

---

# Artificial Neural Network

---

# Neural Network

**Introduction to Neural Networks** :The Biological Neuron, The Perceptron, Multilayer Feed-Forward Networks ,

**Training Neural Networks** :Backpropagation and Forward propagation

**Activation Functions** :Linear ,Sigmoid, Tannh, Hard Tanh, Softmax, Rectified Linear,

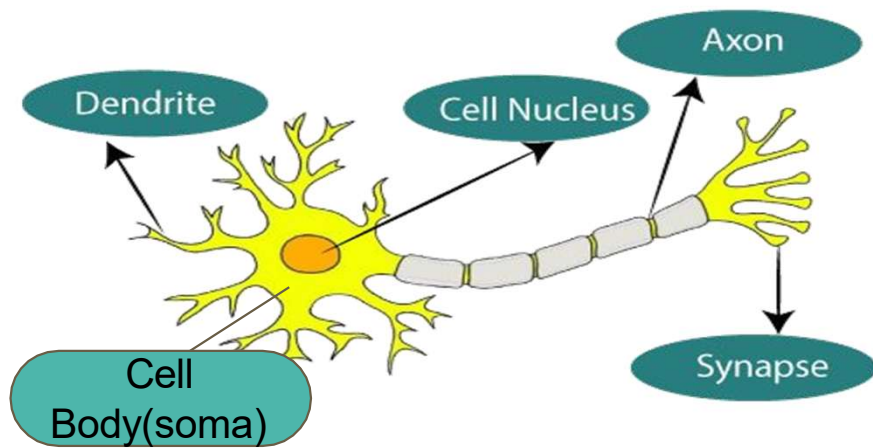
**Loss Functions** :Loss Function Notation , Loss Functions for Regression , Loss Functions for Classification, Loss Functions for Reconstruction,

**Hyperparameters** : Learning Rate, Regularization, Momentum, Sparsity,

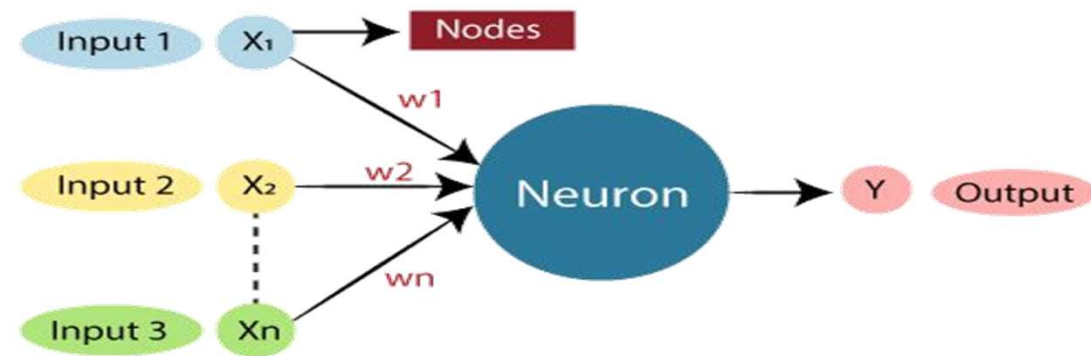
**Deep Feedforward Networks** – Example of Ex OR, Hidden Units, cost functions, error backpropagation, Gradient-Based Learning, Implementing Gradient Descent, vanishing and Exploding gradient descent, Sentiment Analysis, Deep Learning with Pytorch, Jupyter, colab

# Key Terms

- **Neuron:** A building block of ANN. It is responsible for accepting input data, performing calculations, and producing output.
- **Input data:** Information or data provided to the neurons.
- **Artificial Neural Network(ANN):** A computational system inspired by the way biological neural networks in the human brain process information.
- **Deep Neural Network:** An ANN with many layers placed between the input layer and the output layer.
- **Weights:** The strength of the connection between two neurons. Weights determine what impact the input will have on the output.
- **Bias:** An additional parameter used along with the sum of the product of weights and inputs to produce an output.



**Biological Neural Network**



**Artificial Neural Network**

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

# Biological Neural Networks

- The biological neuron is a nerve cell that provides the fundamental functional unit for the nervous systems of all animals.
  - Neurons exist to communicate with one another, and pass electro-chemical impulses across synapses, from one cell to the next, as long as the impulse is strong enough to activate the release of chemicals across a synaptic cleft. The strength of the impulse must surpass a minimum threshold or chemicals will not be released.
  - The neuron is made up of a nerve cell consisting of a soma (cell body) that has many dendrites but only one axon. The single axon can branch hundreds of times.
-

# Biological Neural Networks

- Dendrites are thin structures that arise from the main cell body. Dendrites allow the cell to receive signals from connected neighboring neurons and each dendrite is able to perform multiplication by that dendrite's weight value. Here multiplication means an increase or decrease in the ratio of synaptic neurotransmitters to signal chemicals introduced into the dendrite.
  - Axons are nerve fibers with a special cellular extension that comes from the cell body. Axons are the single, long fibers extending from the main soma. They stretch out longer distances than dendrites and measure generally 1 centimeter in length
  - Synapses are the connecting junction between axon and dendrites. The majority of synapses send signals from the axon of a neuron to the dendrite of another neuron.
-

# Artificial Neural Networks

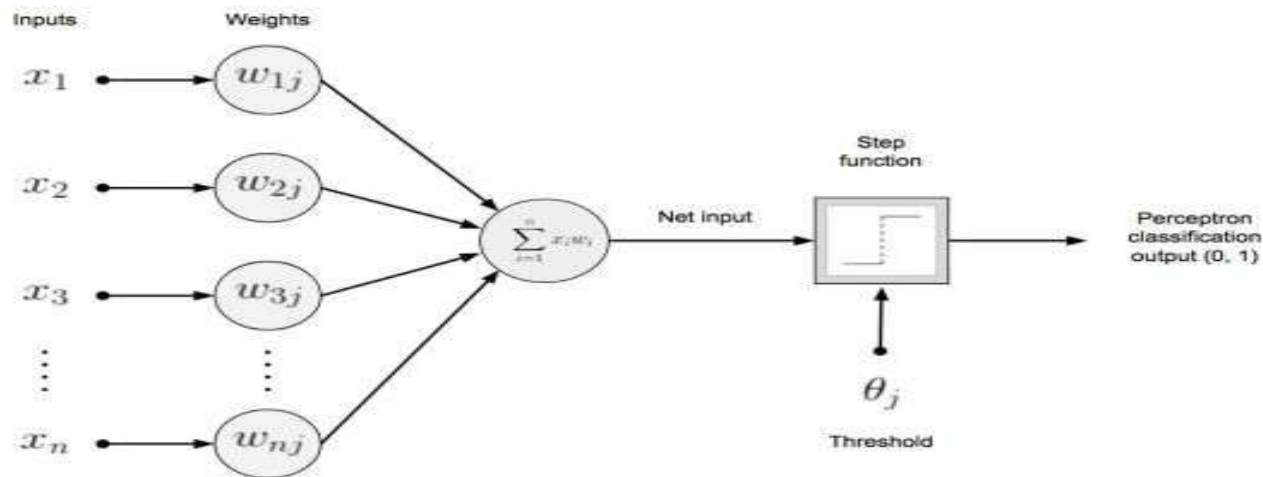
The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes

---

# Perceptron

- Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.
- The perceptron is a linear-model binary classifier with a simple input–output relationship which shows we’re summing n number of inputs times their associated weights and then sending this “net input” to a step function with a defined threshold.



To produce the net input to the activation function we take the dot product of the input and the connection weights.



# Perceptron

The output of the step function (activation function) is the output for the perceptron and gives us a classification of the input values. If the bias value is negative, it forces the learned weights sum to be a much greater value to get a 1 classification output.

The bias term in this capacity moves the decision boundary around for the model. Input values do not affect the bias term, but the bias term is learned through the perceptron learning algorithm.

**Step 1:** Multiply all input values with corresponding weight values and then add to calculate the weighted sum. The following is the mathematical expression of it:

$$\sum w_i * x_i = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + \dots x_4 * w_4$$

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

**Step 2:** An activation function is applied with the above-mentioned weighted sum giving us an output either in binary form or a continuous value as follows:  $Y = f(\sum w_i * x_i + b)$

# Types of Perceptron

## Types of Perceptron models

- **Single Layer Perceptron model:**

- One of the easiest ANN(Artificial Neural Networks) types consists of a feed-forward network and includes a threshold transfer inside the 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.

**Forward Stage:** From the input layer in the on stage, activation functions begin and terminate on the output layer.

**Backward Stage:** In the backward stage, weight and bias values are modified per the model's requirement. The backstage removed the error between the actual output and demands originating backward on the output layer. A multilayer perceptron model has a greater processing power and can process linear and non-linear patterns. Further, it also implements logic gates such as AND, OR, XOR, XNOR, and NOR.

# Advantage and Disadvantage of Perceptron

## Advantages:

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

## Disadvantages:

- In multi-layered perceptron model, computations are time-consuming and complex.
  - It is tough to predict how much the dependent variable affects each independent variable.
  - The model functioning depends on the quality of training.
-

# Characteristics of the Perceptron Model

1. It is a machine learning algorithm that uses supervised learning of binary classifiers.
  2. In Perceptron, the weight coefficient is automatically learned.
  3. Initially, weights are multiplied with input features, and then the decision is made whether the neuron is fired or not.
  4. The activation function applies a step rule to check whether the function is more significant 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.
-

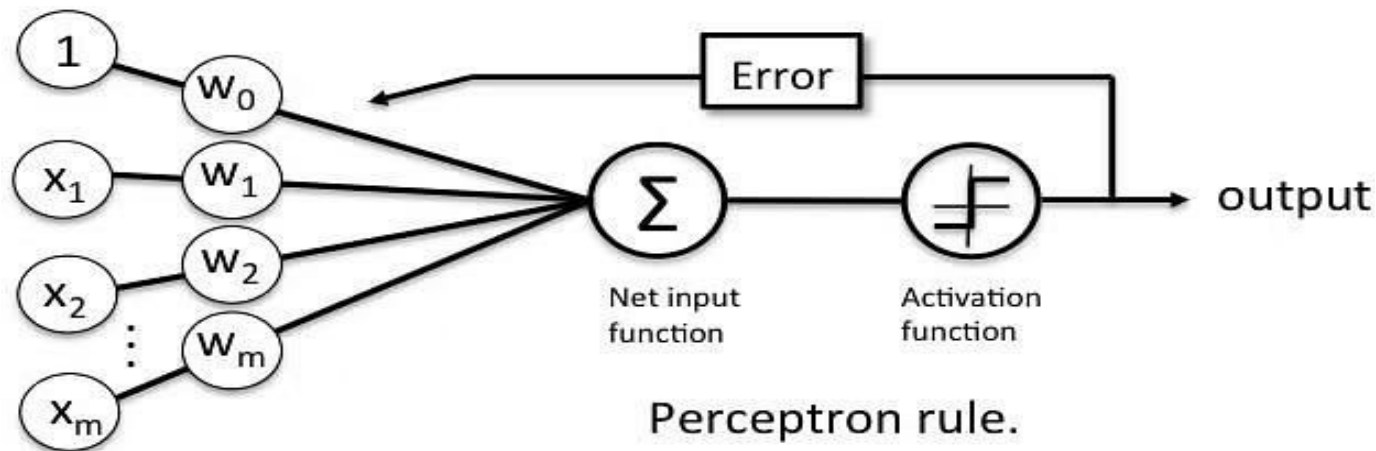
# Limitations of the Perceptron Model

1. The output of a perceptron can only be a binary number (0 or 1) due to the hard-edge transfer function.
  2. It can only be used to classify the linearly separable sets of input vectors. If the input vectors are non-linear, it is not easy to classify them correctly
-

# Perceptron Learning Rule

Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.

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



# Perceptron Function

Perceptron is a function that maps its input “x,” which is multiplied with the learned weight coefficient; an output value “f(x)” is generated

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

- “w” = vector of real-valued weights
- “b” = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
- “x” = vector of input x values

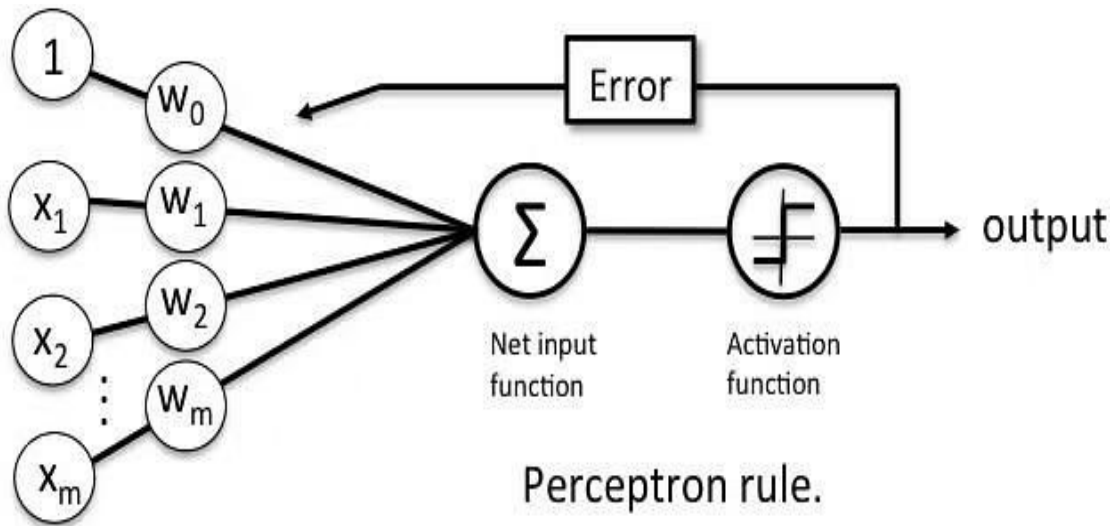
$$\sum_{i=1}^m w_i x_i$$

- “m” = number of inputs to the Perceptron

The output can be represented as “1” or “0.” It can also be represented as “1” or “-1” depending on which activation function is used.

# Inputs of Perceptron

A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result. The image below shows a Perceptron with a Boolean output.



A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False. The summation function " $\Sigma$ " multiplies all inputs of "x" by weights "w" and then adds them up as follows:

$$w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$



# Multilayer Feed Forward Network

The multilayer feed-forward network is a neural network with an input layer, one or more hidden layers, and an output layer. Each layer has one or more artificial neurons. These artificial neurons are similar to their perceptron precursor yet have a different activation function depending on the layer's specific purpose in the network. We'll look more closely at the layer types in multilayer perceptrons later in the chapter. For now, let's look more closely at this evolved artificial neuron that emerged from the limitations of the single-layer perceptron.

## Evolution of the artificial neuron

The artificial neuron of the multilayer perceptron is similar to its predecessor, the perceptron, but it adds flexibility in the type of activation layer we can use. Figure shows an updated diagram for the artificial neuron that is based on the perceptron.

The net input to the activation function is still the dot product of the weights and input features yet the flexible activation function allows us to create different types out of output values

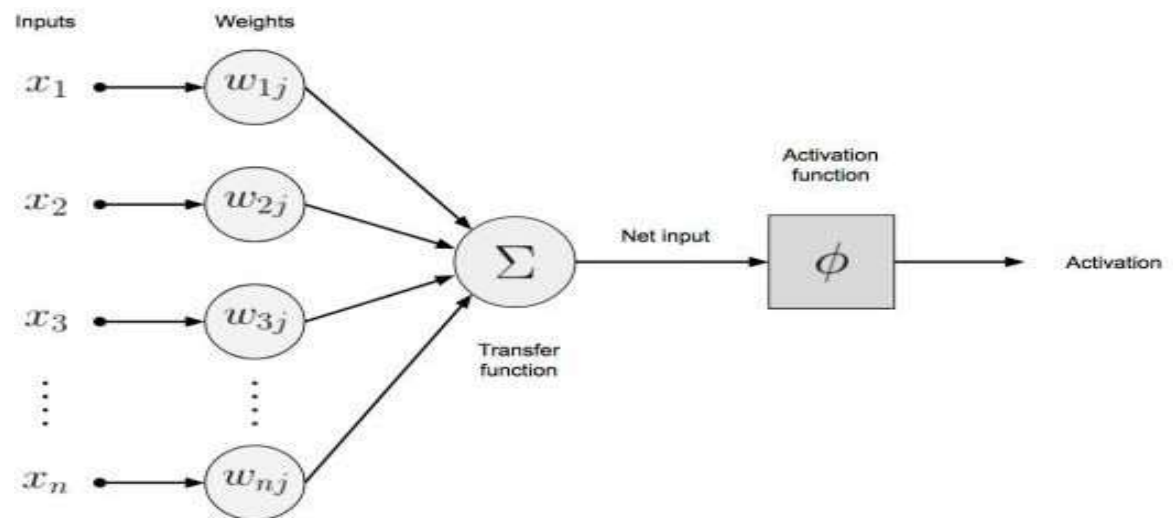


Figure 2-5. Artificial neuron for a multilayer perceptron

# Training Neural Network

**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**.

---

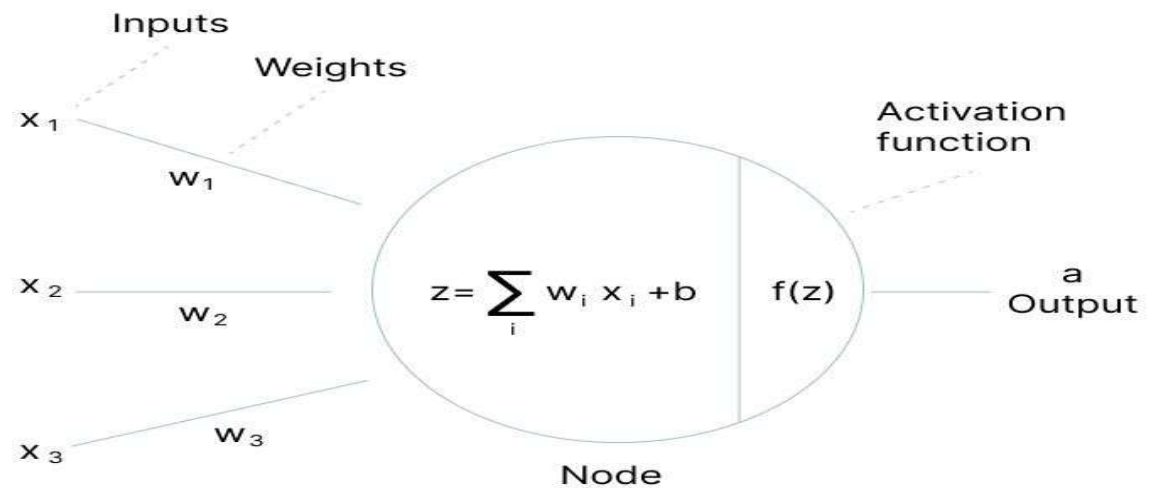
# Activation Function

- We use activation functions to propagate the output of one layer's nodes forward to the next layer (up to and including the output layer). Activation functions are a scalar-to-scalar function, yielding the neuron's activation. We use activation functions for hidden neurons in a neural network to introduce nonlinearity into the network's modeling capabilities.
  - Many activation functions belong to a logistic class of transforms that (when graphed) resemble an S.
-

# Activation Function

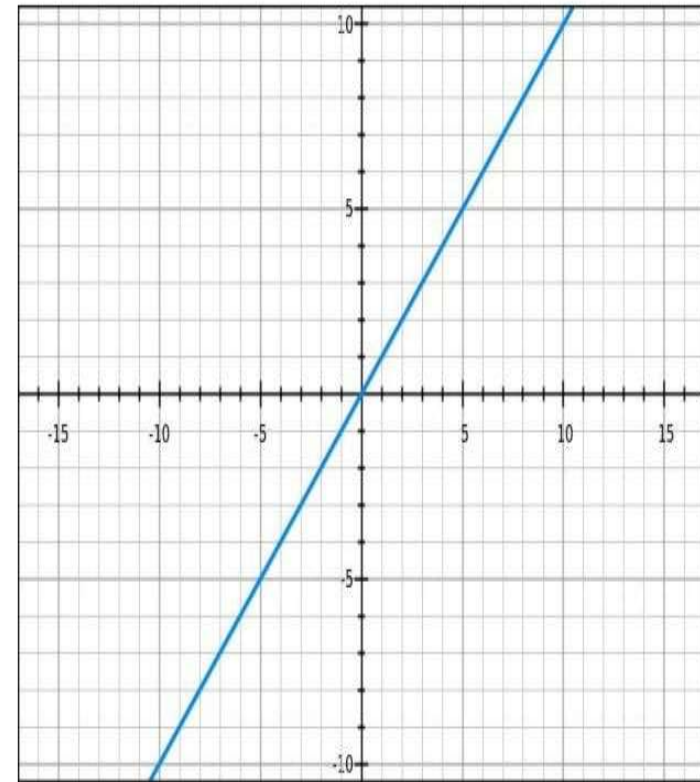
- We use activation functions for hidden neurons in a neural network to introduce nonlinearity into the network's modeling capabilities.

- Linear
- Step
- Sigmoid
- Hyperbolic
- Rectified Linear
- Softmax



# a) Linear Activation Function

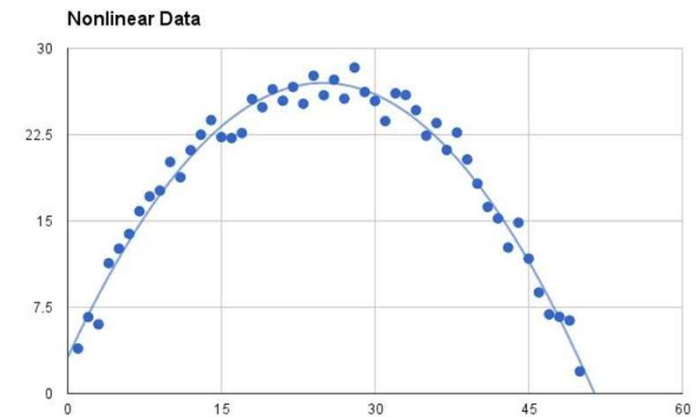
- A linear transform is basically the identity function, and  **$f(\mathbf{x}) = \mathbf{x}$** , where the dependent variable has a direct, proportional relationship with the independent variable. In practical terms, it means the function passes the signal through unchanged.
- **Range : -inf to +inf**
- **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.



## b) Non Linear Activation Function

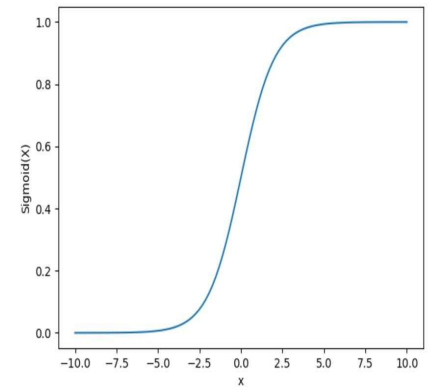
- The Nonlinear Activation Functions are the most used activation functions. Nonlinearity helps to makes the graph look something like this
- It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.
- . The Nonlinear Activation Functions are mainly divided on the basis of their **range or curves**-

1. Sigmoid or Logistic Activation Function
2. Tanh or hyperbolic tangent Activation Function
3. ReLU (Rectified Linear Unit) Activation Function
4. Leaky ReLU



# 1. Sigmoid Activation Function

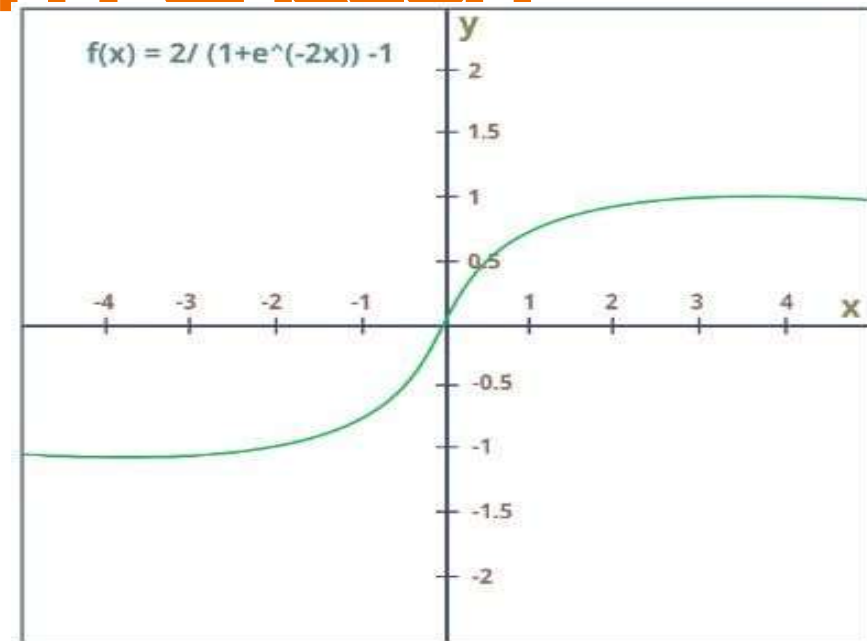
- It is a function which is plotted as 'S' shaped graph.
- **Equation** :  $A = 1/(1 + e^{-x})$
- **Nature** : Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range** : 0 to 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.



## 2. Tanh Activation 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

- **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.

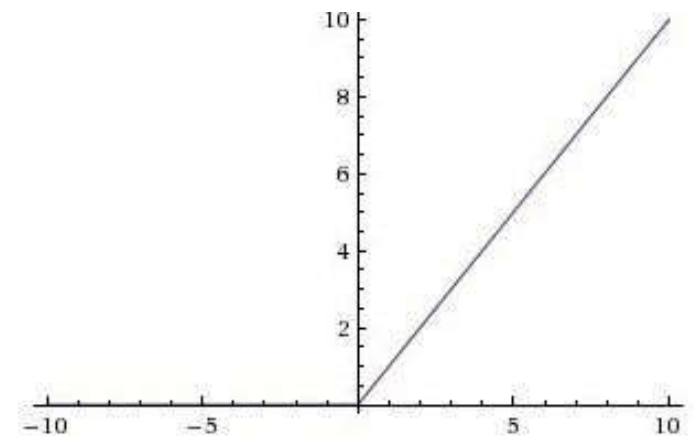


$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



### 3. ReLU(Rectified linear unit) Activation Function

- It is the most widely used activation function. Implemented in *hidden layers* of Neural network.
- **Equation** :-  $A(x) = \max(0, x)$ .
- **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.

# 3. ReLU Activation Function

## c) ReLU Activation Functions

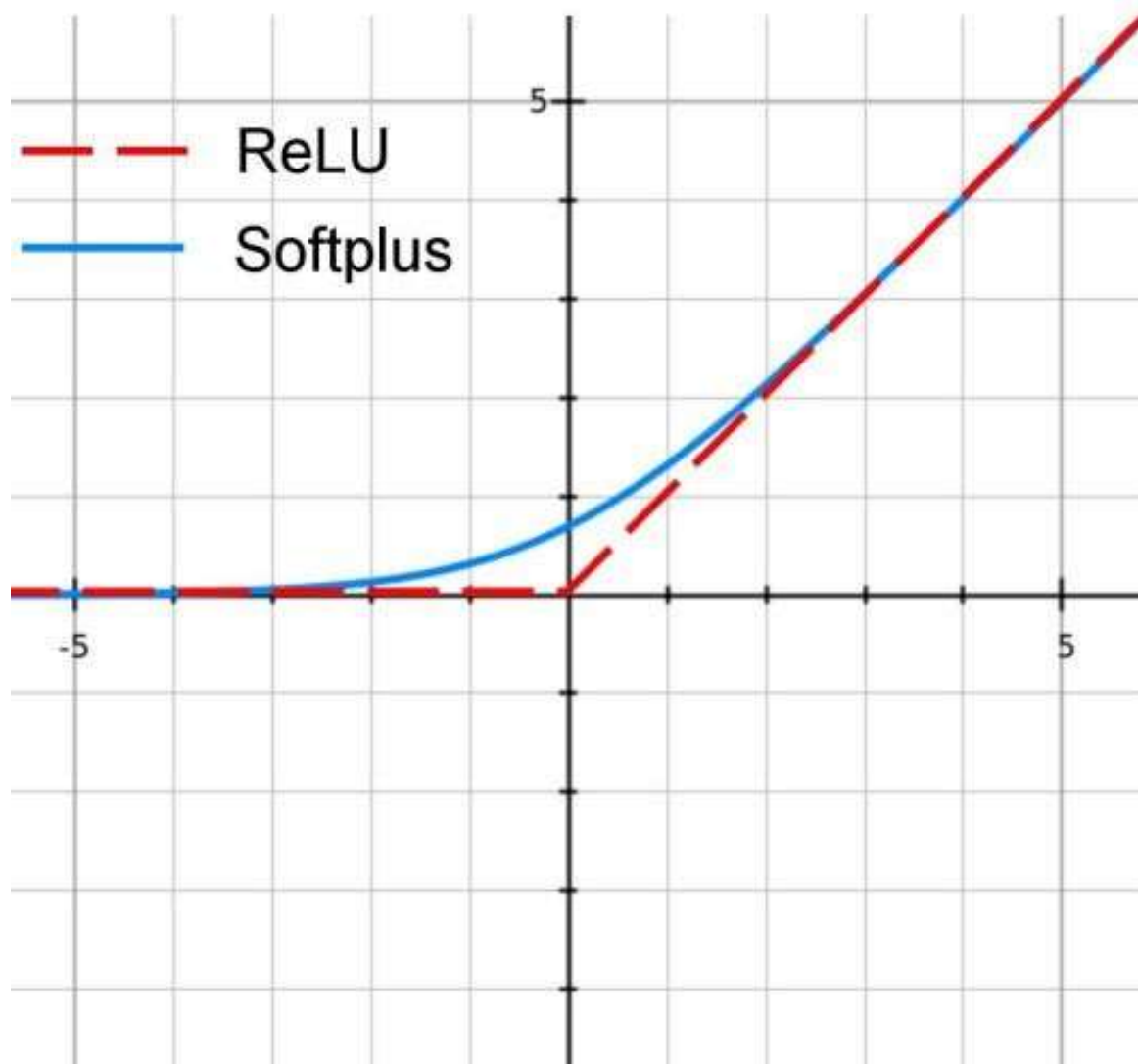
The formula is deceptively simple:  $\max(0, z)$ . Rectified Linear Units, it's not linear and provides the same benefits as Sigmoid but with better performance.

### (i) Leaky Relu

Leaky Relu is a variant of ReLU. Instead of being 0 when  $z < 0$ , a leaky ReLU allows a small, non-zero, constant gradient  $\alpha$  (normally,  $\alpha = 0.01$ ). However, the consistency of the benefit across tasks is presently unclear. Leaky ReLUs attempt to fix the “dying ReLU” problem.

### (ii) Parametric Relu

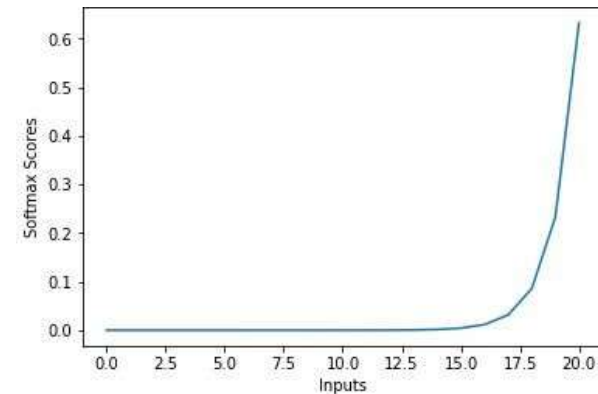
PReLU gives the neurons the ability to choose what slope is best in the negative region. They can become ReLU or leaky ReLU with certain values of  $\alpha$ .



# 4) Softmax Activation Function

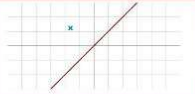
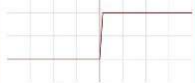

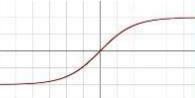
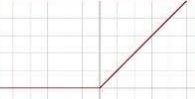

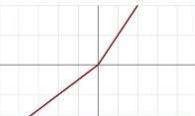

The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems. **Nature :-** non-linear

- **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.



# Note: Activation Function

- 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.
-

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

# Loss Function

## **Loss Functions :**

Loss Function Notation ,

Loss Functions for Regression ,

Loss Functions for Classification,

Loss Functions for Reconstruction,

# Loss Function

- Loss functions quantify how close a given neural network is to the ideal toward which it is training. The idea is simple.
- We calculate a metric based **on the error** we observe in the network's predictions.
- We then **aggregate these errors** over the entire dataset and **average them**
- Now we have a single number representative of how close the neural network is to its ideal.
- Looking for this ideal state is equivalent to finding the parameters (weights and biases) that will minimize the “**loss**” incurred from the errors.
- In this way, loss functions help reframe training neural networks as an optimization problem. In most cases, these parameters cannot be solved for analytically, but, more often than not, they can be approximated well with iterative optimization algorithms like gradient descent



# Loss Function

A **loss function** is a function that **compares** the target and predicted output values; measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target outputs.

The **hyperparameters** are adjusted to minimize the average loss — we find the weights,  $w^T$ , and biases,  $b$ , that minimize the value of  $J$  (average loss).

$$J(w^T, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

## How Loss Functions Are Implemented in TensorFlow

For this article, we will use Google's **TensorFlow** library to implement different loss functions — easy to demonstrate how loss functions are used in models.

In TensorFlow, the loss function the neural network uses is specified as a parameter in `model.compile()` — the final method that trains the neural network.

```
model.compile(loss='mse', optimizer='sgd')
```

The loss function can be inputted either as a String — as shown above — or as a function object — either imported from TensorFlow or written as custom loss functions, as we will discuss later.

```
from tensorflow.keras.losses import mean_squared_error  
model.compile(loss=mean_squared_error, optimizer='sgd')
```

All loss functions in TensorFlow have a similar structure:

```
def loss_function (y_true, y_pred): return losses
```

It must be formatted this way because the `model.compile()` method expects only two input parameters for the loss attribute.

---

# Types of Loss Function

1. **Loss Function for Regression —**
  2. **Loss Function for Classification —**
  3. **Loss Function for Construction —**
-

# Loss Function for Regression

- Involves predicting a specific value that is continuous in nature
- used in regression neural networks; given an input value, the model predicts a corresponding output value (rather than pre-selected labels);
- Estimating the price of a house or predicting stock prices are examples of regression because one works towards building a model that would predict a real-valued quantity.
- **Ex. Mean Squared Error, Mean Absolute Error**

## a) Mean Squared Error

- Mean Squared Error is the mean of squared differences between the actual and predicted value. If the difference is large the model will penalize it as we are computing the squared difference.
- This function has numerous properties that make it especially suited for calculating loss. The difference is squared, which means it does not matter whether the predicted value is above or below the target value.
- However, one disadvantage of this loss function is that it is very sensitive to outliers; if a predicted value is significantly greater than or less than its target value, this will significantly increase the loss.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

## b) Mean Absolute Error

- MAE finds the average of the absolute differences between the target and the predicted outputs.
- This loss function is used as an alternative to MSE in some cases. As mentioned previously, MSE is highly sensitive to outliers, which can dramatically affect the loss because the distance is squared. MAE is used in cases when the training data has a large number of outliers to mitigate this.

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|$$

## c) Mean Bias Error

Mean Bias Error is used to calculate the average bias in the model. Bias, in a nutshell, is [overestimating or underestimating a parameter](#). Corrective measures can be taken to reduce the bias post-evaluating the model using MBE.

Mean Bias Error takes the actual difference between the target and the predicted value, and not the absolute difference. One has to be cautious as the positive and the negative errors could cancel each other out, which is why it is one of the lesser-used loss functions.

$$MBE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)}{n}$$

# Loss Function for Classification

- used in classification problem;
- given an input, the neural network produces a vector of probabilities of the input belonging to various pre-set categories — can then select the category with the highest probability of belonging;
- A mail can be classified as a spam or not a spam and a person's dietary preferences can be put in one of three categories - vegetarian, non-vegetarian and vegan.
- **Ex. Binary Cross-Entropy, Categorical Cross-Entropy**



# a) Binary Cross Entropy Loss

Cross-entropy is the default loss function to use for binary classification problems.

It is intended for use with binary classification where the target values are in the set  $\{0, 1\}$ .

Mathematically, it is the preferred loss function under the inference framework of maximum likelihood. It is the loss function to be evaluated first and only changed if you have a good reason.

Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for predicting class 1. The score is minimized and a perfect cross-entropy value is 0

---

## b)Categorical Cross Entropy Loss

Categorical Cross Entropy loss is essentially Binary Cross Entropy Loss expanded to multiple classes. One requirement when categorical cross entropy loss function is used is that the labels should be [one-hot encoded](#).

This way, only one element will be non-zero as other elements in the vector would be multiplied by zero. This property is extended to an activation function called softmax,

---

## c) Hinge Loss

An alternative to cross-entropy for binary classification problems is the hinge loss function, primarily developed for use with Support Vector Machine (SVM) models.

It is intended for use with binary classification where the target values are in the set  $\{-1, 1\}$ .

The hinge loss function encourages examples to have the correct sign, assigning more error when there is a difference in the sign between the actual and predicted class values.

Reports of performance with the hinge loss are mixed, sometimes resulting in better performance than cross-entropy on binary classification problems.

---

# Loss Functions for Reconstruction

The loss function is used to optimize model performance. It tells us the prediction performance of the model. In the present context, reconstruction loss is a measure to check how good reconstructed images are produced by the proposed autoencoder model

---

|          |   |  |
|----------|---|--|
| <b>1</b> | <b>Loss Function for regression</b>     | <b>Mean Square error,<br/>Mean Absolute error<br/>Huber Loss</b>                   |
| <b>2</b> | <b>Loss function for classification</b> | <b>Binary cross entropy loss<br/>Categorical cross entropy loss<br/>Hinge loss</b> |

# Hyperparameter

- **"Hyperparameters are defined as the parameters that are explicitly defined by the user to control the learning process."**
- Hyperparameter selection **focuses on ensuring that the model neither underfits nor overfits the training dataset**, while learning the structure of the data as quickly as possible.
- Hyperparameters are **important** because they **directly control the behaviour of the training algorithm** and have a **significant impact on the performance of the model is being trained**.
- **"A good choice of hyperparameters can really make an algorithm shine"**.
- It controls the learning process.
- Basic Terms : Learning Rate, Regularization, Momentum, Sparsity,

# Parameter Vs Hyperparameter

| Parameter   | Hyperparameter  |
|---|---|
| Model parameters are the features of training data that will learn on its own during training | Model hyperparameters are the parameters that determine the entire training process |
| They are internal to the model and their values can be estimated from the data.               | They are external to the model and their values can not be estimated from the data. |
| They are not set manually   | They are set manually   |
| <b>Eg. weight and bias</b>  | <b>Eg. Learning rate, Hidden layers, hidden units</b>                               |

# Hyperparameter Categories

- Layer size
  - Magnitude (momentum, learning rate)
  - Regularization (dropout, drop connect, L1, L2)
  - Activations (and activation function families)
  - Weight initialization strategy
  - Loss functions
  - Settings for epochs during training (mini-batch size)
  - Normalization scheme for input data (vectorization)
-



# Layer size

- Layer size is defined by the number of neurons in a given layer.
  - Input and output layers are relatively easy to figure out because they correspond directly to how our modeling problem handles input and output.
  - For the input layer, this will match up to the **number of features in the input vector**.
  - For the output layer, this will **either be a single output neuron or a number of neurons** matching the number of classes we are trying to predict.
-

# Magnitude

- Hyperparameters in the magnitude group involve the gradient, step size, and momentum.  
Learning rate The learning rate in machine learning is how fast we change the parameter vector as we move through search space. If the learning rate becomes too high, we can move toward our goal faster (least amount of error for the function being evaluated),

# Regularization

Regularization is a measure taken against overfitting. Overfitting occurs when a model describes the training set but cannot generalize well over new inputs. Overfitted models have no predictive capacity for data that they haven't seen.

Regularization includes the following:

- Dropout: Dropout is a mechanism used to improve the training of neural networks by omitting a hidden unit. It also speeds training. Dropout is driven by randomly dropping a neuron so that it will not contribute to the forward pass and backpropagation
  - DropConnect: same thing as Dropout, but instead of choosing a hidden unit, it mutes the connection between two neurons.
  - L1 penalty
  - L2 penalty
-

# Regularization

- L1 penalty :

The penalty methods L1 and L2, in contrast, are a way of preventing the neural network parameter space from getting too big in one direction. They make large weights smaller.

L1 regularization is considered computationally inefficient in the nonsparse case, has sparse outputs, and includes built-in feature selection. L1 regularization multiplies the absolute value of weights rather than their squares. This function drives many weights to zero while allowing a few to grow large, making it easier to interpret the weights.

# Regularization

- L2 penalty: L2 regularization is **computationally efficient** due to it having analytical solutions and nonsparse outputs, but it **does not do feature selection** automatically for us.
- The “L2” regularization function, a common and simple hyperparameter, adds a term to the objective function that decreases the squared weights. You multiply half the sum of the squared weights by a coefficient called the weight-cost. L2 improves generalization, smooths the output of the model as input changes, and helps the network ignore weights it does not use.

## Mini Batch

- Here, we send **more than one input vector** (a group or batch of vectors) to be trained in the learning system.
  - This allows us to use **hardware and resources more efficiently** at the computer-architecture level.
  - This method also allows us to compute certain linear algebra operations (specifically matrix-to matrix multiplications) in a vectorized fashion.
  - In this scenario we also have the option of sending the vectorized computations to GPUs if they are present
-

# Learning Rate

- The learning rate **affects the amount by which you adjust parameters during optimization in order to minimize the error of neural network's guesses.**
- It is a coefficient that scales the size of the steps (updates) a neural network takes to its parameter vector  $x$  as it crosses the loss function space.
- It **determines how much of the gradient we want to use for the algorithm's next step.**
- **large error and steep gradient combine with the learning rate to produce a large step.**
- A **large learning rate coefficient (e.g., 1)** will make your parameters take **leaps**, and small ones (e.g., 0.00001) will make it inch along slowly. **Large leaps will save time initially**, but they can be **disastrous** if they **lead us to overshoot our minimum.**
- **small learning rates should lead you eventually to an error minimum** (it might be a local minimum rather than a global one), **but they can take a very long time and add to the burden of** an already computationally intensive process

# Regularization

- **Regularization helps with the effects of out-of-control parameters by using different methods to minimize parameter size over time.**
- In mathematical notation, we see regularization **represented** by the coefficient **lambda**, controlling the trade-off between **finding a good fit and keeping the value of certain feature weights low as the exponents on features increase.**
- Regularization coefficients L1 and L2 help fight **overfitting** by making certain weights **smaller. Smaller-valued weights lead to simpler hypotheses**, and simpler hypotheses are the most generalizable.
- **Unregularized weights with several higher-order polynomials in the feature set tend to overfit the training set.**
- **As the input training set size grows, the effect of regularization decreases and the parameters tend to increase in magnitude.**



# Momentum

- Momentum involves **adding an additional hyperparameter that controls the amount of history (momentum)** to include in the update equation, i.e. the step to a new point in the search space.
  - The value for the hyperparameter is defined in the range **0.0 to 1.0**
  - Momentum is introduced to **speed up the learning process**,
  - Especially for the **gradient with high curvature, small but consistent, which can accelerate the learning process.**
-

|               | Large value   | Small value   |
|---------------|---|---|
| Learning rate | <ul style="list-style-type: none"> <li>• Increase regularization</li> <li>• Increase training speed</li> <li>• Cause instability</li> </ul> | <ul style="list-style-type: none"> <li>• Cause overfitting</li> </ul>       |
| Batch size    | <ul style="list-style-type: none"> <li>• Add less regularization</li> </ul>   | <ul style="list-style-type: none"> <li>• Add more regularization</li> </ul> |
| Momentum      | <ul style="list-style-type: none"> <li>• Closely related to LR</li> </ul>   |   |
| Weight decay  | <ul style="list-style-type: none"> <li>• Increase regularization</li> <li>• Cause instability</li> </ul>                                    | <ul style="list-style-type: none"> <li>• Decrease regularization</li> </ul> |

# Feed Forward Neural Network(FFNN)

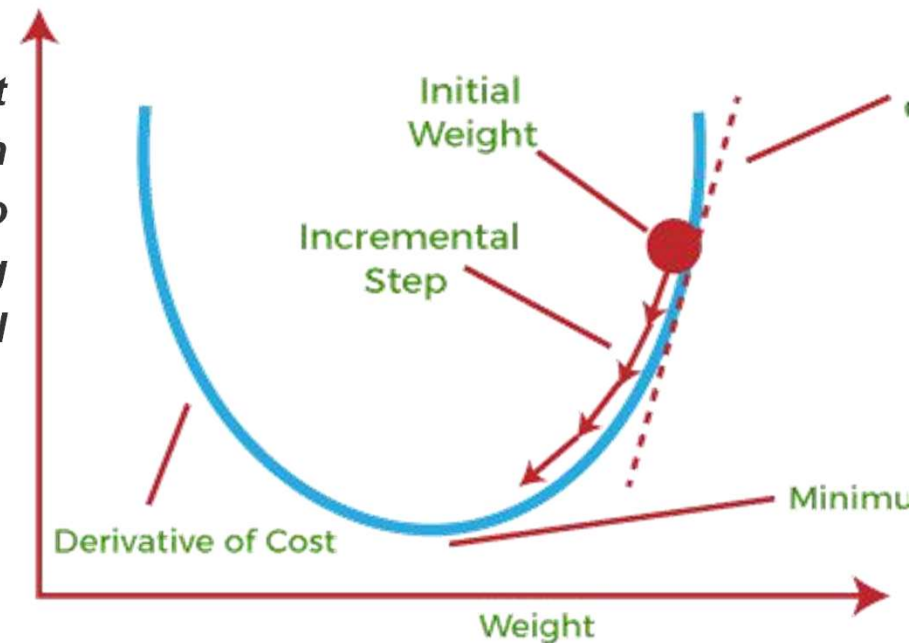
- Feed forward neural networks are **artificial neural networks** in which **nodes do not form loops**. This type of neural network is also known as a **multi-layer neural network** as all information is **only passed forward**.
- During **data flow**, **input nodes receive data, which travel through hidden layers, and exit output nodes**. No links exist in the network that could get used to by sending information back from the output node.
- A Feed Forward Neural Network is commonly seen in its simplest form as a **single layer perceptron**.and is often used in classification tasks.
- In this model, a series of **inputs enter the layer and are multiplied by the weights**. Each value is then **added together to get a sum of the weighted input values**. If the sum of the values is above a **specific threshold**, usually **set at zero**, the value produced is often **1**, whereas if the **sum falls below the threshold**, the output value is **-1**.

# Gradient Descent

- *Discovered by "Augustin-Louis Cauchy"*
- *Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.*

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.



# Gradient 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.
-

# Cost Function

*The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number.*

It helps to **increase and improve machine learning efficiency by providing feedback** to this model so that it can minimize error and find the local or global minimum.

Further, it **continuously iterates along the direction of the negative gradient until the cost function approaches zero**. At this steepest descent point, the model will stop learning further.

The **slight difference** between the loss function and the **cost function** is about **the error within the training of machine learning models**, as **loss function** refers to the **error of one training example**, while a **cost function** calculates the **average error across an entire training set**.

The cost function is **calculated after making a hypothesis with initial parameters and modifying these parameters using gradient descent algorithms** over known data to reduce the cost function.

---

# Cost Function

*The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number.*

It helps to **increase and improve machine learning efficiency by providing feedback** to this model so that it can minimize error and find the local or global minimum.

Further, it **continuously iterates along the direction of the negative gradient until the cost function approaches zero**. At this steepest descent point, the model will stop learning further.

The **slight difference** between the loss function and the **cost function** is about **the error within the training of machine learning models**, as **loss function** refers to the **error of one training example**, while a **cost function** calculates the **average error across an entire training set**.

The cost function is **calculated after making a hypothesis with initial parameters and modifying these parameters using gradient descent algorithms** over known data to reduce the cost function.

---

# Batch Gradient Descent

- Batch gradient descent (BGD) is used **to find the error for each point in the training set and update the model after evaluating all training examples.**
- This procedure is known as the **training epoch.**
- It is a **greedy approach** where we have to sum over all examples for each update.

## **Advantages of Batch gradient descent:**

- It produces less noise in comparison to other gradient descent.
  - It produces stable gradient descent convergence.
  - It is Computationally efficient as all resources are used for all training samples.
-



# Stochastic Gradient Descent

- Stochastic gradient descent (SGD) is a type of gradient descent **that runs one training example per iteration.**
- 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.**
- 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 datasets.**

# Mini Batch Gradient Descent

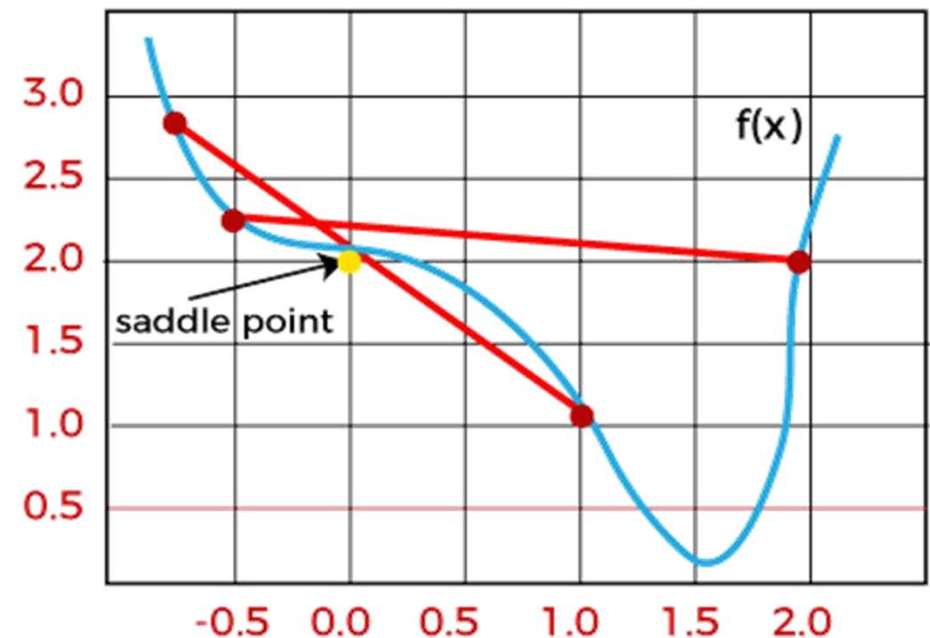
- Mini Batch gradient descent is the **combination** of both **batch gradient descent** and **stochastic gradient descent**.
- It divides the **training datasets** into **small batch sizes** then performs the updates on **those batches separately**.
- **Splitting** training datasets into smaller batches make a balance to maintain the **computational efficiency of batch gradient descent** and **speed of stochastic gradient descent**.
- Hence, we can achieve a special type of gradient descent with **higher computational efficiency and less noisy gradient descent**.

## Advantages of Mini Batch gradient descent:

- It is easier to fit in allocated memory.
- It is computationally efficient.
- It produces stable gradient descent convergence.

# Challenges with the Gradient Descent

1. Local Minima and Saddle Point
2. Vanishing and Exploding Gradient



# Local Minima and Saddle Point

- For **convex** problems, gradient descent can find the global minimum easily, while for **non-convex** problems, it is sometimes difficult to find the global minimum, where the machine learning models achieve the best results
- Whenever the **slope of the cost function is at zero or just close to zero**, this model stops learning **further**. Apart from the global minimum, there occur some scenarios that can show this slop, which is **saddle point and local minimum**. Local minima generate the shape similar to the global minimum, **where the slope of the cost function increases on both sides of the current points**.
- In contrast, **with saddle points, the negative gradient only occurs on one side of the point, which reaches a local maximum on one side and a local minimum on the other side**. The name of a saddle point is taken by that of a horse's saddle.
- The name of **local minima** is because the value of the loss function is minimum at that point in a **local region**. In contrast, the name of the global minima is given so because the value of the loss function is minimum there, globally across the entire domain the loss function.

# Vanishing and Exploding Gradient

- In a deep neural network, if the model is trained with gradient descent and backpropagation, there can occur two more issues other than local minima and saddle point.

## Vanishing Gradients:

- **Vanishing Gradient** occurs when the gradient is smaller than expected. During backpropagation, this gradient becomes smaller that causing the decrease in the learning rate of earlier layers than **the later layer of the network**. Once this happens, the weight parameters update until they become insignificant.

## Exploding Gradient:

- **Exploding gradient** is just opposite to the vanishing gradient as it occurs when the **Gradient is too large and creates a stable model**. Further, in this scenario, model weight increases, and they will be represented as NaN. This problem can be solved using the dimensionality reduction technique, which helps to minimize complexity within the model.

# Sentiment Analysis

Sentiment analysis is a powerful text analysis tool that automatically mines unstructured data (social media, emails, customer service tickets, and more) for opinion and emotion, and can be performed using machine learning and deep learning algorithms.

Deep learning (DL) is considered an evolution of machine learning. It chains together algorithms that aim to simulate how the human brain works, otherwise known as an artificial neural network, and has enabled many practical applications of machine learning, including customer support automation and self-driving cars.

---

# Sentiment Analysis

It is the classification of emotions (positive, negative, and neutral) within data using **text analysis** techniques. Harnessing the power of deep learning, sentiment analysis models can be trained to understand text beyond simple definitions, read for context, sarcasm, etc., and understand the actual mood and feeling of the writer.

Deep learning is hierarchical machine learning that uses multiple algorithms in a progressive chain of events to solve complex problems and allows you to tackle massive amounts of data, accurately and with very little human interaction.

---

## References

<https://www.geeksforgeeks.org/activation-functions-neural-networks/?ref=lbp>

<https://monkeylearn.com/blog/sentiment-analysis-deep-learning/>

---