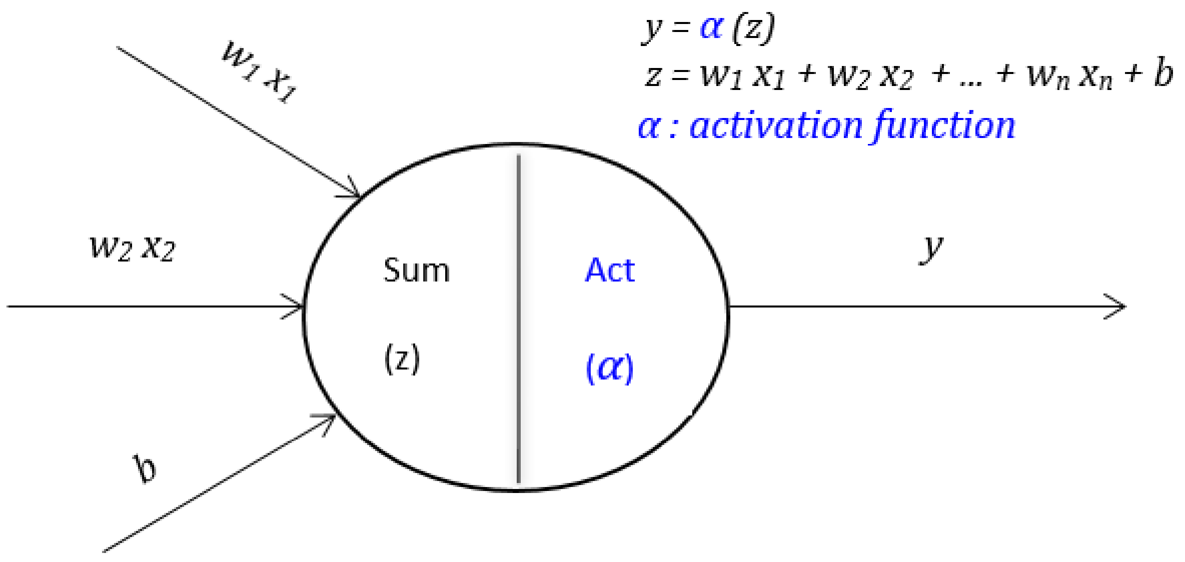
UNIT-3

Activation Function

- In artificial neural networks, each neuron forms a weighted sum of its inputs and passes the resulting scalar value through a function referred to as an activation function.



$y = \alpha(z)$
$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$
α : activation function
$w_1 x_1$
$w_2 x_2$
b
Sum
(z)
Act
(α)
y

- An activation function determines if a neuron should be **activated or not activated**. This implies that it will use some simple mathematical operations to determine if the neuron's input to the network is relevant or not relevant in the prediction process.
- The ability to introduce **non-linearity** to an artificial neural network and generate output from a collection of input values fed to a layer is the purpose of the activation function.
- Activation functions are mathematical equations that determine the output of a neural network model.
- Activation functions also have a major effect on the neural network's ability to converge and the convergence speed, or in some cases, activation functions might prevent neural networks from converging in the first place.
- Activation function also helps to normalize the output of any input in the range between 1 to -1 or 0 to 1.
- Activation function must be efficient and it should reduce the computation time because the neural network sometimes trained on millions of data points.

Let's consider the simple neural network model without any hidden layers.

Here is the output-

Y = ∑ (weights*input + bias)

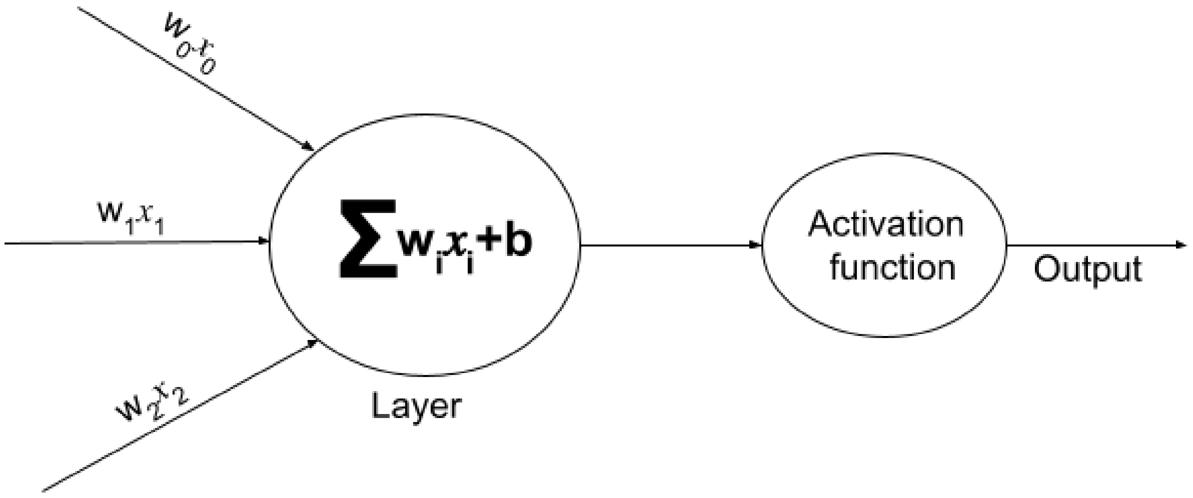
So, if inputs are x1+x2+x3…. xn and the weights are w1+w2 + w3.......wn

then, the activation would be (Activation function (x1*w1+x2*w2+x3*w3……xn*wn) +bias)

and it can range from -infinity to +infinity. So, it is necessary to bound the output to get the desired prediction or generalized results.

Y = Activation function (∑ (weights*input + bias))

So, the activation function is an important part of an artificial neural network. They decide whether a neuron should be activated or not and it is a non-linear transformation that can be done on the input before sending it to the next layer of neurons or finalizing the output.



Types of Activation functions

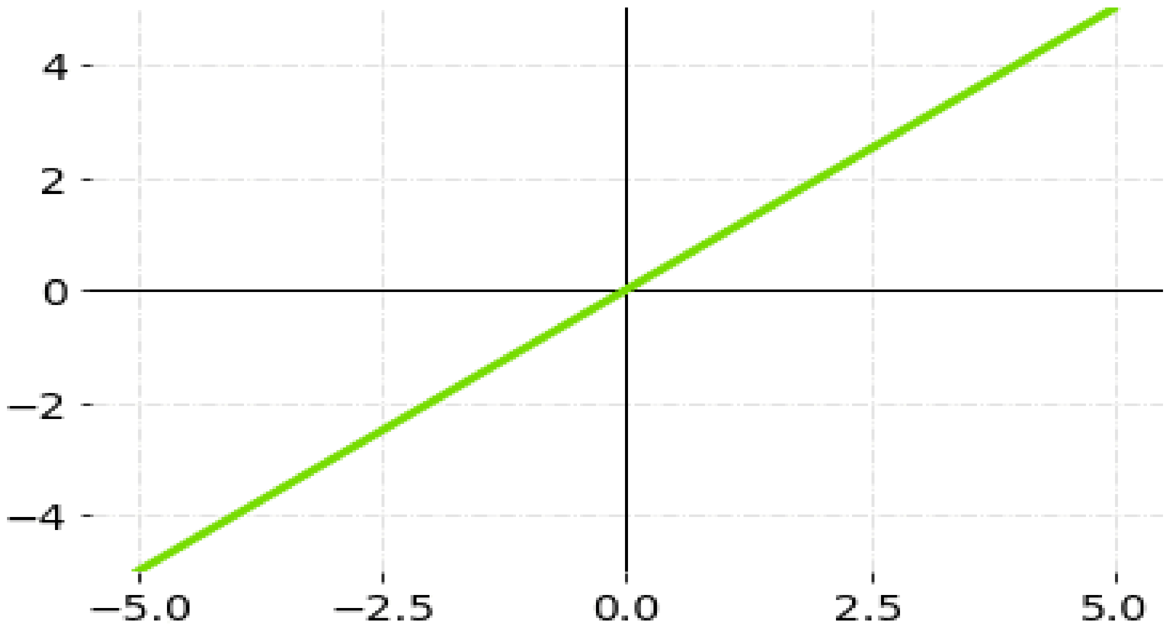
Activation functions can be divided into three types:

1. Linear Activation Function
2. Binary Step Function
3. Non-linear Activation Functions

Linear Activation Function

The linear activation function, often called the **identity activation function**, is proportional to the input. The range of the linear activation function will be (-∞

to ∞). The linear activation function simply adds up the weighted total of the inputs and returns the result.

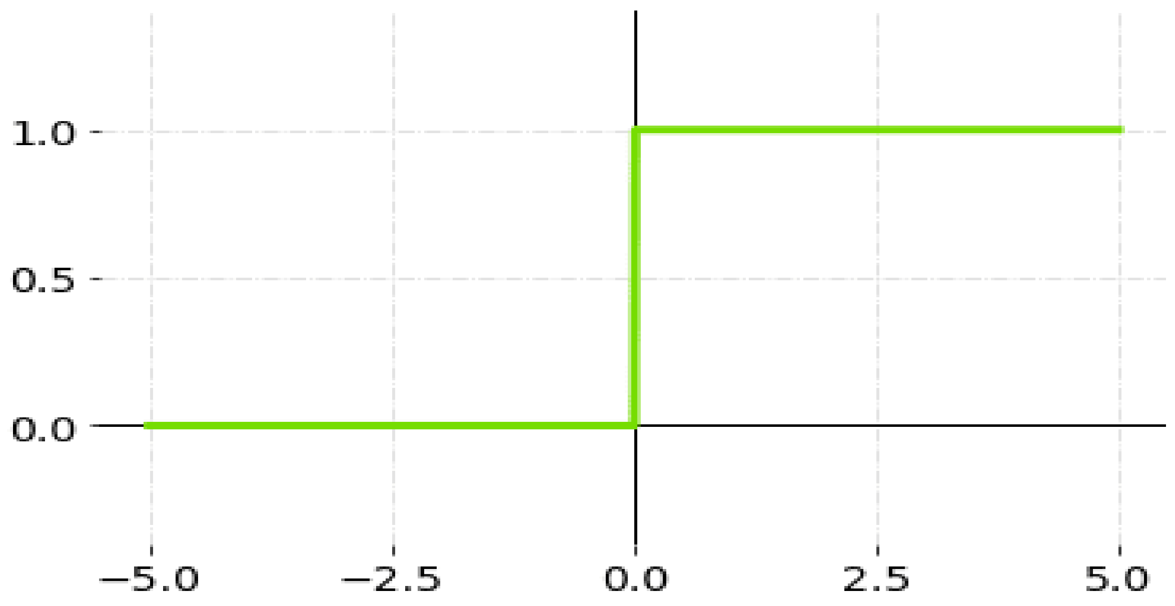


Mathematically, it can be represented as:

$$f(x) = x$$

Binary Step Activation Function
A threshold value determines whether a neuron should be activated or not activated in a binary step activation function.

The activation function compares the input value to a threshold value. If the input value is greater than the threshold value, the neuron is activated. It's disabled if the input value is less than the threshold value, which means its output isn't sent on to the next or hidden layer.



Mathematically, the binary activation function can be represented as:

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

Non-linear Activation Functions
The non-linear activation functions are the most-used activation functions. They make it uncomplicated for an artificial neural network model to adapt to a variety of data and to differentiate between the outputs.

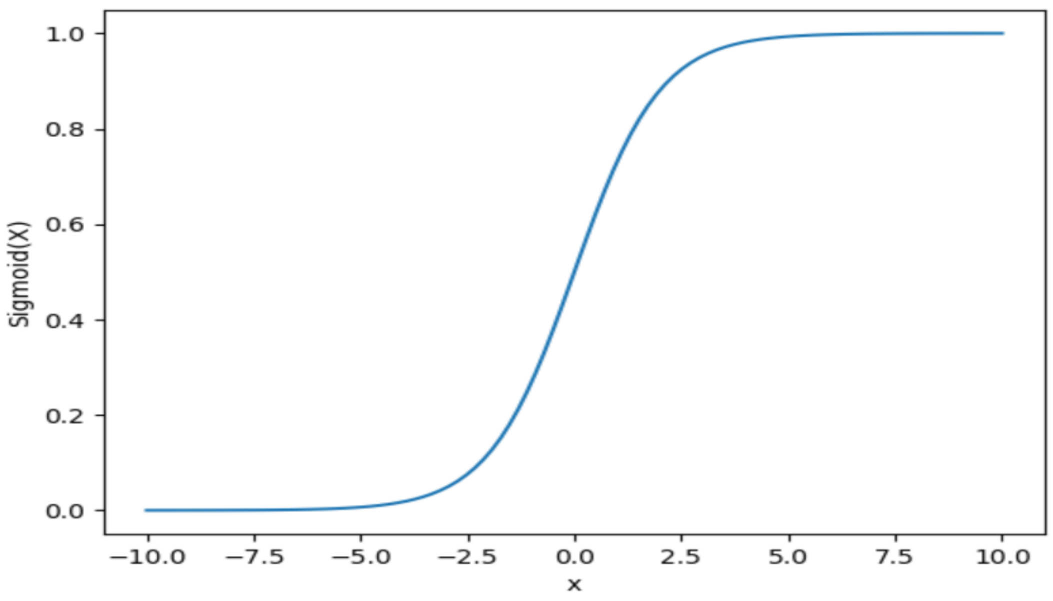
Non-linear activation functions allow the stacking of multiple layers of neurons, as the output would now be a non-linear combination of input passed through multiple layers. Any output can be represented as a functional computation output in a neural network.

These activation functions are mainly divided basis on their range and curves. The remainder of this article will outline the major non-linear activation functions used in neural networks.

Sigmoid Activation Function

- The sigmoid function is defined mathematically as **1/(1+e^(-x))**, where x is the input value and e is the mathematical constant of 2.718.

- The function maps any input value to a value between 0 and 1, making it useful for binary classification and logistic regression problems. The range of the function is (0,1), and the domain is (-infinity,+infinity).
- One of the key properties of the sigmoid function is its “S” shape. As the input value increases, the output value of it starts with a slow increase, then rapidly approaches 1, and finally levels off. This property makes a valuable function for modeling decision boundaries in binary classification problems.
- Another property of the sigmoid is its derivative, commonly used in training neural networks. The derivative of the function is defined as **f(x)(1-f(x))**, where f(x) is the output of the function. The derivative is helpful in training neural networks because it allows the network to adjust the weights and biases of the neurons more efficiently.
- It’s also worth mentioning that the sigmoid function has some limitations. For example, the output of the sigmoid is always between 0 and 1, which can cause problems when the network’s output should be greater than 1 or less than 0. Other activation functions like ReLU and tanh can be used in such cases.
- Visualizing the sigmoid function using graphs can help to understand its properties better. Its graph will show the “S” shape of the function and how the output value changes as the input value changes.

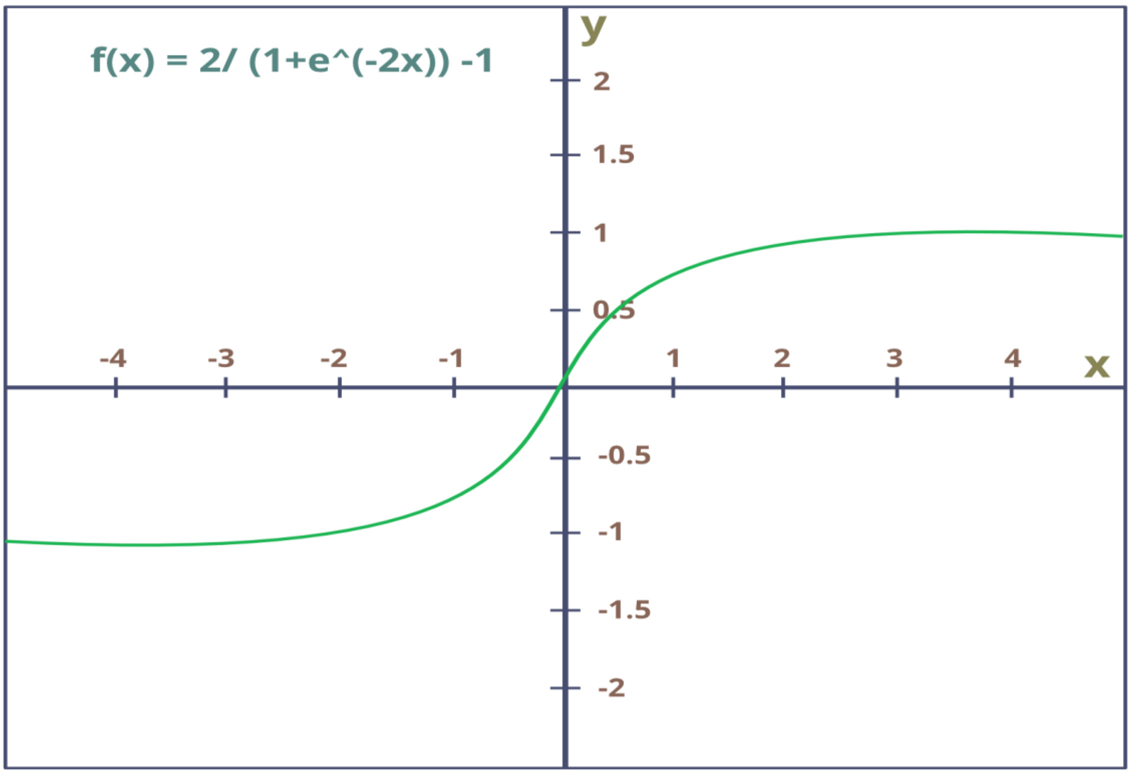


- It is a function which is plotted as **‘S’** shaped graph.
- **Equation :** $A = 1/(1 + e^{-x})$

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

Tanh Activation Function

- The tanh function is just another possible function that can be used as a non-linear activation function between layers of a neural network. It shares a few things in common with the sigmoid activation function.
- Unlike a sigmoid function that will map input values between 0 and 1, the Tanh will map values between -1 and 1.
- Like the sigmoid function, one of the interesting properties of the tanh function is that the derivative of tanh can be expressed in terms of the function itself.



- 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.

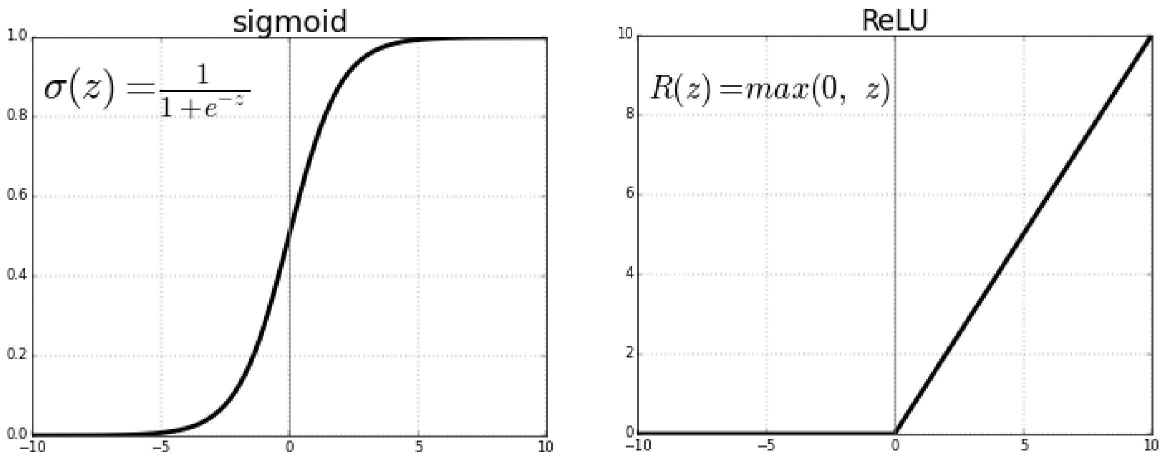
- **Equation: -**

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

- **Value Range: -** -1 to +1
- **Nature: -** non-linear
- **Uses: -** Usually used in hidden layers of a neural network as 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 centring the data by bringing mean close to 0. This makes learning for the next layer much easier.

RELU Activation Function

- A rectified linear unit (ReLU) is an activation function that introduces the property of non-linearity to a deep learning model and solves the vanishing gradients issue.
- It interprets the positive part of its argument. It is one of the most popular activation functions in deep learning.
- **ReLU formula is : f(x) = max(0,x)**



As shown in figure, 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.

ReLU activation function formula

Now how does ReLU transform its input? It uses this simple formula:

f(x)=max(0,x)

ReLU function is its derivative both are monotonic. The function returns 0 if it receives any negative input, but for any positive value x, it returns that value back. Thus, it gives an output that has a range from 0 to infinity.

Now let us give some inputs to the ReLU activation function and see how it transforms them and then we will plot them also.

First, let us define a ReLU function

```
def ReLU(x):
  if x>0:
        return x
  else:
    return 0
```

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.

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

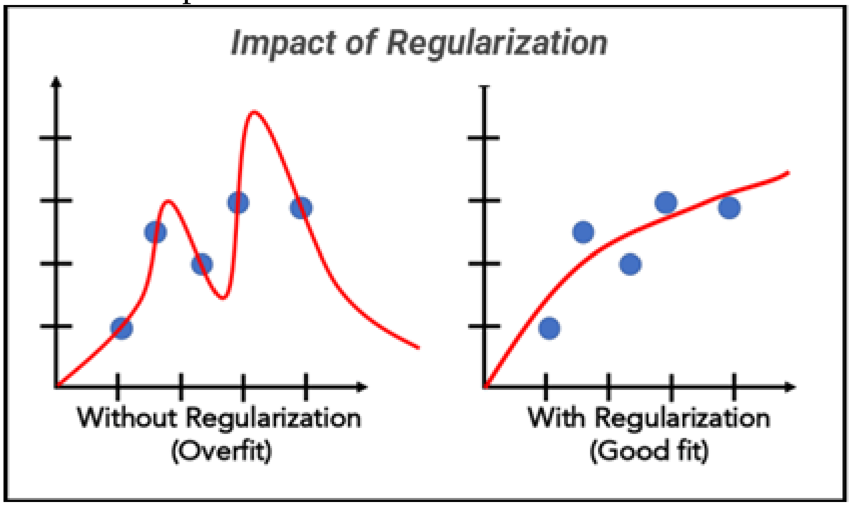
- In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Regularization in Deep Learning

- Regularization is a machine-learning strategy that avoids overfitting. Overfitting happens when a model fits the training data too well and is too complicated yet fails to function adequately on unobserved data.
- The model's loss function is regularized to include a penalty term, which helps prevent the parameters from growing out of control and simplifies the model.
- As a result, the model has a lower risk of overfitting and performs better when applied to new data.
- When working with high-dimensional data, regularization is especially crucial since it lowers the likelihood of overfitting and keeps the model from becoming overly complicated.

What is regularization in machine learning?

- Regularization is a machine-learning approach that prevents overfitting by including a penalty term into the model's loss function.
- Regularization has two objectives: to lessen a model's complexity and to improve its ability to generalize to new inputs.
- Different penalty terms are added to the loss function using numerous regularization methods, including L1 and L2 regularization.
- In contrast to L2 regularization, which adds a punishment term based on the squares of the parameters, L1 regularization adds a penalty term based on the absolute values of the model's parameters.
- Regularization decreases the chance of overfitting and helps keep the model's parameters from going out of control, both of which can enhance the model's performance on untested data.



What is L1 regularization?

- L1 regularization, also known as Lasso regularization, is a machine-learning strategy that inhibits overfitting by introducing a penalty term into the model's loss function based on the absolute values of the model's parameters.

- L1 regularization seeks to reduce some model parameters toward zero in order to lower the number of non-zero parameters in the model.
- L1 regularization is particularly useful when working with high-dimensional data since it enables one to choose a subset of the most important attributes.
- This lessens the risk of overfitting and makes the model easier to understand. The size of a penalty term is controlled by the hyperparameter lambda, which regulates the L1 regularization's regularization strength.
- As lambda rises, more parameters will be lowered to zero, improving regularization.
- L1 Regularization, also called a lasso regression, adds the “absolute value of magnitude” of the coefficient as a penalty term to the loss function.
- **Lasso Regression** (Least Absolute Shrinkage and Selection Operator) adds “*absolute value of magnitude*” of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

What is L2 regularization?

- L2 regularization, also known as Ridge regularization, is a machine learning technique that avoids overfitting by introducing a penalty term into the model's loss function based on the squares of the model's parameters.
- The goal of L2 regularization is to keep the model's parameter sizes short and prevent oversizing.
- In order to achieve L2 regularization, a term that is proportionate to the squares of the model's parameters is added to the loss function.
- This word works as a limiter on the parameters' size, preventing them from growing out of control.
- A hyperparameter called lambda that controls the regularization's intensity also controls the size of the penalty term. The parameters will be smaller and the regularization will be stronger the greater the lambda.
- **Ridge regression** adds “*squared magnitude*” of coefficient as penalty term to the loss function. Here the *highlighted* part represents L2 regularization element.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

- Here, if *lambda* is zero then you can imagine we get back ordinary least squares.
- However, if *lambda* is very large then it will add too much weight and it will lead to under-fitting. Having said that it is important how *lambda* is chosen. This technique works very well to avoid over-fitting issue.

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda\sum_{j=0}^{M}|W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda\sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$