

ALL about Loss function



▼ Table of contents

1. What is loss Function?
2. How to choose loss function?
 1. Regression Problem
 1. Mean Squared Error
 2. Mean Absolute Error Loss
 3. Huber Loss (Smooth Mean Absolute Error)
 3. Binary Classification Problem
 1. Binary Cross Entropy Loss
 2. Hinge Loss
 4. Multi-Class Classification Problem
 1. Categorical Cross Entropy
 2. Sparse Categorical Cross Entropy
 3. Kullback Leibler (KL) Divergence Loss

```
# import numpy, tensorflow, norm from scipy, matplotlib libraries
```

What is loss function?

It is a method of evaluating how well your algorithm models your dataset. If your predictions are totally wrong, your loss function will output a higher number. If they are pretty good, it will output a lower number. As you change pieces of your algorithm to try and improve your model, your loss function will tell you if you are getting close to your goal or getting away.

▼ Note:

A lot of the loss functions that you see implemented in machine learning can get complex and confusing. But if you remember the end goal of all loss functions—measuring how well your algorithm is doing on your dataset—you can keep that complexity in check.

There are many functions that could be used to estimate the error of a set of weights in a neural network.

We need a function that will reach to a set of weights after each iteration. These weights will result in improved performance and lesser loss value.

The loss function gets the weights based on historical training data.

We have a training dataset with one or more input variables and we require a model to estimate model weight parameters that best map examples of the inputs to the output or target variable.

Given input, the model is trying to make predictions that match the data distribution of the target variable. A loss function estimates how closely the distribution of predictions made by a model matches the distribution of target variables in the training data.

Cross Entropy

The error between two probability distributions is measured using cross-entropy.

When modeling a classification problem where we are interested in mapping input variables to a class label, we can model the problem as predicting the probability of an example belonging to each class. In a binary classification problem, there would be two classes, so we may predict the probability of the example belonging to the first class. In the case of multiple-class classification, we can predict a probability for the example belonging to each of the classes.

In the training dataset, the probability of an example belonging to a given class would be 1 or 0, as each sample in the training dataset is a known example from the domain. We know the answer.

We would seek a set of model weights that minimize the difference between the model's predicted probability distribution given the dataset and the distribution of probabilities in the training dataset. This is called the cross-entropy.

How to choose loss function?

Important

The choice of loss function is directly related to the activation function used in the output layer of your neural network. These two design elements are connected.

Think of the configuration of the output layer as a choice about the framing of your prediction problem, and the choice of the loss function as the way to calculate the error for a given framing of your problem.

Regression Problem

Regression is the task of predicting a continuous quantity. A regression algorithm may predict a discrete value, but the discrete value in the form of an integer quantity. Regression predictions can be evaluated using root mean squared error, whereas classification predictions cannot.

Note:

For regression problem the output layer consist of one node with a linear activation unit.

So, based on this we can select a loss function. The loss function for such case is Mean Squared error (MSE)

▼ Mean Squared Error

The Mean Squared Error (MSE) is perhaps the simplest and most common loss function, often taught in introductory Machine Learning courses. To calculate the MSE, you take the difference between your model's predictions and the ground truth, square it, and average it out across the whole dataset.

The MSE will never be negative, since we are always squaring the errors. The MSE is formally defined by the following equation:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2$$

The below code is a working implementation of a function for calculating the mean squared error for a list of actual and a list of predicted real-values quantities.

```
# define a function named mse with parameter y_true, y_pred
```

```

# return mean of (y_true - y_pred)^2

# create a array for true label of random values of any size

# create a array for predicted label same size as compared to true label but values can be

# create a variable to store returned loss value from mse function

# print the mse value

y_true: [0.1 0.2 0.3 0.4 0.5 0.6]
Shape of y_true: (6,)
y_pred: [0.1 0.2 0.3 0.4 0.5 0.6]
Shape of y_pred: (6,)
Mean Squared Error: 0.0

```

From the above output you can see that the error we got is 1.5 because there is one predicted/target value is not matching with actual/true label

▼ Mean Absolute Error Loss

Mean Absolute Error (MAE) is also another important loss function in regression problem. It is defined as the average of the absolute difference between the actual and predicted values.

The formula for MAE:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |x_i - x|$$

MAE is more robust to outliers compared to MSE

it is computationally expensive as modulus error is complex to solve compared to square error.

The gradient becomes large even for small loss as gradient remains same during the process, which is not appropriate for learning. To fix this issues, dynamic learning rate can be used.

```
# import mean_absolute_error from sklearn library

# create a array for true label of random values of any size

# create a array for predicted label same size as compared to true label but values can be

# create a variable to store returned loss value from mean_absolute_error function

# print mean absolute error value

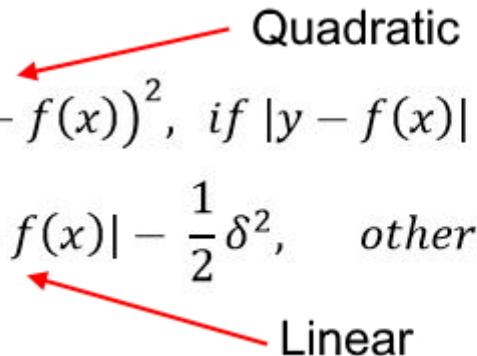
y_true: [ 3. -0.5  2.  7. ]
Shape of y_true: (4,)
y_pred: [2.5 0.  2.  8. ]
Shape of y_pred: (4,)
Mean Absolute Error: 0.5
```

▼ Huber Loss (Smooth Mean Absolute Error)

Huber loss plays an important role by combining both MSE and MAE. It changes the quadratic equation to linear, if the loss is higher. If the error is less than cutoff (epsilon), MSE is used and otherwise MAE can be used.

Formula for Huber Loss:

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


Quadratic
Linear

Huber loss curves around the minima which decreases the gradient, which is better compared to MAE as MAE has a constantly large gradient. This leads to missing minima at the end of training using gradient descent. On the other hand, Huber loss is less sensitive to outliers compared to the mean squared error loss.

Please note, the choice of delta is important as it helps to determine the outlier criteria.

```
# create a list for true label of random values of any size

# create a list for predicted label same size as compared to true label but values can be

# create a variable to store huber loss returned by tensorflow with delta = 1.0

# print huber loss value

y_true: [[0. 1.]
         [1. 1.]
         [1. 1.]]

Shape of y_true: (3, 2)

y_pred: [[ 0.  1.]
         [ 1.  1.]
         [-2. -2.]]

Shape of y_pred: (3, 2)

Huber Loss: tf.Tensor([0.  0.  2.5], shape=(3,), dtype=float32)
```

Binary Classification Problem

Binary classification is the simplest kind of machine learning problem. The goal of binary classification is to categorise data points into one of two buckets: 0 or 1, true or false, to survive or not to survive, blue or no blue eyes, etc.

Note:

For classification problem the output layer consist of one node with a sigmoid activation unit or softmax.

So, based on this we can select a loss function. The loss function for such case is Cross-Entropy

▼ Binary Cross Entropy Loss

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted

probability diverges from the actual label. So predicting a probability of 0.2 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.

Formula for binary cross entropy:

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^N - (y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

```
# create a list for true label of random values of any size

# create a list for predicted label same size as compared to true label but values can be

# create a variable named bin_cross_entropy which is a constructor of BinaryCrossentropy f

# create a variable to store bin_cross_entropy loss returned by tensorflow

# print bin_cross_entropy_value

y_true: [0 1 0 0]

Shape of y_true: (4,)

y_pred: [0.6 0.3 0.2 0.8]

Shape of y_pred: (4,)

BinaryCrossentropy Loss: 0.9882109761238098
```

▼ Hinge Loss

Hinge Loss is another loss function for binary classification problems. It is primarily developed for Support Vector Machine (SVM) models. The hinge loss is calculated based on “maximum-margin” classification.

This loss function is used if the target values are in the set (-1, 1). The target variable must be modified to have values in the set (-1, 1), which means if y has value as 0, it needs to be changed as -1.

Formula for Hinge Loss:

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

The hinge loss function tries to ensure the correct sign by assigning more error if there is a difference in the sign between the actual and predicted class values.

```
# create a list for true label of random values of any size

# create a list for predicted label same size as compared to true label but values can be

# create a variable named hinge_loss which is a constructor of Hinge from tensorflow

# create a variable to store hinge_loss loss returned by tensorflow

# print hinge_loss value

y_true: [[0. 1.]
 [0. 0.]]

Shape of y_true: (2, 2)

y_pred: [[0.6 0.4]
 [0.4 0.6]]

Shape of y_pred: (2, 2)

Hinge Loss: 1.2999999523162842
```

Multi-Class Classification Problem

A classification problem including more than two classes is called Multi-Class classification
example: classify a set of flowers which may be roses, lotus, lily, etc.

Multi-class classification makes the assumption that each sample is assigned to one and only one label

A flower can be either rose or lotus but not both simultaneously.

such as converting the N number of classes to N number binary columns representing each class. By doing so, we can use a binary classifier for Multi Classification problems.

The cross-entropy is then summed across each binary feature and averaged across all examples in the dataset.

Formula for using binary cross entropy for multi-class classification problem.

$$\text{logloss} = - \frac{1}{N} \sum_i^N \sum_j^M y_{ij} \log(p_{ij})$$

- N is the number of rows
- M is the number of classes

▼ Categorical Cross Entropy

```
# create a list for true label of random values of any size
```

```
# create a list for predicted label same size as compared to true label but values are pro
```

```
# create a variable named cat_cross_entropy which is a constructor of CategoricalCrossentr
```

```
# create a variable to store cat_cross_entropy loss returned by tensorflow
```

```
# print cat_cross_entropy_value
```

```
y_true: [[0 1 0]
```

```
[0 0 1]]
```

```
Shape of y_true: (2, 3)
```

```
y_pred: [[0.05 0.95 0. ]  
         [0.1  0.8  0.1 ]]
```

```
Shape of y_pred: (2, 3)
```

```
CategoricalCrossentropy Loss: 1.1769392490386963
```

▼ Sparse Categorical Cross Entropy

As seen in categorical cross entropy we convert the class/labels into binary i.e one hot encode. then we find the loss.

but what if we feed data with class/ label as integer.

For this we have sparse categorical cross entropy function.

Advantage compared Multi-class Cross Entropy: Above example shows that for multi-class cross entropy, the target needs a one hot encoded vector which contains a lot of zero values, leading to significant memory requirement. By using sparse categorical cross entropy, one can save computation time with lower memory requirement because it only requires a single integer for a class, rather than a whole vector.

Disadvantage of Sparse Multi-class Cross Entropy: Multi-class cross entropy can be used in any kind of classification problem. Whereas, Sparse categorical cross entropy can only be used when each input belongs to a single class only.

Note:

The ground truth label(y_true) shape = [batch_size, d0, .. dN-1, 1].

The predicted label(y_pred) shape = [batch_size, d0, .. dN].

```
# create a array for true label of random integer values of any size
```

```
# create a array of shape of rows similar to number of classes
```

```
# create a variable named sparse_cat_cross_entropy which is a constructor of Hinge from te
```

```
# create a variable to store sparse_cat_cross_entropy loss returned by tensorflow
```

```
# print sparse_cat_cross_entropy_value
```

```
y_true: [1 2]
```

```
Shape of y_true: (2,)
```

```
y_pred: [[0.05 0.95 0. ]  
         [0.2  0.8  0.  ]]
```

```
Shape of y_pred: (2, 3)
```

```
SparseCategoricalCrossentropy Loss: 8.084694862365723
```

▼ Kullback Leibler (KL) Divergence Loss

Kullback Leibler Divergence is a measure that shows how much two probability distributions are different from each other.

Formula for Kullback Leibler (KL) Divergence Loss:

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

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

KL divergence loss of 0 suggests the distributions are identical.

KL Divergence is somehow similar to cross-entropy. Like multi-class cross entropy, here also actual targets (y) needs to be one-hot encoded. It calculates how much information is lost if the predicted probability distribution is used to approximate the desired target probability distribution.

KL divergence is mostly used in Variational Autoencoders. Here, the autoencoders learn how to encode the samples into a latent probability distribution, which is further fed into a decoder for generating output. Additionally, KL divergence can be used in multiclass classification.

```
# create a list for true label of random values of any size
```

```
# create a list for predicted label same size as compared to true label but values are pro
```

```
# create a variable named kl which is a constructor of Hinge from tensorflow
```

```
# create a variable to store kl_value loss returned by tensorflow
```

```
# print kl_value
```

```
y_true: [[0 1]
         [0 0]]
```

```
Shape of y_true: (2, 2)
```

```
y_pred: [[0.6 0.4]
         [0.4 0.6]]
```

```
Shape of y_pred: (2, 2)
```

```
KLDivergence Loss: 0.458143025636673
```

▼ Assignment Summary

When creating a neural network we must select a appropriate loss function according to activation function used:

following are the loss function discussed in this assignment

For Regression Problem:

1. Mean Squared Error
2. Mean Absolute Error Loss
3. Huber Loss (Smooth Mean Absolute Error)

For Binary Classification Problem:

1. Binary Cross Entropy Loss
2. Hinge Loss

For Multi-Class Classification Problem:

1. Categorical Cross Entropy
2. Sparse Categorical Cross Entropy
3. Kullback Leibler (KL) Divergence Loss

Congratulations!

shutterstock.com · 1396729610

we have learned all the loss function used in neural network and their properties with code

Please fill the below feedback form about this assignment

<https://forms.zohopublic.in/cloudyml/form/CloudyMLDeepLearningFeedbackForm/formperma/VCFbldnXAnbcgAll0IWv2blgHdSldheO4RfktMdgK7s>