Maximum Likelihood -> Negative Log Loss -> Cross Entropy

This post explains the derivation of Cross Entropy by applying Maximum Likelihood principle to a classification problem.

The objective is to explain why we use Cross Entropy to evaluate the fitness of our classification models. We will first make a review on the concept of maximum likelihood. Later, we will get into the "whys" and the "hows" of applying the cross-entropy loss to train a model with a softmax layer to perform multiclass classification. Finally, we will derive mathematically the expressions for the forward and the backward pass.

Table of Contents

[1. Scenario 2](#_Toc16512106)

[2. Maximizing the likelihood is equivalent to minimizing the negative log likelihood 3](#_Toc16512107)

[Maximum Likelihood 3](#_Toc16512108)

[Negative log loss: 4](#_Toc16512109)

[KL Divergence - Cross Entropy: 4](#_Toc16512110)

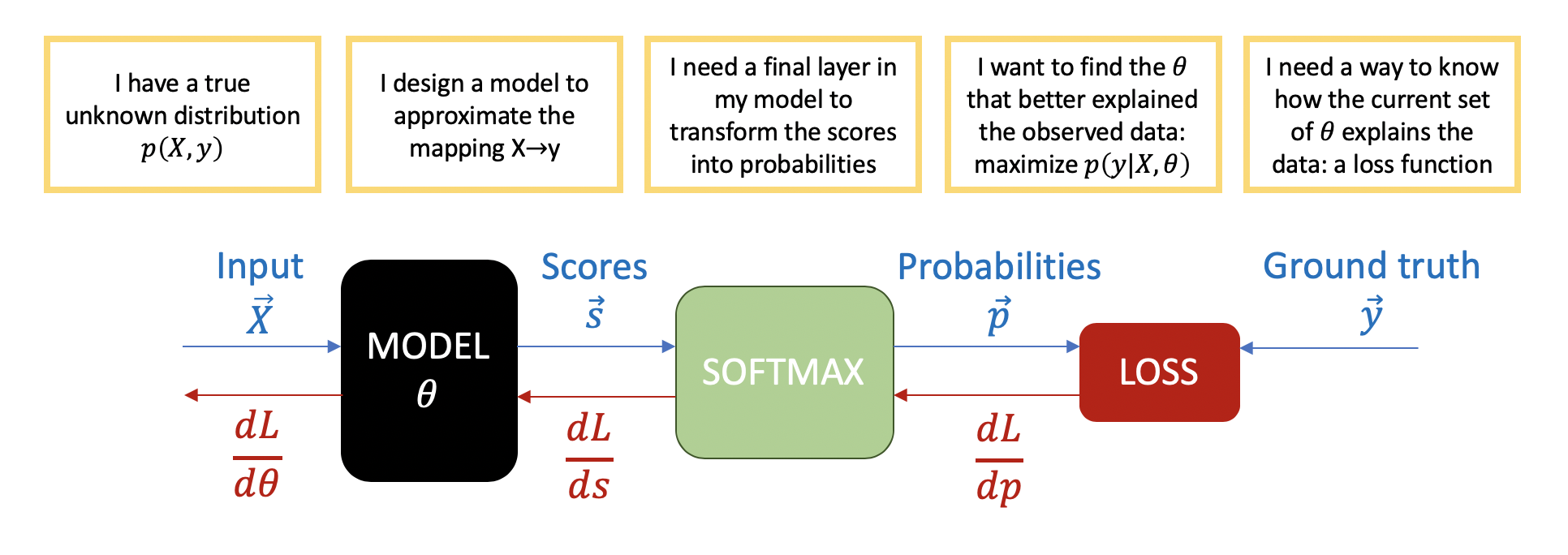
[3. Cross Entropy arises naturally with Maximum Likelihood Estimation 5](#_Toc16512111)

[Forward pass 5](#_Toc16512112)

[Backward pass 6](#_Toc16512113)

## 1. Scenario

We define a starting point for a classification problem to derive the rest of the definitions based on it.



The figure can be described in word as follows:

* We have a dataset of pairs inputs-labels . As an example, think of as an image, and as the corresponding label for a dataset of given labeled images.
* We define a parametrized model which outputs a -dimensional vector, where is number of classes for the classification problem. We want this model to learn the mapping function that relates inputs to labels.
* Because the scores vector will output uncontrollable output values, (we don’t know what the value of the parameters are going to be during the training) we need a layer that normalizes this vector, transforming the scores into probabilities. Each of the entries represent the probability that the input represents each of the classes. Here we notice the need for a Softmax layer.
* Lastly, we need a way to know how good our model is doing. We do this by maximizing the likelihood of the parameters of our model given the observed data. In software applications, we phrase this maximization as the minimization of a cost functions. Here we notice the need of the Cross-Entropy Loss (CEL), which we will see arises naturally when applying maximum likelihood to our current setup.

## 2. Maximizing the likelihood is equivalent to minimizing the negative log likelihood

**Probability**: how probable is to observe that data given the parameters :

**Likelihood:** what is the likelihood that these parameters are correct given the observed data

**Problem setup:**

* There is a true but unknown data distribution we want to model from the observed data .
* There is an empirical distribution defined by this training data, we assumed independently sampled from the true distribution.
* There is a distribution governed by the parameters of the model we define

We then build a model which parameters are , which yield to a new data distribution .

### Maximum Likelihood

We want to find the which maximize the likelihood of the resulting distribution to be the observed. That is why we call Maximum Likelihood to the process of finding the combination of values that are most likely to be the parameters of the true underlying distribution based on the observed data.

#### The i.i.d. assumption:

We assume all the samples are **independent and identically distributed** in the true distribution.

Therefore, the joint probability of all the sample can be decompose as the product of the individual probabilities of each observation:

#### The Log-trick:

Since we are calculating the position where the maximum is found and not the maximum itself, we can conveniently apply the logarithm to previous equation. The value of the maximum will change, but not the values where it is located, .

*Why doing this?*

We want to perform a maximization; therefore, we want to find where the derivative becomes 0. Deriving the product of all those terms becomes very tricky. However, by having the sum of terms it becomes way easier.

#### Conditioned Likelihood:

Maximum Likelihood Estimation (MLE) can be generalized to estimate a conditional probability . This is the most common situation, because it forms the basis of supervised problems, where we want to maximize the joint probability of observing the inputs and the targets. In particular in image classification, to maximize the probability of the observations and their corresponding labels. Our maximum likelihood estimator changes to:

### Log vs neglog.gifNegative log loss:

Log Loss uses the negative logarithm to provide an easy metric for comparison. It takes this approach because the positive log of numbers < 1 returns negative values, which is confusing to work with when comparing the performance of two models.

Because we are setting the negative sign, we change from maximizing the function to minimize the negative of that function.

This equation is showing how maximizing the likelihood (or the joint likelihood) is equivalent to minimize the negative log likelihood, NLL.

### KL Divergence - Cross Entropy:

Another interpretation of the MLE, is to minimize the dissimilarity between the empirical distribution and the model distribution, measured by the KL divergence.

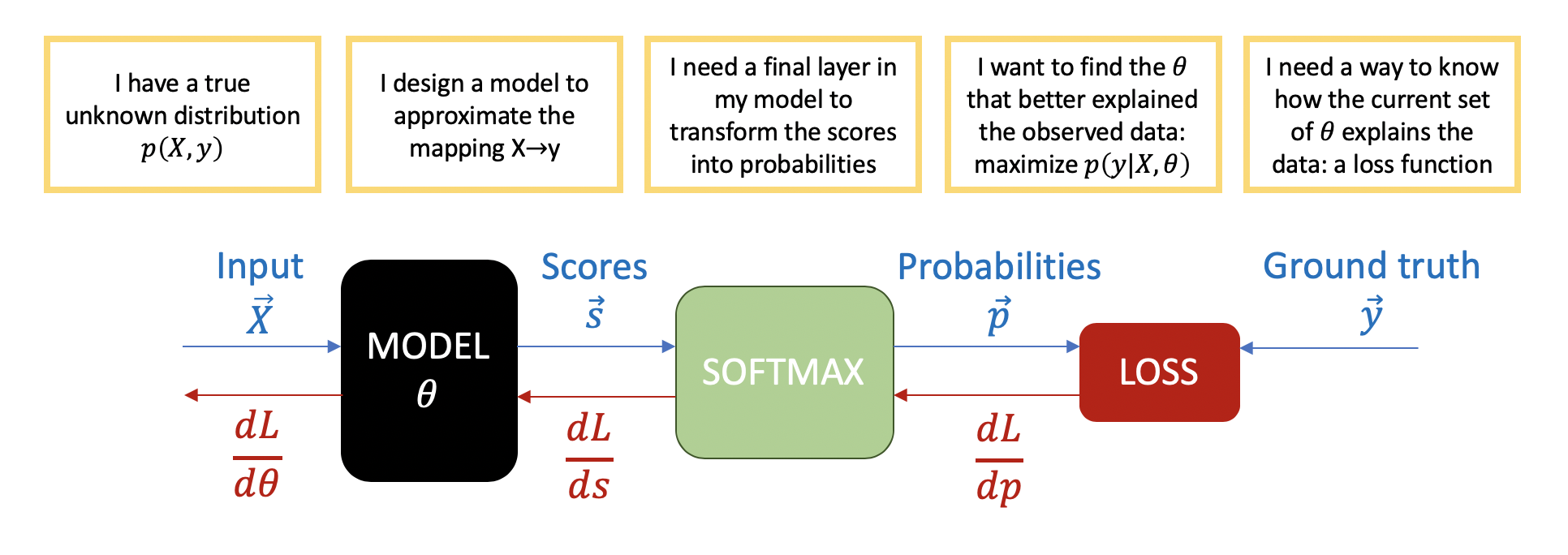
And because does not depend on the model, minimizing the equation above is equivalent to minimize:

Any loss consisting of a negative log likelihood like the above one is a cross-entropy between the empirical distribution defined by the observed data and the probability distribution defined by the model.

For instance, least squares if a cross-entropy between and a Gaussian assumption on the response variable.

## 3. Cross Entropy arises naturally with Maximum Likelihood Estimation

Returning to our scenario:



### Forward pass

[1] SOFTMAX --> Transform scores into probabilities

|  |  |
| --- | --- |
|  | (1) |

Softmax takes an input vector of scores and returns a vector of probabilities .

[2] MAXIMUM LIKELIHOOD --> How close are these probabilities to the ground truth?

At the end of the day, we are interested in the derivative of the loss with respect to the parameters. This derivative is the one we will use to update the parameters (think of the weights of a neural network) to perform better next time (learning).

We are modeling a multinomial distribution and we know that the maximum likelihood solution for a multinomial distribution is equivalent to minimize the negative log-likelihood of our data wr.t. the parameters, which is equivalent to minimize the cross entropy between the true distribution and the distribution generated by our model.

However, let's derive it. The likelihood function for a multinomial distribution

Taking the negative logarithm to have the NLL function we want to minimize (resulting the cross-entropy):

|  |  |
| --- | --- |
|  | (2) |

**Note 1**

is a one-hot encoded vector, where every entry is zero except the -th entry, representing the class. is an matrix with elements

**Note 2**

That solution for the maximum likelihood correspond to the cross-entropy for the multiclass classification problem.

### Backward pass

We are looking for to backpropagate the error into the NN. We can apply the chain rule to decompose:

where the first term is the derivative of the cross-entropy loss and the second term is the derivative of the softmax function. This is the reason why normally the derivative of the loss with respect to the scores is directly taken involving both softmax and cross-entropy together. This simplifies greatly the calculations.

#### [1] DERIVATIVE OF SOFTMAX

To derivate the numerator, we can use:

Therefore, we have different solutions if or not, that we need to derive independently:

- Solution for

- Solution for

where in both cases we have made use of Equation 1. If we combine both solutions making use of [Kronecker Delta](mailto:https://en.wikipedia.org/wiki/Kronecker_delta):

|  |  |
| --- | --- |
|  | (3) |

#### [2] DERIVATIVE OF CROSS ENTROPY

We have derived already the first component of the chain rule, the local gradient. We need know to compute the upcoming gradient to backpropagate it into the network so we can update their weights accordingly to their contribution to the error.

Using Equation 2 for a single prediction:

Substituting 3:

If we decompose the sum now on terms where and :

Note how the above purple selection is 1 as it is the sum of all possible values of a one-hot encoded vector.

|  |  |
| --- | --- |
|  | (4) |