Classification and Cross-Entropy Loss

Jed Guzelkabaagac

April 23, 2021

In the following article I will discuss cross-entropy loss and its relationship with the softmax function. Nomenclature in this area is intially quite confusing, so I'll focus on carefully differentiating between different terminologies. Towards the end I'll touch upon derivatives and their role in gradient descent.

Definitions of Functions

Cross-Entropy

For discrete distributions p, q over the same support X, the cross-entropy of q relative to p is defined as follows:

$$H(p,q) = -\sum_{x \in X} p(x)\, \log\, q(x)$$

This function has a very close relationship with the *likelihood* function.

Aside: Relationship with Likelihood

In classification problems we want to estimate the probability of different outcomes. If the estimated probability of outcome i is $q(i)$, while the frequency (empirical probability) of outcome i in the training set is $p(i)$, and there are N conditionally independent samples in the training set, then the likelihood of the training set is

$$L(\ \underline{q} \mid \text{observations} = Np_i\ ) = \prod_i q_i^{Np_i}$$

The log-likelihood, normalised by the number of samples in the training set N is...

$$\frac{1}{N}\, \log\, \prod_i q_i^{Np_i} = \sum_i p(x)\, \log\, q(x) = -H(p,q)$$

Thus, minimising cross-entropy loss through SGD is equivalent to maximising the likelihood function of the final activation layer probabilities.

Softmax Function

The softmax function $\mathbb{R}^k \rightarrow \mathbb{R}^k$ maps a vector $\mathbf{z} \in \mathbb{Z}^k$ to $\mathbf{q} \in \mathbb{Z}^k$ as follows:

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

Crucially, the resulting vector $\mathbf{q}$ defines a probability distribution, since:

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

Softmax is very useful as it turns numbers, aka *logits*, into probabilities that sum to one. Intuitively, softmax function outputs a vector that represents the probability distributions of a list of potential outcomes.

Conceptualisation of a Classification Network

To understand a multi-level classification network, consider the following high-level conceptualisation:

1. The 'deep' part of the network. Intuitively, early layers capture very elementary and abstract features. Then, using the necessary complexity offered by non-linear activations, more complex features are constructed in deeper layers and used to distinguish between classes.

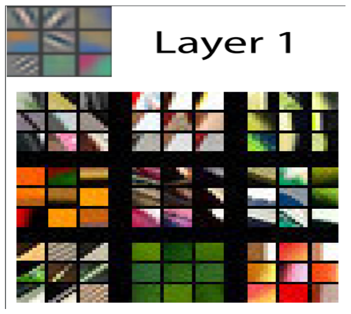

Figure 1: For each layer, the image part with the light gray background shows the reconstructed weights pictures, and the larger section at the bottom shows the parts of the training images that most strongly matched each set of weights. Taken from [1].

2. One or more fully connected layers, followed by 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.

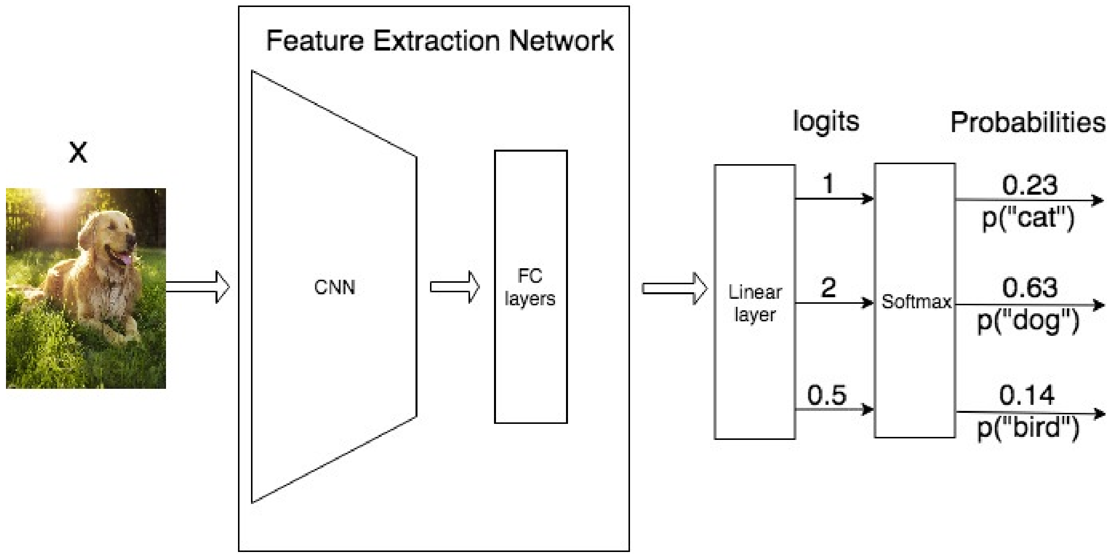

Figure 2: A basic classifier for image classification with K=3 . The feature extraction network is a deep convolutional network, followed by fully connected layers. A linear layer is added to compute the logits from the features and is followed by a softmax activation, normalising the logits and outputting a discrete probability distribution over the classes.

Note the last layer is often called the *logits layer* since it gives the 'raw' prediction values in the range $[-\infty, \infty]$. After this, the softmax squishes the figures into a probability distribution.

Given a model (an architecture and the parameters) we can take an inupt, calculate the logits, and then use softmax 'squishification' to obtain a probability distribution! To train an ANN however, we need to define a differentiable loss function that will assess the quality of the predictions by assigning a low/high loss value in correspondence to a correct/wrong prediction, respectively.

Cross-Entropy Loss

For supervised learning classificaion tasks, it's very common to use softmax in combination with cross-entropy loss. This is since cross-entropy is quite simpistic yet effective, as shown by its relationship with the likelihood function. Additionally, since softmax is defined by the exponential function, it characteristically grows extremely fast. Consequently in classification tasks it's decisive at choosing a 'winning class'.

Usually, a one-hot encoding is used for the true distribution, $\mathbf{p} \in \mathbb{R}^K$. This simply means that if the true label y corresponds to index $1 \leq k \leq K$, then $\mathbf{p}$ is defined by:

$$p_i = \mathbb{1}\{i = k\} \ \ \forall i \in \{1, ..., K\}$$

For the 'estimated probability distribution' $\mathbf{q} \in \mathbb{R}^K$, we use the output of the softmax over the logits z_i:

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