Report
on

Loss Function in Neural Networks
11-Jan-2019

Submitted by

Love Prakash

DronaMaps Private Limited

Contents

1 Introduction to Loss function in Neural Networks:

At its core, a loss function is incredibly simple: it's a method of evaluating how well your algorithm models your dataset. If your predictions are totally off, your loss function will output a higher number. If they're pretty good, it'll 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're getting anywhere.

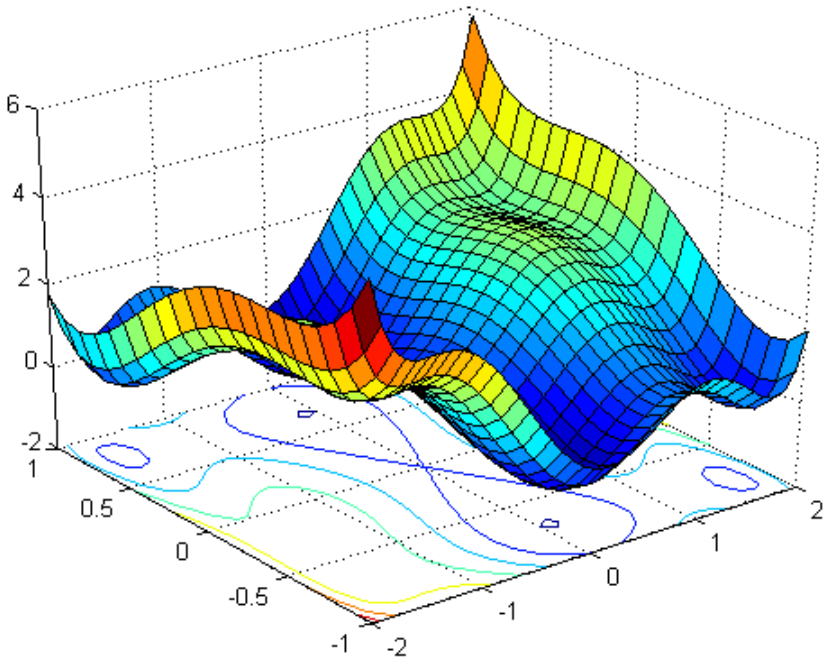

Figure 1: Representation

In fact, we can design our own (very) basic loss function to further explain how it works. For each prediction that we make, our loss function [1]will simply measure the absolute difference between our prediction and the actual value. In mathematical notation, it might look something like $abs(y_{predicted}\breve{}y)$. Loss function is an important part in artificial neural networks, which is used to measure the inconsistency between predicted value and actual label. It is a non-negative value, where the robustness of model increases along with the decrease of the value of loss function. Loss function is the hard core of empirical risk function as well as a significant component of structural risk function. Generally, the structural risk function of a model is consist of empirical risk term and regularization term, which can be represented as [2]

$$\begin{aligned}\theta^* &= \arg\min_{\theta} \mathcal{L}(\theta) + \lambda \cdot \Phi(\theta) \\ &= \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L\big(y^{(i)}, \hat{y}^{(i)}\big) + \lambda \cdot \Phi(\theta) \\ &= \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L\big(y^{(i)}, f(\mathbf{x}^{(i)}, \theta)\big) + \lambda \cdot \Phi(\theta)\end{aligned}$$

Figure 2: Representation

Here we only concentrate on the empirical risk term (loss function)

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} L\big(y^{(i)}, f(\mathbf{x}^{(i)}, \theta)\big)$$

Figure 3: Emperical formula

and introduce the mathematical expressions of several commonly-used loss functions as well as the corresponding expression in DeepLearning.

2 Types of Loss functions:

2.1 Mean Squared Error:

Mean Squared Error (MSE), or quadratic, loss function is widely used in linear regression as the performance measure, and the method of minimizing MSE is called Ordinary Least Squares (OSL), the basic principle of OSL is that the optimized fitting line should be a line which minimizes the sum of distance of each point to the regression line, i.e., minimizes the quadratic sum. The standard form of MSE loss function is defined as

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

Figure 4: MSE Loss function

2.2 Mean Squared Logarithmic Error:

Mean Squared Logarithmic Error (MSLE) loss function is a variant of MSE, MSLE is also used to measure the different between actual and predicted. By taking the log of the predictions and actual values, what changes is the variance that you are measuring. It is

usually used when you do not want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\left(\log(y^{(i)}+1) - \log(\hat{y}^{(i)}+1)\right)^2$$

Figure 5: MSLE Loss function

2.2.1 Another thing is that MSLE penalizes under-estimates more than over-estimates

- If both predicted and actual values are small: MSE and MSLE is same.
- If either predicted or the actual value is big: MSE>MSLE
- If both predicted and actual values are big: MSE>MSLE (MSLE becomes almost negligible).

It is expressed as LossFunctions.LossFunction.MEAN_SQUARED_LOGARITHMIC_ERROR in DeepLearning.

2.2.2 L2

L2 loss function is the square of the L2 norm of the difference between actual value and predicted value. It is mathematically similar to MSE, only do not have division by n.

$$\mathcal{L} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

Figure 6: L2 Loss function

2.3 Mean Absolute Error:

Mean Absolute Error (MAE) is a quantity used to measure how close forecasts or predictions are to the eventual outcomes. where |,|denotes the absolute value. Albeit, both MSE and MAE are used in predictive modeling, there are several differences between them. MSE has nice mathematical properties which makes it easier to compute the gradient. However, MAE requires more complicated tools such as linear programming to compute the gradient. Because of the square, large errors have relatively greater influence on MSE than do the smaller error. Therefore, MAE is more robust to outliers since it does not make use of square. On the other hand, MSE is more useful if concerning about large errors whose consequences are much bigger than equivalent smaller ones. MSE also

corresponds to maximizing the likelihood of Gaussian random variables. In DeepLearning, it is expressed as LossFunctions.LossFunction.Mean_absolute_error.

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}|$$

Figure 7: Mean Absolute Error

2.4 Mean Absolute Percentage Error:

Mean Absolute Percentage Error (MAPE) is a variant of MAE, Although the concept of MAPE sounds very simple and convincing, it has major drawbacks in practical application:

2.4.1 Major Drawback of MAPE:

- It cannot be used if there are zero values (which sometimes happens for example in demand data) because there would be a division by zero.
- For forecasts which are too low the percentage error cannot exceed 100, but for forecasts which are too high there is no upper limit to the percentage error.
- When MAPE is used to compare the accuracy of prediction methods it is biased in that it will systematically select a method whose forecasts are too low. This little-known but serious issue can be overcome by using an accuracy measure based on the ratio of the predicted to actual value (called the Accuracy Ratio), this approach leads to superior statistical properties and leads to predictions which can be interpreted in terms of the geometric mean.

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \left|\frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}}\right| \cdot 100$$

Figure 8: Mean Absolute Percentage Error

It is expressed as LossFunctions.LossFunction.MEAN_ABSOLUTE_PERCENTAGE_ERROR in DeepLearning.

2.4.2 L1

L1 loss function is sum of absolute errors of the difference between actual value and predicted value. Similar to the relation between MSE and L2, L1 is mathematically similar to MAE, only do not have division by n, and it is defined as

$$\mathcal{L} = \sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}|$$

Figure 9: L1 Error

2.5 Kullback Leibler (KL) Divergence:

KL Divergence, also known as relative entropy, information divergence/gain, is a measure of how one probability distribution diverges from a second expected probability distribution.

$$\begin{aligned}\mathcal{L} &= \frac{1}{n}\sum_{i=1}^{n} \mathcal{D}_{KL}(y^{(i)}||\hat{y}^{(i)}) \\ &= \frac{1}{n}\sum_{i=1}^{n} \left[y^{(i)} \cdot \log\left(\frac{y^{(i)}}{\hat{y}^{(i)}}\right)\right] \\ &= \underbrace{\frac{1}{n}\sum_{i=1}^{n} \left(y^{(i)} \cdot \log(y^{(i)})\right)}_{entropy} \underbrace{-\frac{1}{n}\sum_{i=1}^{n} \left(y^{(i)} \cdot \log(\hat{y}^{(i)})\right)}_{cross-entropy}\end{aligned}$$

Figure 10: Kullback Divergence

where the first term is entropy and another is cross entropy (another kind of loss function which will be introduced later). KL divergence is a distribution-wise asymmetric measure and thus does not qualify as a statistical metric of spread. In the simple case, a KL divergence of 0 indicates that we can expect similar, if not the same, behavior of two different distributions, while a KL divergence of 1 indicates that the two distributions behave in such a different manner that the expectation given the first distribution approaches zero. In DeepLearning4J, it is expressed as LossFunctions.LossFunction.KL_DIVERGENCE. Moreover, the implementation of Reconstruction Cross Entropy in DeepLearning4J is same as Kullback Leibler (KL) Divergence, thus, you can also use LossFunctions.LossFunction.RECONSTRUC

2.6 Cross Entropy:

Cross Entropy is commonly-used in binary classification (labels are assumed to take values 0 or 1) as a loss function (For multi-classification, use Multi-class Cross Entropy).Cross entropy measures the divergence between two probability distribution, if the cross entropy is large, which means that the difference between two distribution is large, while if the cross entropy is small, which means that two distribution is similar to each other. As we have mentioned in MSE that it suffers slow divergence when using Sigmoid as activation function, here the cross entropy does not have such problem.

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

Figure 11: Cross Entropy

In this case, when the difference between predicted value and actual value is large, the learning speed, i.e., convergence speed, is fast, otherwise, the difference is small, the learning speed is small, this is our expectation. Generally, comparing to quadratic cost function, cross entropy cost function has the advantages that fast convergence and is more likely to reach the global optimization (like the momentum, it increases the update step). In DeepLearning4J, it is expressed as LossFunctions.LossFunction.XENT. For multi-classification, it is better use LossFunctions.LossFunction.MCXENT.

2.7 Negative Logarithmic Likelihood:

Negative Log Likelihood loss function is widely used in neural networks, it measures the accuracy of a classifier. It is used when the model outputs a probability for each class, rather than just the most likely class. It is a "soft" measurement of accuracy that incorporates the idea of probabilistic confidence. It is intimately tied to information theory. And it is similar to cross entropy (in binary classification) or multi-class cross entropy (in multi-classification) mathematically Negative log likelihood is computed by

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\log(\hat{y}^{(i)})$$

Figure 12: Negative Logarithmic Likelyhood

In DeepLearning4J, it is expressed as LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD. Actually, in Deep Learning, the implementation of MCXENT and NEGATIVELOGLIKELIHOOD is same, since they have almost the mathematically samilar expressions.

2.8 Poisson:

Poisson loss function is a measure of how the predicted distribution diverges from the expected distribution, the poisson as loss function is a variant from Poisson Distribution, where the poisson distribution is widely used for modeling count data. It can be shown to be the limiting distribution for a normal approximation to a binomial where the number of trials goes to infinity and the probability goes to zero and both happen at such a rate that np is equal to some mean frequency for the process. In DL4J, the poisson loss function is computed by

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)} \cdot \log(\hat{y}^{(i)}) \right)$$

Figure 13: Poisson

2.9 Hinge Loss/Multi class SVM Loss:

In simple terms, the score of correct category should be greater than sum of scores of all incorrect categories by some safety margin (usually one). And hence hinge loss is used for maximum-margin classification, most notably for support vector machines. Although not differentiable, it's a convex function which makes it easy to work with usual convex optimizers used in machine learning domain.

$$SVMLoss = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

Figure 14: SVM or hinge loss

3 Conclusion:

Choosing an appropriate loss function is important as it affects the ability of the algorithm to produce optimum results as fast as possible. Machine learning algorithms are designed so that they can "learn" from their mistakes and "update" themselves using the training data we provide them. But how do they quantify these mistakes? This is done via the usage of "loss functions" that help an algorithm get a sense of how erroneous its predictions are when compared to the ground truth. Choosing an appropriate loss function is important as it affects the ability of the algorithm to produce optimum results as fast as possible.

References

[1] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems*, pages 1988–1996, 2014.

[2] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.