Guide For Loss Function in Tensorflow

ADVANCED LIBRARIES MACHINE LEARNING MATHS PROJECT PYTHON

This article was published as a part of the Data Science Blogathon

As we know the main reason behind human evolution is we learned from our past mistakes. What if like humans machines could learn from their mistakes?

But how can machines learn from their past mistake? In neural networks and AI, machines try to find the best predictions but how can someone improve without comparing it with its previous mistakes, so in backpropagation loss function comes into the picture.

We also called it an **error function or cost function.** These are the errors made by machines at the time of training the data and using an optimizer and adjusting weight machines can reduce loss and can predict accurate results.

We are going to see below the loss function and its implementation in python. In Tensorflow API mostly you are able to find all losses in **tensorflow.keras.losses**

Probabilistic Loss Functions:

1. Binary Cross-Entropy Loss:

Binary cross-entropy is used to compute the cross-entropy between the true labels and predicted outputs. It's used when two-class problems arise like cat and dog classification [1 or 0].

Below is an example of Binary Cross-Entropy Loss calculation:

```
## Binary Corss Entropy Calculation import tensorflow as tf #input lables. y_true = [[0.,1.], [0.,0.]] y_pred = [[0.5,0.4], [0.6,0.3]] binary_cross_entropy = tf.keras.losses.BinaryCrossentropy() binary_cross_entropy(y_true=y_true,y_pred=y_pred).numpy()
```

2. Categorical Crossentropy Loss:

The Categorical crossentropy loss function is used to compute loss between true labels and predicted labels.

It's mainly used for multiclass classification problems. For example Image classification of animal-like cat, dog, elephant, horse, and human.

```
# inputs y_true = [[0, 1, 0], [0, 0, 1]] y_pred = [[0.05, 0.95, 0.56], [0.1, 0.4, 0.1]] categorical_cross_entropy = tf.keras.losses.CategoricalCrossentropy() categorical_cross_entropy(y_true=y_true,y_pred=y_pred).numpy()
```

This is how we can calculate categorical cross-entropy loss.

3. Sparse Categorical Crossentropy Loss:

It is used when there are two or more classes present in our classification task. similarly to categorical crossentropy. But there is one minor difference, between categorical crossentropy and sparse categorical crossentropy that's in sparse categorical cross-entropy labels are expected to be provided in integers.

The implementation for Sparse Categorical Crossentory loss is as below:

```
#input Labels y_true = [1, 2] #Predicted Lables y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]] #Implementation of Sparse Categorical Crossentropy tf.keras.losses.sparse_categorical_crossentropy(y_true,y_pred).numpy()
```

Rather than using Sparse Categorical crossentropy we can use **one-hot-encoding** and convert the above problem into categorical crossentropy.

4. Poisson loss:

The poison loss is the mean of elements of tensor. we can calculate poison loss like **y_pred – y_true*log(y_true)**

The Tensorflow Implementation for the same is as follows.

```
#input Labels y_true = [[0., 1.], [0., 0.]] #Predicted Lables y_pred = [[1., 1.], [1., 0.]] # Using 'auto'/'sum_over_batch_size. p = tf.keras.losses.Poisson() p(y_true, y_pred).numpy()
```

5. Kullback-Leibler Divergence Loss:

Also, called KL divergence, it's calculated by doing a negative sum of probability of each event P and then multiplying it by the log of the probability of an event.

KL(P || Q) = – sum x in X P(x) * log(Q(x) / P(x))

Tensorflow Implementation for KL divergence Loss:

```
#input Labels y_true = [[0, 1], [0, 0]] #Predicted Lables y_pred = [[0.7, 0.8], [0.4, 0.8]] #KL divergen loss kl = tf.keras.losses.KLDivergence() kl(y_true, y_pred).numpy()
```

Regression Losses:

6. Means Squared Error (MSE):

MSE tells, how close a regression line from predicted points. And this is done simply by taking distance from point to the regression line and squaring them. The squaring is a must so it'll remove the negative sign problem.

Tensorflow implementation for MSE:

```
# Input Labels y_true = [[10., 10.], [0., 0.]] # Predicted Labels y_pred = [[10., 10.], [1., 0.]] #Mean Sqaured Error Loss mse = tf.keras.losses.MeanSquaredError() mse(y_true, y_pred).numpy()
```

7. Mean Absolute Error:

MAE simply calculated by taking distance from point to the regression line. The MAE is more sensitive to outliers. So before using MAE confirm that data doesn't contain outliers.

Tensorflow Implementation for MAE:

```
# Input Labels y_true = [[10., 20.], [30., 40.]] # Predicted Labels y_pred = [[10., 20.], [30., 0.]] mae = tf.keras.losses.MeanAbsoluteError() mae(y_true, y_pred).numpy()
```

8. Cosine Similarity Loss:

Cosine similarity is a measure of similarity between two non-zero vectors. This loss function calculates the cosine similarity between labels and predictions.

- It's just a number between 1 and -1
- when it's a negative number between -1 and 0 then, 0 indicates orthogonality, and values closer to -1 show greater similarity.

Tensorflow Implementation for Cosine Similarity is as below:

```
# Input Labels y_true = [[10., 20.], [30., 40.]] # Predicted Labels y_pred = [[10., 20.], [30., 0.]] cosine_loss = tf.keras.losses.CosineSimilarity(axis=1) cosine_loss(y_true, y_pred).numpy()
```

9. Huber Loss:

The Huber loss function is quadratic for small values and linear for larger values,

For each value of **X the error = y_true-y_pred**

Loss = 0.5 * X^2 if |X| <= d Loss = 0.5 * d^2 + d (|X| - d) if |X| > d

Tensorflow Implementation for Huber Loss:

```
# Input Labels y_true = [[10., 20.], [30., 40.]] # Predicted Labels y_pred = [[10., 20.], [30., 0.]] hub_loss = tf.keras.losses.Huber() hub_loss(y_true, y_pred).numpy()
```

10. LogCosh Loss:

The LogCosh loss computes the log of the hyperbolic cosine of the prediction error.

The Tensorflow Implementation for LogCosh Loss:

```
#input Labels y_true = [[0., 1.], [0., 0.]] #Predicted Lables y_pred = [[1., 1.], [1., 0.]] l = tf.keras.losses.LogCosh() l(y_true, y_pred).numpy()
```

Hinge Losses for 'Maximum – Margin' Classification:

11. Hinge Loss

It's mainly used for problems like maximum-margin most notably for support vector machines.

In Hinge loss values are expected to be -1 or 1. In the case of binary i.e. 0 or 1 it'll get converted into -1 and 1.

Tensorflow implementation for Hing Loss:

```
#input Labels y_true = [[0., 1.], [0., 0.]] #Predicted Lables y_pred = [[0.5, 0.4], [0.4, 0.5]] h_loss = tf.keras.losses.Hinge() h_loss(y_true, y_pred).numpy()
```

12. Squared Hinge Loss:

The Square Hinge loss is just square of hinge loss.

Tensorflow implementation for Squared Hinge loss:

```
#input Labels y_true = [[0., 1.], [0., 0.]] #Predicted Lables y_pred = [[0.5, 0.4], [0.4, 0.5]] h = tf.keras.losses.SquaredHinge() h(y_true, y_pred).numpy()
```

13. Categorical Hinge Loss:

It calculates the categorical hing loss between y_true and y_pred labels.

Tensorflow Implementation for categorical Hing loss:

```
#input Labels y_true = [[0., 1.], [0., 0.]] #Predicted Lables y_pred = [[0.5, 0.4], [0.4, 0.5]] h = tf.keras.losses.CategoricalHinge() h(y_true, y_pred).numpy()
```

Conclusion

We have discussed almost all major loss function which is supported by Tensorflow API, For more information, you can check official documents.

Keras Documentation:

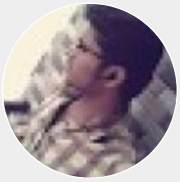
https://keras.io/api/losses/

Tensorflow Documentation:

https://www.tensorflow.org/api_docs/python/tf/keras/losses

The media shown in this article are not owned by Analytics Vidhya and are used at the Author's discretion.

Article Url - https://www.analyticsvidhya.com/blog/2021/05/guide-for-loss-function-in-tensorflow/



pravin31