Thomas Kurbiel Follow

Apr 22, 2021 · 6 min read · Listen

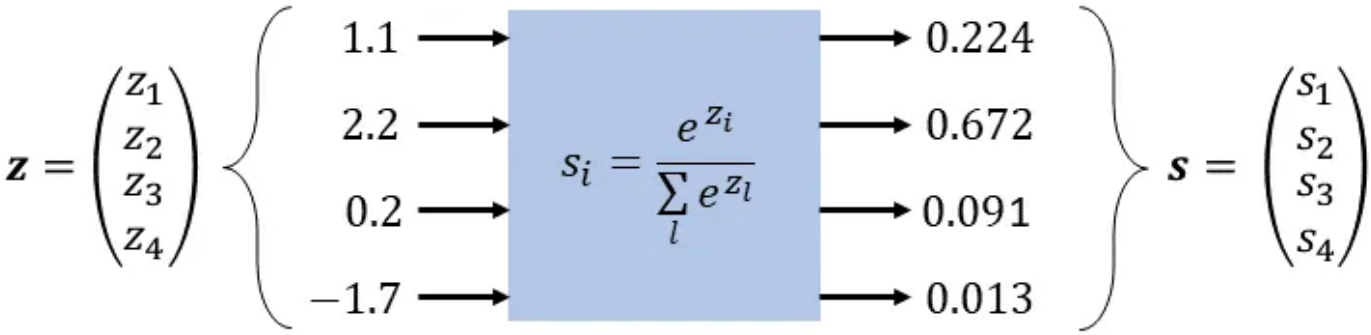
Derivative of the Softmax Function and the Categorical Cross-Entropy Loss

A simple and quick derivation

In this short post, we are going to compute the Jacobian matrix of the softmax function. By applying an elegant computational trick, we will make the derivation super short. Using the obtained Jacobian matrix, we will then compute the gradient of the categorical cross-entropy loss.

Softmax Function

The main purpose of the softmax function is to grab a vector of arbitrary real numbers and turn it into probabilities:



(Image by author)

The exponential function in the formula above ensures that the obtained values are non-negative. Due to the normalization term in the denominator the obtained values sum to 1. Furthermore, all values lie between 0 and 1. An important property of the softmax function is that it preser ts input values:

$$-1.7 < 0.2 < 1.1 < 2.2 \quad \rightarrow \quad 0.013 < 0.091 < 0.224 < 0.672$$

Jacobian of the Softmax Function

Formally, the softmax function is a so called *vector function*, which takes a vector as input and produces a vector as output:

$$\text{softmax: } \mathbb{R}^n \rightarrow \mathbb{R}^n$$

Therefore, when talking about the derivative of the softmax function, we actually talk about its Jacobian matrix, which is the matrix of all first-order partial derivatives:

$$J_{softmax} = \begin{pmatrix} \frac{\partial s_1}{\partial z_1} & \frac{\partial s_1}{\partial z_2} & \cdots & \frac{\partial s_1}{\partial z_n} \\ \frac{\partial s_2}{\partial z_1} & \frac{\partial s_2}{\partial z_2} & \cdots & \frac{\partial s_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial s_n}{\partial z_1} & \frac{\partial s_n}{\partial z_2} & \cdots & \frac{\partial s_n}{\partial z_n} \end{pmatrix}$$

where

$$s_i = \frac{e^{z_i}}{\sum_{l=1}^{n} e^{z_l}}, \qquad \forall i = 1, \ldots, n$$

Notice, how each output of the softmax function depends on all the input values (due to the denominator). For this reason the off-diagonal elements of the Jacobian aren't zero.

Since the outputs of the softmax function are strictly positive values, we can make the following derivation super short, by applying the following trick: instead of taking the partial derivative of the output, we take the partial derivative of the log of the output (also called "logarithmic derivative"):

$$\frac{\partial}{\partial z_j} log(s_i) = \frac{1}{s_i} \cdot \frac{\partial s_i}{\partial z_j}$$

where the expression on the right-hand side follows directly from the chain rule. Next, we rearrange the upper formula and obtain:

$$\frac{\partial s_i}{\partial z_j} = s_i \cdot \frac{\partial}{\partial z_j} log(s_i)$$

The left-hand side is exactly the partial derivative we're looking for. As we will shortly see, the right-hand side simplifies the computation of the derivative, such that we don't require the quotient rule of derivatives. We must first take the logarithm of *s*:

$$log\, s_i = log\left(\frac{e^{z_i}}{\sum_{l=1}^{n} e^{z_l}}\right) = z_i - log\left(\sum_{l=1}^{n} e^{z_l}\right)$$

The partial derivative of the resulting expression is:

$$\frac{\partial}{\partial z_j} log\, s_i = \frac{\partial z_i}{\partial z_j} - \frac{\partial}{\partial z_j} log\left(\sum_{l=1}^{n} e^{z_l}\right)$$

Let's have a look at the first term on the right-hand side:

$$\frac{\partial z_i}{\partial z_j} = \begin{cases} 1, & if\ i = j \\ 0, & otherwise \end{cases}$$

which can be concisely written using the indicator function 1{·}. The indicator function takes on a value of 1 if its argument is true, and 0 otherwise.

The second term on the right-hand side can be evaluated by applying the chain rule:

$$\frac{\partial}{\partial z_j} log\, s_i = 1\{i = j\} - \frac{1}{\sum_{l=1}^{n} e^{z_l}} \cdot \left(\frac{\partial}{\partial z_j} \sum_{l=1}^{n} e^{z_l} \right)$$

In the step above we used the derivative of the natural logarithm:

$$\frac{d}{dx} log(x) = \frac{1}{x}$$

Obtaining the partial derivative of the sum is trivial:

$$\frac{\partial}{\partial z_j} \sum_{l=1}^{n} e^{z_l} = \frac{\partial}{\partial z_j} \left[e^{z_1} + e^{z_2} + \cdots + e^{z_j} + \cdots + e^{z_n} \right] = \frac{\partial}{\partial z_j} \left[e^{z_j} \right] = e^{z_j}$$

Plugging the result into the formula yields:

$$\frac{\partial}{\partial z_j} log\, s_i = 1\{i = j\} - \frac{e^{z_j}}{\sum_{l=1}^{n} e^{z_l}} = 1\{i = j\} - s_j$$

Finally, we have to multiply the upper expression with s, as shown at the beginning of this section:

$$\frac{\partial s_i}{\partial z_j} = s_i \cdot \frac{\partial}{\partial z_j} log(s_i) = s_i \cdot \left(1\{i = j\} - s_j \right)$$

This concludes our derivation. We've obtained a formula for all the elements of the Jacobi matrix (both diagonal and off-diagonal). For the special case of $n = 4$ we get:

$$J_{softmax} = \begin{pmatrix} s_1 \cdot (1 - s_1) & -s_1 \cdot s_2 & -s_1 \cdot s_3 & -s_1 \cdot s_4 \\ -s_2 \cdot s_1 & s_2 \cdot (1 - s_2) & -s_2 \cdot s_3 & -s_2 \cdot s_4 \\ -s_3 \cdot s_1 & -s_3 \cdot s_2 & s_3 \cdot (1 - s_3) & -s_3 \cdot s_4 \\ -s_4 \cdot s_1 & -s_4 \cdot s_2 & -s_4 \cdot s_3 & s_4 \cdot (1 - s_4) \end{pmatrix}$$

See how the diagonal elements differ from the off-diagonal elements.

Categorical Cross-Entropy Loss

Categorical cross-entropy loss is closely related to the softmax function, since it's practically only used with networks with a softmax layer at the output. Before we formally introduce the categorical cross-entropy loss (often also called softmax loss), we shortly have to clarify two terms: **multi-class classification** and **cross-entropy**.

Classification problems can be subdivided into the following two categories:

- **multi-class classification**, where each sample belongs to only one class (mutually exclusive)
- **multi-label classification**, where each sample may belong to multiple classes (or to no class)

The categorical cross-entropy loss is exclusively used in multi-class classification tasks, where each sample belongs exactly to one of the C classes. The true label assigned to each sample consists hence of a single integer value between 0 and C -1. The label can be represented by an one-hot encoded vector of size C, which has the value one for the correct class and zero everywhere else, see example below for $C = 4$:

$$\text{label} = 2 \;\rightarrow\; \boldsymbol{y} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

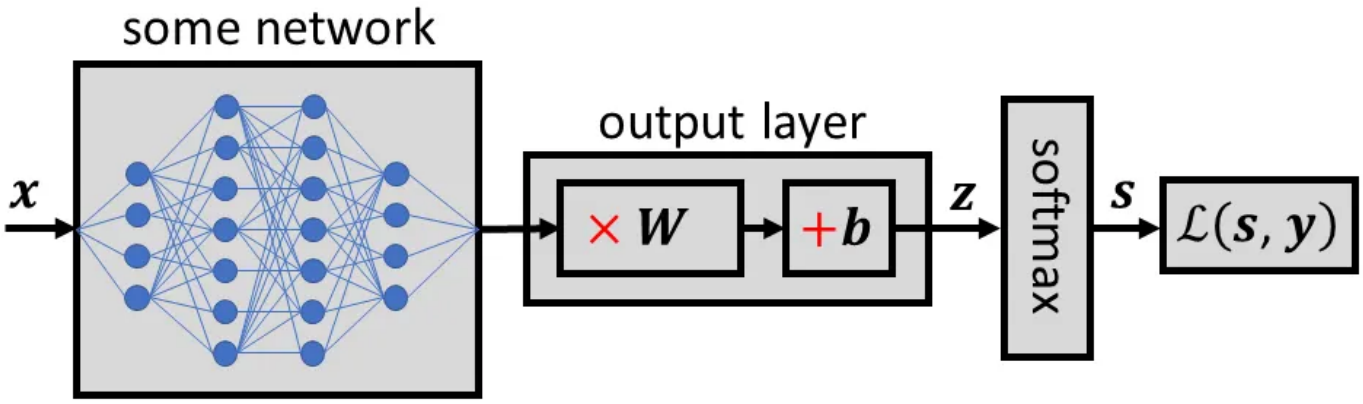
Cross-entropy takes as input two discrete probability distributions (simply vectors whose elements lie between 0,..,1 and sum to 1) and outputs a single real-valued (!) number representing the similarity of both probability distributions:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{s}) = -\sum_{i=1}^{C} y_i \cdot log(s_i)$$

where C denotes the number of different classes and the subscript i denotes i-th element of the vector. The smaller the cross-entropy, the more similar the two probability distributions are.

When cross-entropy is used as loss function in a multi-class classification task, then $\boldsymbol{y}$ is fed with the one-hot encoded label and the probabilities generated by the softmax layer are put in s. This way round we won't take the logarithm of zeros, since mathematically softmax will never really produce zero values.

By minimizing the loss during training, we basically force the predicted probabilities to gradually resemble the true one-hot encoded vectors.



(Image by author)

In order to kick off the backpropagation process, as described in this post, we have to calculate the derivative of the loss w.r.t to *weighted input z* of the output layer, see figure above:

$$\frac{\partial \mathcal{L}}{\partial z_j} = -\frac{\partial}{\partial z_j}\sum_{i=1}^{C} y_i \cdot log(s_i) = -\sum_{i=1}^{C} y_i \cdot \frac{\partial}{\partial z_j} log(s_i) = -\sum_{i=1}^{C} \frac{y_i}{s_i} \cdot \frac{\partial s_i}{\partial z_j}$$

Let us plug in the derivatives we have obtained in the last section:

$$\frac{\partial \mathcal{L}}{\partial z_j} = -\sum_{i=1}^{C} \frac{y_i}{s_i} \cdot s_i \cdot \left(1\{i=j\} - s_j\right) = -\sum_{i=1}^{C} y_i \cdot \left(1\{i=j\} - s_j\right)$$

and expand the product in the last term:

$$\frac{\partial \mathcal{L}}{\partial z_j} = \sum_{i=1}^{C} y_i \cdot s_j - \sum_{i=1}^{C} y_i \cdot 1\{i=j\}$$

The indicator function $1\{\cdot\}$ takes on a value of 1 for $i = j$ and 0 everywhere else:

$$\frac{\partial \mathcal{L}}{\partial z_j} = \sum_{i=1}^{C} y_i \cdot s_j - y_j$$

Next, we pull s out of the sum, since it does not depend on index i:

$$\frac{\partial \mathcal{L}}{\partial z_j} = s_j \cdot \sum_{i=1}^{C} y_i - y_j = s_j - y_j$$

In the last step we used the fact, that the one-hot encoded vector $\boldsymbol{y}$ sums to 1. Remember that a one-hot encoded vector can be interpreted as a probability distribution with the probability mass centered around a single value. In the concise vector notation we get:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}} = \boldsymbol{s} - \boldsymbol{y}$$

More readings

Eli Bendersky's website

Raúl Gómez blog

Data Science Machine Learning Artificial Intelligence Optimization