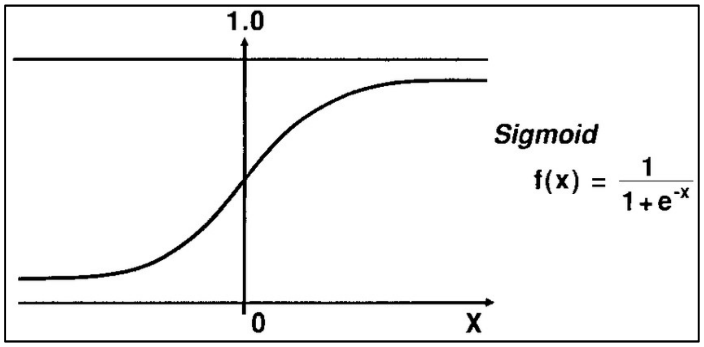
Activation functions –

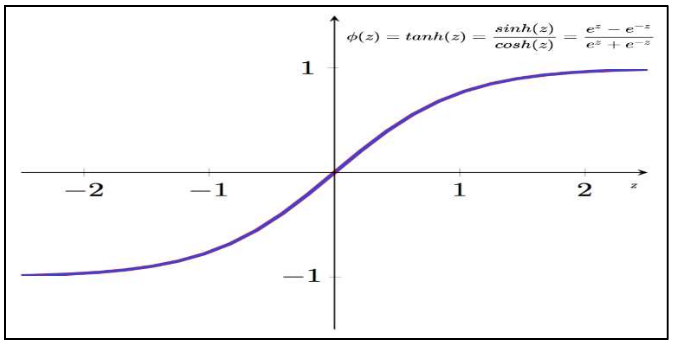
The activation function is applied to every node of the neural network to convert the linear relation of (weights * inputs + bias) into a non-linear one. If it is not applied, then the model will give the output tantamount to a linear regression. So, to induce the non-linearity and to capture non-linear relations of the input data (which is always the case) we introduce activation functions. The output of activation function is then sent as input to the subsequent layer. One import property which is cardinal for activation functions is that it should be differentiable. This is required for the consideration of back-propagation. Different types of activation functions are -

1. **Sigmoid** – This is also called logistic function or the 's' function. It always gives values between 0 and 1 as per the formula mentioned in the below picture. It is advisable not to use this function in the hidden layers, it is suitable for the output layer where segmentation needs to be done for two classes. This is used to be one of the most import activation functions of yesteryears.



   a. Advantages –
      i. It is easy to understand theoretically.
      ii. It is having a very smooth gradient and thus the smooth output values.
      iii. It gives values between 0 and 1.
   b. Dis-advantages –
      i. It has exponential power, so from the computation perspective, it is quite expensive.
      ii. The derivate of this function ranges between 0 to 0.25. This has a profound impact in back propagation. If a network has multiple layers, the change (delta for weights) in one epoch that reaches the first layer of neurons is very less numerically. This causes the longer time in convergence. This problem is also known by the name of 'vanishing gradient' problem.
      iii. It is not a zero centric function, due to this updates that are made to gradients swing heavily in both the sides.
      iv. Overall convergence of sigmoid is slower.

2. **Tan h – Hyperbolic tangent –** This is another activation function which is used in the neural networks. It ranges from -1 to 1. It appears in shape very close to that of sigmoid function. It is used in the hidden layer.



a. Advantages –
   i. It is zero centered, thus the gradients do not fluctuate wildly in the both the sides.
   ii. Like sigmoid, it is having a very smooth gradient and thus the smooth output values.
   iii. It gives values between -1 and 1 and it is steeper than sigmoid.
b. Dis-advantages –
   i. It has exponential power, so from the computation perspective, it is quite expensive.
   ii. It also suffers from vanishing gradient problem.

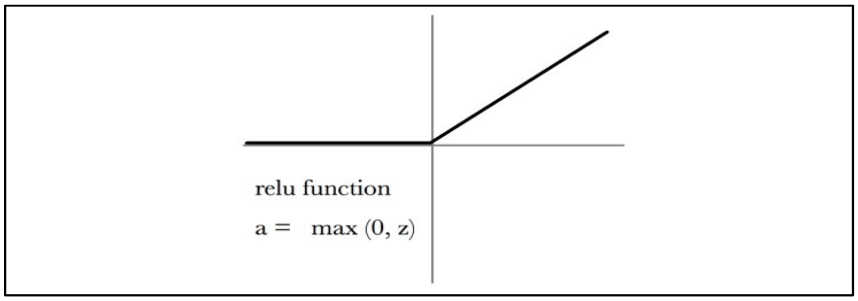
3. **<u>RELU – Rectified linear unit-</u>** This function has gained very popularity in the recent times due to its fast rate of convergence. It is very simple function and is free from some of the problems associated with the sigmoid and tanh functions. Though its name contains word 'linear' but it is not a linear function.

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \qquad \text{RELU}$$

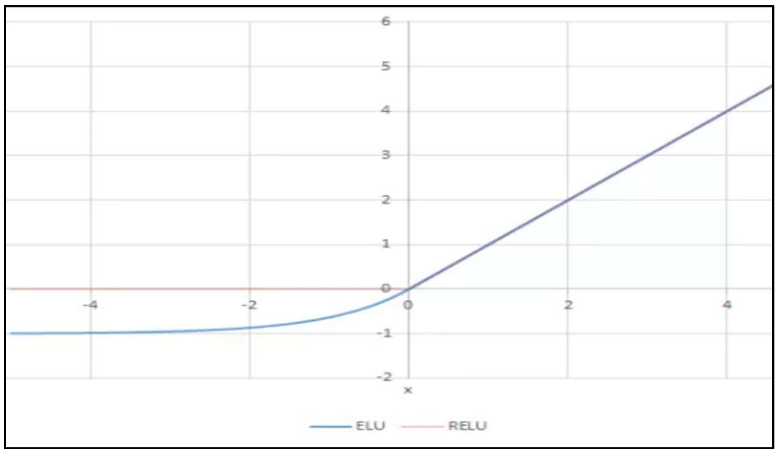
$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \qquad \text{Derivative of RELU}$$

As it is clear from the above equation of its derivative, it is either 0 or 1.

a. Advantages –
   i. It is easy to understand theoretically.
   ii. It is computationally very light.
   iii. It takes care and solves problem of vanishing gradient/gradient saturation.
b. Dis-advantages –
   i. Its derivate has either 0 or 1, which means it turns of some neurons in the network causing the problem of dead neurons.
   ii. It can only the used in the hidden layer and not in the last layer.
   iii. It is not a zero centric function.
   iv. As its range is from zero to infinity, it has it is marred with exploding gradient problem.



4. **ELU – Exponential linear unit –** It is like RELU for the positive values. The difference lies in the negative values, as shown in the below equations.



$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$$ ELU

$$f'(x) = \begin{cases} 1\ , & x > 0 \\ \alpha . e^x\ , & x \leq 0 \end{cases}$$ Derivative of ELU

There is one additional positive factor α (generally picked between .1 to .4) which is selected to be multiplied with the negative values so that after the application of this activation function it does not have zero as the output and thus averts problem of dead neurons.

a. Advantages –
   i. It is easy to understand theoretically.
   ii. Unlike RELU, it can compute negative values too.
   iii. It is computationally very light.
   iv. The mean of the output is close to zero, so it is a zero centric function.
   v. It takes care and solves problem of vanishing gradient and the problem of dead neurons in RELU.

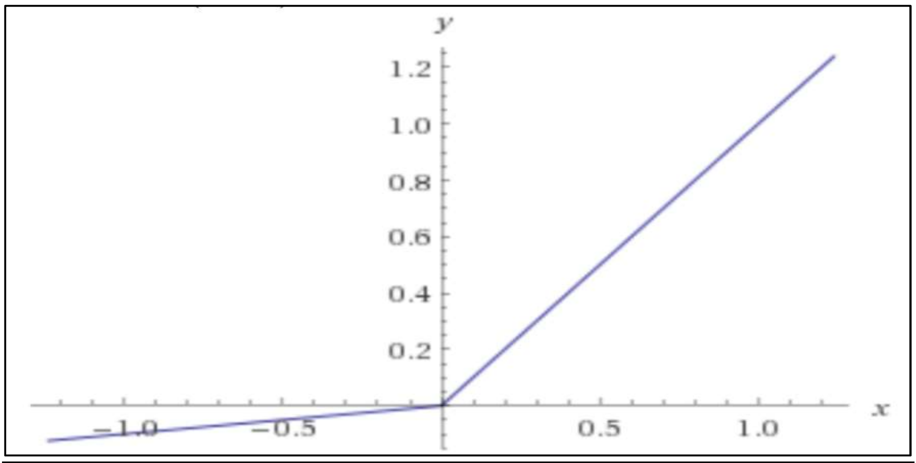
b. Dis-advantages –

i. It can only the used in the hidden layer and not in the last layer.
ii. Compared to RELU, it is expensive from computational point of view for negative values.
iii. The alpha value is decided by the user and not by the neural network.
iv. Same as RELU, it has exploding gradient problem.

5. **Leaky RELU-** This function is very similar to RELU function. It has only a factor α multiplied with the negative values as per the below equations. The value of α is 0.01.

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$ ELU

$$f'(x) = \begin{cases} 1 \ , & x > 0 \\ \alpha \ , & x \leq 0 \end{cases}$$ Derivative of ELU



a. Advantages –
   i. It is easy to understand theoretically.
   ii. Unlike RELU, it can compute negative values too.
   iii. It is computationally light than sigmoid and tanh
   iv. It takes care and solves problem of vanishing gradient/gradient clipping/gradient saturation and the problem of dead neurons in RELU.

b. Dis-advantages –
   i. It can only the used in the hidden layer and not in the last layer.
   ii. Compared to RELU, it is expensive from computation point of view.
   iii. The alpha value is decided by the user and not by the neural network.
   iv. Same as RELU, it has exploding gradient problem.

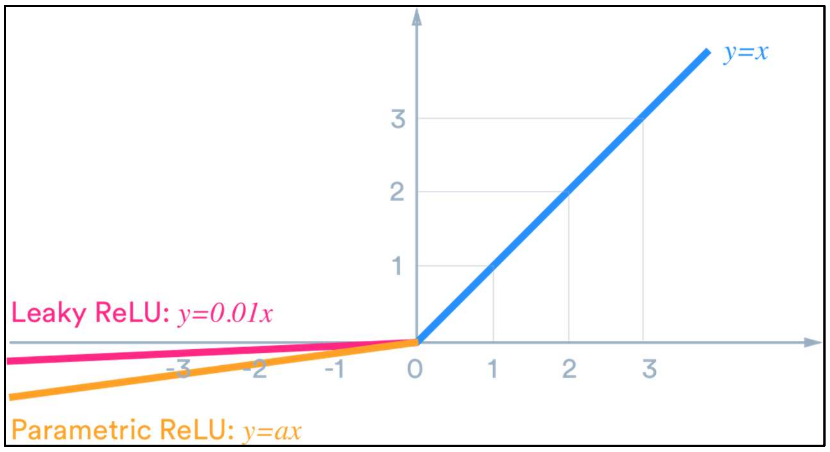
6. **Parametric RELU or P-RELU -** This function is again derived from RELU function. It has only a factor α , (which is a learnable parameter and the model try to learn it with respect to gradient or slope) is multiplied with the negative values as per the below equations.

$$f(x) = \begin{cases} x,\ x > 0 \\ \alpha x,\ x \le 0 \\ where\ \alpha\ is\ a\ learnable\ parameter \end{cases}$$

P RELU

$$f'(x) = \begin{cases} 1\ ,\ x > 0 \\ \alpha\ ,\ x \le 0 \\ where\ \alpha\ is\ a\ learnable\ parameter \end{cases}$$

Derivative of P RELU

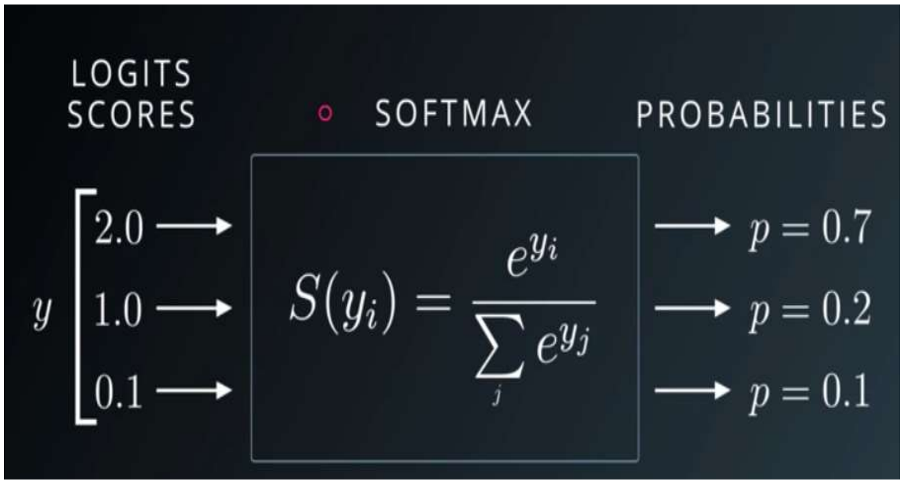


a. Advantages –
   i. It is easy to understand theoretically.
   ii. The alpha value is not decided by the user by is learned in the model training.
   iii. Unlike RELU, it can compute negative values too.
   iv. It is computationally light than sigmoid and tanh
   v. It takes care and solves problem of vanishing gradient/gradient clipping/gradient saturation and the problem of dead neurons in RELU.
   vi. In the negative side, it is sort of linear function.

b. Dis-advantages –
   i. It can only the used in the hidden layer and not in the last layer.
   ii. Compared to RELU, it is expensive from computation point of view.
   iii. Same as RELU, it has exploding gradient problem.

7. **SoftMax –** This function is usually used in the output layer when the classification needs to be done for more than two classes. In this function, probabilities for all the outcomes are computed against all the possible classes that are present in the model and the highest probability is given as output for the model. This function gives output between 0 to 1 as this is

the range of the probability for any class. This function is also know as multinomial logistic regression or maximum entropy classifier.



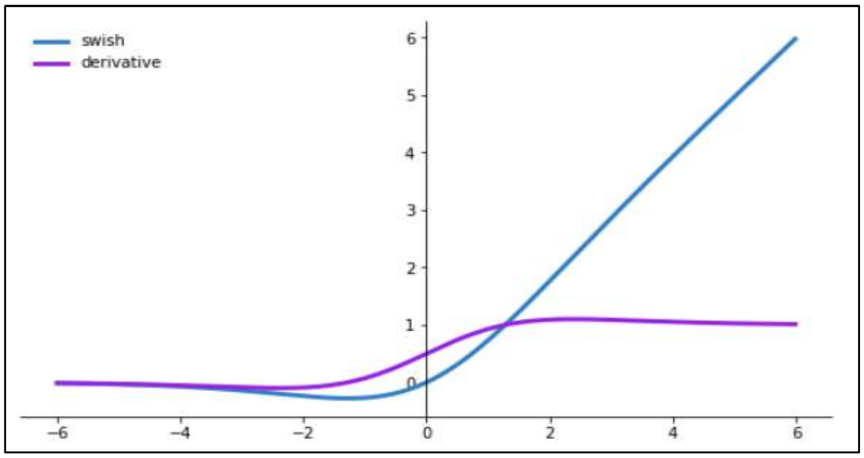
a. Advantages –
   i. It is used for multi-class classification.
   ii. The output lies between 0 and 1. The sum of all the output probabilities is equal to 1.

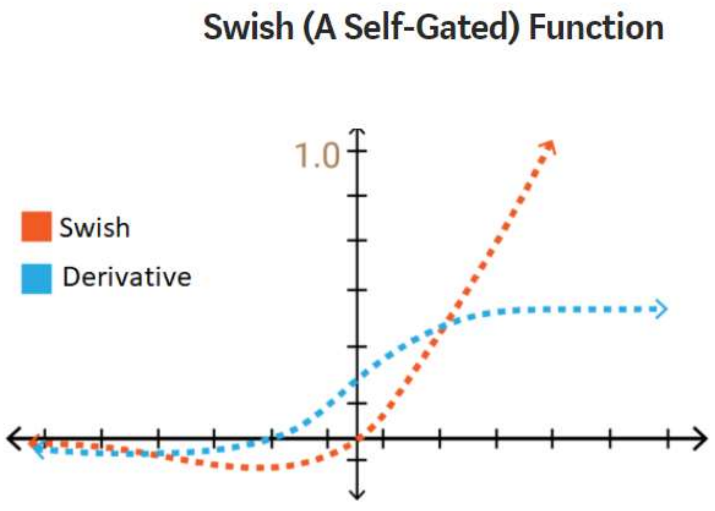
b. Dis-advantages –
   i. It cannot compute properly if the data has null values.
   ii. It is used in the output layer and not in the hidden layer.

8. **Swish – Self gated activation function –** It was developed by researchers working with Google brain. It is theoretically better than RELU and avoids the trap of vanishing gradient problem.

$$f(x) = x[\frac{1}{1+exp(-x)}]$$ **Swish**

$$f'^{(x)} = f(x) + \left[\frac{1}{1+exp(-x)}\right] * (1 - f(x))$$ **Derivate of Swish**





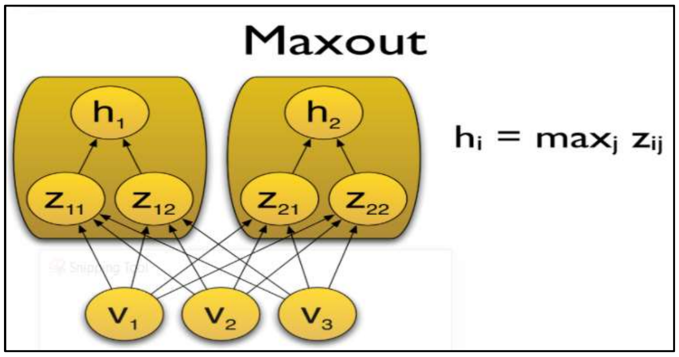
Swish Function and Derivative

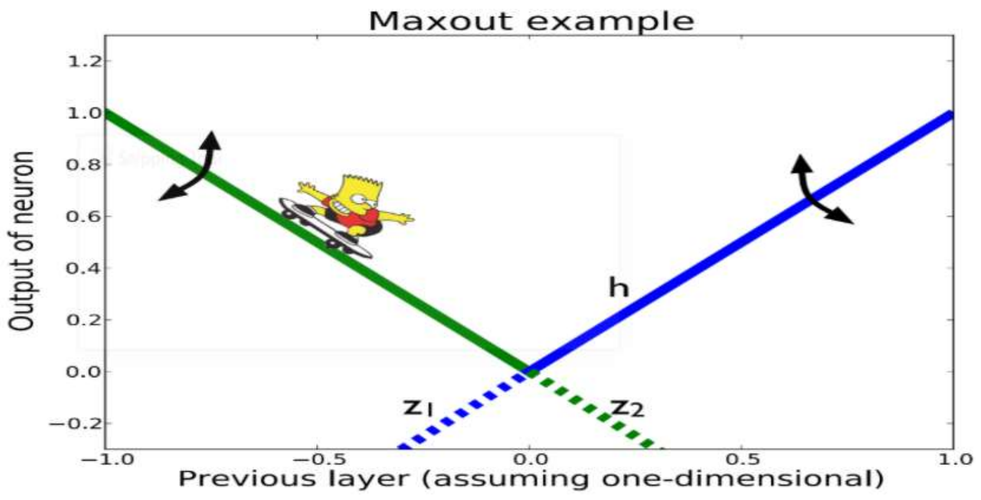
a. Advantages –
   i. Performs better than RELU.
   ii. Unlike RELU, it can deal with negative inputs.
   iii. It is free from vanishing gradient problem.
   iv. It will always give a smooth output. There is no steepness in the gradients like the RELU has when the input value is close to 0.
   v. A unique property of swish function is that even though the input can be increasing but the output may not necessarily increase but can decrease. Thus, for a very large output, there is no saturation of the gradients, means it is free from exploding gradient problem.

b. Dis-advantages –
   i. Takes more time to compute compare to RELU.
   ii. Cannot be used in the output layer.

9. **MaxOut –** This function is used by Ian Goodfellow, creator of GANs. He created this function to be used with the dropout layer.

$$\boldsymbol{f(x) = \max(w_1 x_1 + b_1,\ w_2 x_2 + b_2\ )}$$





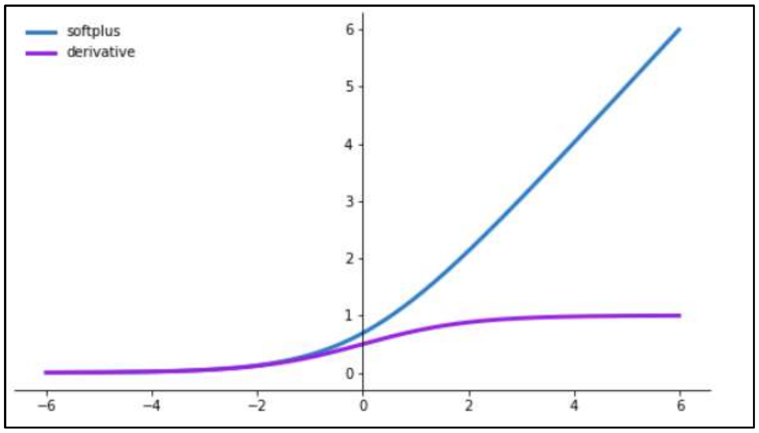
a. Advantages –
   i. It does not depend upon the pre-defined scenarios.
   ii. It is quite light from computation point of view, as it is a linear combination of weights, input and biases.
   iii. It takes care of the vanishing gradient problem.
   iv. It learns itself form the model and adjust itself based on the weights and biases that are being trained in the model.
   v. It takes care of the negative values.
   vi. It is free from dying neuron problem.
b. Dis-advantages –
   i. It requires double the number of parameters to be trained for each neuron.

10. **SoftPlus–** This function is used by Ian Goodfellow, creator of GANs. He created this function to be used with the dropout layer.

$$\boldsymbol{f(x) = ln(1 + exp(x)\,)} \qquad \textbf{Softplus}$$

$$\boldsymbol{f'(x) = (1/(1 + exp(-x))} \qquad \textbf{Derivate of Softplus}$$

Surprisingly, the derivate of the softplus is a sigmoid function as can be viewed in the below graph.



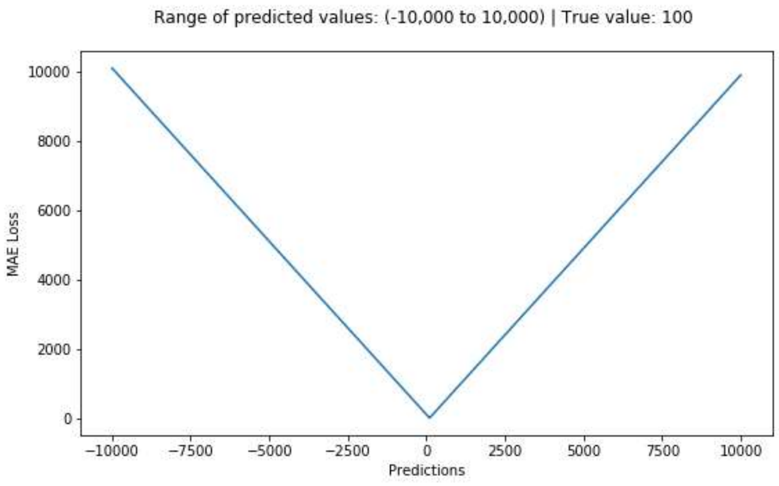
a. Advantages –
    i. It is smoother function compared with RELU. Gradients are smooth.
b. Dis-advantages –
    i. It is computationally costlier than RELU.
    ii. Its use is discouraged by Ian Goodfellow in his book as he found that during his research, though theoretically softplus looks promising then RELU but the later performed well. (reference 6.3.3 Other hidden units - http://www.deeplearningbook.org/contents/mlp.html)

Loss functions –

Loss functions are the functions which calculate the degree of difference between the model output ($\hat{y}$) with the actual value. The duty of this function is to let optimizer know how much the model is currently deviating or giving error from reaching near to the original input data. Along with activation and optimizer functions, loss function constitutes three pillars of back propagation (which itself is the backbone of Neural networks). **The primary task for the loss function is to give huge error for the incorrect classification and low error for the correct classification.** Major types of loss functions are -

1. **L1 loss –** This is called the **"least absolute deviation"**. The absolute value of the error is calculated from the output that the model gives.

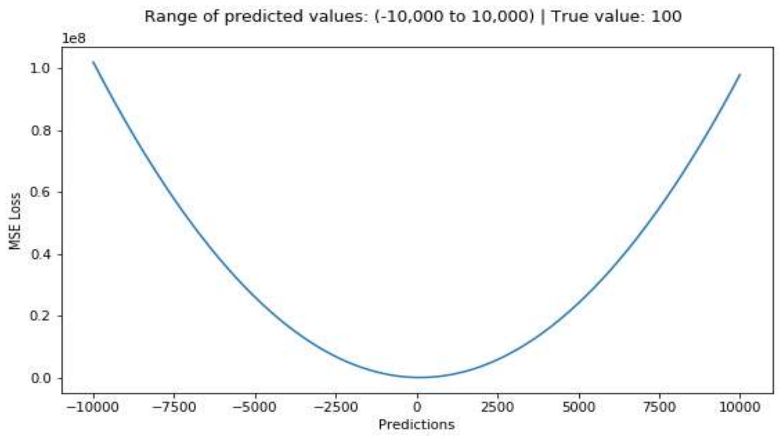
$$\boldsymbol{L1 = \sum_{i=0}^{n} |y - \hat{y}\,|}$$



a. Advantages –
    i. It is simple to compute and computationally light.
    ii. It is not affected by the presence of outliers.
b. Dis-advantages –
    i. Due to the steep and constant slope, the gradient is constant throughout the range of input data. Even for a smaller value, the gradient will be larger.

2. **L2 loss–** This is called the **"least squared errors**". It gives the square of the errors between the predicted values and the original values.

$$\boldsymbol{L2} = \sum_{i=0}^{n} (\boldsymbol{y} - \ \widehat{\boldsymbol{y}})^2$$



a. Advantages –
   i. It is simple to compute and computationally light.
   ii. If the outliers are not present in the data, it converges fast.
   iii. For this loss function, the gradient is larger for a larger input value and consequently tries to reach zero.
b. Dis-advantages –
   i. It is affected by the outliers as the errors will be squared.

3. **Huber loss–** In this loss function, the attempt has been made to remove the disadvantages of L2 loss function when dealing with outliers and incorporating the advantages of L1 loss function. In this function, we need to set a value δ which is threshold value for the outliers. The output is compared with the real input value and if the loss is less than the threshold, the first part of the below equation (1) is used otherwise, the second part (2) is used.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \quad \text{.........1} \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \quad \text{..........2} \end{cases}$$

a. Advantages –
   i. It has all the advantages of L1 and L2 losses.
   ii. It can deal with the input data having outliers.
   iii. It deals with the problem associated with the L1 losses of having large gradients which are constantly generated. For these cases, it considers the calculations mentioned in equation no 1.
b. Dis-advantages –
   i. δ is something which model do not learns itself, but it is provided form outside based on the rationale. To train it in model, parameter tuning needs to be done which can takes time.

4. **Pseudo Huber losses–** It is not a very popular function. It is used for extra smoothening for the gradient.

$$S = \sum_{i=1}^{n} \delta^2 \left(\sqrt{1 + \left(\frac{y_i - f(x_i)}{\delta}\right)^2} - 1 \right)$$

a. Advantages –
   i. It has all the advantages of L1 and L2 losses.
   ii. It has smoother gradient.
b. Dis-advantages –

   i. δ is something which model do not learns itself, but it is provided form outside based on the rationale. To train it in model, parameter tuning needs to be done which can takes time.

5. **Hinge loss–** It is used when we are trying to model a classifier. It is maximum margin classifier.

$$f(x) = max\ [\ 0\ , (1 - t * y)]$$

where 't' is the number of classes for classification.

6. **Cross entropy loss–** It is used for binary classification problems. The output value will range between [0,1].

$$L(y, \hat{y}\ ) = -\frac{1}{N}\sum_{i=0}^{N}[\ y_i(log(\hat{y}_i) + (1 -\ y) * (1 - log(\hat{y}_i))\ ]$$

where 'N' is the number of classes for classification.

In the above formula, either of the terms will be zero and thus will give output between 0 and 1.

7. **Sigmoid Cross entropy loss–** It is again very similar to cross entropy loss function. The log is replaced by sigmoid function. The output value will range between [0,1].

$$L(y, \hat{y}\ ) = -\frac{1}{N}\sum_{i=0}^{N}[\ y_i(sigmoid(\hat{y}_i) + (1 -\ y_i) * (1 - sigmoid(\hat{y}_i))\ ]$$

where 'N' is the number of classes for classification.

In the above formula, either of the terms will be zero and thus will give output between 0 and 1.

8. **SoftMax Cross entropy loss–** It is used when we are trying to model a classifier. It is maximum margin classifier.

$$L = -\mathrm{lo\,g}\left[\frac{e^{f_{y_i}}}{\sum_J e^{f_j}}\right]$$

Optimizer –

1. **Gradient Descent**
2. **Stochastic gradient descent**
3. **Batch gradient descent**
4. **Mini batch gradient descent**
5. **Adam**