Loss Functions in Artificial Neural Networks

2017-06-07 | Updated on 2017-09-21 | Count 2,709 words | Reading 17 min | Deep Learning

Loss function is an important part in artificial neural networks, which is used to measure the inconsistency between predicted value ($\hat{y}$y^) and actual label (yy). 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

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

θ∗=argminθL(θ)+λ⋅Φ(θ)=argminθ1n∑i=1nL(y(i),y^(i))+λ⋅Φ(θ)=argminθ1n∑i=1nL(y(i),f(x(i),θ))+λ⋅Φ(θ)

where $\Phi(\theta)$ Φ(θ) is the regularization term or penalty term, θ θ is the parameters of model to be learned, $f(\cdot)$ f(⋅) represents the activation function and $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \ldots, x_m^{(i)}\} \in \mathbb{R}^m$ x(i)={x1(i),x2(i),…,xm(i)}∈Rm denotes the a training sample.

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

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

L(θ)=1n∑i=1nL(y(i),f(x(i),θ))

and introduce the mathematical expressions of several commonly-used loss functions as well as the corresponding expression in DeepLearning4J.

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

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

L=1n∑i=1n(y(i)−y^(i))2

where $(y^{(i)} - \hat{y}^{(i)})$ (y(i)−y^(i)) is named as residual, and the target of MSE loss function is to minimize the residual sum of squares. In DeepLearning4J, it is `LossFunctions.LossFunction.MSE` or `LossFunctions.LossFunction.SQUARED_LOSS` (they are same in DL4J). However, if using Sigmoid as the activation function, the quadratic loss function would suffer the problem of slow convergence (learning speed), for other activation funtions, it would not have such problem.

For example, by using Sigmoid, $\hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(\theta^T x^{(i)})$ y^(i)=σ(z(i))=σ(θTx(i)), simply, we only consider one sample, say, $(y - \sigma(z))^2$ (y−σ(z))2, and it derivative is computed by

$$\frac{\partial L}{\partial \theta} = -(y - \sigma(z))\cdot\sigma'(z)\cdot x$$

∂L∂θ=−(y−σ(z))⋅σ′(z)⋅x

according to the shape and feature of Sigmoid (see my another blog: Activation Functions in Artificial Neural Networks), when $\sigma(z)$ σ(z) tends to 0 or 1, $\sigma'(z)$ σ′(z) is close to zero, and when $\sigma(z)$ σ(z) close to 0.5, $\sigma'(z)$ σ′(z) will reach it maximum. In this case, when the difference between predicted value and true label $(y - \sigma(z))$ (y−σ(z)) is large, $\sigma'(z)$ σ′(z) will close to 0, which decreases the convergence speed, this is improper, since we expect that the learning speed should be fast when the error is large.

Mean Squared Logarithmic Error

Mean Squared Logarithmic Error (MSLE) loss function is a variant of MSE, which is defined as

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

L=1n∑i=1n(log(y(i)+1)−log(y^(i)+1))2

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

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

In DeepLearning4J, it is expressed as `LossFunctions.LossFunction.MEAN_SQUARED_LOGARITHMIC_ERROR` .

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 nn, it is computed by

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

L=∑i=1n(y(i)−y^(i))2

For more details, typically in mathematic, please read the paper: On Loss Functions for Deep Neural Networks in Classification, which gives comprehensive explanation about several commomly-used loss functions, including L2, L1 loss function.

In DeepLearning4J, it is expressed as `LossFunctions.LossFunction.L2` .

Mean Absolute Error

Mean Absolute Error (MAE) is a quantity used to measure how close forecasts or predictions are to the eventual outcomes, which is computed by

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

L=1n∑i=1n|y(i)−y^(i)|

where $|\cdot|$ |·| 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 DeepLearning4J, it is expressed as `LossFunctions.LossFunction.MEAN_ABSOLUTE_ERROR` .

Mean Absolute Percentage Error

Mean Absolute Percentage Error (MAPE) is a variant of MAE, it is computed by

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

L=1n∑i=1n|y(i)−y^(i)y(i)|·100

Although the concept of MAPE sounds very simple and convincing, it has major drawbacks in practical application:

1. 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.
2. For forecasts which are too low the percentage error cannot exceed 100100, but for forecasts which are too high there is no upper limit to the percentage error.
3. 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.

In DeepLearning4J, it is expressed as `LossFunctions.LossFunction.MEAN_ABSOLUTE_PERCENTAGE_ERROR`.

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 nn, and it is defined as

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

L=∑i=1n|y(i)−y^(i)|

In DeepLearning4J, it is expressed as `LossFunctions.LossFunction.L1`.

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. KL divergence loss function is computed by

$$L = \frac{1}{n}\sum_{i=1}^{n} D_{KL}(y^{(i)} || \hat{y}^{(i)}) = \frac{1}{n}\sum_{i=1}^{n} [y^{(i)} \cdot \log(\frac{y^{(i)}}{\hat{y}^{(i)}})]$$
$$= \underbrace{\frac{1}{n}\sum_{i=1}^{n} (y^{(i)} \cdot \log(y^{(i)}))}_{\text{entropy}} \underbrace{-\frac{1}{n}\sum_{i=1}^{n} (y^{(i)} \cdot \log(\hat{y}^{(i)}))}_{\text{cross−entropy}}$$

L=1n∑i=1nDKL(y(i)||y^(i))=1n∑i=1n[y(i)⋅log(y(i)y^(i))]=1n∑i=1n(y(i)⋅log(y(i)))⏟entropy−1n∑i=1n(y(i)⋅log(y^(i)))⏟cross−entropy

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. For more details, please visit the wikipedia: [link].

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.RECONSTRUCTION_CROSSENTROPY`.

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**), which is computed by

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

L=−1n∑i=1n[y(i)log(y^(i))+(1−y(i))log(1−y^(i))]

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. Samely, $\hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(\theta^{T} x^{(i)})$ y^(i)=σ(z(i))=σ(θTx(i)), and we only consider one training sample, by using Sigmoid, we have $L = y\log(\sigma(z)) + (1-y)\log(1-\sigma(z))$ L=ylog(σ(z))+(1−y)log(1−σ(z)), and compute it derivative as

$$\frac{\partial L}{\partial \theta} = (y - \sigma(z)) \cdot x$$

∂L∂θ=(y−σ(z))·x

compare to the derivative in MSE, it eliminates the term $\sigma'(z)$ σ′(z), where the learning speed is only controlled by $(y - \sigma(z))$ (y−σ(z)). 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). For the mathematical details, see wikipedia: [link].

In DeepLearning4J, it is expressed as `LossFunctions.LossFunction.XENT`. For multi-classification, it is better use `LossFunctions.LossFunction.MCXENT`.

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

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

L=−1n∑i=1nlog(y^(i))

More details about Negative Log Likelihood and the relation of KL Divergence, Cross Entropy and Negative Log Likelihood, you can visit this post: [link].

In DeepLearning4J, it is expressed as `LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD`. Actually, in DL4J, the implementation of `MCXENT` and `NEGATIVELOGLIKELIHOOD` is same, since they have almost the mathematically samilar expressions.

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

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

L=1n∑i=1n(y^(i)−y(i)·log(y^(i)))

In DL4J, it is expressed as `LossFunctions.LossFunction.POISSON`. Moreover, the implementation of Exponential Log Likelihood in DeepLearning4J is same as Poisson, so you can also use `LossFunctions.LossFunction.EXPLL`.

Cosine Proximity

Cosine Proximity loss function computes the cosine proximity between predicted value and actual value, which is defined as

$$L = -\frac{y \cdot \hat{y}}{||y||_2 \cdot ||\hat{y}||_2} = -\frac{\sum_{i=1}^{n} y^{(i)} \cdot \hat{y}^{(i)}}{\sqrt{\sum_{i=1}^{n} (y^{(i)})^2} \cdot \sqrt{\sum_{i=1}^{n} (\hat{y}^{(i)})^2}}$$

L=−y·y^||y||2·||y^||2=−∑i=1ny(i)·y^(i)∑i=1n(y(i))2·∑i=1n(y^(i))2

where $y = \{y^{(1)}, y^{(2)}, \dots, y^{(n)}\} \in R^n$ y={y(1),y(2),...,y(n)}∈Rn, and $\hat{y} = \{\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(n)}\} \in R^n$ y^={y^(1),y^(2),...,y^(n)}∈Rn. It is same as Cosine Similarity, which is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. In this case, note that unit vectors are maximally "similar" if they're parallel and maximally "dissimilar" if they're orthogonal (perpendicular). This is analogous to the cosine, which is unity (maximum value) when the segments subtend a zero angle and zero (uncorrelated) when the segments are perpendicular.

In DeepLearning4J, it is expressed as `LossFunctions.LossFunction.COSINE_PROXIMITY` .

Hinge

Hinge Loss, also known as max-margin objective, is a loss function used for training classifiers. The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs). For an intended output $y^{(i)} = \pm 1$ y(i)=±1, i.e., binary classification and a classifier score $\hat{y}^{(i)}$ y^(i), the hinge loss of the prediction y is defined as

$$L = \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y^{(i)} \cdot \hat{y}^{(i)})$$

L=1n∑i=1nmax(0,1−y(i)⋅y^(i))

Note that $\hat{y}^{(i)}$ y^(i) should be the "raw" output of the classifier's decision function, not the predicted class label. It can be seen that when $y^{(i)}$ y(i)and $\hat{y}^{(i)}$ y^(i) have the same sign (meaning $\hat{y}^{(i)}$ y^(i) predicts the right class) and $|\hat{y}^{(i)}| > 1$ |y^(i)|>1, the hinge loss equals to zero, but when they have opposite sign, hinge loss increases linearly with $\hat{y}^{(i)}$ y^(i) (one-sided error). And in DeepLearning4J, this formula is expressed as `LossFunctions.LossFunction.HINGE` (in ND4J codes, the `HINGE` loss function is implemented by the formula above). However, there is a more general expression

$$L = \frac{1}{n} \sum_{i=1}^{n} \max(0, m - y^{(i)} \cdot \hat{y}^{(i)})$$

L=1n∑i=1nmax(0,m−y(i)⋅y^(i))

where m m (margin) is a customized value. More details about extending to multi-classification, optimization, you can visit Hinge loss's wikipedia: [link].

Squared Hinge

Squared Hinge Loss function is a variant of Hinge Loss, it solves the problem in hinge loss that the derivative of hinge loss has a discontinuity at $y^{(i)} \cdot \hat{y}^{(i)} = 1$ y(i)⋅y^(i)=1. Squared Hinge Loss is computed by

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(\max(0, 1 - y^{(i)} \cdot \hat{y}^{(i)})\right)^2$$

L=1n∑i=1n(max(0,1−y(i)⋅y^(i)))2

as the definition in DL4J. In DeepLearning4J, it is expressed as `LossFunctions.LossFunction.SQUARED_HINGE` .

Reference

- On Loss Functions for Deep Neural Networks in Classification
- Loss functions
- ND4J Loss Functions
- Losses - Keras Documentation
- What is the difference between an RMSE and RMSLE
- Machine Learning-Loss Function
- Neural Network-Loss Function
- Cross Entropy Cost Function
- What is the difference between squared error and absolute error?
- Mean Absolute Percentage Error
- KL-divergence as an objective function
- Poisson regression
- A Study on L2-Loss (Squared Hinge-Loss) Multi-Class SVM
- Why Minimize Negative Log Likelihood?

> **Author:** Isaac Changhau
> **Link:** https://isaacchanghau.github.io/2017/06/07/Loss-Functions-in-Artificial-Neural-Networks/