TYPES OF ACTIVATION FUNCTION

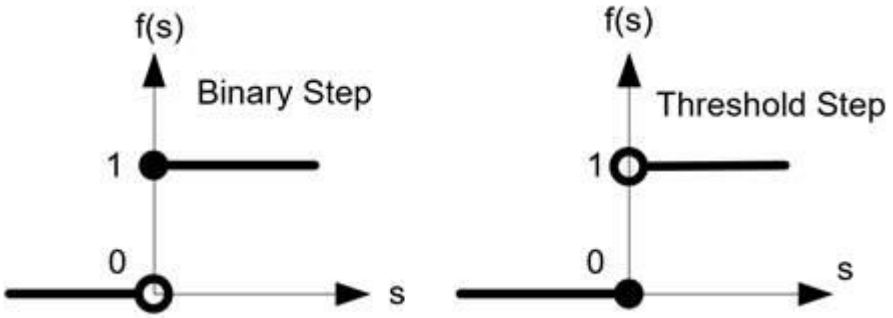
The purpose of an activation function is to add non-linearity to the neural network.

There are 3 types of neural networks activation functions:

Binary step function:

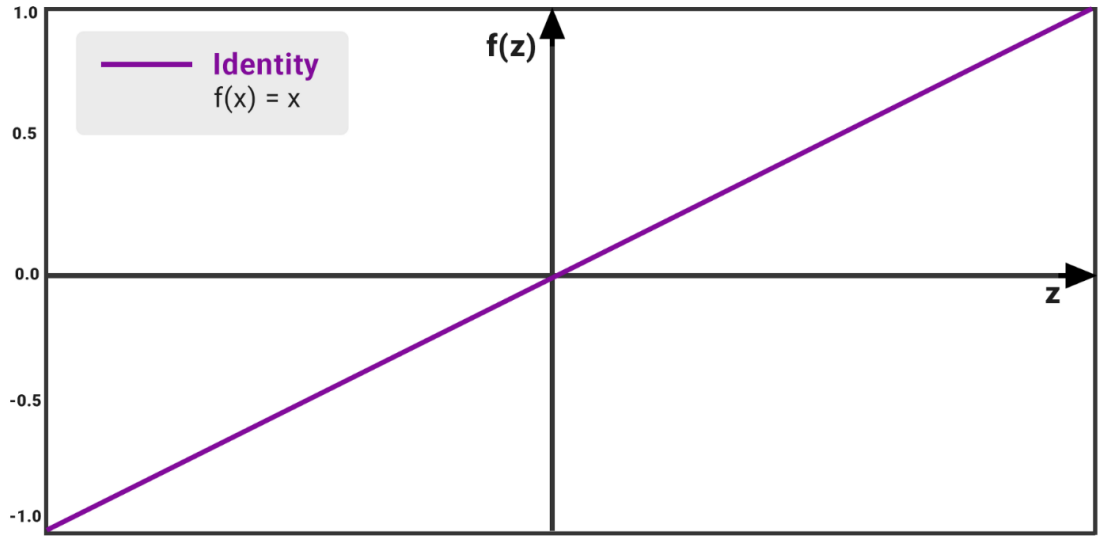
Binary step function is a threshold-based activation function which means after a certain threshold neuron is activated and below the said threshold neuron is deactivated.

The step function can be used in binary classification problems and works well for linearly severable data.



Linear Activation Function:

The linear activation function is also known as Identity Function where the activation is proportional to the input. The weights get multiplied by 1, so there is no updating of weights. Linear activation function turns the neural network into just one layer.



Non-linear activation functions

These are the most commonly used activation function as due to non-linearity the neural network can adapt to a variety of data and to differentiate between the outcomes.

These activation function solve the following limitations of linear activation functions:

- They allow backpropagation because now the derivative function would be related to the input, and it's possible to go back and understand which weights in the input neurons can provide a better prediction.
- They 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 in a neural network.

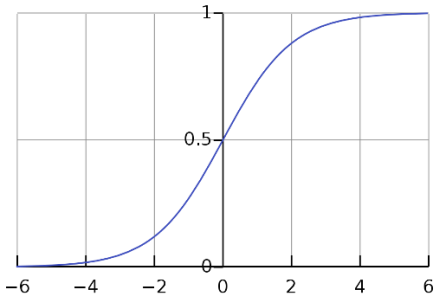
SOME OF THE COMMONLY USED NON- LINEAR ACTIVATION FUNCTION:

SIGMOID / LOGISTIC ACTIVATION FUNCTION:

This function takes any real value as input and outputs values in the range of 0 to 1.

The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0

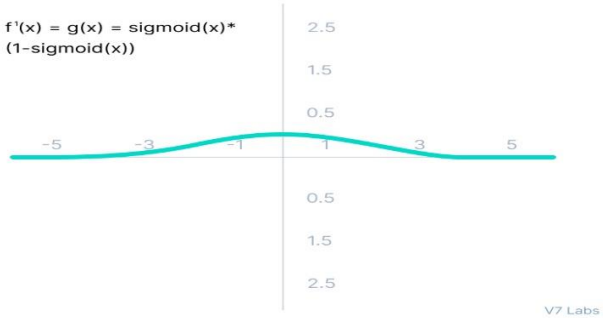
It is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.



Sigmoid / Logistic

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

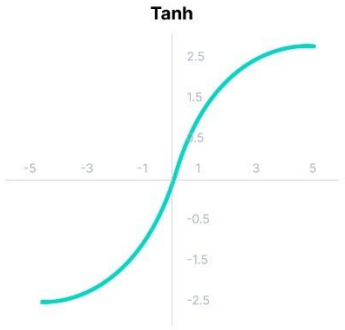
The differentiation graph of sigmoid function is as below:



As we can see from the graph that the gradient values are only significant for the range -3 to +3. It implies that for values greater than 3 or less than -3, the function will have very small gradients. As the gradient value approaches zero, the neural network ceases to learn and suffers from the ***Vanishing gradient*** **problem.**

TANH FUNCTION (HYPERBOLIC TANGENT):

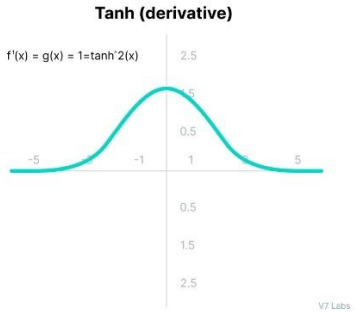
Tanh function is very similar to the sigmoid activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.



Advantages:

The output of the tanh activation function is Zero cantered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.

Usually used in hidden layers of a neural network as its values lie between -1 to 1; therefore, the mean for the hidden layer comes out to be 0 or very close to it. It helps in cantering the data and makes learning for the next layer much easier.

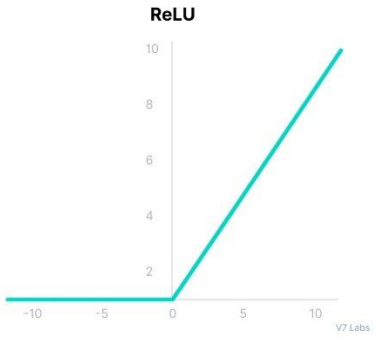


It also faces the problem of ***vanishing gradients*** like the sigmoid activation function. Plus, the gradient of the tanh function is much steeper as compared to the sigmoid function.

ReLU FUNCTION (Rectified Linear Unit)

ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.

The main catch here is that the ReLU function does not activate all the neurons at the same time. The neurons will only be deactivated if the output of the linear transformation is less than 0.



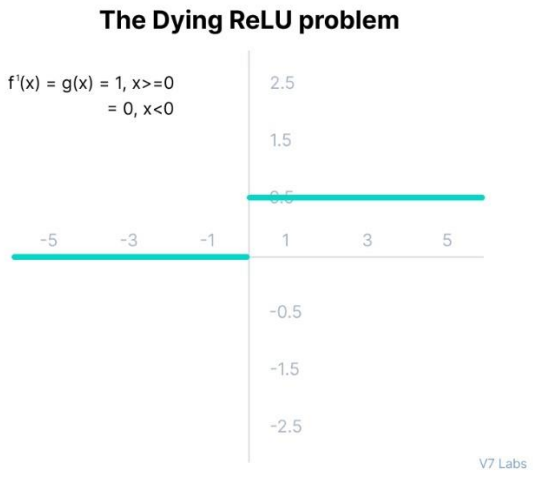
$$ReLU$$

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

The advantages of using ReLU as an activation function are as follows:

- Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh functions.
- ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear, non-saturating property.

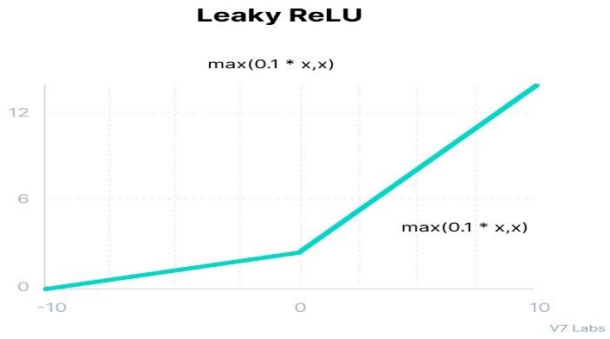
The limitations faced by this function are: The ***Dying ReLU*** problem explained below.

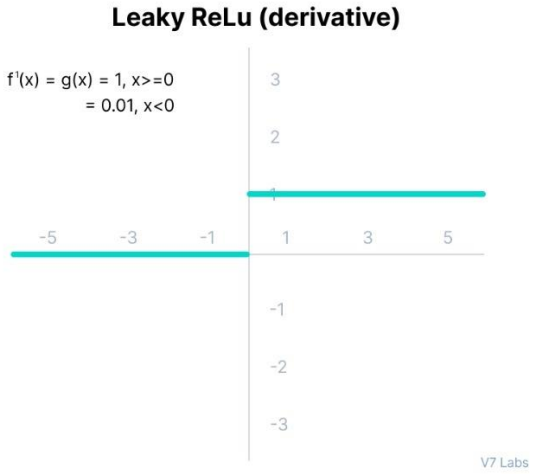


The negative side of the graph makes the gradient value zero. Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated. All the negative input values become zero immediately, which decreases the model's ability to fit or train from the data properly.

LEAKY ReLU FUNCTION

Leaky ReLU is an improved version of ReLU function to solve the Dying ReLU problem as it has a small positive slope in the negative area.



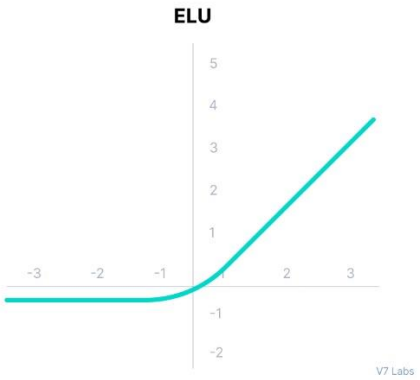


The advantages of Leaky ReLU are same as that of ReLU, in addition to the fact that it does enable backpropagation, even for negative input values.

By making this minor modification for negative input values, the gradient of the left side of the graph comes out to be a non-zero value. Therefore, we would no longer encounter dead neurons in that region.

EXPONENTIAL LINEAR UNITS (ELUS) FUNCTION

ELU is also a variant of ReLU that modifies the slope of the negative part of the function. It uses a log curve to define the negative values unlike the leaky ReLU functions with a straight line.



$$ELU$$

$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha\left(e^{x}-1\right) & for\ x < 0 \end{cases}$$

ELU is a strong alternative for ReLU because of the following advantages:

- ELU becomes smooth slowly until its output equal to -α whereas RELU sharply smoothens.
- Avoids dead ReLU problem by introducing log curve for negative values of input. It helps the network nudge weights and biases in the right direction.

The limitations of the ELU function are as follow:

- It increases the computational time because of the exponential operation included
- Exploding gradient problem

SOFTMAX FUNCTION

SoftMax converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. The Softmax function is described as a combination of multiple sigmoids.

It calculates the relative probabilities. Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class. It is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification.

Let us assume that we have three classes, meaning that there would be three neurons in the output layer. Now, suppose that our output from the neurons are [1.8, 0.9, 0.68].

Applying the softmax function over these values to give a probabilistic view will result in the following outcome: [0.58, 0.23, 0.19].

The function returns 1 for the largest probability index while it returns 0 for the other two array indexes. Here, giving full weight to index 0 and no weight to index 1 and index 2. So the output would be the class corresponding to the 1st neuron (index 0) out of three.