# Cross Entropy Loss Derivative

Roei Bahumi

In this article, I will try and explain the concept of the Cross-Entropy Loss, commonly called the "Softmax Classifier", its usage in Deep Learning classification tasks and go through the mathematics of the function derivatives required for the Gradient Descent algorithm.

## A brief overview of relevant functions

### Cross-Entropy

Information Theory's Cross-Entropy function is a function that measures the difference between the true distribution $p$ and the estimated distribution $q$:

$$H(p,q) = -\sum_x p(x) \log q(x)$$

Note that the cross-entropy is not a distance function because $H(p,q) \neq H(q,p)$.

### Softmax

The Softmax function is a function $\mathbb{R}^K \to \mathbb{R}^K$ that maps a vector $z \in \mathbb{R}^K$ to a vector $q \in \mathbb{R}^K$ such that:

$$q_i(z) = \frac{e^{z_i}}{\sum_{j \in \{1,..K\}} e^{z_j}} \quad \forall i \in \{1,..K\}$$

Note that the denominator of each element in $q$ is the sum of numerators of all the elements, which satisfy:

$$0 \leq q_i \leq 1 \quad \forall i \in \{1,..K\}$$
$$and \quad \sum_{i \in \{1,..K\}} q_i = 1$$

and therefore is a suffice discrete probability distribution over $K$ values.

The Softmax function can normalize any real vector $z$ into a probability distribution $q$. The input vector $z$ can be interpreted as the unnormalized log probabilities, and the output $q$ as a probability vector over the $K$ values, which is exponentially proportional to $z$.

## Softmax in Supervised Learning Classification

The Softmax function is commonly used as a normalization function in Supervised Learning Classification task in the following high-level structure:

1. A deep ANN is used as a feature extractor. This network's task is to take the raw input and create a non-linear mapping that can be used as features to a classifier.

2. A fully connected linear layer with K units (where K is the number of classes). This layer's output can be interpreted as the unnormalized log probabilities a.k.a logits.

3. A softmax activation function with K output units, which can be interpreted as the normalized probability that the current sample belongs to each of the $K$ classes.

Figure 1 shows an example of a basic classifier for an image classification task with $K = 3$

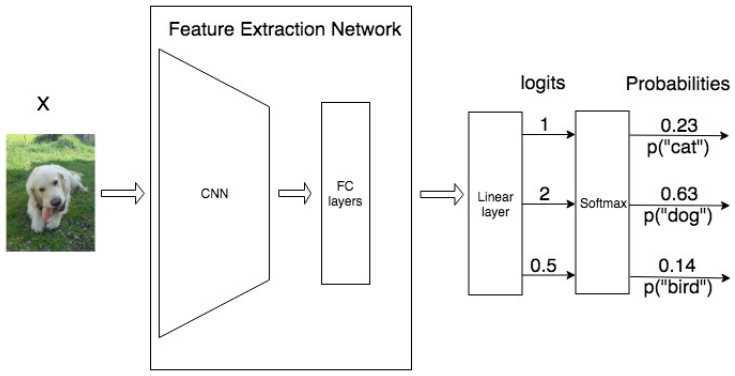



Figure 1: An example of a basic classifier for image classification task with $K = 3$. The feature extraction network is a deep convolutional network, followed by a few fully connected layers. A linear layer is added to compute the logits from the extracted features and is followed by a softmax activation which normalizes the logits and outputs a discrete probability distribution over the 3 classes.

## Cross-Entropy Loss Function

In order to train an ANN, we need to define a differentiable loss function that will assess the network predictions quality by assigning a high/low loss value in correspondence to a correct/wrong prediction respectively. When training the network with the backpropagation algorithm, this loss function is the last computation step in the forward pass, and the first step of the gradient flow computation in the backward pass.

In a Supervised Learning Classification task, we commonly use the cross-entropy function on top of the softmax output as a loss function. We use a 1-hot encoded vector for the true distribution $p$, where the 1 is at the index of the true label ($y$):

$$p_i(x) = \begin{cases} 1 & \text{if y=i} \\ 0 & \text{otherwise} \end{cases}$$

and the output of the softmax function over the logits ($z(x)$) as our $q$:

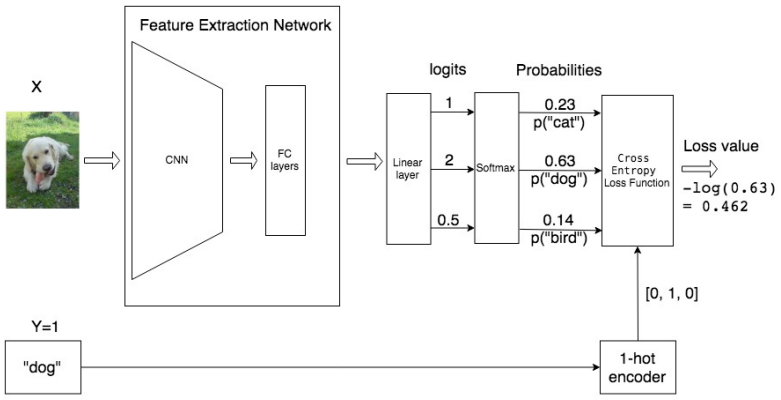
$$q_i(z) = \frac{e^{z_i}}{\sum_{j \in \{1,..K\}} e^{z_j}} \quad \forall i \in \{1,..K\}$$

This loss function is sometimes also referred to as the Softmax Classifier.

Figure 2 shows an example for a cross entropy loss calculation of an image classification task with $K = 3$ classes and the following index mapping: {0:"cat", 1:"dog", 2:"bird"}. Given an input image $x$, the logits layer outputs the unnormalized log probabilities vector: $(1, 2, 0.5)$, and the corresponding softmax output (using a 2 digit precision) is $q(x) = (0.23, 0.63, 0.14)$. Given the true label "dog" ($y = 1$), we generate the relevant 1-hot encoding vector: $p(x) = (0, 1, 0)$. The cross-entropy loss value for these $p(x)$ and $q(x)$ is then:

$$\begin{aligned} H(p, q) &= -\sum_x p(x) \log q(x) \\ &= -0 * log(0.23) - 1 * log(0.63) - 0 * log(0.14) \\ &= -log(0.63) = 0.462 \end{aligned}$$

Note that the 1-hot encoded vector $p(x)$ acts as a selector, and the loss can be written as $-log(q_y)$ where $y$ is the index of the true label. Because $0 \leq q_y \leq 1$, and $log(0) = \infty$ and $log(1) = 0$, the loss value resides within the interval $[0, +\infty)$. The loss is infinite when the classifier assigns 0 probability to the true class and is equal to 0 when the classifier assigns it a probability of 1.




Class index mapping: $\{0 : "cat", 1 : "dog", 2 : "bird"\}$

$p(x) = (0, 1, 0)$

$z(x) = (1, 2, 0.5)$

$q(x) = softmax(z(x)) = (0.23, 0.63, 0.14)$

$$Loss = -\sum_x p(x) \log q(x) = -0 * log(0.23) - 1 * log(0.63) - 0 * log(0.14) = -log(0.63) = 0.462$$

Figure 2: An example of a cross entropy loss calculation of an image classification task with $K = 3$ classes.

## Cross-Entropy derivative

The forward pass of the backpropagation algorithm ends in the loss function, and the backward pass starts from it. In this section we will derive the loss function gradients with respect to $z(x)$.

Given the true label $Y = y$, the only non-zero element of the 1-hot vector $p(x)$ is at the $y$ index, which in practice makes the $p(x)$ vector a selector for the $y$ index in the $q(x)$ vector. Therefore, the loss function for a single sample then becomes:

$$Loss = -\log(q_y) = -\log(\frac{e^{z_y}}{\sum_j e^{z_j}}) = -z_y + \log \sum_j e^{z_j}$$

Calculating the derivative for each $z_i$:

$$
\begin{aligned}
\nabla_{z_i} Loss &= \nabla_{z_i}(-z_y + \log \sum_j e^{z_j}) \\
&= \nabla_{z_i} \log \sum_j e^{z_j} - \nabla_{z_i} z_y \\
&= \frac{1}{\sum_j e^{z_j}} \nabla_{z_i} \sum_j e^{z_j} - \nabla_{z_i} z_y \qquad \text{from} \quad \frac{d}{dx} ln[f(x)] = \frac{1}{f(x)} \frac{d}{dx} f(x) \\
&= \frac{e^{z_i}}{\sum_j e^{z_j}} - \nabla_{z_i} z_y \\
&= q_i - \nabla_{z_i} z_y \\
&= q_i - \mathbb{1}(y = i) \\
&\text{where} \quad \mathbb{1}(y = i) = \begin{cases} 1 & \text{if y=i} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

These results show:

- $\nabla_{z_y} Loss = q_y - 1$
  The gradient for the true label's logit is non-positive and decrease proportionally in magnitude as $q_y$ increases.

- $\nabla_{z_i} Loss = q_i \quad \forall i \neq y$
  The rest of the logits gradient ($q_i$) will be non-negative and increase proportionally as $q_i$ increases.

- In the specific case of perfect classification where $q_y = 1$, the gradient will be $\vec{0}$ and thus non of the network's parameters will be modified.

Gradients are directed towards the maximal value increase of their function. As expected for our loss function, increasing the probability of the true label's class will decrease the loss, and increasing the probability of each of the incorrect classes will increase the loss. When running gradient decent we will update the network parameters in the counter direction to the gradient in order to minimize the loss. As a result, the network will try to move all the probability mass towards the correct class, which will reduce the current training batch loss, and (hopefully) will generalize and improve the classification of new unseen inputs.

Figure 2 showed a forward pass of backpropagation on a single example, for which the true label was $Y = 1$ and the softmax output $(0.23, 0.63, 0.14)$. Figure 3 show the backward backpropagation pass for the same example. The initial gradient is 1, and the logits gradients for each class $i$ are $q_i - \mathbb{1}(y = i)$.

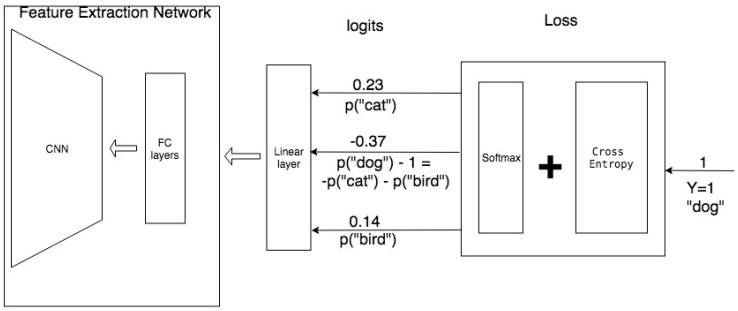



Figure 3: Figure 2 showed a forward pass of backpropagation on a single example, for which the true label was $Y = 1$ and the softmax output $(0.23, 0.63, 0.14)$. Here we see the backward backpropagation pass for the same example. The initial gradient is 1, and the logits gradients for each class $i$ are $q_i - \mathbb{1}(y = i)$.