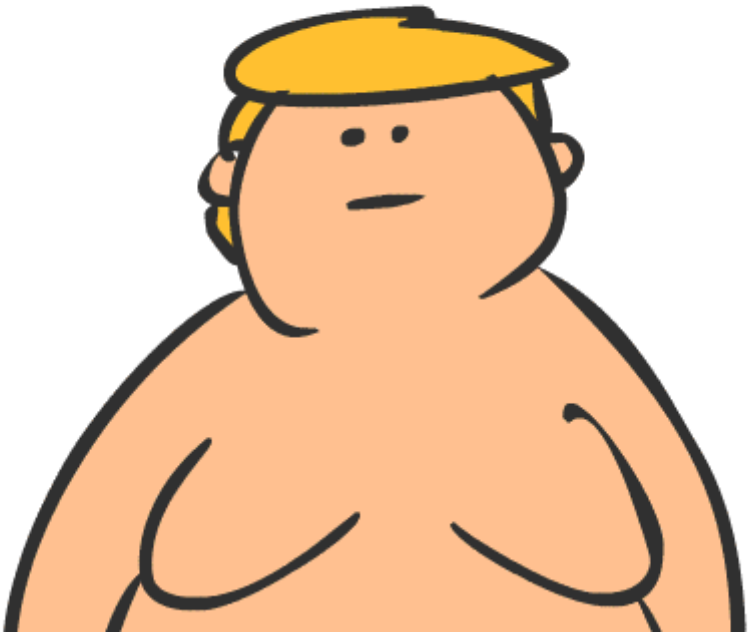
skaran Following

in read · Listen

Save

Lost function and its type !!!



Reduce the loss for better modelling

Lets begins with ….

What is loss ?

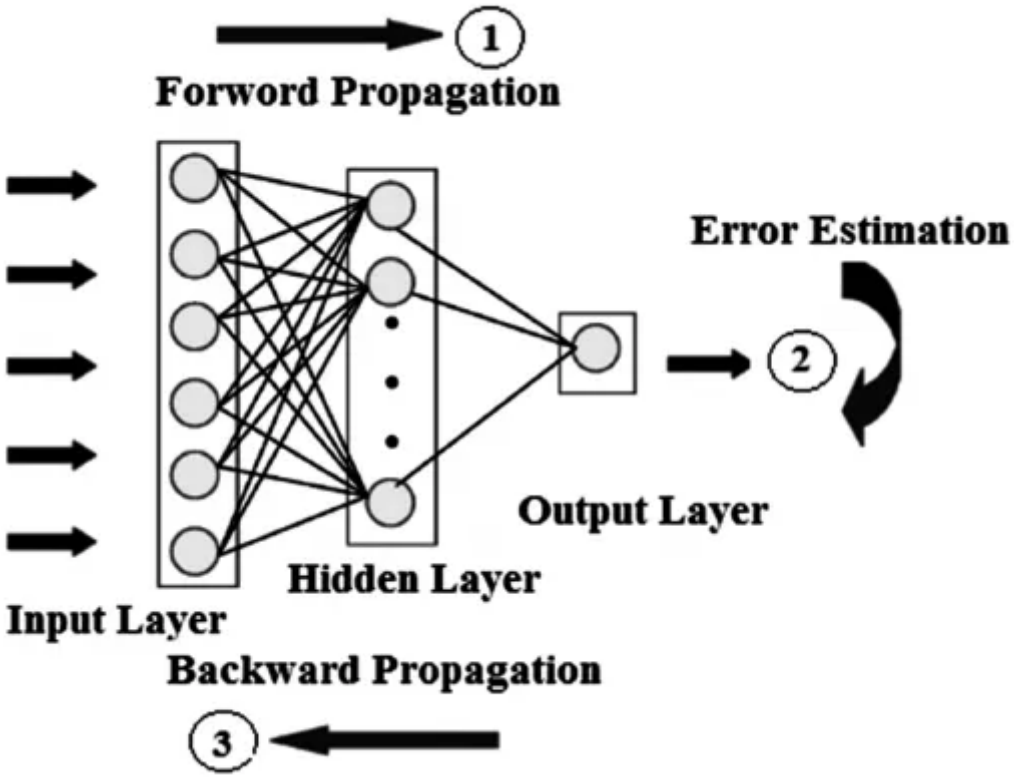
> '**Loss**' helps us to understand how much the predicted value differ from actual value

Function used to calculate the loss is called as "Loss function"

Loss function is a method of ev *algorithm models your dataset"*. If your predictions are totally ll output a higher number. If

they’re pretty good, it’ll output a lower number. As you tune your algorithm to try and improve your model, your loss function will tell you if you’re improving or not.

“Loss functions are helpful to train a neural network”



Working of Neural Nets

Types of Loss function:

1. Regression Loss Function
2. Binary Classification Loss Functions
3. Multi-class Classification Loss Functions

1. Regression Loss Function:

Regression models deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of room.Loss function used in regression problem are called “**Regression Loss Function**”.

Most used Regression loss function are below,

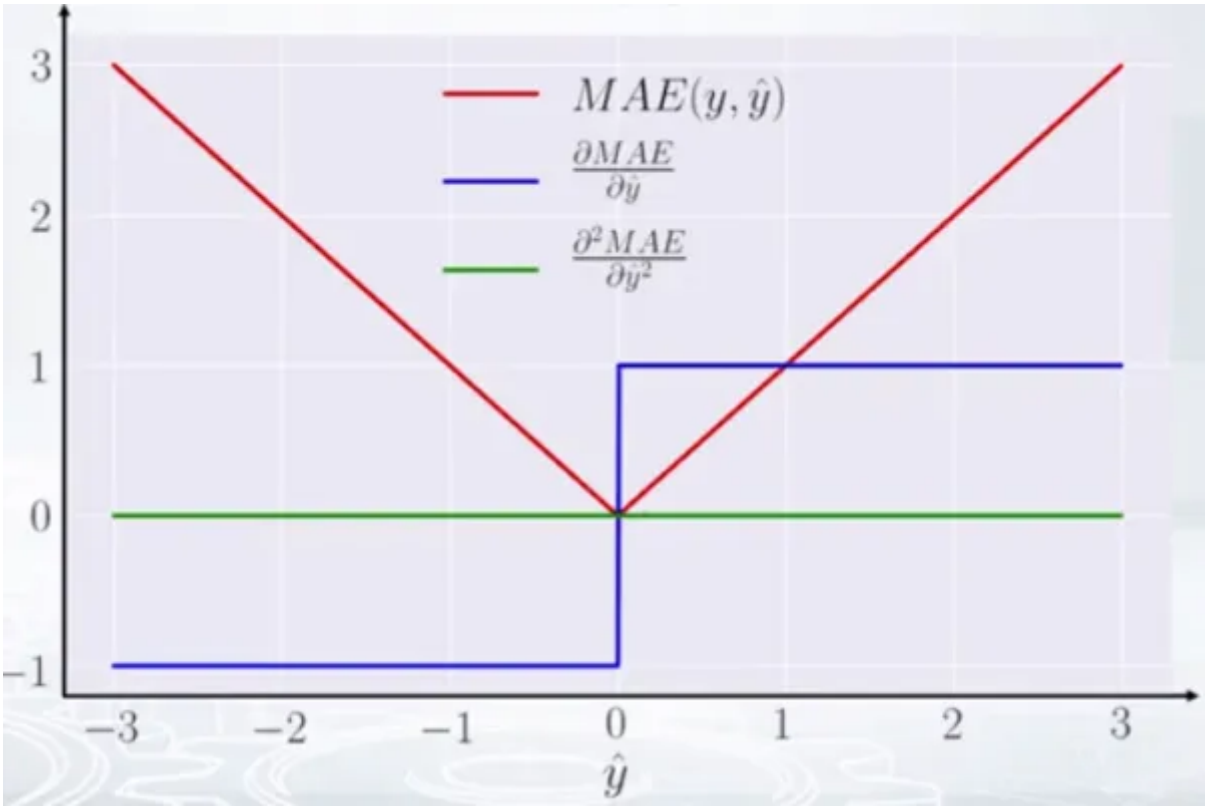
1.1 Mean Absolute Error (MAE) / L1 Loss function:

- ***Mean absolute error,*** is measured as the average of sum of absolute differences between predictions and actual observations.
- It measures the average "***magnitude***" of errors in a set of predictions, "without considering their "***directions***".
- Robust to outlier.
- MAE output ranges between 0 and +inf.
- Derivatives of MAE are not continuous, making it inefficient to find the solution.

> MAE is also known as "Least Absolute Deviations" or "LAD"

$$Loss = \frac{\sum_{i=0}^{n} |Y_i - \hat{Y}_i|}{n}$$

MAE Loss



MAE function, Derivatives (I ,II order)

Code Snippet in Python:

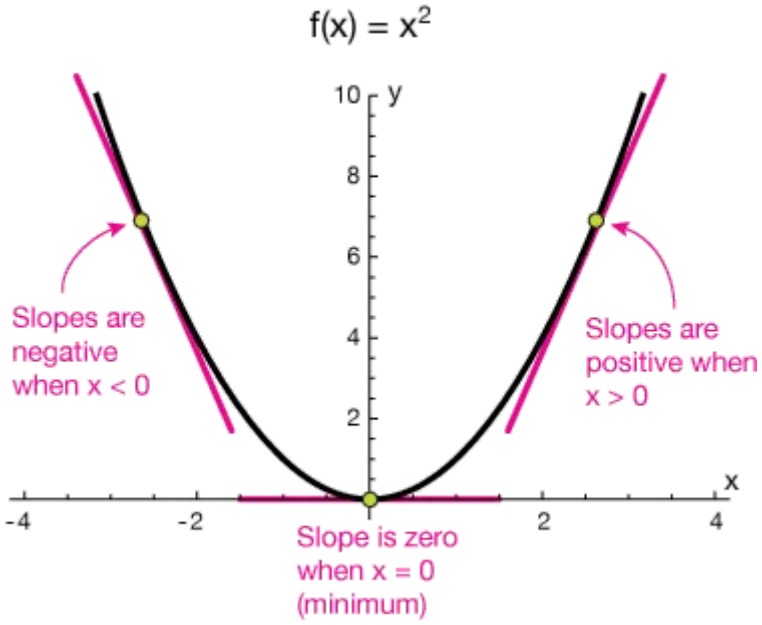
```python
def mae(predictions, targets):
    differences = predictions - targets
    absolute_differences = np.absolute(differences)
    mean_absolute_differences = absolute_differences.mean()
    return mean_absolute_differences
```

1.2 Mean Square Error (MSE) / L2 Loss function:

- *Mean Square error*, is measured as the average of sum of squared differences between predictions and actual observations.
- It measures the average "*magnitude*" of errors in a set of predictions, "without considering their "*directions*".
- **Less robust to outliers.**
- **MSE loss is stable then MAE**
- L2 loss output ranges between 0 and +inf.
- Derivatives of MSE are continuous, making it efficient to find the solution.

"MSE is the most commonly used regression loss function"

$$Loss = \frac{\sum_{i=0}^{n}(Y_i - \hat{Y}_i)^2}{n}$$



X²=(Yi-Y^i)², Function and Derivative of MSE

Code Snippet in Python:

```
def MSE(y_predicted, y):
        squared_error = (y_predicted - y) ** 2
        sum_squared_error = np.sum(squared_error)
        mse = sum_squared_error / y.size
        return(mse)
```

1.3 Root Mean Square Error (RMSE) / RMS Error:

- ***Root Mean Square error** is the extension of MSE* - measured as the average of square root of sum of squared differences between predictions and actual observations.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Code Snippet in Python:

```
def RMSE(y_predicted, y):
        squared_error = sqrt((y_predicted - y)**2)
        sum_squared_error = np.sum(squared_error)
        mse = sum_squared_error / y.size
        return(mse)
```

1.4 Mean Bias Error:

- MBE is less commonly used in practice.
- MBE is similar to MSE with the only difference that we don't take absolute values.Meaning we are considering the direction which not considered in MSE and MAE.
- *Caution :* Positive and negative errors could cancel each other out at times hence need to be handled with high care.Due to this reason its not been in use.

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}$$

Code Snippet in Python:

```
def MSE(y_predicted, y):
    error = (y_predicted - y)
    sum_squared_error = np.sum(squared_error)
    mbe = sum_squared_error / y.size
    return(mse)
```

1.5 Huber loss :

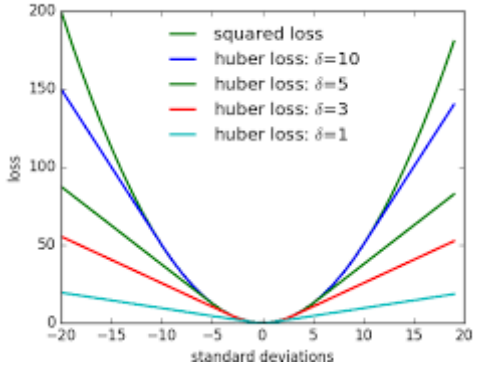
- Huber loss is less sensitive to outliers in data than the squared error loss.
- This function is quadratic for small values of error /residual, and linear for large values.
- What is that small value for the error to make the function quadratic depends on the "*hyperparameter*", δ (delta), which can be tuned.
- Huber loss approaches **MAE when δ ~ 0 and MSE when δ ~ ∞ (large numbers)**
- Choice of delta is important because it decide what value you are willing set as an outlier.
- Error >delta (***outlier cases***)are minimized with L1 (***Reason : Robust to outlier***), while error < delta (***inlier cases***) are minimized "appropriately" with L2.
- It is used in **Robust Regression, M-estimation and Additive Modelling**.

$$L_\delta = \begin{cases} \frac{1}{2}(y - f(x))^2, & if\ |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & otherwise \end{cases}$$

Quadratic

Linear

Huber function



$$Loss_\delta = \frac{\sum_{i=0}^{n} L_i}{n}$$

$$where,\ if\ |L_i - \hat{L}_i| < \delta\ ,\ L_i = \frac{(L_i - \hat{L}_i)^2}{2}$$

$$else\ L_i = \delta|L_i - \hat{L}_i| - \frac{\delta^2}{2}$$

Huber loss

Code Snippet in Python :

```
def Huber_loss(y_predict,y,delta):
    # derivative of quadratic for small values and of linear for large values
    if abs(Y[i] - m*X[i] - b) <= delta:
        error=(y_predict -y)**2
    else:
        error= delta * ((y_predict -y)) - (0.5 * delta)
    return error
```

2. Binary Classification Loss Function:

- Binary classification is a prediction algorithm where the output can be either one of two items, indicated by 0 or 1
- Output of binary classification algorithms is a prediction score (mostly).
- So the classification happens based on the threshold the value (default value is 0.5).If the prediction score > threshold then 1 else 0.

Most used binary classification loss function are below,

2.1 Binary Cross-Entropy or Log-loss error:

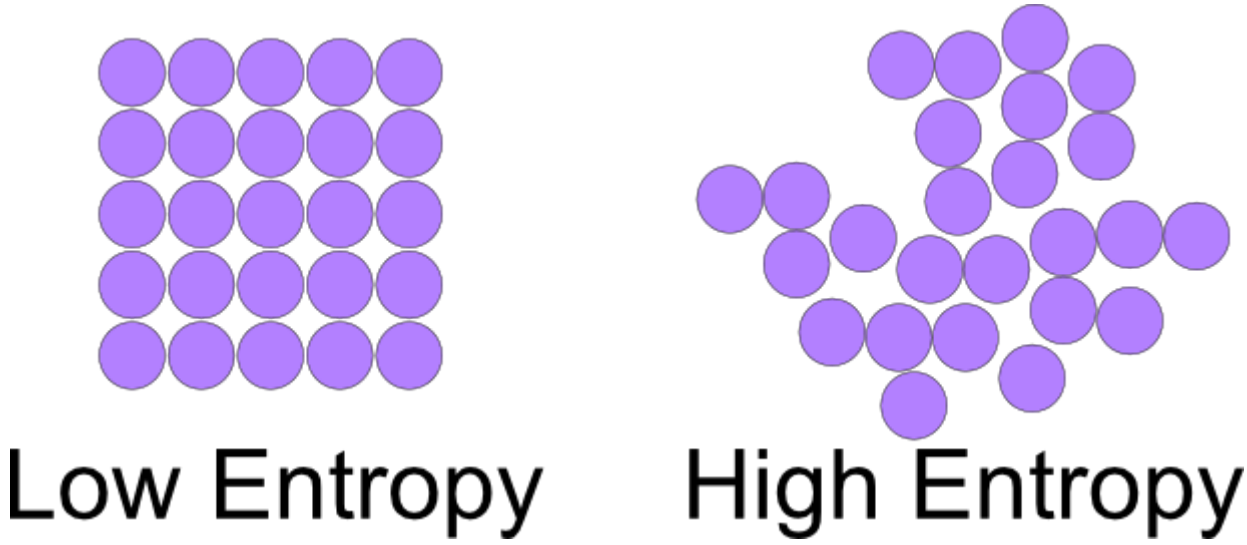
What is Entropy(S)*?*

> According to Wikipedia,***Entropy*** *indicates disorder or uncertainty / lack of order.*
>
> *"**Entropy** is a measure of the randomness in the information being processed"*

Mathematically, it is measured for a random variable X with probability distribution p(X) as follows

$$S = \begin{cases} -\int p(x).\log p(x)\,.dx, & \text{if } x \text{ is continuous} \\ -\sum_{x} p(x).\log p(x), & \text{if } x \text{ is discrete} \end{cases}$$

S represent entropy



- If Entropy value is high indicates greater uncertainty in the distribution
- If Entropy value is low indicates high certainty in the distribution

Reason for log in entropy : All events have the same probability(1/p) hence,it makes sense to use log(1/p) as a "*measure of information"*

Reason for negative sign:Entropy function *use* negative log(negative sign)to provide an ***easy metric for comparison.*** *Reason for negative sign is that the positive log of numbers < 1 returns negative values,* which is confusing to work with when comparing the performance of two models.

Since the probability value(p(x)) always lies between 0 and 1 function will return negative value.so by adding the negative sign value becomes positive.

Example : log(0.2) =-0.7 so to make it comparable do -log(0.2) = 0.7

At this point we understood- what is entropy? Now let discuss on Binary cross entropy.

According to MLcheatsheet,**Binary** *Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.*

"In other terms , ***Binary Cross-entropy loss / log loss aims to reduce the entropy of the predicted probability distribution in binary classification problems"***

$$Loss = (Y)(-log(Y_{pred})) + (1 - Y)(-log(1 - Y_{pred})$$

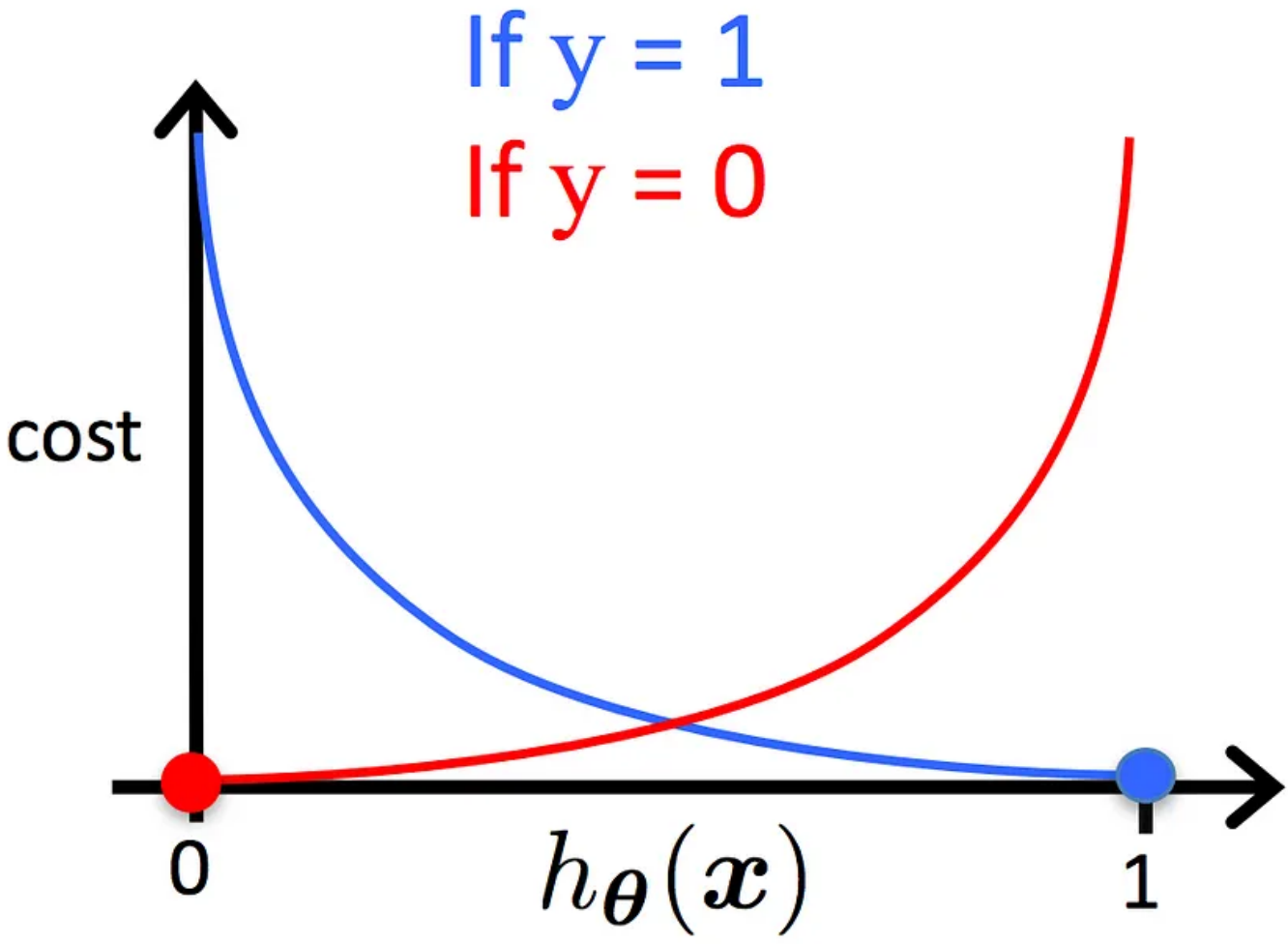
Remains when Y = 1

Removed when Y = 0

Remains when Y = 0

Removed when Y = 1

Ypred ==> p



Log Loss

The probability *(p/ Y pred)* can be calculated using sigmoid function.

$$S(z) = \frac{1}{1 + e^{-z}}$$

Why sigmoid function ?

The output value of sigmoid function ranges between 0 and 1. Hence this property make sigmoid function suitable for calculating the probability*(p)*.

Code Snippet in Python:

```
import math

def sigmoid(x):
  return 1 / (1 + math.exp(-x))


y_predicted =sigmoid(z)



def CrossEntropyloss(y_predicted, y):
    if y == 1:
      return -log(y_predicted)
    else:
      return -log(1 - y)
```

2.2 Hinge loss:

- Hinge loss is most popular loss function during pre-deep learning era.
- Hinge loss is often used for binary classification problems.
- This type of loss is primarily used in SVM classifiers where the target values are in the set {-1, 1}.

> *Note : When using hinge loss it is important to change the class label to -1 and +1.*

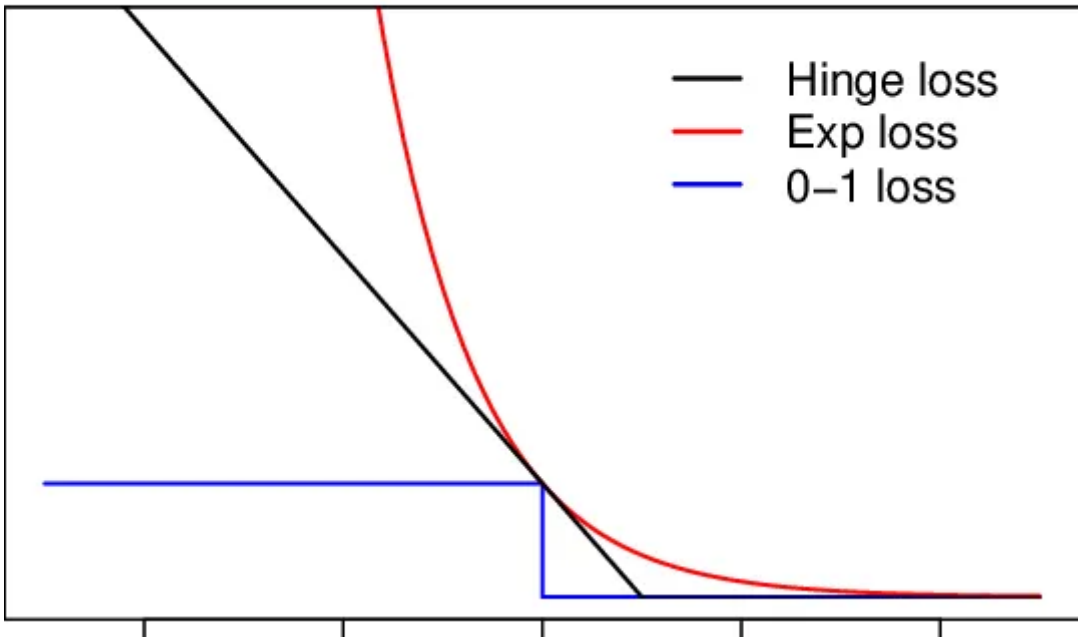
"According to Analytics Vidhya, ***Hinge Loss not only penalizes the wrong predictions but also the right predictions that are not confident.***"

$$\ell(y) = \max(0, 1 - t \cdot y)$$

t ==> Predicted output ; y ==> actual

- Hinge loss output the value ranges from -inf to 1

- Hinge loss is used for maximum-margin classification,.
- Though hinge loss is not differentiable, it's convex function which makes it easy to work with usual convex optimizers used in machine learning domain.



Code Snippet in Python:

```
def Hinge(y_predicted, y):
    return np.max(0, 1 - y_predicted * y)
```

3. Multi-class Classification Loss Function:

- *Multi-Class classification* are those predictive modeling problems where there are more target variables/class.
- It is just the extension of binary classification problem.

3.1 Multi-class Cross Entropy Loss:

Most time,*Multi-class Cross Entropy Loss* is used as a loss function.

- The generalised form of cross entropy loss is the multi-class cross entropy loss.

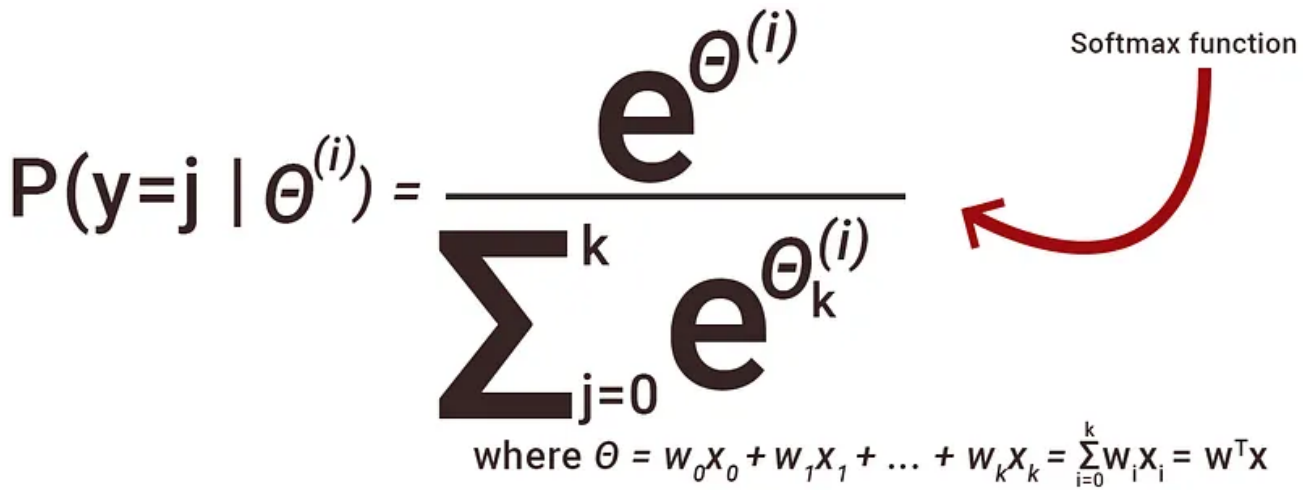
$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Fig 3.1

- M — No of classes
- y — binary indicator (0 or 1) if class label c is the correct classification for input o
- p — predicted probability score of input o belonging to class c

Reason for the usage of negative sign and log : "Refer log loss section"

The probability *(p o,c)* can be calculated using softmax function.



probability of y belonging to J(c) class given inputs(o)

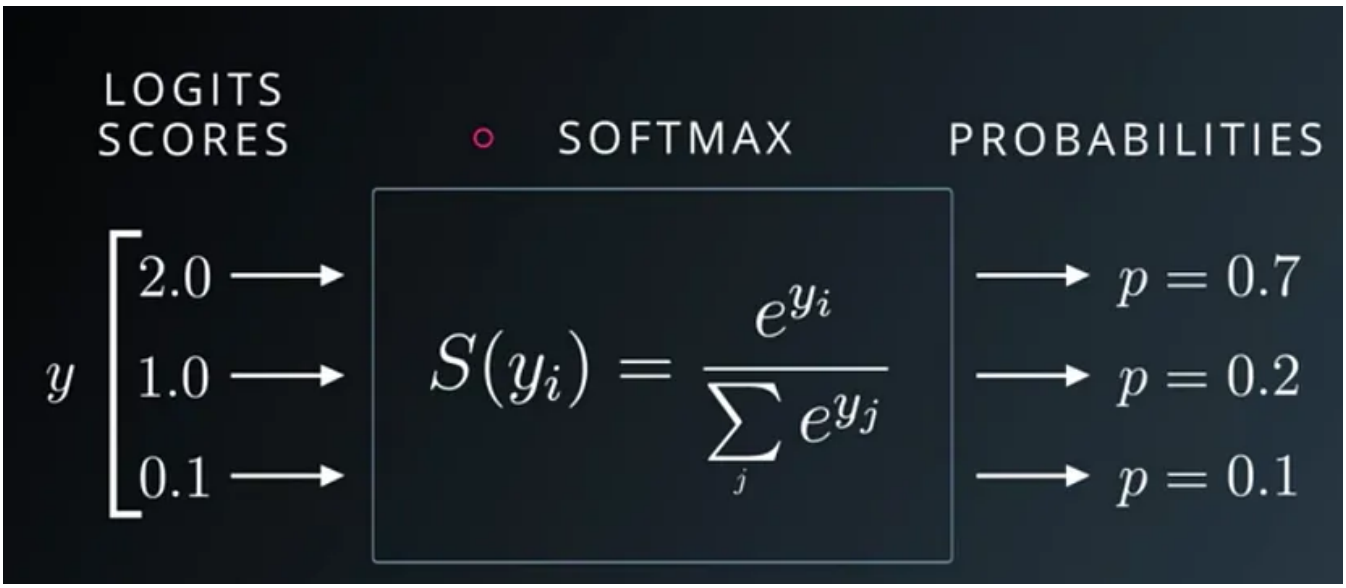
As per the fig 3.1 , J==>c (Class) and theta ==> o (inputs)

Why Softmax function ?

Softmax implements a vector of 'squashes' k-dimensional real value to the [0,1] range of k-dimensional, while ensuring that the cumulative sum is 1.

> "Softmax is implemented through a neural network layer just before the output layer. The Softmax layer

must have the same number of nodes as the output layer.” Google Developer’s Blog”



3.2 Kullback-Liebler Divergence (KL-Divergence):

- KL Divergence is a measure of how a probability one distribution differs from another distribution.

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

$$D_{KL}(P||Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx$$

1) for discrete function 2) for continuous function

KL Divergence is also know as “**Relative entropy**”.

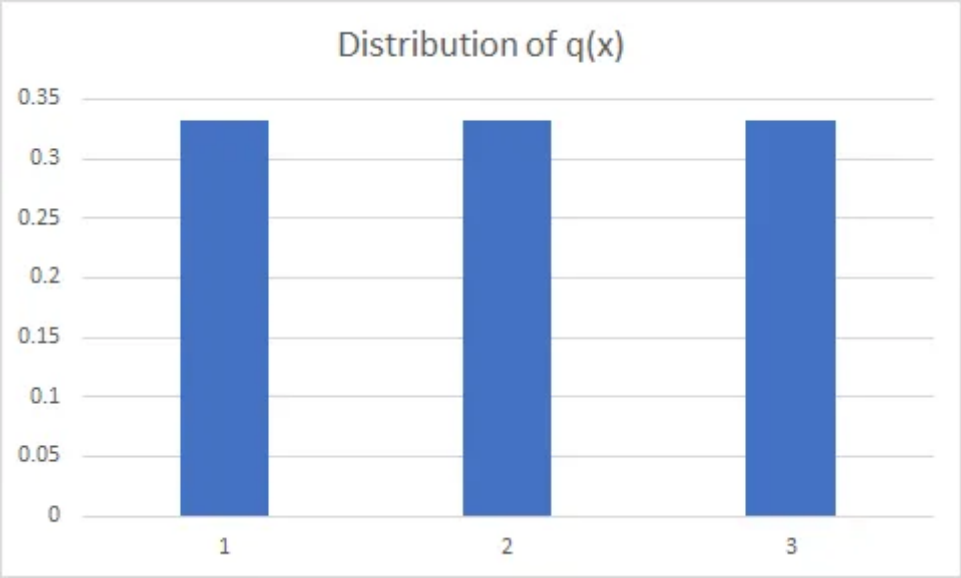
Dkl(P||Q) is read as Divergence from Q to P.

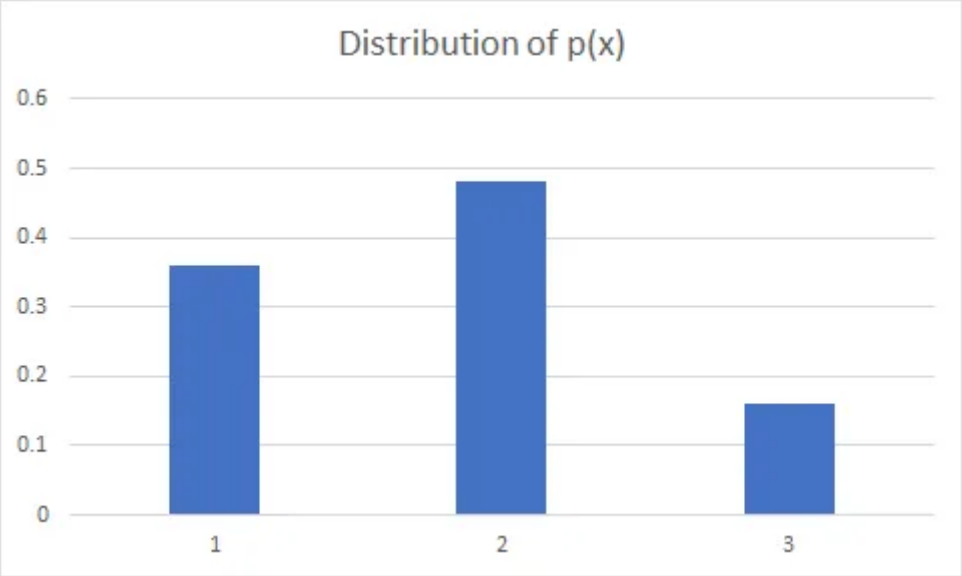
The example below is discrete random function.

x	Distribution of p(x)	Distribution of q(x)
0	0.36	0.333
1	0.48	0.333
2	0.16	0.333

Distribution of p(x) is from binominal distribution with p=0.4 and N=2

Distribution of q(x) is from uniform distribution with p=1/3





Let solve this ,

$$KL(P,Q) = \sum_i P_i \ln \frac{P_i}{Q_i}$$

Applying values to the above formula,

Dkl(P||Q) => 0.36 ln(0.36/0.333)+0.48 ln(0.48/0.333)+0.16 ln(0.16/0.333) = 0.086

Dkl(Q||P) =>0.333 ln(0.333/0.36)+0.333 ln(0.333/0.48)+0.333 ln(0.333/0.16)=0.096

> *How to interpret the above values of KL Divergence ?*

Dkl(P||Q) is interpreted as the information gain when distribution Q is used instead of distribution P .

Dkl(Q||P) is interpreted as the information gain when distribution P is used instead of distribution Q.

Information gain (IG) *measures how much "information" a feature gives us about the class.*

Now let's talk about few properties of KL Divergence,

1. Dkl(P||Q) !=Dkl(Q||P). Divergence from Q to P is not symmetric to Divergence from P to Q.

Eg: Dkl(P||Q) =0.086 ;Dkl(Q||P)=0.096

It clear from the example that Dkl(P||Q) not symmetric to Dkl(Q||P).

2. Dkl(P||P) =0 meaning both have identical distribution and not information is gained from other distribution.

3.The value of Dkl(P||Q) will be greater than or equal to zero.

So, the goal of the **KL divergence loss** is to approximate the **true probability distribution P** of our target variables with respect to the input features, given some **approximate distribution Q.This Can be achieved by** minimizing the Dkl(P||Q) then it is called **forward KL.** If we are minimizing Dkl(Q||P) then it is called **backward KL.**

Forward KL → applied in Supervised Learning

Backward KL → applied in Reinforcement learning

> *Why KL divergence is not considered as distance metric?*

Reason is simple KL divergence is not symmetric.

KL- Divergence is functionally similar to multi-class Cross entropy .

$$D_{KL}(P||Q) = -\sum_{x}\big(P(x).logQ(x) - P(x).logP(x)\big) = H(P,Q) - H(P,P)$$

$H(P,P)$ *is the entropy of* P

$H(P,Q)$ *is the cross* $-$ *entropy of* P *and* Q

Explaination

Conclusion:

Finally we come to the end of the discussion on Loss function. Discussed loss function are some of the important and popular loss function.There are other loss function which are in use as well.

Reference:

A Detailed Guide to 7 Loss Functions for Machine Learning Algorithms with Python Code	

Overview What are loss functions? And how do they work in machine learning algorithms? Find out in this article Loss... www.analyticsvidhya.com	

https://www.wikipedia.org/

Loss Function Classification Entropy Cross Entropy KI Divergence