A Brief Tutorial on the Relationship Between Maximum Likelihood Estimation and Cross-Entropy Loss

Thomas Phan
thomas.phan@acm.org

Technical Report
June 3, 2021

1 Introduction

Machine learning practitioners and engineers using neural network libraries such as TensorFlow and PyTorch usually follow a best practice recipe for training their models. For supervised classification tasks, a de facto step involves setting up an optimizer to minimize the *cross-entropy loss* between true and predicted labels. Practitioners should not take this step for granted; instead, they should understand the theoretical justification for using this loss function.

This report seeks to provide an explanation by giving a brief overview of the relationship between the two machine learning concepts of *maximum likelihood estimation* and cross-entropy loss. These concepts are relevant in categorical classification problems, where the input is an observation and the output is a label from one of K discrete possible classes.

2 Maximum Likelihood Estimation

Consider an example of Natural Language Processing (NLP) text classification, specifically sentiment analysis, where a machine learning system take as input an English sentence and then infers (or predicts) its sentiment, which can be one of K=3 discrete sentiment classes (Positive, Neutral, or Negative). [1]

Suppose we have three training examples and their respective ground-truth sentiment labels as shown in Table 1. Each ground-truth label is expressed as a one-hot probability distribution. For example, the one-hot distribution for training instance 1 is shown in Table 2.

That one-hot distribution can be interpreted to mean that training instance 1 has 100% probability of being Positive, 0% probability of being Neutral, and 0% probability of being Negative. We denote this ground-truth distribution as $y = [1.0, 0.0, 0.0]$.

[1] Another interesting problem comes from computer vision: given an image of an animal, a machine learning system infers what kind of animal it is out of a set of K animals.

	Input	True Label
1	The movie Interstellar is brilliant in its interweaving of science and humanity.	Positive
2	Inception has an acceptable plot and tolerable acting performances.	Neutral
3	Tenet is full of pretentious, overly-complex pseudoscience.	Negative

Table 1: Example training instances for text classification.

P(Positive)	P(Neutral)	P(Negative)
1.0	0.0	0.0

Table 2: Ground-truth probability distribution y for training instance 1.

P(Positive)	P(Neutral)	P(Negative)
0.7	0.2	0.1

Table 3: Example probability distribution $\hat{y}$ predicted by a ML model for training instance 1.

When the machine learning model is run to make an inference, the prediction's output takes the form of a probability distribution over the three labels. In deep learning, a neural network may output raw (unnormalized) scores for each class, which can then be normalized into a probability distribution by applying the softmax function. As an example, the machine learning model may output the distribution shown in Table 3. We denote this predicted distribution as $\hat{y} = [0.7, 0.2, 0.1]$.

The distribution $\hat{y}$ in Table 3 represents the likelihood (probability) of correctly predicting training example 1. More broadly, training example i has a predicted distribution $\hat{y_i}$ expressed as:

$$\text{Likelihood of training instance } i = \hat{y_i} = P(y_i|x_i;\theta) \tag{1}$$

We can interpret this expression to mean that for training instance i, the machine learning model computes $\hat{y_i}$, the predicted probability that y_i is the correct label for the input x_i, and that the model uses parameters θ, such as the weights and biases in a neural network.

Of course, we are interested in the predictions for all the training instances, not just the i^{th} one. Suppose there are N training instances. We would now like to compute the likelihood that all N predictions are correct. If the training instances are drawn from an independent and identically distributed (i.i.d.) population, then the joint likelihood that all of them are predicted correctly is the product of the likelihoods that each one of them is predicted correctly.

$$\text{Likelihood all instances predicted correctly} = \prod_{i}^{N} P(y_i|x_i;\theta) \tag{2}$$

Note that these probabilities are tiny floating-point numbers. When we use a real computer to perform the multiplication of these tiny numbers, we may get a numeric underflow, meaning that the value is too small to be accurately represented in memory (or in the hardware registers) of a computer's CPU. Instead, we can replace the likelihood (product of probabilities) with the numerical equivalent of log-likelihood (sum of log probabilities):

$$\text{Log-likelihood all instances predicted correctly} = \sum_{i}^{N} \log P(y_i|x_i;\theta) \tag{3}$$

A note about the logarithm base: We will use the log function throughout this paper. It does not matter what logarithm base is used as long as it is used consistently. As it happens, in the Python programming language the default `log()` and the `numpy.log()` functions compute the natural logarithm (base e). We use the natural logarithm in this paper. ■

When training a machine learning model, we would like to find parameters θ that produce the best results. That is, we would like to find θ that maximizes the likelihood that all instances are predicted correctly. We say that the *maximum likelihood estimate* (MLE) of the parameters is the parameter setting that maximizes this probability.

$$\theta_{MLE} = \underset{\theta}{\text{argmax}} \prod_{i}^{N} P(y_i|x_i;\theta) \qquad \text{Maximize likelihood} \qquad (4)$$

$$= \underset{\theta}{\text{argmax}} \sum_{i}^{N} \log P(y_i|x_i;\theta) \qquad \text{Maximize log-likelihood} \qquad (5)$$

Machine learning optimizers are commonly designed to search for a minimum instead of a maximum, so the parameters equivalently satisfy finding parameters that minimize the negative of the log-likelihood.

$$= \underset{\theta}{\text{argmin}} -\sum_{i}^{N} \log P(y_i|x_i;\theta) \qquad \text{Minimize negative log-likehood} \qquad (6)$$

Those parameters must also minimize the negative log-likelihood averaged over all the training instances.

$$= \underset{\theta}{\text{argmin}} -\frac{1}{N}\sum_{i}^{N} \log P(y_i|x_i;\theta) \qquad \text{Minimize average negative log-likehood} \qquad (7)$$

A note about terminology: In statistics literature, MLE is described as finding the hypothesis h that maximizes the probability of observing the training data given a hypothesis: $h_{MLE} = \underset{h}{\text{argmax}}(\mathcal{D}|h)$ (Chapter 3 of [Murphy 2012], Chapter 4 of [Murphy 2021]). The hypothesis h comprises both a model and the parameters of the model. The model in our context is a neural network, and the parameters are its weights and biases. In supervised machine learning, the training data $\mathcal{D}$ comprises pairs of (x, y) features and labels, so "probability of observing the data" means $P(y|x)$. ■

At this point, we have shown that training a machine learning algorithm is equivalent to finding the maximum likelihood estimate of the parameters, meaning those parameters that minimize the average negative log-likelihood. If we use those parameters θ_{MLE} in the model and make a prediction on the training set instances, then we achieve the highest likelihood that all those predictions are correct.

3 Cross-Entropy Loss

This concept of maximum likelihood estimation is directly related to cross-entropy loss, which is the standard metric to be minimized while training a machine learning classification model.

Cross-entropy loss measures the dissimilarity between two probability distributions p and q where p_k and q_k are the respective probabilities in each distribution for class k. By definition cross-entropy $H(p, q)$ has the form:

$$H(p, q) = -\sum_{k}^{K} p_k \log q_k \tag{8}$$

In our case, we are specifically interested in the two distributions y (the one-hot ground-truth distribution) and $\hat{y} = P(y_i|x_i; \theta)$ (the machine learning model's predicted distribution). For one training instance, the cross-entropy loss is:

$$H(y_i, \hat{y}_i) = -\sum_{k}^{K} y_{i,k} \log \hat{y}_{i,k} \tag{9}$$

where $y_{i,k}$ and $\hat{y}_{i,k}$ are the ground-truth and predicted probabilities, respectively, for the i^{th} training instance to have class label k. Recall the earlier examples of our two probability distributions $y = [1.0, 0.0, 0.0]$ and $\hat{y} = [0.7, 0.2, 0.1]$ for one of the sentiment analysis training instances. We can apply the formula above to find that the cross entropy loss between them is $H(y_i, \hat{y}_i) = -(1.0 \times \log 0.7 + 0.0 \times \log 0.2 + 0.0 \times \log 0.1) = 0.357$.

Note the following:

- Because we are using the natural log (base e), the units of cross-entropy here are in *nats*; that is, the cross-entropy loss is 0.357 nats. If we had instead used the logarithm with base 2, then the units would have been in *bits*.
- Among the K=3 classes in our earlier sentiment analysis example, the ground-truth class for this training instance is Positive, as indicated by the one-hot distribution. We denote the ground-truth class for a training example as k_{gt}.

We are interested in the total cross-entropy loss over all N training instances, which is defined as the mean of the individual cross-entropy losses:

$$H_{\text{total}}(y, \hat{y}) = -\frac{1}{N}\sum_{i}^{N} H(y_i, \hat{y}_i) \tag{10}$$

$$= -\frac{1}{N}\sum_{i}^{N}\sum_{k}^{K} y_{i,k} \log \hat{y}_{i,k} \tag{11}$$

We already know that the distribution y_k is one-hot, so we can remove the inner summation over K since only the probability of the ground-truth class k_{gt} in y_k is non-zero.

$$= -\frac{1}{N}\sum_{i}^{N} \log \hat{y}_{i,k_{gt}} \tag{12}$$

The expression $\hat{y}_{i,k_{gt}}$ is the probability distribution $P(y_i|x_i;\theta)$.

$$= -\frac{1}{N}\sum_{i}^{N} \log P(y_i|x_i;\theta) \tag{13}$$

During training of our machine learning model, we set up an optimizer to find parameters θ_{opt} that minimize the cross-entropy loss $H_{\text{total}}(y, \hat{y})$:

$$\theta_{opt} = \underset{\theta}{\text{argmin}} \; -\frac{1}{N}\sum_{i}^{N} \log P(y_i|x_i;\theta) \tag{14}$$

At this point we have reached the understanding that cross-entropy loss is a means to measure the dissimilarity of two probability distributions. In machine learning practice, we have two distributions of interest: the ground-truth one-hot distribution y and the predicted distribution $\hat{y}$. The role of the machine learning optimizer during training is then to find the parameters θ_{opt} that minimizes the cross-entropy loss to get the predicted distribution to be as close as possible to the ground-truth distribution.

4 The relationship between MLE and Cross-Entropy Loss

Recall that Equation 7 defines the parameters θ_{MLE} as those that minimize the average negative log-likelihood of the data, while Equation 14 defines the optimal parameters θ_{opt} that minimizes the cross-entropy loss. We repeat them here for convenience:

$$\theta_{MLE} = \underset{\theta}{\text{argmin}} \; -\frac{1}{N}\sum_{i}^{N} \log P(y_i|x_i;\theta)$$

$$\theta_{opt} = \underset{\theta}{\text{argmin}} \; -\frac{1}{N}\sum_{i}^{N} \log P(y_i|x_i;\theta)$$

The two equations are exactly the same! This equivalence shows that when we train a machine learning model with an optimizer that minimizes the cross-entropy loss, we are equivalently training a model that minimizes the average negative log likelihood of the data, which in turn means that we are maximizing the likelihood of the data. Thus, minimizing the cross-entropy loss is exactly what we want: we are finding parameters that can be used to predict the observed data as closely as possible.

Another way to think about the relationship is that the MLE negative log-likelihood term is equivalent to the cross-entropy loss $H(y, \hat{y})$ where the ground-truth distribution y is implicitly one-hot.

5 Additional Notes

5.1 Kullback-Leibler Divergence

The Kullback-Leibler Divergence is a measurement for the dissimilarity between two probability distributions. In fact, one of its components is cross-entropy loss.

KL Divergence is defined on two probability distributions p and q like this:

$$KL(p||q) = -\sum k^K p_k \log q_k + \sum_{k}^{K} p_k \log p_k \tag{15}$$

$$= -\sum k^K p_k \log q_k - (-\sum_{k}^{K} p_k \log p_k) \tag{16}$$

Note that the right hand side's first time has two terms: the first term is the cross-entropy loss $H(p, q)$ that we saw earlier, while the second term is the negative of the cross-entropy loss with a single distribution, $H(p, p)$, which is known as simply the entropy $H(p) = -\sum_k^K p_k \log p_k$.

$$= H(p, q) - H(p) \tag{17}$$

5.2 One-hot alternative

The one-hot distribution is used pervasively in machine learning to represent the ground-truth distribution. In some statistics contexts [Murphy 2021], the ground-truth distribution is represented by a *delta function*, which has approximately the same semantics as a one-hot distribution. One such delta function is the Dirac delta function defined as:

$$\delta(x) = \begin{cases} +\infty & x = 0 \\ 0 & x \neq 0 \end{cases}, \text{ where } \int_{-\infty}^{\infty} \delta(x)\ dx = 1$$

5.3 Likelihood alternative form

We saw in Equation 2 one way to express the joint likelihood of the training data. An alternative formalism [Webb 2017] expresses the same joint likelihood in a way that makes its relationship to cross-entropy loss more immediate:

$$\text{Likelihood all instances predicted correctly} = \prod_{i}^{N} \prod_{k}^{K} (\hat{y}_{i,k})^{y_{i,k}} \tag{18}$$

where $y_{i,k}$ and $\hat{y}_{i,k}$ are the ground-truth and predicted probabilities, respectively, for the i^{th} training instance to have class label k.

(19)

References

[Deisenroth et al. 2020] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. 2020. *Mathematics for Machine Learning*. Cambridge University Press.

[Goodfellow et al. 2016] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press Adaptive Computation and Machine Learning series.

[Murphy 2012] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press Adaptive Computation and Machine Learning series.

[Murphy 2021] Kevin P. Murphy. 2021. *Probabilistic Machine Learning: An Introduction*. MIT Press Adaptive Computation and Machine Learning series.

[Webb 2017] Andrew M. Webb. 2017. *Cross Entropy and Log Likelihood*. `http://www.awebb.info/probability/2017/05/18/cross-entropy-and-log-likelihood.html`, retrieved June 2, 2021.