Activation Function in Neural Networks

★ **In the process of building a neural network, one of the choices you get to make is what Activation Function uses in the hidden layer as well as at the output layer of the network.This article discusses some of the choices.**

Elements of a Neural Network

Input Layer: This layer accepts input features.It provides Information from the outside world to the network, no computation is performed at this layer,nodes here just pass on the information to the hidden layer.

Hidden layer: Nodes of this layer are not exposed to the outer world, they are part of the abstraction provided by any neural networks.The hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the output layer.

Output Layer: This Layer brings up the information learned by the network to the outer world.

What is an activation function and why use them?

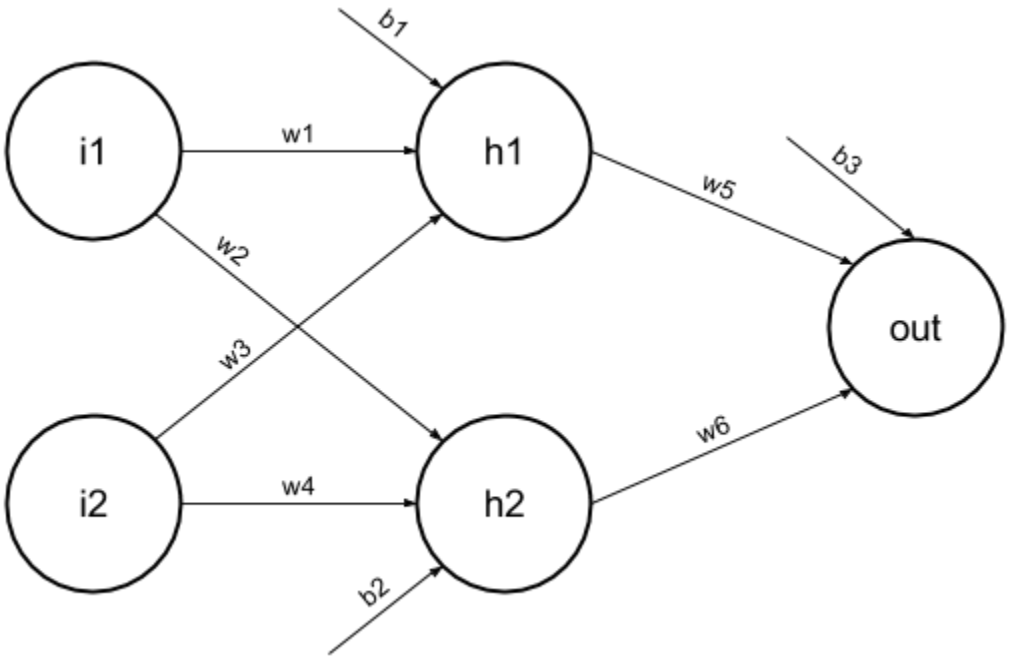
- The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it.
- The purpose of the activation functions is to introduce non-linearity into the output of a neuron.
- In a neural network, we should update the weights and biases of the neurons on the basis of the error at the output.This process is known as back-propagation.

Why do we need Non-linear activation function?

★ 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.

Mathematical Proof

Suppose we have a Neural network like this:-



Elements of the diagram are as follows:

Hidden layer i.e. layer 1:

z(1) = w(1)x + b(1) a(1)

Here,

- z(1) is the vectorized output of layer 1
- w(1) be the vectorized weights assigned to neurons of hidden layer i.e. w1,w2,w3 and w4
- X be the vectorized input features i.e. i1 and i2

- b is the vectorized bias assigned to neurons in hidden layer i.e. b1 and b2
- a(1) is the vectorized form of any linear function.

Layer 2 i.e. output layer:-

Note: Input for layer 2 is output from layer 1

z(2) = w(2)a(1) +b(2)
a(2) = z(2)

Calculate 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

Final output : z(2) = w*x + b
Which is again linear function

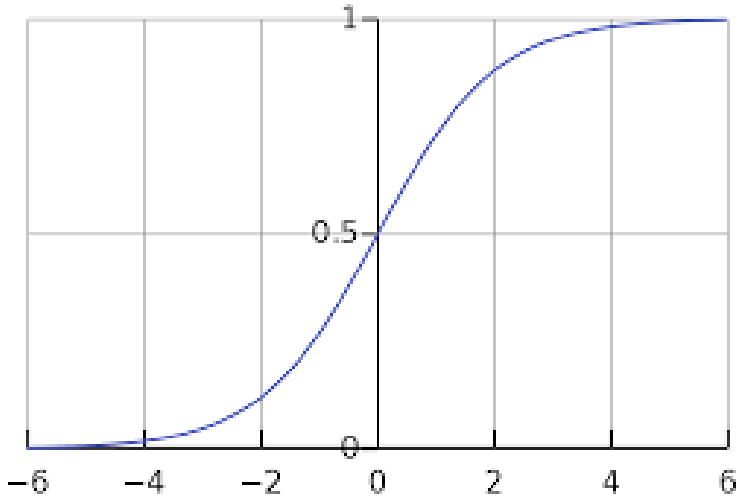
- This observation results again in a linear function even after applying a hidden layer, hence we can conclude that,doesn't matter how many hidden layers we attach in the neural net.
- All layers will behave the same way we attach in a neural net, all layers will behave the same way because the composition of two linear functions is a linear function itself.
- A non-linear activation function will let it learn as per the difference with respect to error.Hence we need an activation 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 the last layer is nothing but just a linear function of the first layer.
- **Range** : -inf to +inf
- **Uses** : Linear activation function is used at just one place i.e. output layer.
- **Issues** : If we differentiate linear function to bring non-linearity, result will no longer depend on input “x” and function will become constant,it won’t introduce any ground breaking behavior to our algorithm.

Sigmoid Function

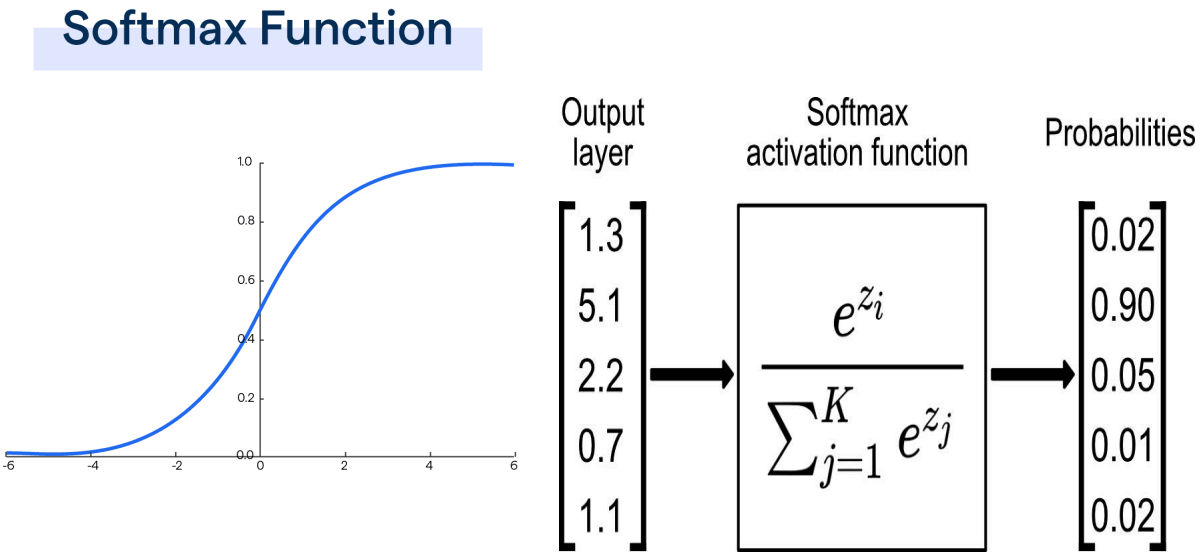


- It is a function which is plotted as an ‘**s**’ shaped graph.
- **Equation** : A = $1/(1+e^{-x})$
- **Nature** : Nonlinear. Notice that X values lie between -2 to 2,Y values are very steep.This means,small changes in x would also bring about large changes in the value of Y.
- **Value Range** : 0 to 1
- **Uses** : Usually used in the output layer of a binary classification ,where the result is either 0 or 1, as the value for the sigmoid function lies between 0

and 1 only, so the result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.

- **Vanishing Gradients** : If you use sigmoid function in hidden layers, at some point for low X value the gradient becomes zero.Due to gradients becoming zero ,the weight update does not happen.

Softmax 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
- **Equation** :

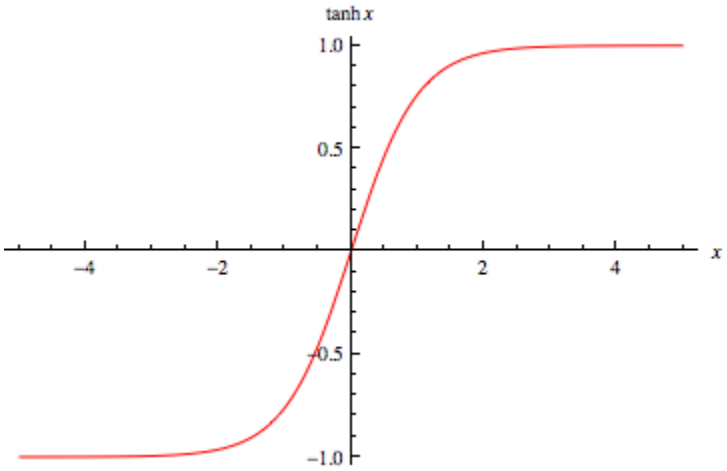
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- **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.

- **Value Rage** : 0 to 1
- **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.
- If your output is for multi-class classification then,Softmax is very useful to predict the probabilities of each class.

Tanh Function:-



The activation that works almost always better than the sigmoid function is the Tanh function also known as the **Tangent Hyperbolic function**.Its actually mathematically shifted version of the sigmoid function, Both are similar and can be derived from each other.

Equation :

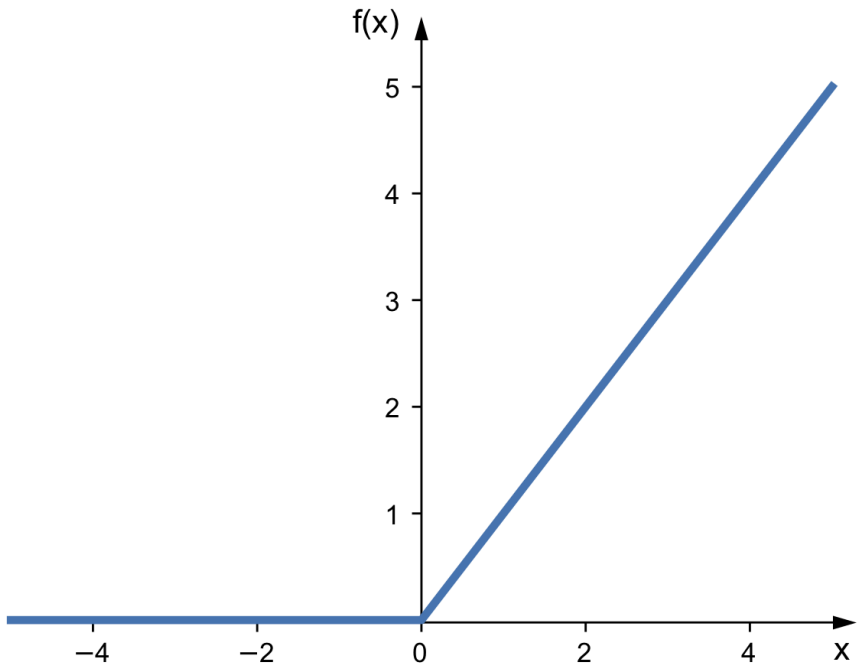
$$g(x) = \frac{e^x}{1 + e^x}$$

$$tanh(x) = 2g(2x) - 1$$

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

- **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 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(Rectified Linear Unit):-



It Stands for Rectified linear unit .It is the most widely used activation function.Chiefly implemented in hidden layers of Neural networks.

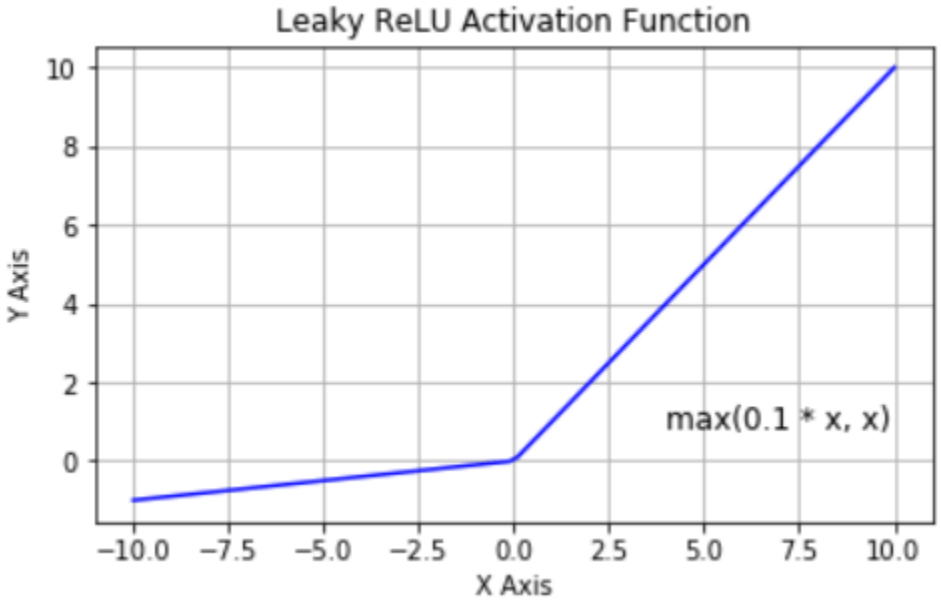
- **Equation** :

  $$A(x) = max(0, x)$$

  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 and used in hidden layers.
- **Pseudo code** :
  $max(0, x)$
  $If\ Input\ >\ 0$
  $x\ =\ Input$
  $else$
  $x\ =\ 0$

Leaky ReLU:-



Leaky ReLU is an activation function that introduces the property of nonlinearity to a deep learning model and solves the vanishing gradients issue.

- Equation :
  $$f(x)\ =\ max(0.01\ *\ x, x)$$
- Value Range : (-infinity to infinity)
- Nature : Leaky ReLU is a type of activation function based on a ReLU , but it has a small slope for negative values instead of a flat slope.
- Uses : Leaky ReLU is an activation function used in artificial neural networks to introduce non-linearity among the outputs between layers of a

neural network.This activation function was created to solve the dying ReLU problem using the standard ReLU function that makes the neural network die during training.