Loss Functions in Deep Learning

A Guide on the Concept of Loss Functions in Deep Learning — What they are, Why we need them...

Artem Oppermann Follow
Mar 7 · 10 min read ★

Source: https://unsplash.com/photos/yNvVnPcurD8

This in-depth article addresses the questions of why we need loss functions in deep learning and which loss functions should be used for which tasks.

In Short: *Loss functions in deep learning are used to measure how well a neural network model performs a certain task.*

Table of Content

1. Why do we need Loss Functions in Deep Learning?

Before we discuss different kinds of loss functions used in Deep Learning, it would be a good idea to address the question of why we need loss functions in the first place.

I think you must be familiar by now with the mathematical operations which are happening inside a neural network. Basically, there are just two:

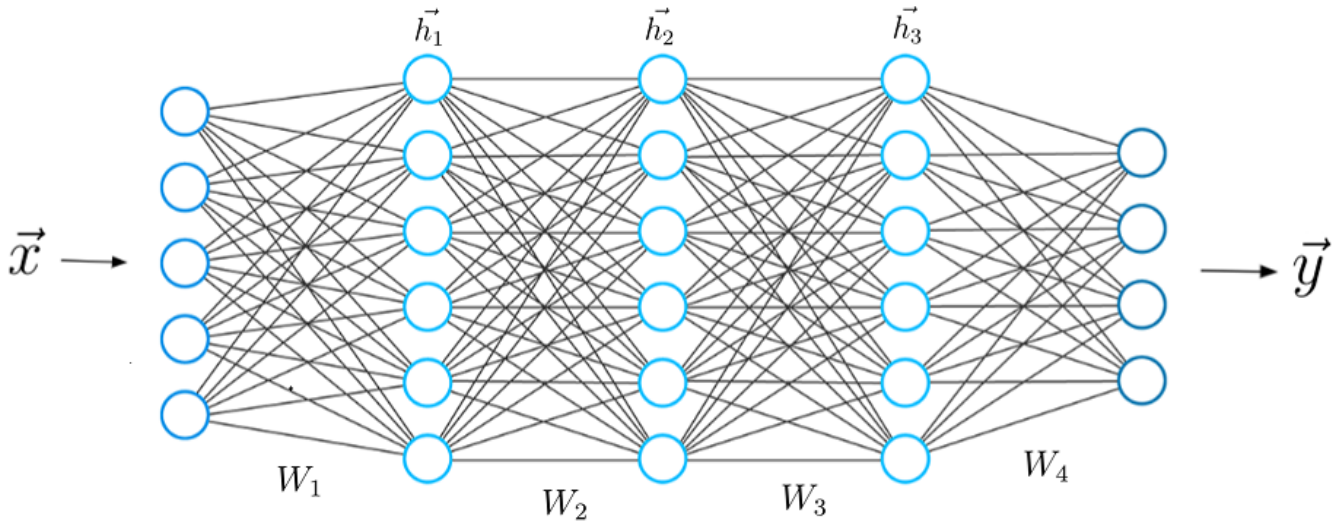
- Forward Propagation
- Backpropagation with Gradient Descent

While forward propagation refers to the computational process of predicting an output for a given input vector $\boldsymbol{x}$, backpropagation and gradient descent describe the process of improving the weights and biases of the network in order to make better predictions.

Recap: Forward Propagation

> *For a more in-depth explanation of Forward Propagation and Backpropagation in neural networks, please refer to my other article What is Deep Learning and How does it work?*

For a given input vector $\boldsymbol{x}$ the neural network predicts an output, which is generally called a prediction vector $\boldsymbol{y}$.



Fig, 1. Feedforward neural network. Source: Author's own image.

The equations which describe the mathematics that is happening during the computation of the prediction vector loos as follows:

$$\vec{h_1} = \sigma(\vec{x} \cdot W_1)$$
$$\vec{h_2} = \sigma(\vec{h_1} \cdot W_2)$$
$$\vec{h_3} = \sigma(\vec{h_2} \cdot W_3)$$
$$\vec{y} = \sigma(\vec{h_3} \cdot W_4)$$

Eq.1 Forward propagation. Source: Author's own image.

We must compute a dot-product between the input vector x and the weight matrix ***W1*** that connects the first layers with the second. After that, we apply a non-linear activation function to the result of the dot-product. (A detailed guide on the different

activation functions in deep learning can be found in my other article Activation Functions in Deep Learning)

Depending on the task we want the network to do, this prediction vector represents different things. For regression tasks, which are basically predictions of continuous variables (e.g. stock price, expected demand for products) the output vector ***y*** would contain continuous numbers.

For classification tasks, on the other hand, such as customer segmentation or image classification the output vector ***y*** would represent probability scores between 0.0 and 1.0.

The value that we want the neural network to predict is called a *ground truth label,* which is usually represented as ***y_hat***. A predicted value ***y*** that is closer to the label suggests a better performance of the neural network.

Regardless of the task, we somehow have to measure how close our predictions are to the ground truth label.

This is where the concept of a loss function comes into play.

Mathematically, we can measure the difference (or error) between the prediction vector ***y*** and the label ***y_hat*** by defining a loss function which value depends on this difference.

An example of a general loss function is the quadratic loss:

$$\mathcal{L}(\theta) = \frac{1}{2}(\vec{y}(\theta) - \hat{\vec{y}})^2 \qquad \theta = (W_1, W_2, W_3, W_4)$$

Eq. 2 Quadratic loss function. Source: Authors own image.

Since the prediction vector ***y(θ)*** is a function of the weights of the neural network (which we abbreviate to **θ**), the loss is also a function of the weights.

The value of this loss function depends on the difference between the label ***y_hat*** and ***y***. A higher difference means a higher loss value, a smaller difference means a smaller loss

value. Minimizing the loss function directly leads to more accurate predictions of the neural network, as the difference between the prediction and the label decreases.

In fact, the minimization of the loss function is the only objective that the neural network tries to achieve.

A neural network solves tasks without being explicitly programmed with a task-specific rule. This is possible because *minimizing the loss function as a goal is universal and does not depend on the task or task circumstances*.

> Minimizing the loss function automatically causes the neural network model to make better predictions regardless of the exact characteristics of the task at hand.

You only have to select the right loss function for the task. From my experience, I can tell that there, fortunately, are only three loss functions that you should know about to solve almost any problem that you will encounter in practice. We will discuss these loss functions in the following.

Since the loss depends on the weights, we must find a certain set of weights for which the value of the loss function is as small as possible. The method of minimizing the loss function is achieved mathematically by a method called gradient descent. (If you are interested in this topic please refer to the article *What is Deep Learning and how does it work?*, Where I explain gradient descent in much more detail.)

If we have found a proper loss function to measure the error the neural network makes, we can proceed with the backpropagation and gradient descent step where we improve the weights and biases of the network. However, this topic is not important for this article right now.

Previously I introduced the squared error loss function which I used to give you a better understanding of the general concept of a loss function. Now I will extend your

knowledge and present to you three of the most common loss functions that are used in the field of deep learning. The first one is called the mean squared error.

2. Mean Squared Error Loss Function

Mean squared error (MSE) loss function is the sum of squared differences between the entries in the prediction vector ***y*** and the ground truth vector ***y_hat***.

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2$$

y_i : entries in the prediction vector $\vec{y}$

$\hat{y}_i$: entries in the ground truth label $\hat{\vec{y}}$

Eq. 3 MSE Loss Function. Source: Author's own image.

The sum of squared differences is divided by ***N***, which corresponds to the length of the vectors. If the output ***y*** of your neural network is a vector with multiple entries then ***N*** is the number of the vector entries and ***y_i*** being one particular entry in the output vector.

When should You use Mean Squared Error loss?

The mean squared error loss function is the perfect loss function if you are dealing with a regression problem. That is if you want your neural network to predict a continuous scalar value.

An example of a regression problem would be predictions of …

- … the number of products needed in a supply chain
- … future real estate prices under certain market conditions,
- … a stock value.

Here is a code snippet where MSE loss is calculated in python:

```
import numpy as np

# The prediction Vector of the neural network
```

```
y_pred = [0.6,1.29,1.99,2.69,3.4]

# The ground truth label
y_hat = [1,1,2,2,4]

# Mean Squared Error
MSE = np.sum(np.square(np.subtract(y_hat, y_pred))) / len(y_hat)

print(MSE) # The result is 0.21606
```

3. Cross-Entropy Loss Function

Regression is only one of two areas where feedforward networks enjoy great popularity. The other area is classification.

In classification tasks, we are dealing with predictions of probabilities. Meaning the output of a neural network must be in a range between 0 and 1. A loss function that can measure the error between a predicted probability and the label which represents the actual class is called the cross-entropy loss function.

One important thing that we need to discuss before continuing with the cross-entropy is how exactly the ground truth vector does look like in the case of a classification problem.

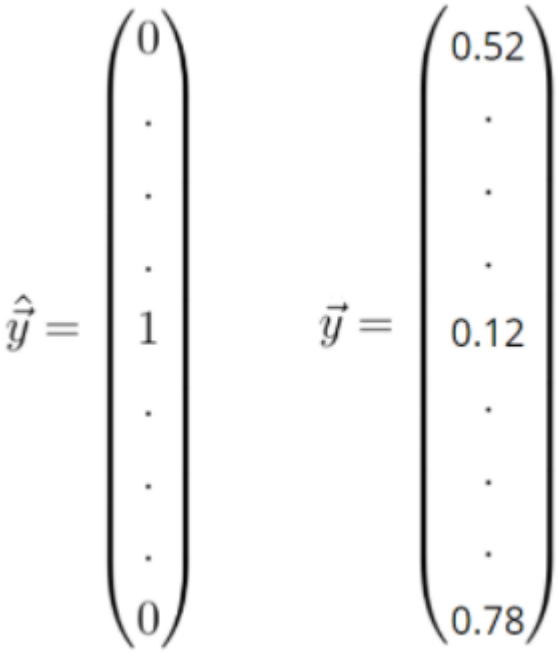


Fig. 2 One-hot-encoded vector (left), prediction vector (right). Source: Authors own image.

The label vector ***y_hat*** is one-hot-encoded which means that the values in this vector can only take discrete values of either 0 or 1. The entries in this vector represent different classes. The values of these entries are zero, except for one entry which is one. This entry of 1 tells us the class, we want to classify the input feature vector ***x*** into.

The prediction ***y***, however, can take continuous values between 0 and 1.

Given the prediction vector ***y*** and the ground truth vector ***y_hat*** you can compute the cross-entropy loss between those two vectors as follows:

$$\mathcal{L}(\theta) = -\sum_{i=0}^{N} \hat{y}_i \cdot log(y_i)$$

y_i : entries in the prediction vector $\vec{y}$

$\hat{y}_i$: entries in the ground truth label $\hat{\vec{y}}$

Eq. 4 Cross-entropy loss function. Source: Author's own image.

First, we need to sum up the products between the entries of the label vector ***y_hat*** and the logarithms of the entries of the predictions vector ***y***. In the second step, we must negate the sum to get a positive value of the loss function.

One interesting thing to consider is the plot of the cross-entropy loss function. In the following graph, the can see the value of the loss function (y-axis) vs. the predicted probability ***y_i. y_i takes values between 0 and 1.***

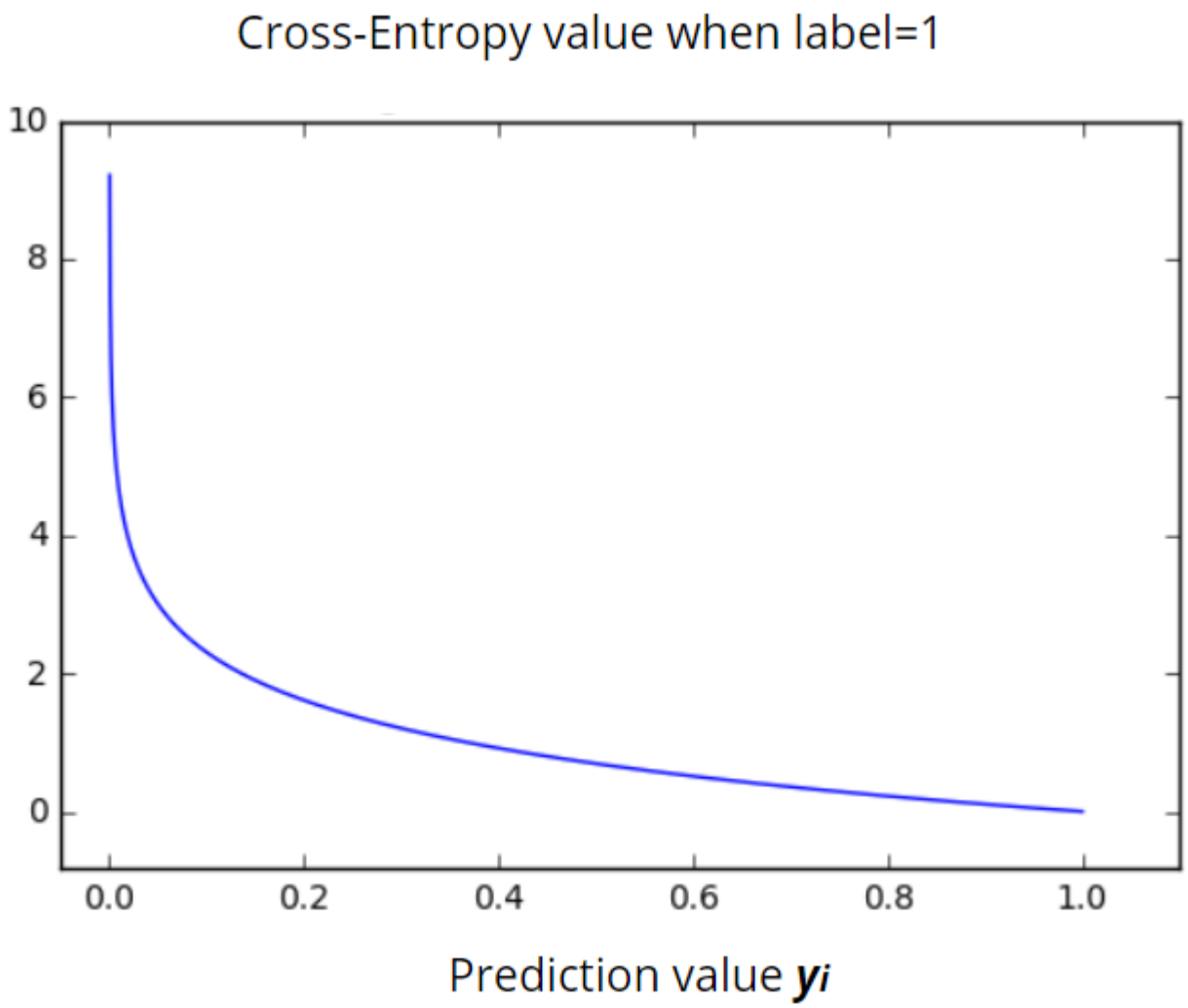


Fig. 3. Cross-Entropy function depending on prediction value. Source: Authors own image.

We can clearly see that the cross-entropy loss function grows exponentially for lower values of the predicted probability ***y_i.*** For ***y_i=0*** the function becomes infinite, while for ***y_i=1*** the neural network makes an accurate probability prediction and the loss value goes to zero.

Here is once more a code snippet in python, where the cross-entropy loss function is calculated:

```
import numpy as np

# The probabilities predicted by the neural network
```

```
# The probabilities predicted by the neural network
y_pred = [0.1, 0.3, 0.4 ,0.2]

# one-hot-encoded ground truth label
y_hat = [0, 1, 0, 0]

cross_entropy= - np.sum(np.log(y_pred)*y_hat)

print(cross_entropy)  # The result is 1.20
```

When should You use Cross-Entropy loss?

> You should always use cross-entropy loss if probabilities are involved. Meaning if you are doing some kind of classification.

Another thing to remember is that the label vector is one hot encoded and the values in this vector can only take discrete values of either a 0 or 1. The entries in this vector represent different classes.

The values of the entries are zero, except for one entry which is one. This entry tells us the class, we want to classify the input features into.

The prediction, however, can take continuous values between 0 and 1.

4. Mean Absolute Percentage Error

The third loss function that I would like to present to you is the Mean Absolute Percentage Error (MAPE) loss function. This loss function gets not that much attention in deep learning. It is used for the most part to measure the performance of a neural network during demand forecasting tasks.

But first, what is demand forecasting?

Demand forecasting is the area of predictive analytics dedicated to predicting the expected demand for a good or service in the near future. For example:

- In retail, one can use demand forecasting models to determine the amount of particular product that should be available and at what price
- In Industrial manufacturing, it could be predicted how much of each product should be produced, the amount of stock that should be available at various points in time, when maintenance should be performed
- In the travel and tourism industry demand forecasting models could be used to assess, in light of available capacity, what price should be assigned (for hotels, flights), which destinations should be highlighted, what types of packages should be advertised, etc.

Although demand forecasting is also a regression task and the minimization of the MSE loss function is an adequate training goal, this type of loss function to measure the performance of the model during training is not suitable for demand forecasting.

Why is that?

Well, imagine the MSE loss function gives you a value of 100. Can you tell if this is generally a good result? No, because it depends on the situation. If the prediction ***y*** of the model is 1000 and the actual ground truth label ***y_hat*** is 1010, then the MSE loss of 100 would be in fact a very small error and the performance of the model would be quite good.

However in the case where the prediction would be 5 and the label is 15, you would have the same loss value of 100 but the relative deviation to the ground-truth value would be much higher than in the previous case.

This example shows the shortcoming of the mean squared error function as the loss function for the demand forecasting models. For this reason, I would strongly recommend using another loss function, such as Mean Absolute Percentage Error (MAPE).

The mean absolute percentage error, also known as mean absolute percentage deviation (MAPD) usually expresses accuracy as a percentage, and is defined by the following equation:

$$MAPE = \frac{100\%}{N} \sum_{i=0}^{N} \frac{|y_i - \hat{y}_i|}{\hat{y}_i}$$

Source: Authors own image.

In this equation, ***y_i*** is the predicted value and ***y_hat*** is the label. The difference between ***y_i*** and ***y_hat*** is divided by the actual value ***y_hat*** again. Multiplying by 100% makes it a percentage error.

Applying this equation to the previous example gives you a more meaningful answer to the model's performance. In the first case, the deviation from the ground truth label would be only 1%, while in the second case the deviation would be 66%:

Source: Authors own image.

We see that the performance of these two models is very different. In the meantime, the MSE loss function would indicate that the performance of both models is the same.

The following python code snipped shows how MAPE can be calculated:

When should you use MAPE?

- Mean Absolute Percentage Error loss function should be used during demand forecasting tasks to evaluate the performance of the model during training time
- Meanwhile, the minimization of the MSE loss function can still be used as a training objective

Take-Home-Message

- A loss function measures how good a neural network model is in performing a certain task, which in most cases is regression or classification
- We must minimize the value of the loss function during the backpropagation step in order to make the neural network better
- The cross-entropy loss function is only used in classification tasks when we want the neural network to predict probabilities
- For regression tasks, when we want the network to predict continuous numbers we must use the mean squared error loss function
- Mean Absolute Percentage Error loss function is used during demand forecasting to keep an eye on the performance of the network during training time