Course: Artificial Intelligence and Machine Learning Code: 20CS51I

WEEK - 10: DEEP LEARNING

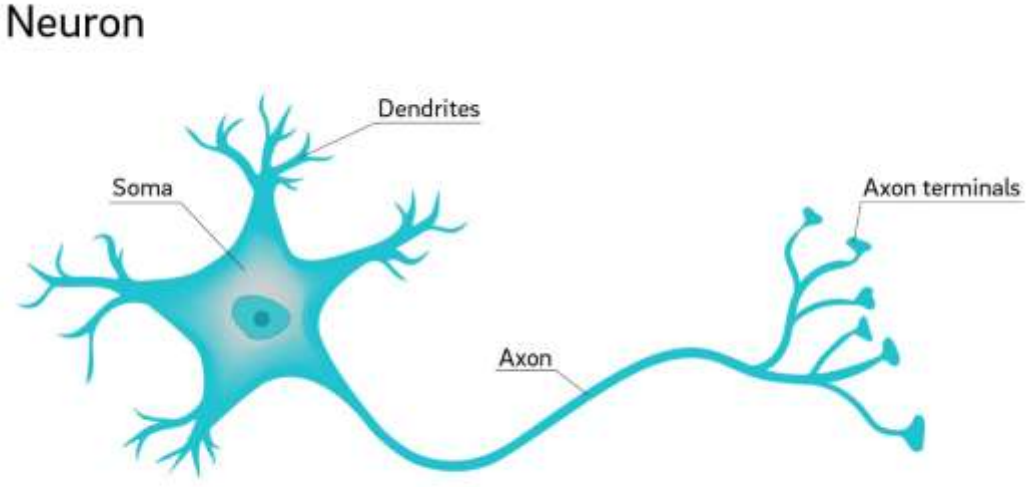
- **Introduction to Neural Networks.**
- **Biological Neurons**
- **Artificial neuron / Perceptron**
- **Working of perceptron**
- **Neural network**
- **Activation function**
- **Cost function**

Session No. 2

Biological Neurons:

In living organisms, the brain is the control unit of the neural network, and it has different subunits that take care of vision, senses, movement, and hearing. The brain is connected with a dense network of nerves to the rest of the body's sensors and actors. There are approximately 10^{11} neurons in the brain, and these are the building blocks of the complete central nervous system of the living body.

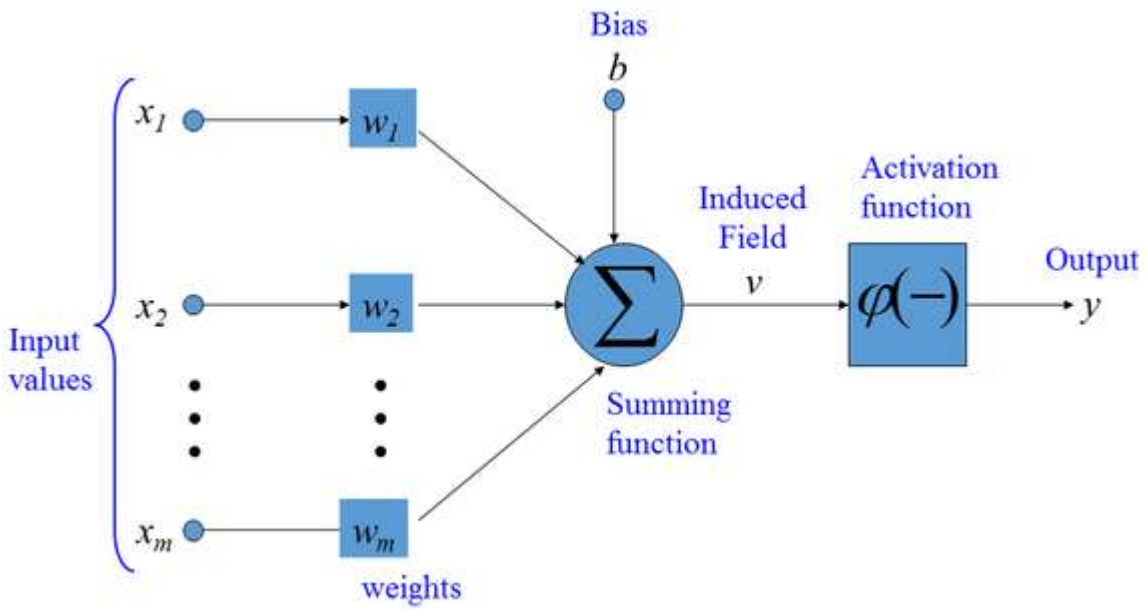
A biological neural network is **a network of neurons that are connected together by axons and dendrites**. The connections between neurons are made by synapses.



An artificial neuron

An **artificial neuron** is a mathematical function conceived as a model of biological neurons, a neural network. Artificial neurons are elementary units in an artificial neural network.

The artificial neuron receives one or more inputs (representing excitatory postsynaptic potentials and inhibitory postsynaptic potentials at neural dendrites) and sums them to produce an output (or activation, representing a neuron's action potential which is transmitted along its axon). Usually each input is separately weighted, and the sum is passed through a non-linear function known as an activation function or transfer function. The transfer functions usually have a sigmoid shape, but they may also take the form of other non-linear functions, piecewise linear functions, or step functions.



Perceptron :

A Perceptron is **an Artificial Neuron**. It is the simplest possible Neural Network. Neural Networks are the building blocks of Machine Learning.

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.

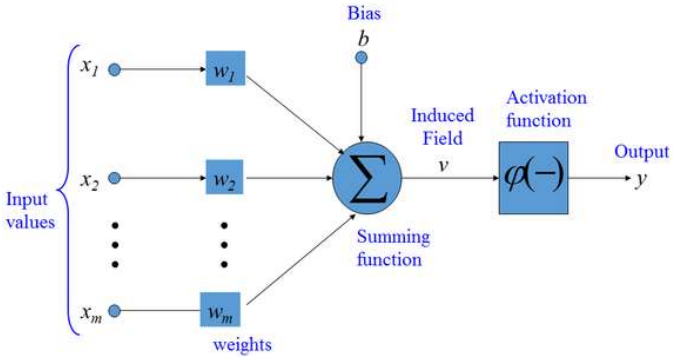
Types of Perceptron :

1. Single layer: Single layer perceptron can learn only linearly separable patterns.
2. Multilayer: Multilayer perceptrons can learn about two or more layers having a greater processing power.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then 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** and is represented by **'f'**.



This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

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

Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.

$\sum wi*xi + b$

Step-2

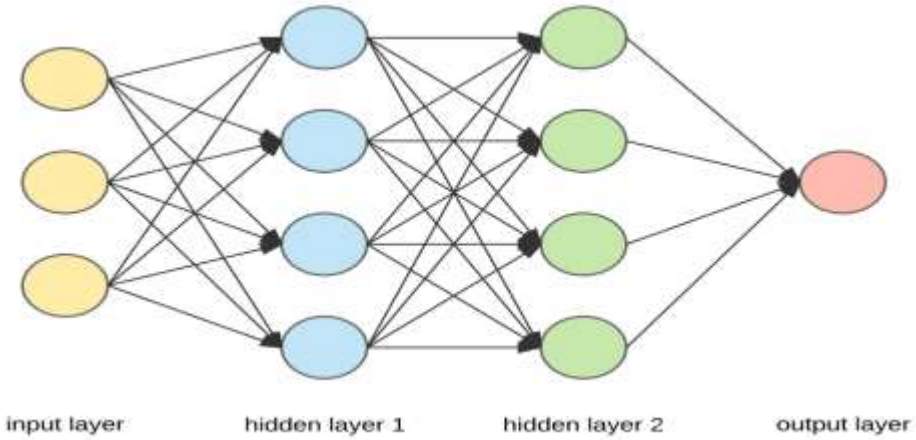
In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows: **$Y = f(\sum wi*xi + b)$**

Neural Networks:

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Architecture:

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.



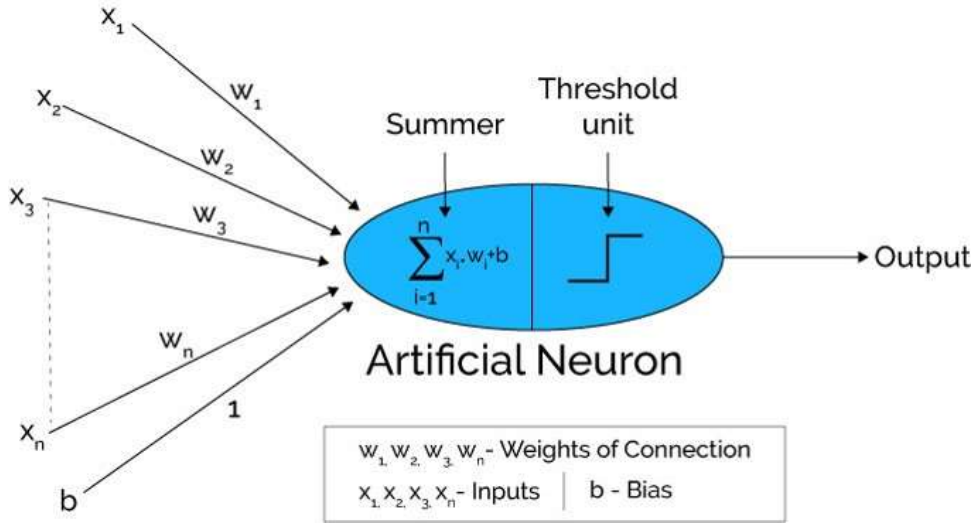
Input Layer: Also known as Input nodes are the inputs/information from the outside world is provided to the model to learn and derive conclusions from. Input nodes pass the information to the next layer i.e Hidden layer.

Hidden Layer: Hidden layer is the set of neurons where all the computations are performed on the input data. There can be any number of hidden layers in a neural network. The simplest network consists of a single hidden layer.

Output layer: The output layer is the output/conclusions of the model derived from all the computations performed. There can be single or multiple nodes in the output layer. If we have a binary classification problem the output node is 1 but in the case of multi-class classification, the output nodes can be more than 1.

How do neural networks work?

Step by Step Working of the Artificial Neural Network :



Source: Xenonstack.com

1. In the first step, **Input units are passed i.e data is passed with some weights attached to it to the hidden layer**. We can have any number of hidden layers. In the above image inputs $x_1,x_2,x_3,\ldots x_n$ is passed.

2. Each hidden layer consists of neurons. All the inputs are connected to each neuron.
3. After passing on the inputs, **all the computation is performed in the hidden layer** (Blue oval in the picture) Computation performed in hidden layers are done in two steps which are as follows :
    a. First of all, **all the inputs are multiplied by their weights**. Weight is the gradient or coefficient of each variable. It shows the strength of the particular input. After assigning the weights, a bias variable is added. **Bias** is a constant that helps the model to fit in the best way possible.

    $$\mathbf{Z_1 = W_1*In_1 + W_2*In_2 + W_3*In_3 + W_4*In_4 + W_5*In_5 + b}$$

    W_1, W_2, W_3, W_4, W5 are the weights assigned to the inputs In_1, In_2, In_3, In_4, In_5, and b is the bias.

    b. Then in the second step, the **activation function is applied to the linear equation Z1.** The activation function is a nonlinear transformation that is applied to the input before sending it to the next layer of neurons. The importance of the activation function is to inculcate nonlinearity in the model.

4. The whole process described in point 3 is performed in each hidden layer. After passing through every hidden layer, **we move to the last layer i.e our output layer which gives us the final output.**

The process explained above is known as forwarding Propagation.

5. After getting the predictions from the output layer, the **error is calculated i.e the difference between the actual and the predicted output.** If the error is large, then the steps are taken to minimize the error and for the same purpose, **Back Propagation is performed.**

Forward and backword propagation in Neural Networks:

Forward Propagation is the way to move from the Input layer (left) to the Output layer (right) in the neural network.

The process of moving from the right to left i.e backward from the Output to the Input layer is called the **Backward Propagation.**

Backward Propagation is the preferable method of adjusting or correcting the weights to reach the minimized loss function. The backpropagation algorithm, on the other hand, does this through a series of Back Propagation Algorithm Steps, which include:

- **Choosing Input and Output:** The backpropagation algorithm's first step is to choose a process input and set the desired output.

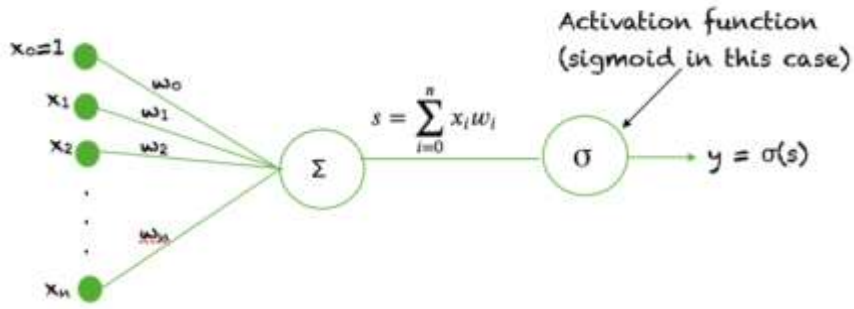
- **Setting Random Weights:** After the input and output values have been determined, random weights are assigned in order to alter the input and output values. Following that, each neuron's output is estimated via forwarding propagation, which involves:

- ✓ Input Layer
- ✓ Hidden Layer
- ✓ Output Layer
    - **Error Calculation:** This is an important step that determines the total error by determining how far and suitable the actual output is from the required output. This is accomplished by calculating the output neuron's mistakes.
    - **Error Minimization:** Based on the observations made in the previous step, the goal here is to reduce the error rate as much as possible so that accurate output can be supplied.
    - **Updating Weights and Other Parameters:** If the error rate is large, the delta rule or gradient descent are used to adjust and update parameters (weights and biases) in order to lower the rate of error.

Activation Functions:

1. Sigmoid

The sigmoid function is used as an activation function in neural networks. Just to review what is an activation function, the figure below shows the role of an activation function in one layer of a neural network. A weighted sum of inputs is passed through an activation function and this output serves as an input to the next layer.

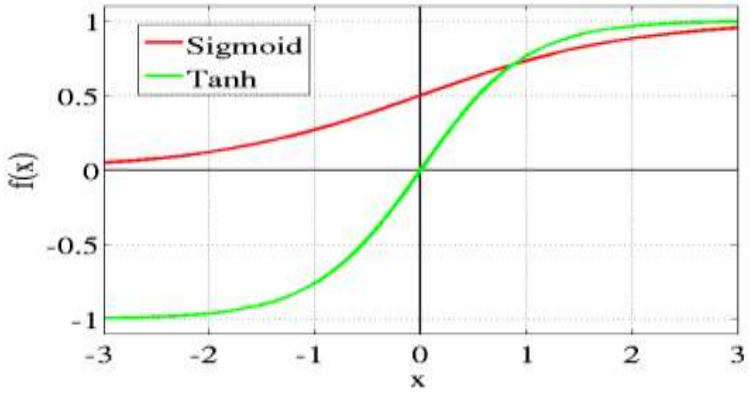


A sigmoid unit in a neural network

When the activation function for a neuron is a sigmoid function it is a guarantee that the output of this unit will always be between 0 and 1. Also, as the sigmoid is a non-linear function, the output of this unit would be a non-linear function of the weighted sum of inputs. Such a neuron that employs a sigmoid function as an activation function is termed as a sigmoid unit.

2. Tanh or hyperbolic tangent Activation Function

tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).

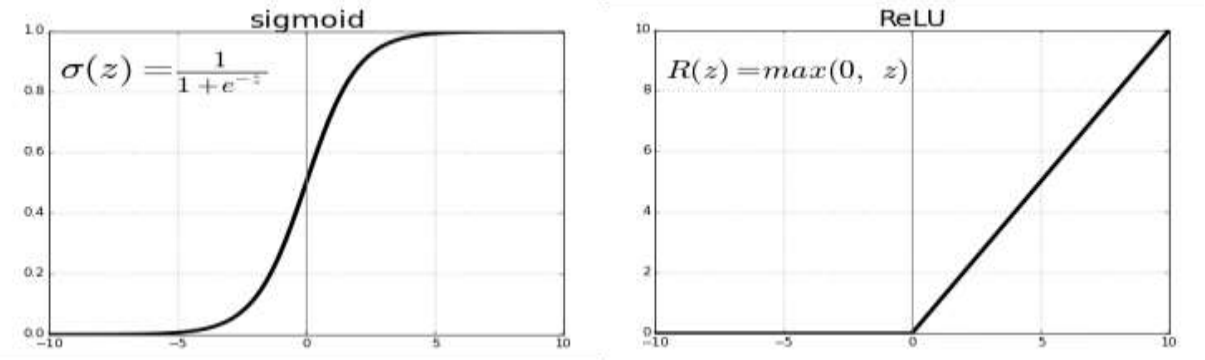


tanh v/s Logistic Sigmoid

- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.
- The function is **differentiable**.
- The function is **monotonic** while its **derivative is not monotonic**.
- The tanh function is mainly used classification between two classes.

3. ReLU (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.



ReLU v/s Logistic Sigmoid

As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

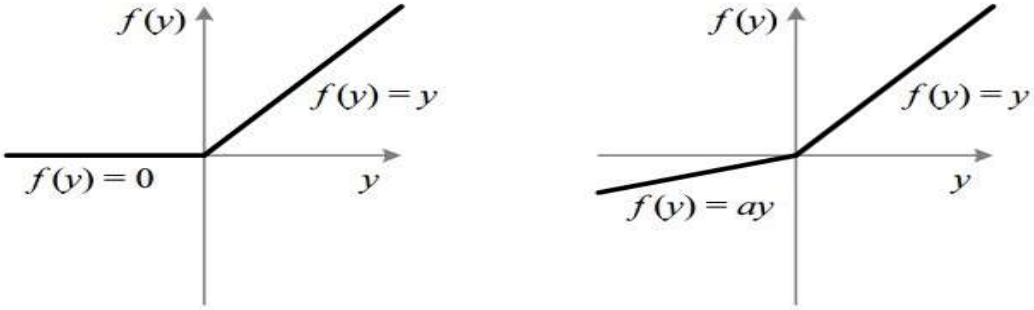
Range: [0 to infinity)

The function and its derivative **both are monotonic**.

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

4. Leaky ReLU(Rectified Linear Unit)

It is an attempt to solve the dying ReLU problem



ReLU v/s Leaky ReLU

- The leak helps to increase the range of the ReLU function. Usually, the value of **a** is 0.01 or so.
- When **a is not 0.01** then it is called **Randomized ReLU**.
- Therefore, the **range** of the Leaky ReLU is (-infinity to infinity).
- Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives also monotonic in nature.

Cost Function:

A Cost Function is used to measure just how wrong the model is in finding a relation between the input and output. It tells you how badly your model is behaving/predicting

Consider a robot trained to stack boxes in a factory. The robot might have to consider certain changeable parameters, called Variables, which influence how it performs. Let's say the robot comes across an obstacle, like a rock. The robot might bump into the rock and realize that it is not the correct action.

How to measure loss?

The formula for the loss is fairly straightforward. It is just the squared difference between the expected value and the predicted value.

L = (y_i - \hat y_i)^2$L=(yi−y^i)2$

Suppose you have a model that helps you predict the price of oil per gallon. If the actual price of the house is $2.89 and the model predicts $3.07, you can calculate the error.

L = (2.89 - 3.07)^2 = 0.032$L=(2.89−3.07)2=0.032$

The cost is again calculated as the average overall losses for the individual examples.

C = \frac{1}{N} \sum_{i=1}^N (y_i - \hat y_i)^2$C=N1i=1∑N(yi−y^i)2$

How to reduce loss?

- Hyperparameters are the configuration settings used to tune how the model is trained.

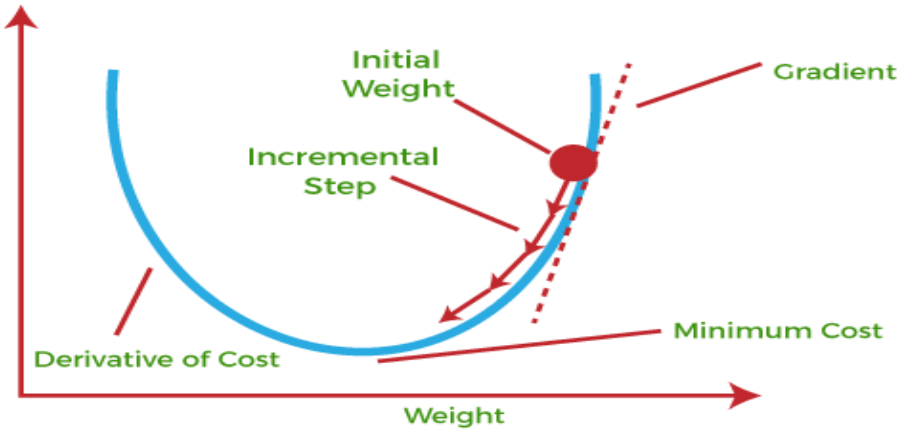
- The derivative of (y - y')2 with respect to the weights and biases tells us how loss changes for a given example
    - Simple to compute and convex
- So we repeatedly take small steps in the direction that minimizes loss
    - We call these **Gradient Steps** (But they're really negative Gradient Steps)
    - This strategy is called **Gradient Descent**

Gradient Descent or Steepest Descent:

Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

- If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.
- Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the **local maximum** of that function.



This entire procedure is known as Gradient Ascent, which is also known as steepest descent. ***The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.*** To achieve this goal, it performs two steps iteratively:

- Calculates the first-order derivative of the function to compute the gradient or slope of that function.
- Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate. It is a tuning parameter in the optimization process which helps to decide the length of the steps.