP) and Modern Neural Networks:

The multilayer perceptron (MLP) extends the simple perceptron by adding multiple layers of neurons (hidden layers) and using non-linear activation functions (like sigmoid, ReLU). This allows MLPs to solve more complex problems that are not linearly separable, including the XOR problem.

Backpropagation is used to train multilayer perceptrons, allowing them to learn from errors and adjust the weights across multiple layers.

Perceptron in Machine Learning Algorithms:

Binary Classification: The perceptron is used for binary classification tasks where the data can be separated by a hyperplane.

Foundation for Support Vector Machines (SVMs): The concept of maximizing the margin between classes in SVMs is closely related to the perceptron.

Deep Learning: The perceptron inspired the development of deeper neural networks, which are now widely used in deep learning applications.

Backpropagation Algorithm Machine Learning

BACKPROPAGATION (training_example, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form (x, t), where (x) is the vector of network input values, and (t) is the vector of target network output values.

η is the learning rate (e.g., 0.05).

n_i, is the number of network inputs,

n_{hidden} the number of units in the hidden layer, and

n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

Steps in Backpropagation algorithm

1. Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.

2. Initialize all network weights to small random numbers

3. Until the termination condition is met, Do

    For each (x, t), in training examples, Do

        Propagate the input forward through the network:

1. Input the instance x, to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network

2. For each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each network unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$\mathbf{w_{ji}} \leftarrow \mathbf{w_{ji}} + \Delta\, \mathbf{w_{ji}}$$

Where

$$\Delta\, \mathbf{w_{ji}} = \eta \delta_j x_{\mathbf{ji}}$$

Gradient Descent and Delta Rule in ANN

Gradient Descent and the Delta Rule is used separate the Non-Linearly Separable data.

Weights are updated using the following rule,

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

Where,

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

Gradient Descent Algorithm

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, *where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - For each linear unit weight w_i, Do

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

Gradient descent is an important general paradigm for learning.

It is a strategy for searching through a large or infinite hypothesis space that can be applied whenever

1. the hypothesis space contains continuously parameterized hypotheses (e.g., the weights in a linear unit), and
2. the error can be differentiated with respect to these hypothesis parameters.

The key practical difficulties in applying gradient descent are

1. Converging to a local minimum can sometimes be quite slow (i.e., it can require many thousands of gradient descent steps), and
2. If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

What is an Activation Function?

An activation function in the context of neural networks is a mathematical function applied to the output of a neuron. The purpose of an activation function is to introduce non-linearity into the model, allowing the network to learn and

represent complex patterns in the data. Without non-linearity, a neural network would essentially behave like a linear regression model, regardless of the number of layers it has.

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Explanation: We know, the neural network has neurons that work in correspondence with *weight, bias,* and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as ***back-propagation***. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

Elements of a Neural Network

Input Layer: This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.

Hidden Layer*:* Nodes of this layer are not exposed to the outer world, they are part of the abstraction provided by any neural network. The hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the output layer.

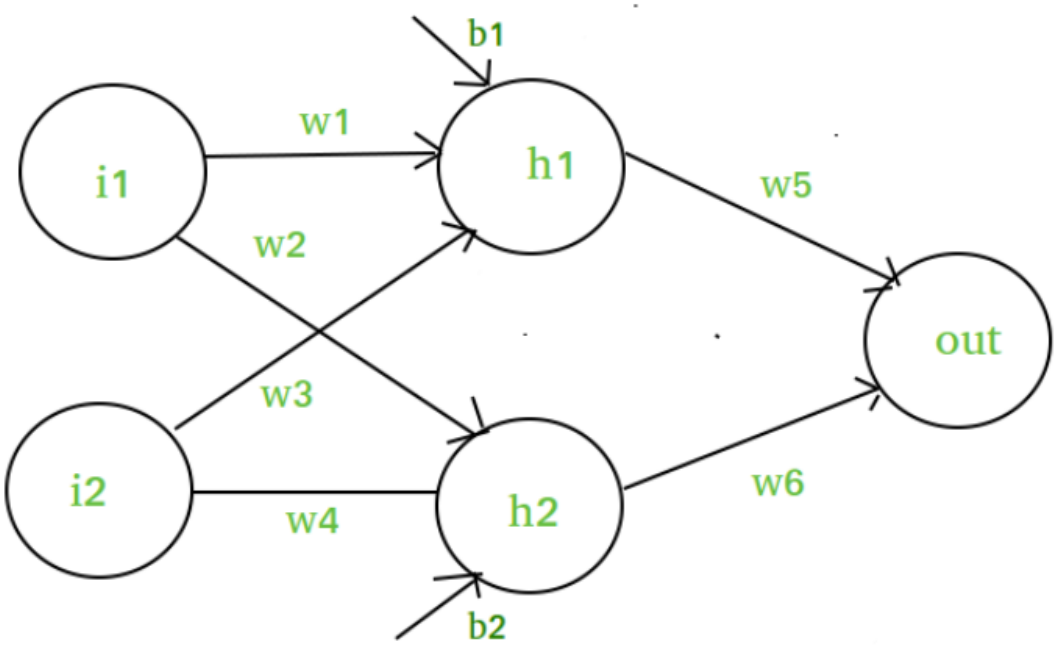
Output Layer: This layer bring up the information learned by the network to the outer world.

Why do we need Non-linear activation function?

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Mathematical proof

Suppose we have a Neural net like this :-



Elements of the diagram are as follows:

Hidden layer i.e. layer 1:

z(1) = W(1)X + b(1) a(1)

Here,

- *z(1) is the vectorized output of layer 1*
- *W(1) be the vectorized weights assigned to neurons of hidden layer i.e. w1, w2, w3 and w4*
- *X be the vectorized input features i.e. i1 and i2*
- *b is the vectorized bias assigned to neurons in hidden layer i.e. b1 and b2*
- *a(1) is the vectorized form of any linear function.*

*(**Note:** We are not considering activation function here)*

***Note :** Input for layer 2 is output from layer 1*

z(2) = W(2)a(1) + b(2)

a(2) = z(2)

Calculation at Output layer

*z(2) = (W(2) * [W(1)X + b(1)]) + b(2)*

*z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]*

Let,

*[W(2) * W(1)] = W*

*[W(2)*b(1) + b(2)] = b*

*Final output : z(2) = W*X + b*

which is again a linear function

This observation results again in a linear function even after applying a hidden layer, hence we can conclude that, doesn't matter how many hidden layer we attach in neural net, all layers will behave same way because ***the composition of two linear function is a linear function itself***. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error. **Hence we need an activation function.**

Variants of Activation Function

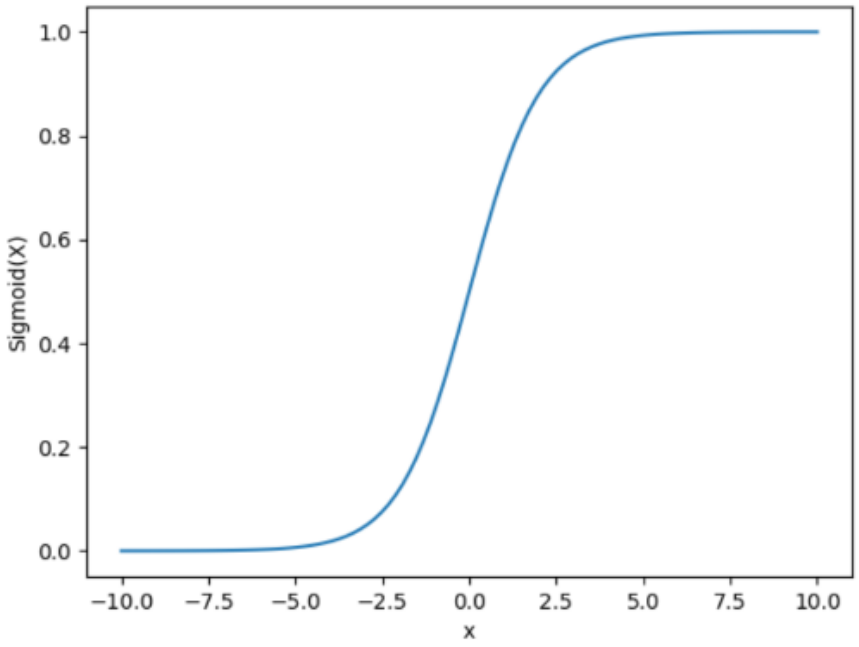
Linear Function

- **Equation :** Linear function has the equation similar to as of a straight line i.e. **y = x**
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- **Range :** -inf to +inf
- **Uses : Linear activation function** is used at just one place i.e. output layer.
- **Issues :** If we will differentiate linear function to bring non-linearity, result will no more depend on *input "x"* and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

For example : Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output

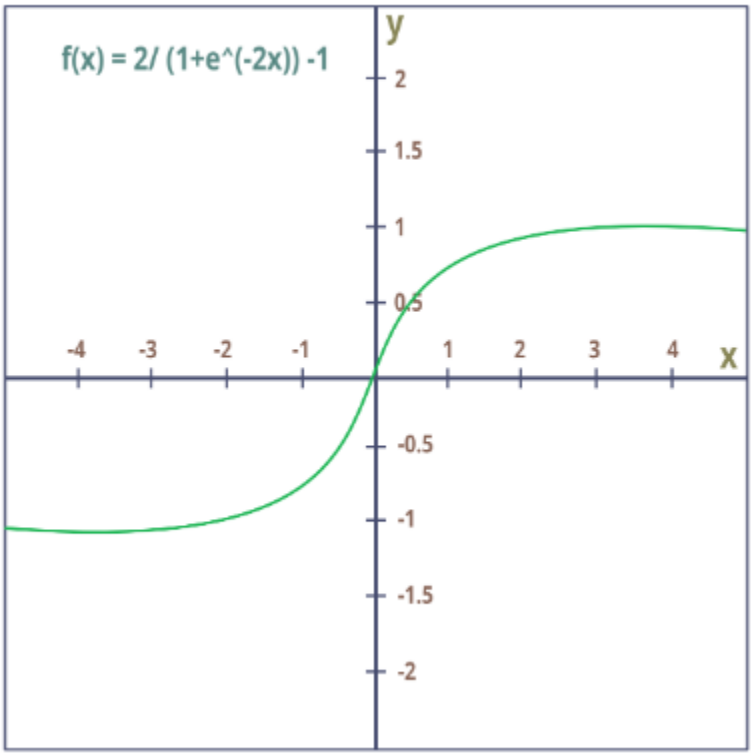
layer. Even in this case neural net must have any non-linear function at hidden layers.

Sigmoid Function



- It is a function which is plotted as **'S'** shaped graph.
- **Equation :** A = 1/(1 + e-x)
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

Tanh Function



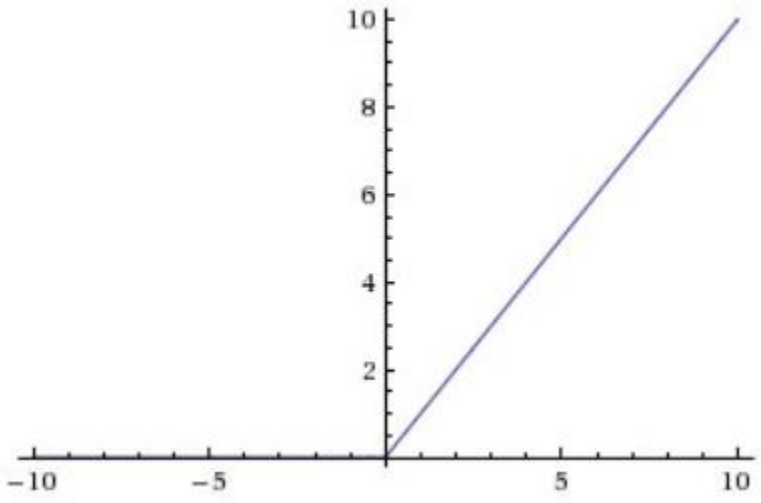
- The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation :-**
  f(x) = tanh(x) = 2/(1 + e-2x) – 1
  OR
  tanh(x) = 2 * sigmoid(2x) – 1
- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0

or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

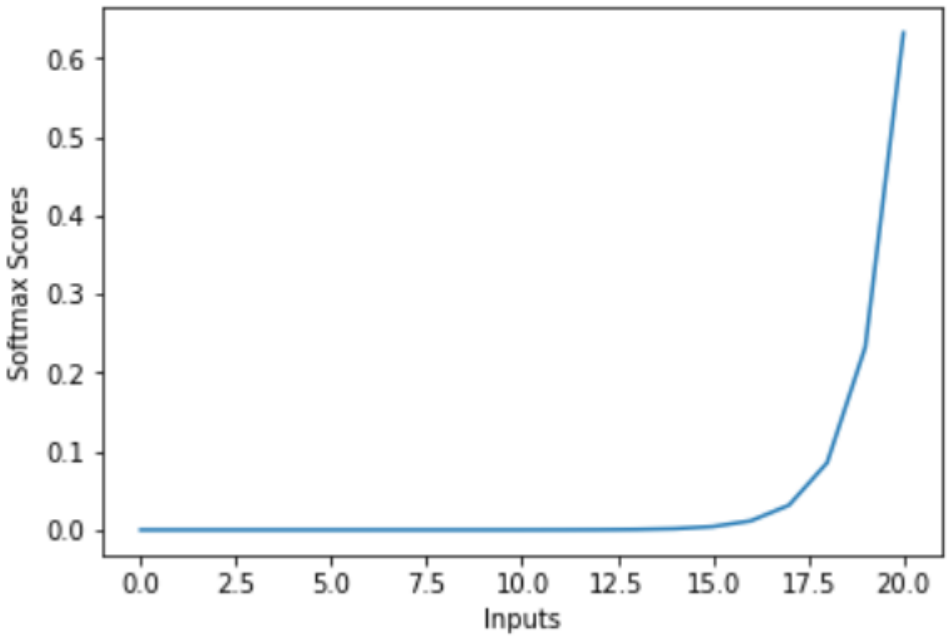
RELU Function

- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :- *A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.



Softmax Function



The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Output:-** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.
- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

Usecase based -Activation Functions and Loss Functions for neural networks

Problem Type	Last-layer Output Nodes	Hidden-layer activation	Last-layer activation	Loss function
Binary classification	1	RELU (first choice), Tanh (for RNNs)	Sigmoid	Binary Crossentropy
Multi-class, single-label classification	Number of classes		Softmax	Categorical Crossentropy
Multi-class, multi-label classification	Number of classes		Sigmoid (one for each class)	Binary Crossentropy
Regression to arbitrary values	1		None	MSE
Regression to values between 0 and 1	1		Sigmoid	MSE/Binary Crossentropy