Loss Functions in neural networks [Habiba Shera]

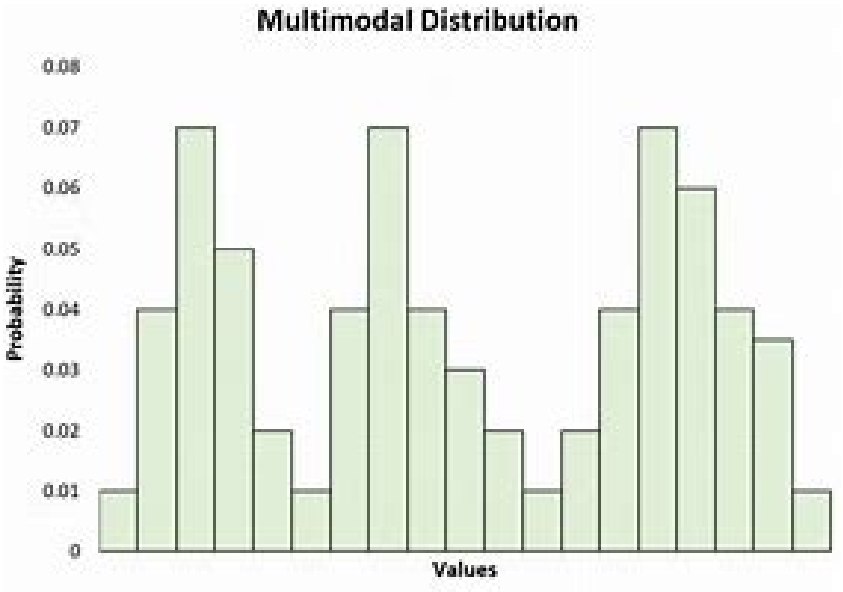
Loss function is a very important thing while creating a neural networks. It's the different between actual value and predicted value. It helps optimizer while updating weights on its backpropagation.

Note :

> The choice of loss function depends on the type of problem. knowing the difference between these types of loss functions is appropriate for each problem is important.

- **Mean Absolute Error (MAE):**
  - for regression problems.
  - *not sensitive towards outliers*
  - It can also be useful if you know that your distribution is multimodal

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$

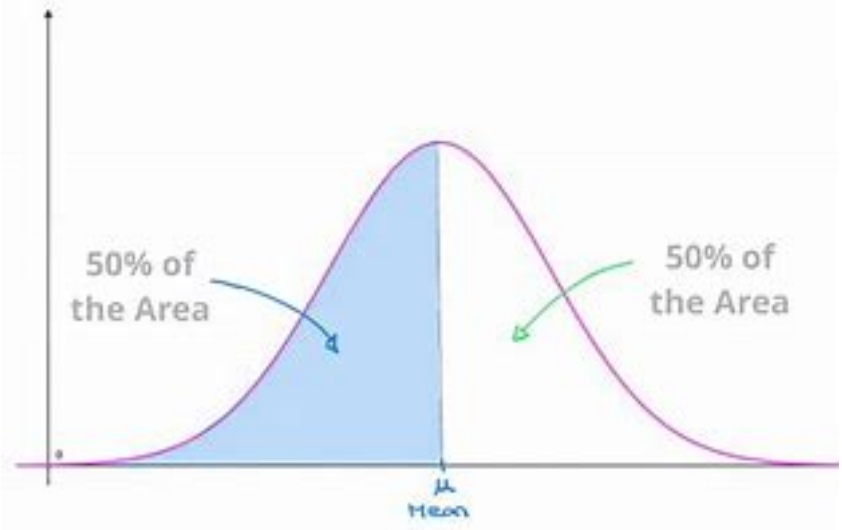



```
odel.compile(loss='mean_absolute_error', optimizer='adam')
```

Mean Squared Error (MSE):

- for regression problems.
- s*ensitive towards outliers*
- It can be useful if you know that your distribution is normally distributed

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$



Normal Distribution

```
model.compile(loss='mean_squared_error', optimizer='adam')
```

- **Huber Loss:**
  - for regression problems.
  - It's less sensitive to outliers than the MSE
  - It makes balancing the MSE and MAE together.
  - **for loss values less than (*δ*) delta, use the MSE, and for loss values greater than delta, use the MAE (**This way Huber loss provides the best of both MAE and MSE.)

$$L_\delta = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & if\ |(y-\hat{y})| < \delta \\ \delta((y-\hat{y}) - \frac{1}{2}\delta) & otherwise \end{cases}$$

Here, **(*δ*)**
delta → hyperparameter defines the range for MAE and MSE.

```
model.compile(loss=tensorflow.keras.losses.Huber(delta=1.5), optimizer='adam')
```

- Cross Entropy:
  - for classification model
  - A perfect model would have a log loss (cross entropy) of 0.

$$H = -\sum p(x) \log p(x)$$

- Binary Cross-Entropy
  - used in binary classification tasks
  - can be used in **Multi-label classification**
  - where the number of classes **M** equals **2**, cross-entropy can be calculated as:
  - Note1 : S**igmoid** is the only activation function compatible with the binary cross-entropy loss function. You must use it on the last block before the target block.
  - Note2 : The softmax activation function is the only one to guarantee that the output.

$$-(y \log(p) + (1 - y) \log(1 - p))$$

```
model.compile(loss='binary_crossentropy', optimizer='adam')
```

- Categorical Cross-Entropy
  - used in multi-class classification tasks.

- Note : **Softmax** is the only activation function recommended to use with the categorical cross-entropy loss function.

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

- **M**—number of classes (dog, cat, fish)
- **log**—the natural log
- **y**—binary indicator (0 or 1) if class label **c** is the correct classification for observation **o**
- **p**—predicted probability observation **o** is of class ***c***

$$L = \sum_{j=1}^{M} y_j \log(\hat{y}_j)$$

```
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

- **Sparse Categorical Cross-Entropy**
  - has the same loss function of **Categorical Cross Entropy**
  - The difference is :
    - If your **Y-True** are one-hot encoded, use categorical_crossentropy. Examples for a 3-class classification: **[1,0,0] , [0,1,0], [0,0,1]**
    - But if your **Y-True** are integers, use sparse_categorical_crossentropy. Examples for above 3-class classification problem: **[1] , [2], [3]**
  - One advantage of using sparse categorical cross-entropy is it saves time in memory as well as computation because it simply uses a single integer for a class, rather than a whole vector.

```
model.compile(optimizer = 'adam',loss='sparse_categorical_crossentropy', metrics =['accuracy'])
```

- **Hinge Loss**

- used when the target variable has 1 or -1 as class labels. (difference in the sign)
- Your labeled classes should be { -1, 1}

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

t={−1,1} is the label

```
model.compile(loss='hinge', optimizer='adam')
```

The most popular loss functions for deep learning **classification** models are **binary cross-entropy and sparse categorical cross-entropy.**

References:

- Loss Functions in Neural Networks (theaidream.com)
- An Introduction to Neural Network Loss Functions - Programmathically
- Artificial Neural Network in TensorFlow - GeeksforGeeks
- Loss Functions in Deep Learning: An Overview (analyticsindiamag.com)