

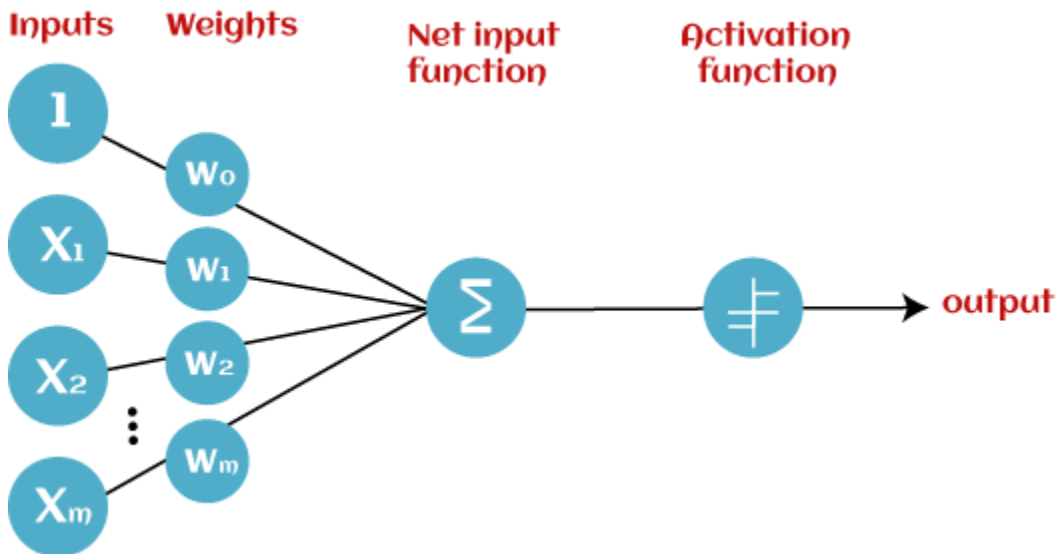
UNIT-5 (8Marks and 16Marks)

1. Explain in detail about Perceptrons and its types?

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence. Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., input values, weights and Bias, net sum, and an activation function.

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



- **Input Nodes or Input Layer:**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Weight and Bias:**

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sin function
- Step function, and
- Sigmoid function

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

•Single Layer Perceptron Model:

➤ This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

➤ In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with in constantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

➤ If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

•Multi-Layered Perceptron Model:

A multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- Forward Stage: Activation functions start from the input layer in the forward stage and terminate on the output layer.
- Backward Stage :In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

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

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Perceptron Function

Perceptron function "f(x)" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

$$f(x)=1; \text{if } w \cdot x + b > 0 \quad ; \quad \text{otherwise, } f(x)=0$$

'w' represents real-valued weights vector

'b' represents the bias

'x' represents a vector of input x values.

Characteristics of Perceptron

The perceptron model has the following characteristics.

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied within put features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

2. Explain about ReLu, Hyperparameter tuning, Normalization, Regularization, Dropout?

i) Hyperparameter Tuning in Deep Learning

The first hyperparameter to tune is the number of neurons in each hidden layer. In this case, the number of neurons in every layer is set to be the same. It also can be made different. The number of neurons should be adjusted to the solution complexity. The task with a more complex level to predict needs more neurons. The number of neurons range is set to be from 10 to 100.

- An activation function is a parameter in each layer. Input data are fed to the input layer, followed by hidden layers, and the final output layer. The output layer contains the output value. The input values moving from a layer to another layer keep changing according to the activation function.
- The activation function decides how to compute the input values of a layer into output values. The output values of a layer are then passed to the next layer as input values again. The next layer then computes the values into output values for another layer again. There are 9 activation functions to tune in to this demonstration. Each activation function has its own formula (and graph) to compute the input values. It will not be discussed in this article.
- The layers of a neural network are compiled and an optimizer is assigned. The optimizer is responsible to change the learning rate and weights of neurons in the neural network to reach the minimum loss function. Optimizer is very important to achieve the possible highest accuracy or minimum loss. There are 7 optimizers to choose from. Each has a different concept behind it.
- One of the hyperparameters in the optimizer is the learning rate. We will also tune the learning rate. Learning rate controls the step size for a model to reach the minimum loss function. A higher learning rate makes the model learn faster, but it may miss the minimum loss function and only reach the surrounding of it. A lower learning rate gives a better chance to find a minimum loss function. As a tradeoff lower learning rate needs higher epochs, or more time and memory capacity resources.

ii) ReLU- Rectified Linear Unit

A Rectified Linear Unit is a form of activation function used commonly in deep learning models. In essence, the function returns 0 if it receives a negative input, and if it receives a positive value, the function will return back the same positive value. The function is understood as:

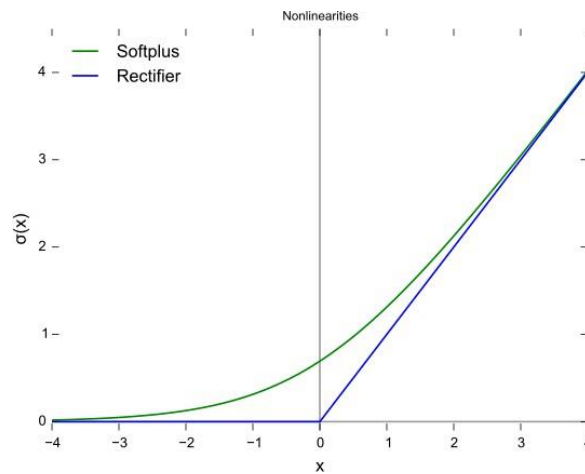
$$f(x) = \max(0, x)$$

The rectified linear unit, or ReLU, allows for the deep learning model to account for non-linearities and specific interaction effects. The image above displays the graphic representation of the ReLU

function. Note that the values for any negative X input result in an output of 0, and only once positive values are entered does the function begin to slope upward.

How does a Rectified Linear Unit work?

To understand how a ReLU works, it is important to understand the effects it has on variable interaction effects. An interaction effect is when a variable affects a prediction depending on the value of associated variables. For example, comparing IQ scores of two different schools may have interaction effects of IQ and age. The IQ of a student in high school is better than the IQ of an elementary school student, as age and IQ interact with each other regardless of the school. This is known as an interaction effect and ReLUs can be applied to minimize interaction effects. For example, if $A=1$ and $B=2$, and both have the respective associated weights of 2 and 3, the function would be, $f(2A+3B)$. If A increases, the output will increase as well. However, if B is a large negative value, the output will be 0.



Benefits of using the ReLU function

Its simplicity leads it to be a relatively cheap function to compute. As there is no complicated math, the model can be trained and run in a relatively short time. Similarly, it converges faster, meaning the slope doesn't plateau as the value for X gets larger. This vanishing gradient problem is avoided in ReLU, unlike alternative functions such as sigmoid or tanh. Lastly, ReLU is sparsely activated because for all negative inputs, the output is zero. Sparsity is the principle that specific functions only are activated in concise situations. This is a desirable feature for modern neural networks, as in a sparse network it is more likely that neurons are appropriately processing valuable parts of a problem. For example, a model that is processing images of fish may contain a neuron that is specialized to identify fish eyes. That specific neuron would not be activated if the model was processing images of airplanes instead. This specified use of neuron functions accounts for network sparsity.

iii) Regularization:

- Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it. Sometimes the machine learning

model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

- This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model. It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "In regularization technique, we reduce the magnitude of the features by keeping the same number of features."

How does Regularization Work?

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + b$$

In the above equation, Y represents the value to be predicted X1, X2, ...Xn are the features for Y. $\beta_0, \beta_1, \dots, \beta_n$ are the weights or magnitude attached to the features, respectively. Here β_0 represents the bias of the model, and b represents the intercept. Linear regression models try to optimize the β_0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares**.

Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

- **Ridge Regression**
- **Lasso Regression**

Ridge Regression

- Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.
- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
- In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called **Ridge Regression penalty**. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.
- The equation for the cost function in ridge regression will be:

$$\sum_{i=1}^M (y_i - y'_i)^2 = \sum_{i=1}^M \left(y_i - \sum_{j=0}^n \beta_j * x_{ij} \right)^2 + \lambda \sum_{j=0}^n \beta_j^2$$

- In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
- As we can see from the above equation, if the values of λ **tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ , the model will resemble the linear regression model.
- A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.
- It helps to solve the problems if we have more parameters than samples.

Lasso Regression:

- Lasso regression is another regularization technique to reduce the complexity of the model. It stands for **Least Absolute and Selection Operator.**
- It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
- Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
- It is also called as **L1 regularization.** The equation for the cost function of Lasso regression will be:

$$\sum_{i=1}^M (y_i - y'_i)^2 = \sum_{i=1}^M \left(y_i - \sum_{j=0}^n \beta_j * x_{ij} \right)^2 + \lambda \sum_{j=0}^n |\beta_j|$$

- Some of the features in this technique are completely neglected for model evaluation.
- Hence, the Lasso regression can help us to reduce the overfitting in the model as well as the feature selection.

Key Difference between Ridge Regression and Lasso Regression

- **Ridge regression** is mostly used to reduce the overfitting in the model, and it includes all the features present in the model. It reduces the complexity of the model by shrinking the coefficients.
- **Lasso regression** helpstoreducetheoverfittinginthemodelaswellasfeatureselection.

3. Explain about Error Backpropagation?

Backpropagation, or backward propagation of errors, is an algorithm that is designed to test for errors working back from output nodes to input nodes. It is an important mathematical tool for improving the accuracy of predictions in data mining and machine learning. Essentially, backpropagation is an algorithm used to calculate derivatives quickly.

There are two leading types of backpropagation networks:

1. **Static backpropagation:**

Static backpropagation is a network developed to map static inputs for static outputs. Static backpropagation networks can solve static classification problems, such as optical character recognition (OCR).

2. **Recurrent backpropagation.**

The recurrent backpropagation network is used for fixed-point learning. Recurrent backpropagation activation feeds forward until it reaches a fixed value.

What is a backpropagation algorithm in a neural network?

Artificial neural networks use backpropagation as a learning algorithm to compute a gradient descent with respect to weight values for the various inputs. By comparing desired output to achieved system outputs, the systems are tuned by adjusting connection weights to narrow the difference between the two as much as possible. The algorithm gets its name because the weights are updated backward, from output to input.

Advantages

- It does not have any parameters to tune except for the number of inputs.
- It is highly adaptable and efficient and does not require any prior knowledge about the network.
- It is a standard process that usually works well.
- It is user-friendly, fast and easy to program.
- Users do not need to learn any special functions.

DISADVANTAGES

It prefers a matrix-based approach over a mini-batch approach.

- Data mining is sensitive to noise and irregularities.
- Performance is highly dependent on input data.
- Training is time- and resource-intensive.

Features of Backpropagation:

1. It is the gradient descent method as used in the case of simple perceptron network with the differentiable unit.
2. It is different from other networks in respect to the process by which the weights are calculated during the learning period of the network.
3. Training is done in the three stages:
 - The feed-forward of input training pattern
 - The calculation and backpropagation of the error
 - Updation of the weight

Working of Backpropagation:

Neural networks use supervised learning to generate output vectors from input vectors that the network operates on. It compares generated output to the desired output and generates an error report if the result does not match the generated output vector. Then it adjusts the weights according to the bug report to get your desired output.

Backpropagation Algorithm:

Step1: Inputs X, arrive through the preconnected path.

Step2: The input is modeled using true weights W. Weights are usually chosen randomly.

Step3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

Step4: Calculate the error in the outputs

$$\text{Backpropagation Error} = \text{Actual Output} - \text{Desired Output}$$

Step5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.

Step6: Repeat the process until the desired output is achieved.

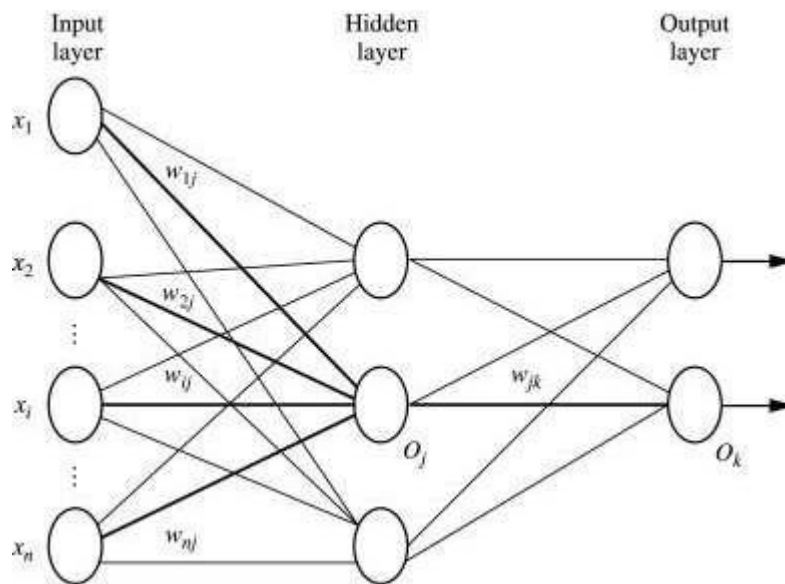


Fig: Error backpropagation

- x = inputs training vector $x = (x_1, x_2, \dots, x_n)$.
- t = target vector $t = (t_1, t_2, \dots, t_n)$.
- δ_k = error at output unit.
- δ_j = error at hidden layer.
- α = learning rate.
- V_{0j} = bias of hidden unit j .

Training Algorithm:

Step1: Initialize weight to small random values.

Step2: While the steps stopping condition is to be false do step 3 to 10.

Step3: For each training pair do step 4 to 9(Feed-Forward).

Step4: Each input unit receives the signal unit and transmits the signal x_i signal to all the units.

Step5: Each hidden unit Z_j ($j=1$ to a) sums its weighted input signal to calculate its net input

$$z_{inj} = v_{0j} + \sum x_i v_{ij} \quad (i=1 \text{ to } n)$$

Applying activation function $z_j = f(z_{inj})$ and sends this signals to all units in the layer about i.e output units

For each output l -unit $y_k = (k=1$ to $m)$ sums its weighted input signals.

$$y_{ink} = w_{0k} + \sum z_i w_{jk} \quad (j=1 \text{ to } a)$$

and applies its activation function to calculate the output signals.

$$y_k = f(y_{ink})$$

Backpropagation Error:

Step6: Each output unit K ($k=1$ to n) receives a target pattern corresponding to an input pattern then error is calculated as:

$$\delta_k = (t_k - y_k) + y_{ink}$$

Step7: Each hidden unit Z_j ($j=1$ to a) sums its input from all units in the layer above

$$\delta_{inj} = \sum \delta_j w_{jk}$$

The error information term is calculated as :

$$\delta_j = \delta_{inj} + z_{inj}$$

Updation of weight and bias:

Step8: Each output unit y_k ($k=1$ to m) updates its bias and weight ($j=1$ to a). The weight correction term is given by :

$$\Delta w_{jk} = \alpha \delta_k z_j$$

and the bias correction term is given by $\Delta w_k = \alpha$

δ_k . therefore $w_{jk(\text{new})} = w_{jk(\text{old})} + \Delta w_{jk}$

$$w_{0k(\text{new})} = w_{0k(\text{old})} + \Delta w_{0k}$$

for each hidden unit z_j ($j=1$ to a) update its bias and weights ($i=0$ to n) the weight connection term

$$\Delta v_{ij} = \alpha \delta_j x_i$$

And the bias connection on term

$$\Delta v_{0j} = \alpha$$

δ_j Therefore $v_{ij(\text{new})} = v_{ij(\text{old})} + \Delta v_{ij}$

$$v_{0j(\text{new})} = v_{0j(\text{old})} + \Delta v_{0j}$$

Step9: Test the stopping condition. The stopping condition can be the minimization of error, number of epochs.

4. Explain in detail about activation functions?

ACTIVATION FUNCTIONS

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.

✓ 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.

✓ 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.

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,

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

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

$$\text{Final output : } z(2) = W * X + b$$

which is again a linear function

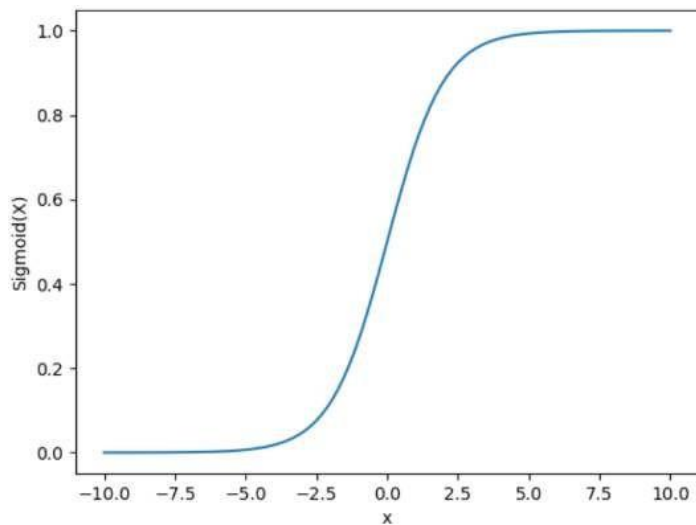
Variants of Activation Function Linear Function

- **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:** $-\infty$ to $+\infty$
- **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:** $A = 1/(1+e^{-x})$



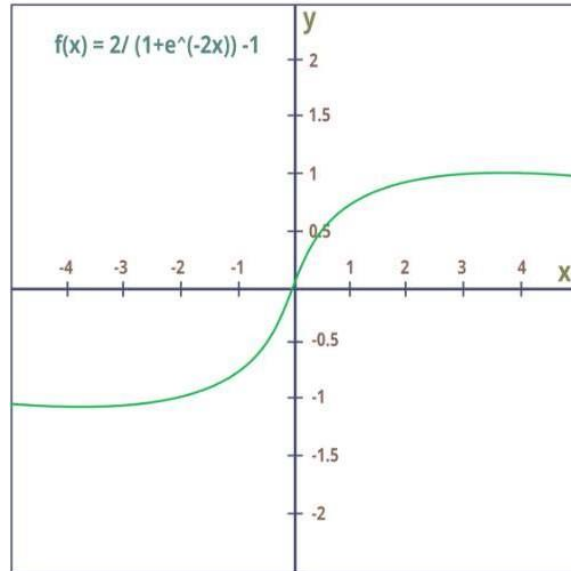
- **Nature:** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **ValueRange:** 0 to 1
- **Uses:** Usually used in output layer of a binary classification, where result is either 0 or 1, as a 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:-**

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

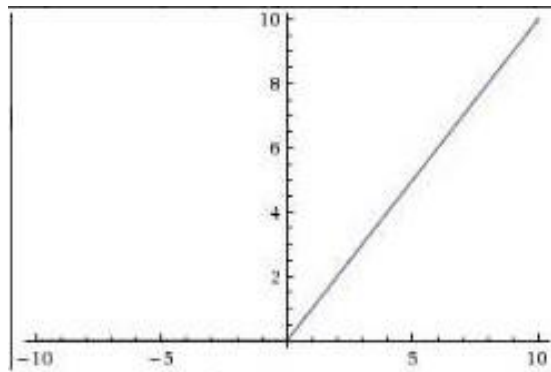
- **Value Range:-** -1 to +1
- **Nature:-** non-linear
- **Uses:-** Usually used in hidden layers of a neural network as its values lies between **-1 to 1** hence the mean for the hidden layer comes out to be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes 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:-** $[0, \infty)$
- **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.

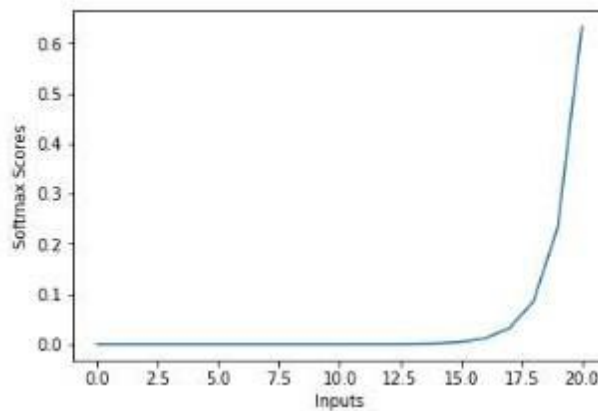


Softmax Function

The softmax function is also a type of sigmoid function but is used 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.



5. Explain in detail about gradient descent optimization?

Gradient descent optimization

✓ Gradient descent is an optimization algorithm in gadget mastering used to limit a feature with the aid of iteratively moving towards the minimal fee of characteristic.

✓ We essentially use this algorithm when we have to locate the least possible values which could fulfill a given free function. In gadget getting to know, greater regularly that not we try to limit loss features (like mean squared error). By minimizing the loss characteristic, we will improve our model and gradient descent is one of the most popular algorithms used for this cause.

✓ The graph above shows how exactly a gradient descent set of rules works.

✓ We first take a factor in the value function and begin shifting in steps in the direction of the minimum factor. The size of the step, or how quickly we ought to converge to the minimum factor is defined by learning rate.

✓ We can cover more location with better learning rate but at the risk of overshooting the minima. On the opposite hand, small steps/ smaller gaining knowledge of charges will eat a number of times to attain the lowest point.

✓ Now, the direction where in algorithm has to transport is also important. We calculate this by way of using derivatives. You need to be familiar with derivatives from calculus. A spinoff is largely calculated because the slope of the graph at any specific factor. We get that with the aid of finding the tangent line to the graph at that point. The extra steep the tangent, would suggest that more steps would be needed to reach minimum point; much less steep might suggest lesser steps are required to reach the minimum factor.

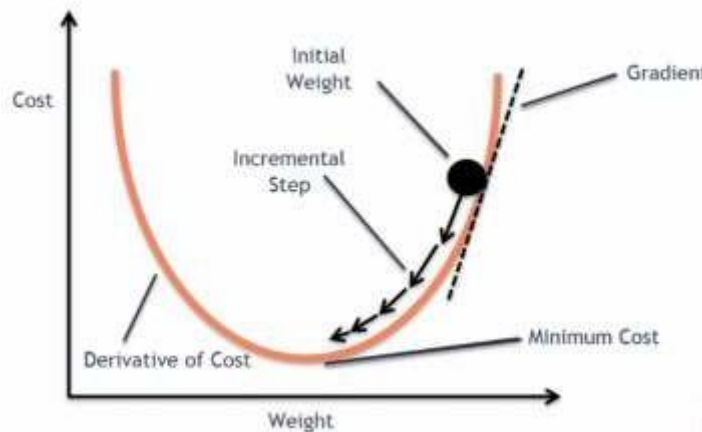


Fig: Gradient descent optimization

Stochastic gradient descent

The word stochastic means a system or a process that is linked with a random probability. Hence, in stochastic gradient descent, a few samples are selected randomly instead of the whole data set for each iteration.

- ✓ Stochastic gradient descent is a type of gradient descent that runs one training example per iteration. It processes a training epoch for each example within a dataset and updates each training example's parameters one at a time.
- ✓ As it requires only one training example at a time, hence it is easier to store in allocated memory. However, it shows some computational efficiency losses in comparison to batch gradient systems as it shows frequent updates that require more detail and speed.

- ✓ Further, due to frequent updates, it is also treated as a noisy gradient. However, sometimes it can be helpful in finding the global minimum and also escaping the local minimum.

Advantages of stochastic gradient descent:

It is easier to allocate in desired memory.

It is relatively fast to compute than batch gradient descent.

It is more efficient for large dataset.

Disadvantages of stochastic gradient descent:

SGD require a number of hyperparameters such as the regularization parameter and the number of iterations.

SGD is sensitive to feature scaling.