

Agenda :

1. How to train MLP
2. Memoization
3. Backpropagation
4. Vanishing Gradient Problem
5. Expanding Gradient Problem
6. Overfitting Underfitting
7. Activation Function
8. Regularization
9. MCQ

Vanishing Gradient

In backpropagation, the new weight(w_{new}) of a node is calculated using the old weight(w_{old}) and product of the learning rate(η) and gradient of the loss function ($\frac{\partial C}{\partial w}$)

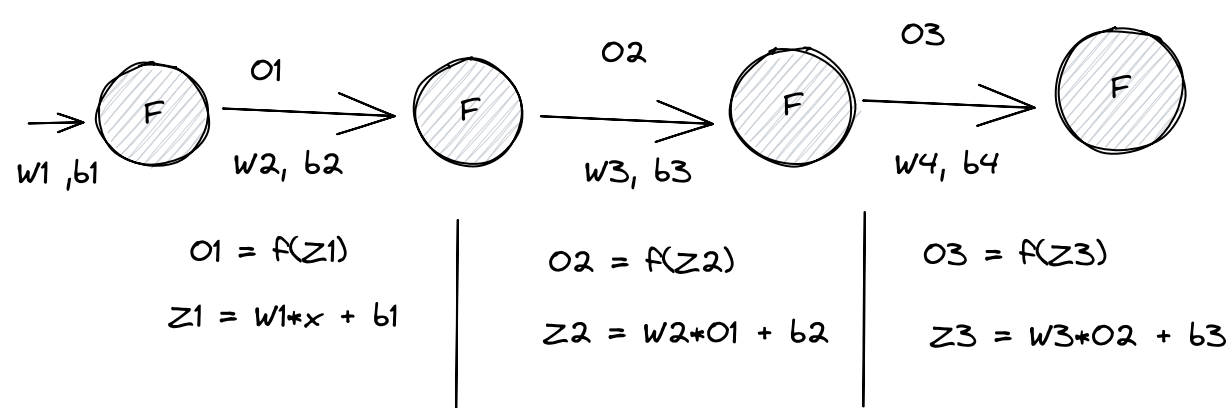
$$W_{new} = W_{old} - \eta * \frac{\partial C}{\partial w}$$

When there are more layers in the network, the value of the product of derivative decreases until at some point the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes. We call this the vanishing gradient problem.

-- Because of chain rule we get the vanishing gradient problem the value ($\frac{\partial C}{\partial w}$) becomes very very small.

Expanding Gradient Problem

-- Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training.

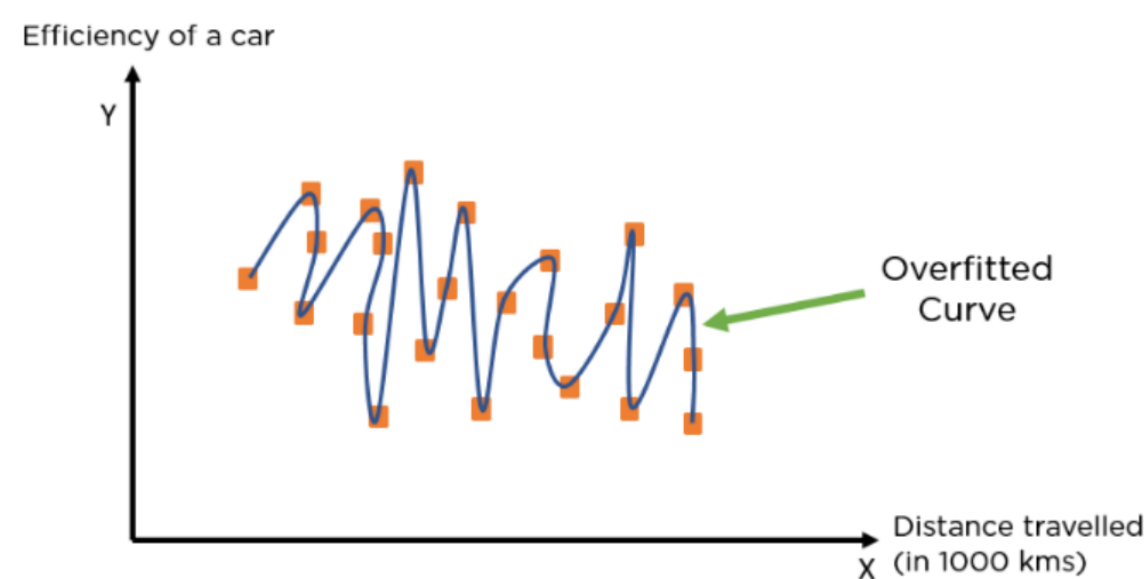


-- If we initiate value of w and b very high then the multiplication outcome will be very high in this case. This condition will be called as expanding gradient.

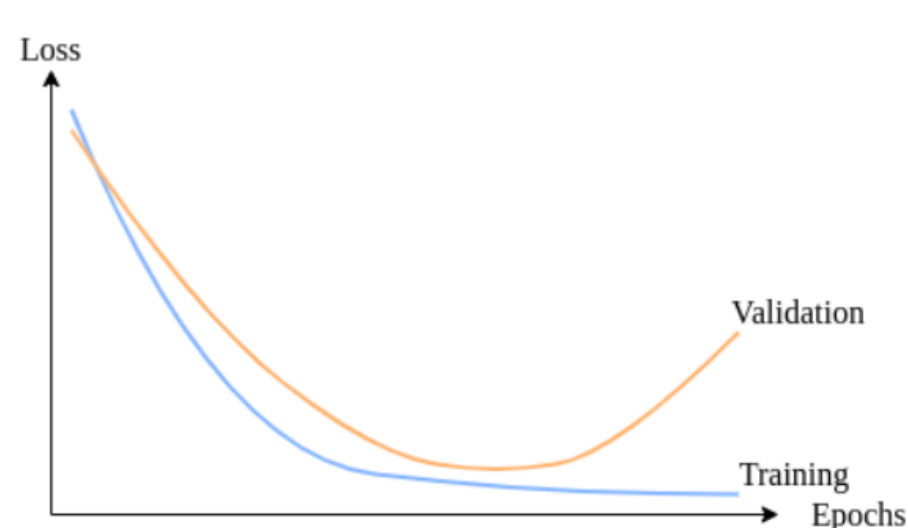
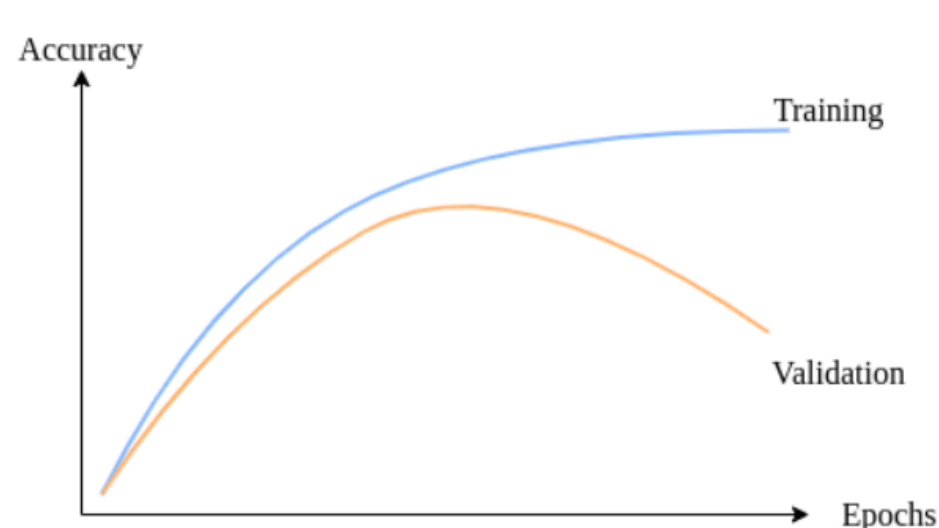
Overfitting and Underfitting

Overfitting :

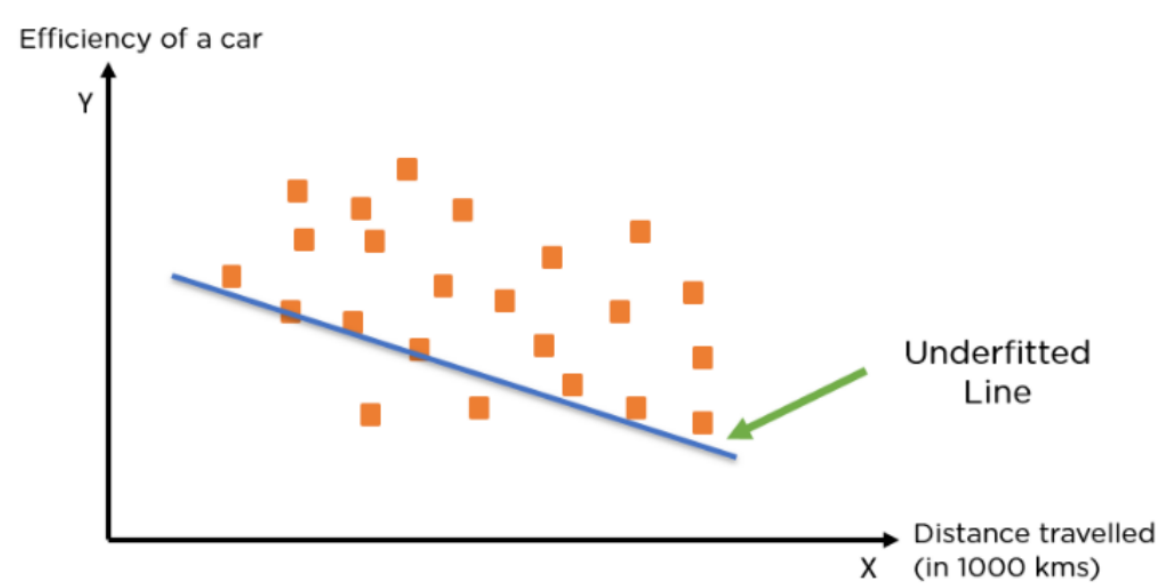
When a model performs very well for training data but has poor performance with test data (new data), it is known as overfitting. In this case, the machine learning model learns the details and noise in the training data such that it negatively affects the performance of the model on test data.



We need to closely watch model loss and accuracy to decide how the model is fitted to the dataset:



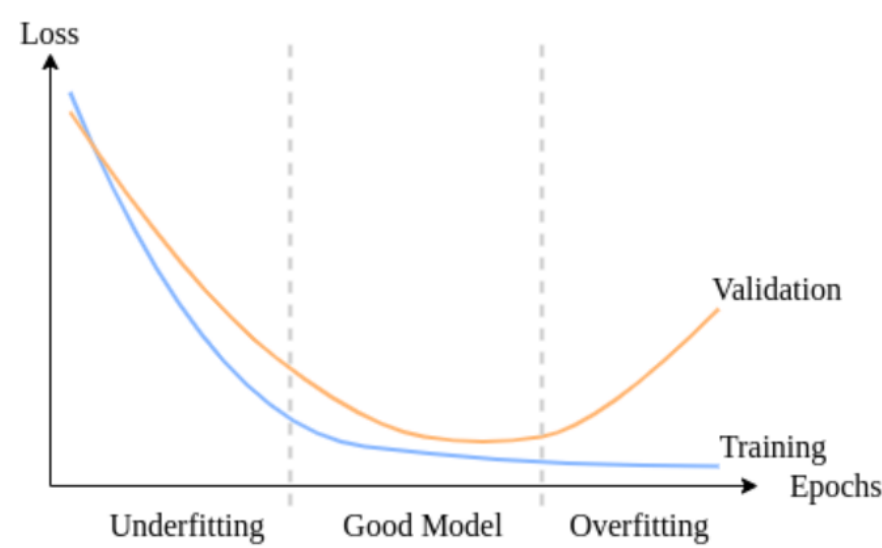
Underfitting :



Let's summarize what we've discussed so far in a comparison table:

Overfitting	Underfitting
Model is too complex	Model is not complex enough
Accurate for training set	Not accurate for training set
Not accurate for validation set	Not accurate for validation set
Need to reduce complexity	Need to increase complexity
Reduce number of features	Increase number of features
Apply regularization	Reduce regularization
Reduce training	Increase training
Add training examples	Add training examples

Conditions



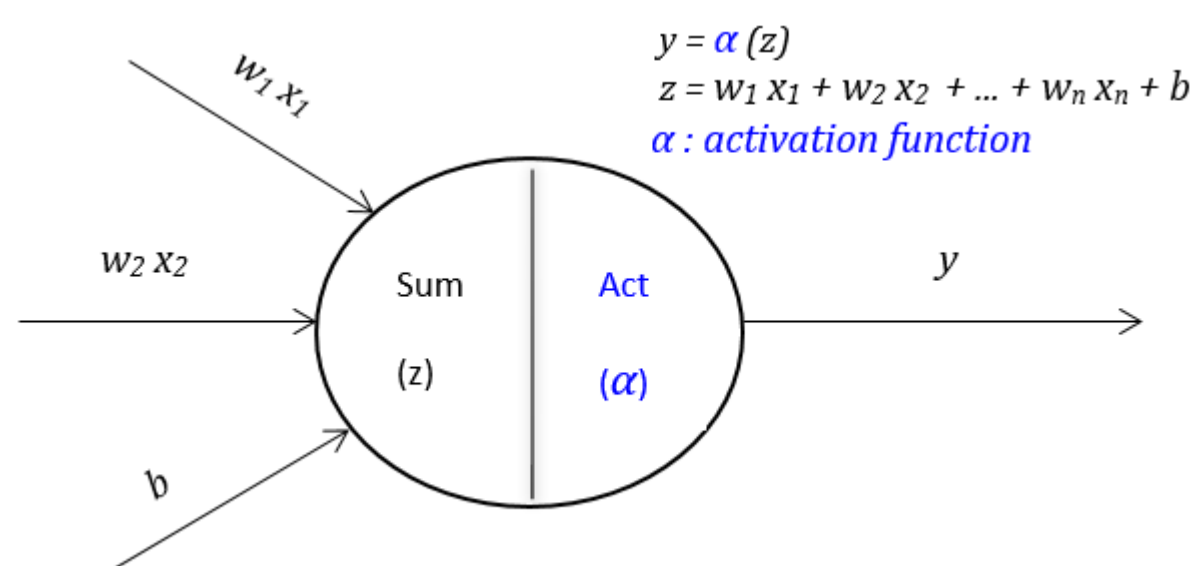
Activation Function

What it actually is?

- Activation functions are functions used in a neural network to compute the weighted sum of inputs and biases, which is in turn used to decide whether a neuron can be activated or not.
- It manipulates the presented data and produces an output for the neural network that contains the parameters in the data.
- It's just a thing function that you use to get the output of node. It is also known as Transfer Function.

Why we use Activation Functions with Neural Networks?

- It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).



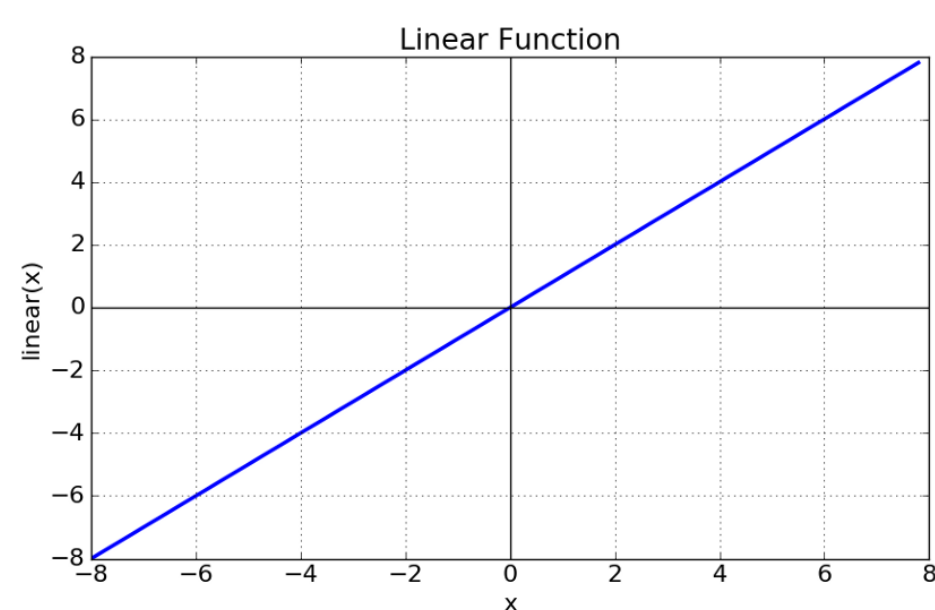
How to use it?

--- In a neural network every neuron will do two computations:

1. Linear summation of inputs: In the above diagram, it has two inputs x_1, x_2 with weights w_1, w_2 , and bias b . And the linear sum $z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$
2. Activation computation: This computation decides, whether a neuron should be activated or not, by calculating the weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Types of Activation Functions

1. Linear or Identity Activation Function

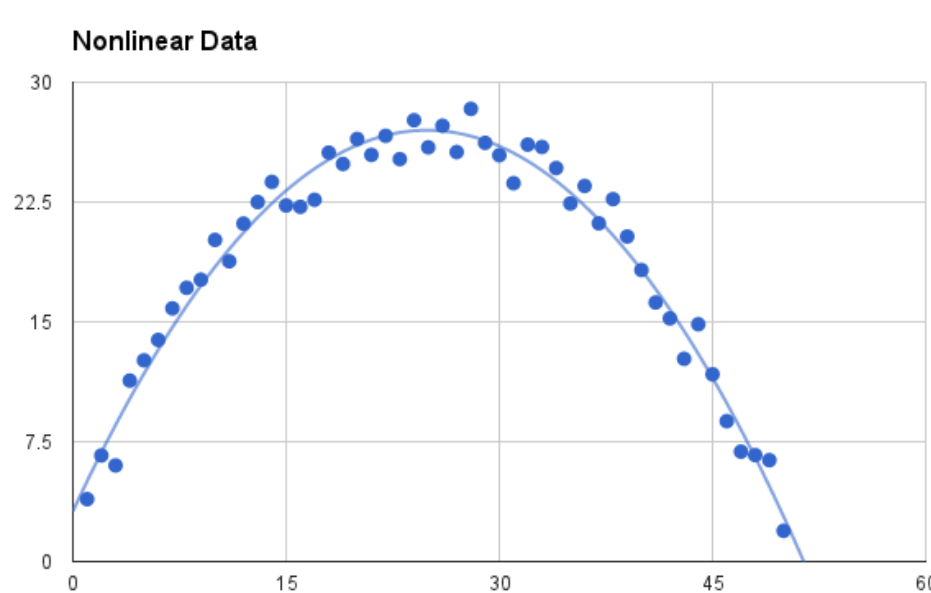


Equation : $f(x) = x$

Range : (-infinity to infinity)

- It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.

2. Non-linear Activation Function

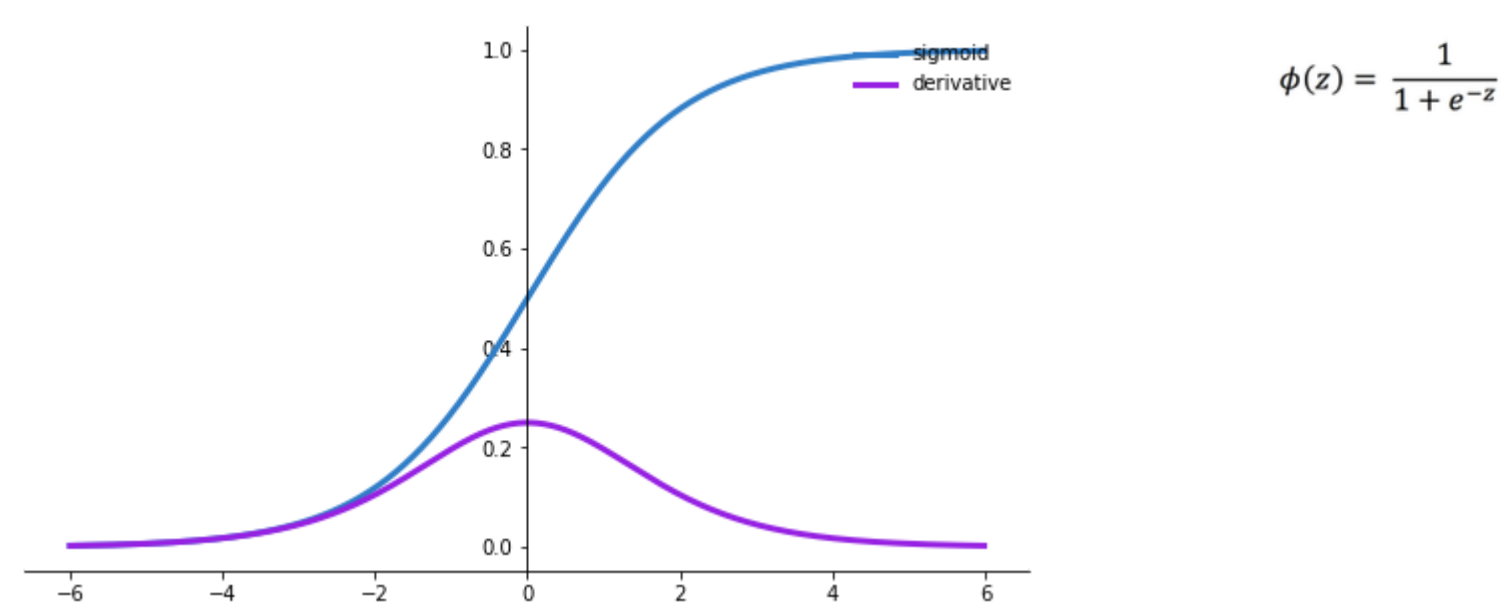


- The Nonlinear Activation Functions are the most used activation functions.

The main terminologies needed to understand for nonlinear functions are:

1. Derivative or Differential: Change in the y-axis w.r.t. change in the x-axis. It is also known as slope.
2. Monotonic function: A function that is either entirely non-increasing or non-decreasing.

1. Sigmoid or Logistic Activation Function

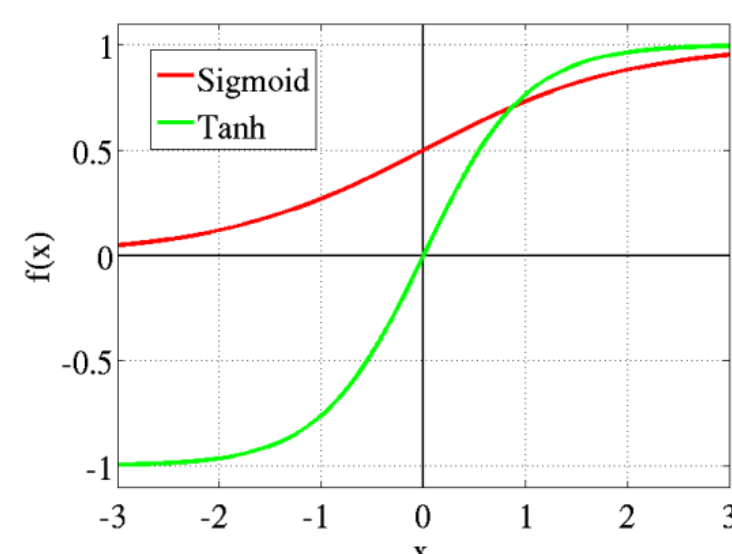
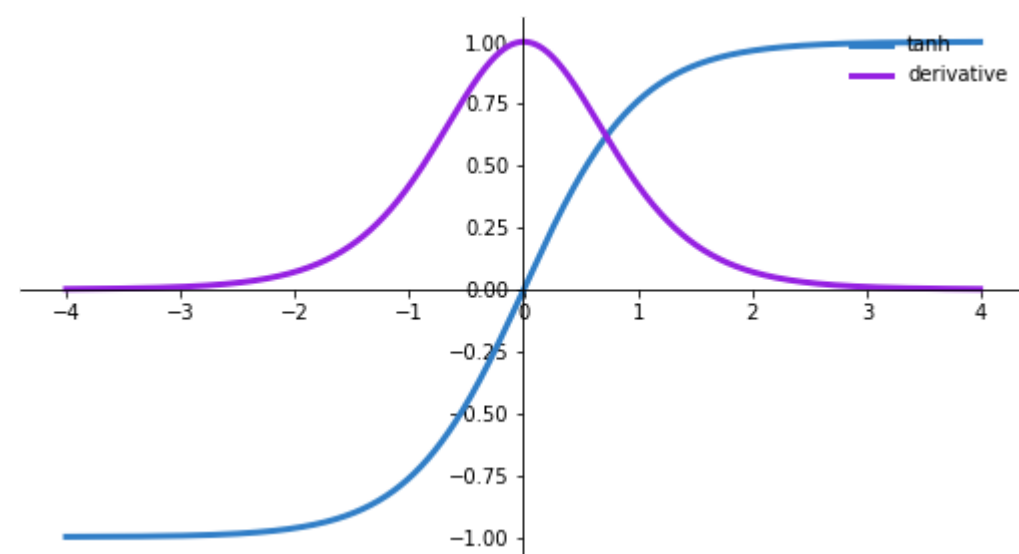


Observations:

- The sigmoid function has values between 0 to 1.
- The function is differentiable. That means, we can find the slope of the sigmoid curve at any two points.
- At the top and bottom level of sigmoid functions, the curve changes slowly, the derivative curve above shows that the slope or gradient it is zero
- The function is monotonic but function's derivative is not.

Problem :- Sigmoids saturate and kill gradients. It gives rise to a problem of "vanishing gradients"
The network refuses to learn further or is drastically slow

2. Tanh Activation Function

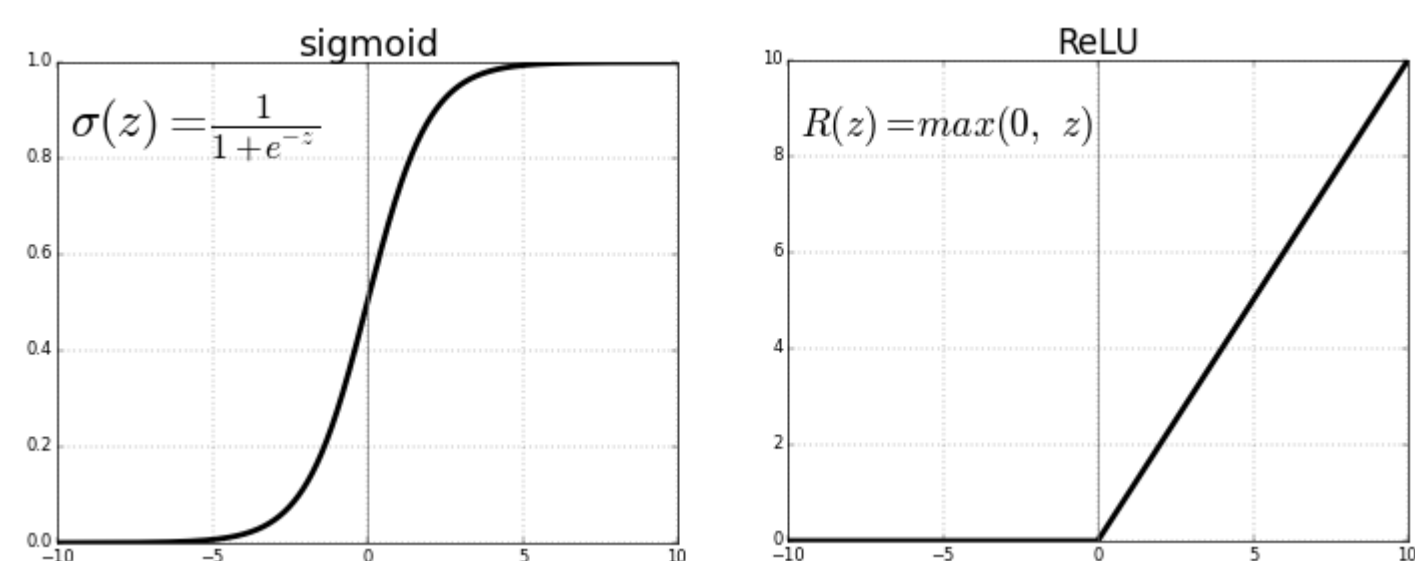


- Tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).
- The function is differentiable.
- The function is monotonic while its derivative is not monotonic.
- The tanh function is mainly used classification between two classes.
- Both tanh and logistic sigmoid activation functions are used in feed-forward nets.

Problem: Tanh also has a vanishing gradient problem.

3. ReLU (Rectified Linear Unit) Activation Function

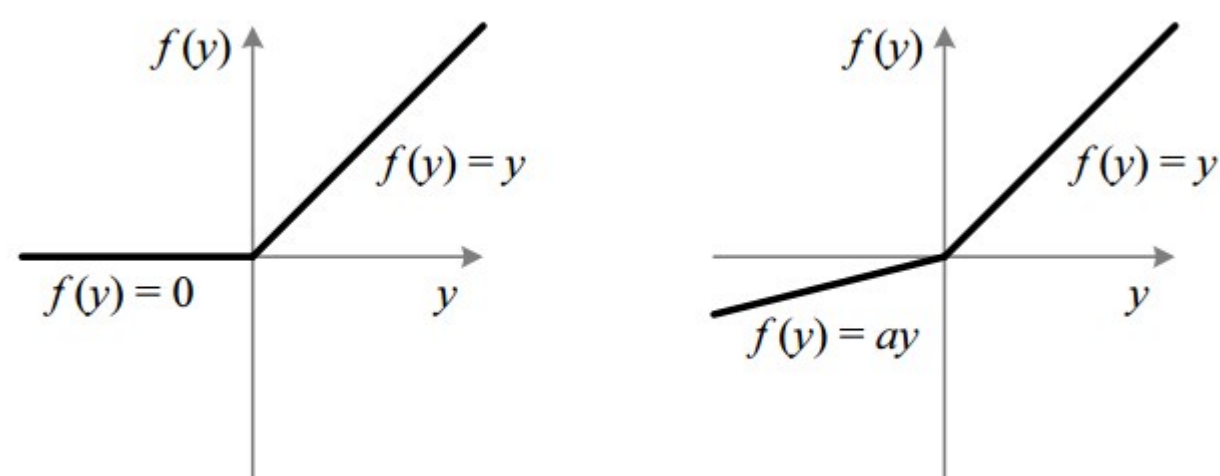
- The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning.



- 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.
- Dead Activation zone.

4. Leaky ReLU










It is an attempt to solve the dying ReLU problem



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

Why derivative/differentiation is used ?

When updating the curve, to know in which direction and how much to change or update the curve depending upon the slope. That is why we use differentiation in almost every part of Machine Learning and Deep Learning.

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

