**Activation function**

Activation function decides, whether a neuron should be activated or not by calculating 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.

**Explanation**

We know, neural network has neurons that work in correspondence of weight, bias and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as back-propagation. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

**Why do we need Non-linear activation functions**

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.

**Types of Activation Function**

**1). Linear Function**

* **Equation :**Linear function has the equation similar to as of a straight line i.e. **y = ax**
* No matter how many layers, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
* **Range :** -inf to +inf
* **Uses : Linear activation function** is used at just one place i.e. output layer.
* **Issues :**If we will differentiate linear function to bring non-linearity, result will no more depend on input “x” and function will become constant, it won’t introduce any ground-breaking behavior to our algorithm.

**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.

**2). Sigmoid 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.

**3). Tanh Function :-**The activation that works almost always better than sigmoid function is Tanh function also knows 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) = 2/(1 + e-2x) - 1

OR

tanh(x) = 2 \* sigmoid(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 centering the data by bringing mean close to 0. This makes learning for the next layer much easier.

**4). RELU :-**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.

Problem: In the negative side of the graph, gradient value is zero. Due to this reason, during the backpropogation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated. This is taken care of by the ‘Leaky’ ReLU function.

**5). Leaky ReLU**

Leaky ReLU function is nothing but an improved version of the ReLU function. As we saw that for the ReLU function, the gradient is 0 for x<0, which would deactivate the neurons in that region.

Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for negative values of x, we define it as an extremely small linear component of x. Here is the mathematical expression-

**Equation :-**

f(x)= 0.01x, x<0

= x, x>=0

**6).Parameterised ReLU**

This is another variant of ReLU that aims to solve the problem of gradient’s becoming zero for the left half of the axis. The parameterised ReLU, as the name suggests, introduces a new parameter as a slope of the negative part of the function. Here’s how the ReLU function is modified to incorporate the slope parameter-

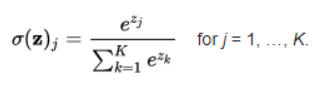
**Equation :-**

f(x) = x, x>=0

= ax, x<0

**7). Softmax Function :-**The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems.

Equation:



* **Nature :-**non-linear
* **Uses :-**Usually used when trying to handle multiple classes. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
* **Ouput:-**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.

**8). Binary Step** **Function**

If the input to the activation function is greater than a threshold, then the neuron is activated, else it is deactivated, i.e. its output is not considered for the next hidden layer.

Equation : f(x) = 1, x>=0

= 0, x<0

Uses: The binary step function can be used as an activation function while creating a binary classifier. As you can imagine, this function will not be useful when there are multiple classes in the target variable. That is one of the limitations of binary step function.

**9).Exponential Linear Unit**

Exponential Linear Unit or ELU for short is also a variant of Rectified Linear Unit (ReLU) that modifies the slope of the negative part of the function. Unlike the leaky relu and parametric ReLU functions, instead of a straight line, ELU uses a log curve for defining the negative values. It is defined as

Equation : f(x) = x, x>=0

= a(e^x-1), x<0

**10).Swish**

Swish is a lesser known activation function which was discovered by researchers at Google. Swish is as computationally efficient as ReLU and shows better performance than ReLU on deeper models.  The values for swish ranges from negative infinity to infinity. The function is defined as –

Equation : f(x) = x\*sigmoid(x)

f(x) = x/(1-e^-x)

A unique fact about this function is that swish function is not monotonic. This means that the value of the function may decrease even when the input values are increasing.

**Loss Function**

**Loss** is nothing but a prediction error of Neural Net. And the method to calculate the loss is called Loss Function. The Loss is used to calculate the gradients. And gradients are used to update the weights of the Neural Net. This is how a Neural Net is trained.

# Mean Squared Error

**MSE** loss is used for regression tasks. As the name suggests, this loss is calculated by taking the mean of squared differences between actual(target) and predicted values.

## Example

For Example, we have a neural network which takes house data and predicts house price. In this case, you can use the MSE loss. Basically, in the case where the output is a real number, you should use this loss function.

# Binary Cross entropy

**BCE** loss is used for the binary classification tasks. If you are using BCE loss function, you just need one output node to classify the data into two classes. The output value should be passed through a sigmoid activation function and the range of output is (0 – 1).

## Example

For example, we have a neural network that takes atmosphere data and predicts whether it will rain or not. If the output is greater than 0.5, the network classifies it as rain and if the output is less than 0.5, the network classifies it as not rain. (it could be opposite depending upon how you train the network). More the probability score value, the more the chance of raining.

While training the network, the target value fed to the network should be 1 if it is raining otherwise 0.

# Categorical Cross entropy

When we have a multi-class classification task, one of the loss function you can go ahead is this one. If you are using CCE loss function, there must be the same number of output nodes as the classes. And the final layer output should be passed through a softmax activation so that each node output a probability value between (0–1).

## Example

For example, we have a neural network that takes an image and classifies it into a cat or dog. If the cat node has a high probability score then the image is classified into a cat otherwise dog. Basically, whichever class node has the highest probability score, the image is classified into that class.

For feeding the target value at the time of training, we have to one-hot encode them. If the image is of cat then the target vector would be (1, 0) and if the image is of dog, the target vector would be (0, 1). Basically, the target vector would be of the same size as the number of classes and the index position corresponding to the actual class would be 1 and all others would be zero.

# Sparse Categorical Crossentropy

This loss function is almost similar to CCE except for one change.

When we are using SCCE loss function, you do not need to one hot encode the target vector. If the target image is of a cat, you simply pass 0, otherwise 1. Basically, whichever the class is you just pass the index of that class.