Loss and Loss Functions for Training Deep Learning Neural Networks

by **Jason Brownlee** on January 28, 2019 in **Deep Learning Performance**

Last Updated on October 23, 2019

Neural networks are trained using stochastic gradient descent and require that you choose a loss function when designing and configuring your model.

There are many loss functions to choose from and it can be challenging to know what to choose, or even what a loss function is and the role it plays when training a neural network.

In this post, you will discover the role of loss and loss functions in training deep learning neural networks and how to choose the right loss function for your predictive modeling problems.

After reading this post, you will know:

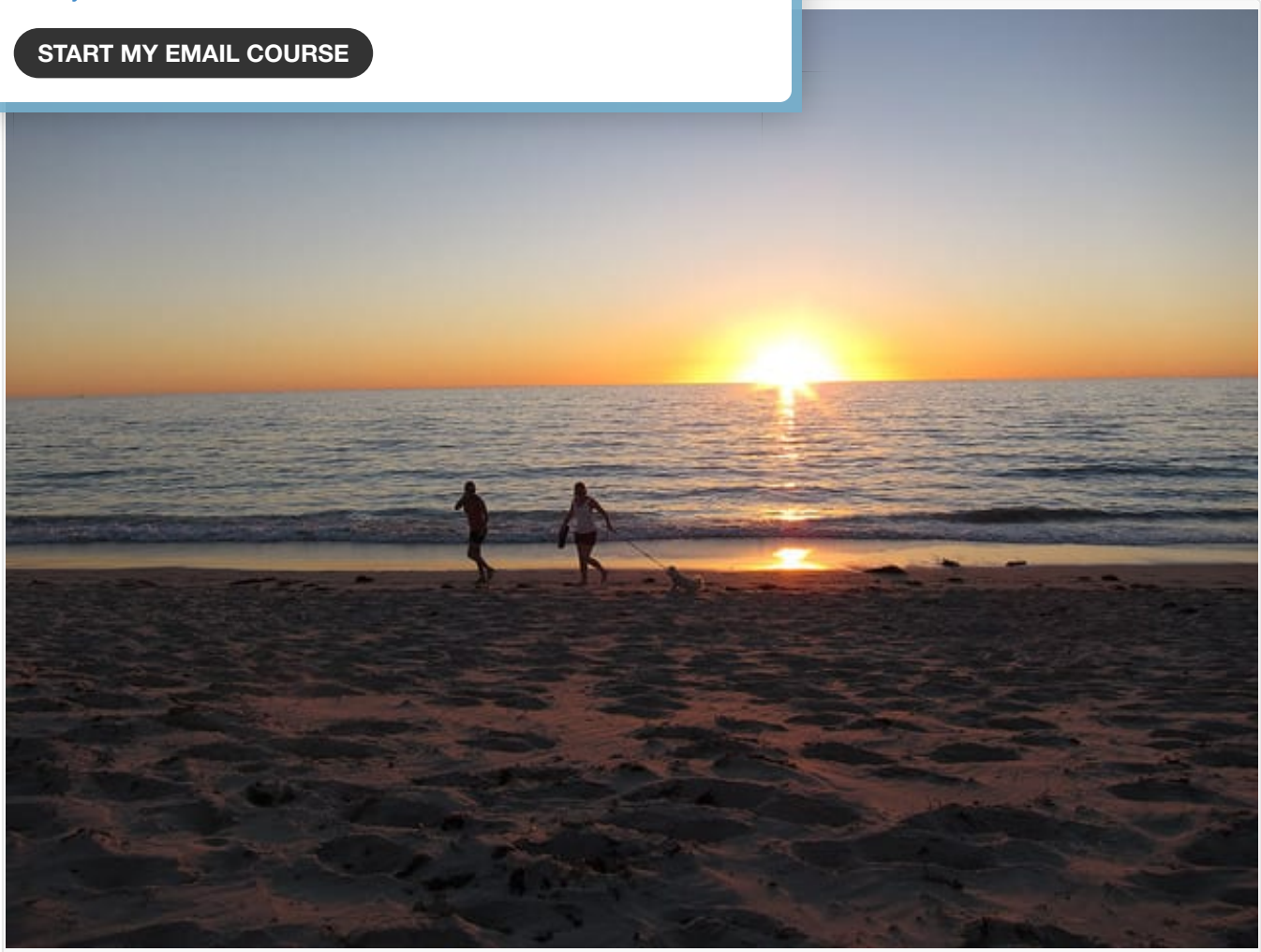
ess that requires a loss function

g a loss function when training
al.
n types of loss functions to use

rning, including *step-by-step*

special offers by email. For more information, see the Privacy Policy.

START MY EMAIL COURSE



Loss and Loss Functions for Training Deep Learning Neural Networks
Photo by Ryan Albrey, some rights reserved.

Overview

This tutorial is divided into seven parts; they are:

1. Neural Network Learning as Optimization
2. What Is a Loss Function and Loss?
3. Maximum Likelihood
4. Maximum Likelihood and Cross-Entropy
5. What Loss Function to Use?
6. How to Implement Loss Functions
7. Loss Functions and Reported Model Performance

We will focus on the theory behind loss functions.

For help choosing and implementing different loss functions, see the post:

- How to Choose Loss Functions When Training Deep Learning Neural Networks

Neural Network Learning as Optimization

A deep learning neural network learns to map a set of inputs to a set of outputs from training data.

We cannot calculate the perfect weights for a neural network; there are too many unknowns. Instead, the problem of learning is cast as a search or optimization problem and an algorithm is used to navigate the space of possible sets of weights the model may use in order to make good or good enough predictions.

Typically, a neural network model is trained using the stochastic gradient descent optimization algorithm and weights are updated using the backpropagation of error algorithm.

The "*gradient*" in gradient descent refers to an error gradient. The model with a given set of weights is used to make predictions and the error for those predictions is calculated.

The gradient descent algorithm seeks to change the weights so that the next evaluation reduces the error, meaning the optimization algorithm is navigating down the gradient (or slope) of error.

Now that we know that training neural nets solves an optimization problem, we can look at how the error of a given set of weights is calculated.

What Is a Loss Function and Loss?

In the context of an optimization algorithm, the function used to evaluate a candidate solution

(i.e. a set of weights) is referred to as the objective function.

We may seek to maximize or minimize the objective function, meaning that we are searching for a candidate solution that has the highest or lowest score respectively.

Typically, with neural networks, we seek to minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply "*loss*."

> *The function we want to minimize or maximize is called the objective function or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function.*

— Page 82, Deep Learning, 2016.

The cost or loss function has an important job in that it must faithfully distill all aspects of the model down into a single number in such a way that improvements in that number are a sign of a better model.

> *The cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared.*

— Page 155, Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, 1999.

In calculating the error of the model during the optimization process, a loss function must be chosen.

This can be a challenging problem as the function must capture the properties of the problem and be motivated by concerns that are important to the project and stakeholders.

> *It is important, therefore, that the function faithfully represent our design goals. If we choose a poor error function and obtain unsatisfactory results, the fault is ours for badly specifying the goal of the search.*

— Page 155, Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, 1999.

Now that we are familiar with the loss function and loss, we need to know what functions to

use.

Maximum Likelihood

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

We prefer a function where the space of candidate solutions maps onto a smooth (but high-dimensional) landscape that the optimization algorithm can reasonably navigate via iterative updates to the model weights.

Maximum likelihood estimation, or MLE, is a framework for inference for finding the best statistical estimates of parameters from historical training data: exactly what we are trying to do with the neural network.

> *Maximum likelihood seeks to find the optimum values for the parameters by maximizing a likelihood function derived from the training data.*

— Page 39, Neural Networks for Pattern Recognition, 1995.

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. Under maximum likelihood, a loss function estimates how closely the distribution of predictions made by a model matches the distribution of target variables in the training data.

> *One way to interpret maximum likelihood estimation is to view it as minimizing the dissimilarity between the empirical distribution […] defined by the training set and the model distribution, with the degree of dissimilarity between the two measured by the KL divergence. […] Minimizing this KL divergence corresponds exactly to minimizing the cross-entropy between the distributions.*

— Page 132, Deep Learning, 2016.

A benefit of using maximum likelihood as a framework for estimating the model parameters (weights) for neural networks and in machine learning in general is that as the number of examples in the training dataset is increased, the estimate of the model parameters improves. This is called the property of "*consistency*."

> *Under appropriate conditions, the maximum likelihood estimator has the property of consistency […], meaning that as the number of training examples approaches infinity, the maximum likelihood estimate of a parameter converges to the true value of the parameter.*

— Page 134, Deep Learning, 2016.

Now that we are familiar with the general approach of maximum likelihood, we can look at the error function.

Maximum Likelihood and Cross-Entropy

Under the framework maximum likelihood, 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.

Therefore, under maximum likelihood estimation, 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.

> *In most cases, our parametric model defines a distribution […] and we simply use the principle of maximum likelihood. This means we use the cross-entropy between the training data and the model's predictions as the cost function.*

— Page 178, Deep Learning, 2016.

Technically, cross-entropy comes from the field of information theory and has the unit of "bits." It is used to estimate the difference between an estimated and predicted probability distributions.

In the case of regression problems where a quantity is predicted, it is common to use the mean squared error (MSE) loss function instead.

> *A few basic functions are very commonly used. The mean squared error is popular for function approximation (regression) problems […] The cross-entropy error function is often used for classification problems when outputs are interpreted as probabilities of membership in an indicated class.*

— Page 155-156, Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, 1999.

Nevertheless, under the framework of maximum likelihood estimation and assuming a Gaussian distribution for the target variable, mean squared error can be considered the cross-entropy between the distribution of the model predictions and the distribution of the target variable.

> *Many authors use the term "cross-entropy" to identify specifically the negative log-likelihood of a Bernoulli or softmax distribution, but that is a misnomer. Any loss consisting of a negative log-likelihood is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by model. For example, mean squared error is the cross-entropy between the empirical distribution and a Gaussian model.*

— Page 132, Deep Learning, 2016.

Therefore, when using the framework of maximum likelihood estimation, we will implement a cross-entropy loss function, which often in practice means a cross-entropy loss function for classification problems and a mean squared error loss function for regression problems.

Almost universally, deep learning neural networks are trained under the framework of maximum likelihood using cross-entropy as the loss function.

> *Most modern neural networks are trained using maximum likelihood. This means that the cost function is […] described as the cross-entropy between the training data and the model distribution.*

— Page 178-179, Deep Learning, 2016.

In fact, adopting this framework may be considered a milestone in deep learning, as before being fully formalized, it was sometimes common for neural networks for classification to use

a mean squared error loss function.

> *One of these algorithmic changes was the replacement of mean squared error with the cross-entropy family of loss functions. Mean squared error was popular in the 1980s and 1990s, but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learning community.*

— Page 226, Deep Learning, 2016.

The maximum likelihood approach was adopted almost universally not just because of the theoretical framework, but primarily because of the results it produces. Specifically, neural networks for classification that use a sigmoid or softmax activation function in the output layer learn faster and more robustly using a cross-entropy loss function.

> *The use of cross-entropy losses greatly improved the performance of models with sigmoid and softmax outputs, which had previously suffered from saturation and slow learning when using the mean squared error loss.*

— Page 226, Deep Learning, 2016.

What Loss Function to Use?

We can summarize the previous section and directly suggest the loss functions that you should use under a framework of maximum likelihood.

Importantly, 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.

> *The choice of cost function is tightly coupled with the choice of output unit. Most of the time, we simply use the cross-entropy between the data distribution and the model distribution. The choice of how to represent the output then determines the form of the cross-entropy function.*

— Page 181, Deep Learning, 2016.

We will review best practice or default values for each problem type with regard to the output layer and loss function.

Regression Problem

A problem where you predict a real-value quantity.

- **Output Layer Configuration**: One node with a linear activation unit.
- **Loss Function**: Mean Squared Error (MSE).

Binary Classification Problem

A problem where you classify an example as belonging to one of two classes.

The problem is framed as predicting the likelihood of an example belonging to class one, e.g. the class that you assign the integer value 1, whereas the other class is assigned the value 0.

- **Output Layer Configuration**: One node with a sigmoid activation unit.
- **Loss Function**: Cross-Entropy, also referred to as Logarithmic loss.

Multi-Class Classification Problem

A problem where you classify an example as belonging to one of more than two classes.

The problem is framed as predicting the likelihood of an example belonging to each class.

- **Output Layer Configuration**: One node for each class using the softmax activation function.
- **Loss Function**: Cross-Entropy, also referred to as Logarithmic loss.

How to Implement Loss Functions

In order to make the loss functions concrete, this section explains how each of the main types of loss function works and how to calculate the score in Python.

Mean Squared Error Loss

Mean Squared Error loss, or MSE for short, is calculated as the average of the squared differences between the predicted and actual values.

The result is always positive regardless of the sign of the predicted and actual values and a perfect value is 0.0. The loss value is minimized, although it can be used in a maximization optimization process by making the score negative.

The Python function below provides a pseudocode-like working implementation of a function for calculating the mean squared error for a list of actual and a list of predicted real-valued quantities.

```
# calculate mean squared error
def mean_squared_error(actual, predicted):
    sum_square_error = 0.0
    for i in range(len(actual)):
        sum_square_error += (actual[i] - predicted[i])**2.0
    mean_square_error = 1.0 / len(actual) * sum_square_error
    return mean_square_error
```

For an efficient implementation, I'd encourage you to use the scikit-learn mean_squared_error() function.

Cross-Entropy Loss (or Log Loss)

Cross-entropy loss is often simply referred to as "*cross-entropy*," "*logarithmic loss*," "*logistic loss*," or "*log loss*" for short.

Each predicted probability is compared to the actual class output value (0 or 1) and a score is calculated that penalizes the probability based on the distance from the expected value. The penalty is logarithmic, offering a small score for small differences (0.1 or 0.2) and enormous score for a large difference (0.9 or 1.0).

Cross-entropy loss is minimized, where smaller values represent a better model than larger values. A model that predicts perfect probabilities has a cross entropy or log loss of 0.0.

Cross-entropy for a binary or two class prediction problem is actually calculated as the average cross entropy across all examples.

The Python function below provides a pseudocode-like working implementation of a function for calculating the cross-entropy for a list of actual 0 and 1 values compared to predicted probabilities for the class 1.

```
from math import log

# calculate binary cross entropy
def binary_cross_entropy(actual, predicted):
    sum_score = 0.0
    for i in range(len(actual)):
        sum_score += actual[i] * log(1e-15 + predicted[i])
    mean_sum_score = 1.0 / len(actual) * sum_score
    return -mean_sum_score
```

Note, we add a very small value (in this case 1E-15) to the predicted probabilities to avoid

ever calculating the log of 0.0. This means that in practice, the best possible loss will be a value very close to zero, but not exactly zero.

Cross-entropy can be calculated for multiple-class classification. The classes have been one hot encoded, meaning that there is a binary feature for each class value and the predictions must have predicted probabilities for each of the classes. The cross-entropy is then summed across each binary feature and averaged across all examples in the dataset.

The Python function below provides a pseudocode-like working implementation of a function for calculating the cross-entropy for a list of actual one hot encoded values compared to predicted probabilities for each class.

```
from math import log

# calculate categorical cross entropy
def categorical_cross_entropy(actual, predicted):
    sum_score = 0.0
    for i in range(len(actual)):
        for j in range(len(actual[i])):
            sum_score += actual[i][j] * log(1e-15 + predicted[i][j])
    mean_sum_score = 1.0 / len(actual) * sum_score
    return -mean_sum_score
```

For an efficient implementation, I'd encourage you to use the scikit-learn log_loss() function.

Loss Functions and Reported Model Performance

Given a framework of maximum likelihood, we know that we want to use a cross-entropy or mean squared error loss function under stochastic gradient descent.

Nevertheless, we may or may not want to report the performance of the model using the loss function.

For example, logarithmic loss is challenging to interpret, especially for non-machine learning practitioner stakeholders. The same can be said for the mean squared error. Instead, it may be more important to report the accuracy and root mean squared error for models used for classification and regression respectively.

It may also be desirable to choose models based on these metrics instead of loss. This is an important consideration, as the model with the minimum loss may not be the model with best metric that is important to project stakeholders.

A good division to consider is to use the loss to evaluate and diagnose how well the model is learning. This includes all of the considerations of the optimization process, such as overfitting, underfitting, and convergence. An alternate metric can then be chosen that has

meaning to the project stakeholders to both evaluate model performance and perform model selection.

- **Loss**: Used to evaluate and diagnose model optimization only.
- **Metric**: Used to evaluate and choose models in the context of the project.

The same metric can be used for both concerns but it is more likely that the concerns of the optimization process will differ from the goals of the project and different scores will be required. Nevertheless, it is often the case that improving the loss improves or, at worst, has no effect on the metric of interest.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Books

- Deep Learning, 2016.
- Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, 1999.
- Neural Networks for Pattern Recognition, 1995.

Articles

- Maximum likelihood estimation, Wikipedia.
- Kullback–Leibler divergence, Wikipedia.
- Cross entropy, Wikipedia.
- Mean squared error, Wikipedia.
- Log Loss, FastAI Wiki.

Summary

In this post, you discovered the role of loss and loss functions in training deep learning neural networks and how to choose the right loss function for your predictive modeling problems.

Specifically, you learned:

- Neural networks are trained using an optimization process that requires a loss function to calculate the model error.
- Maximum Likelihood provides a framework for choosing a loss function when training neural networks and machine learning models in general.
- Cross-entropy and mean squared error are the two main types of loss functions to use when training neural network models.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

More On This Topic

How to Choose Loss Functions When Training Deep…