Softmax Activation Function — How It Actually Works

Kiprono Elijah Koech Sep 30, 2020 · 5 min read ★

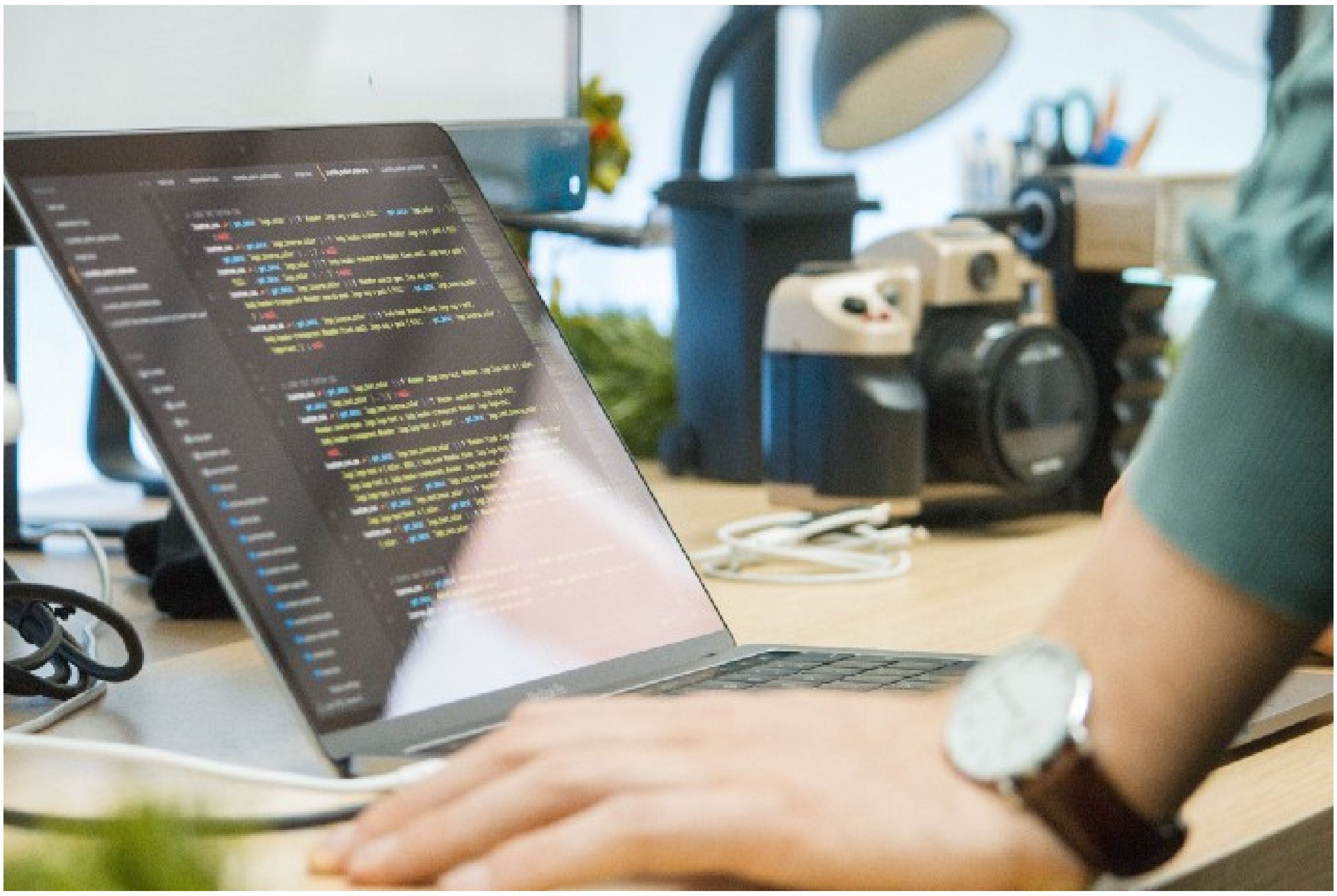


Photo by Fatos Bytyqi on Unsplash

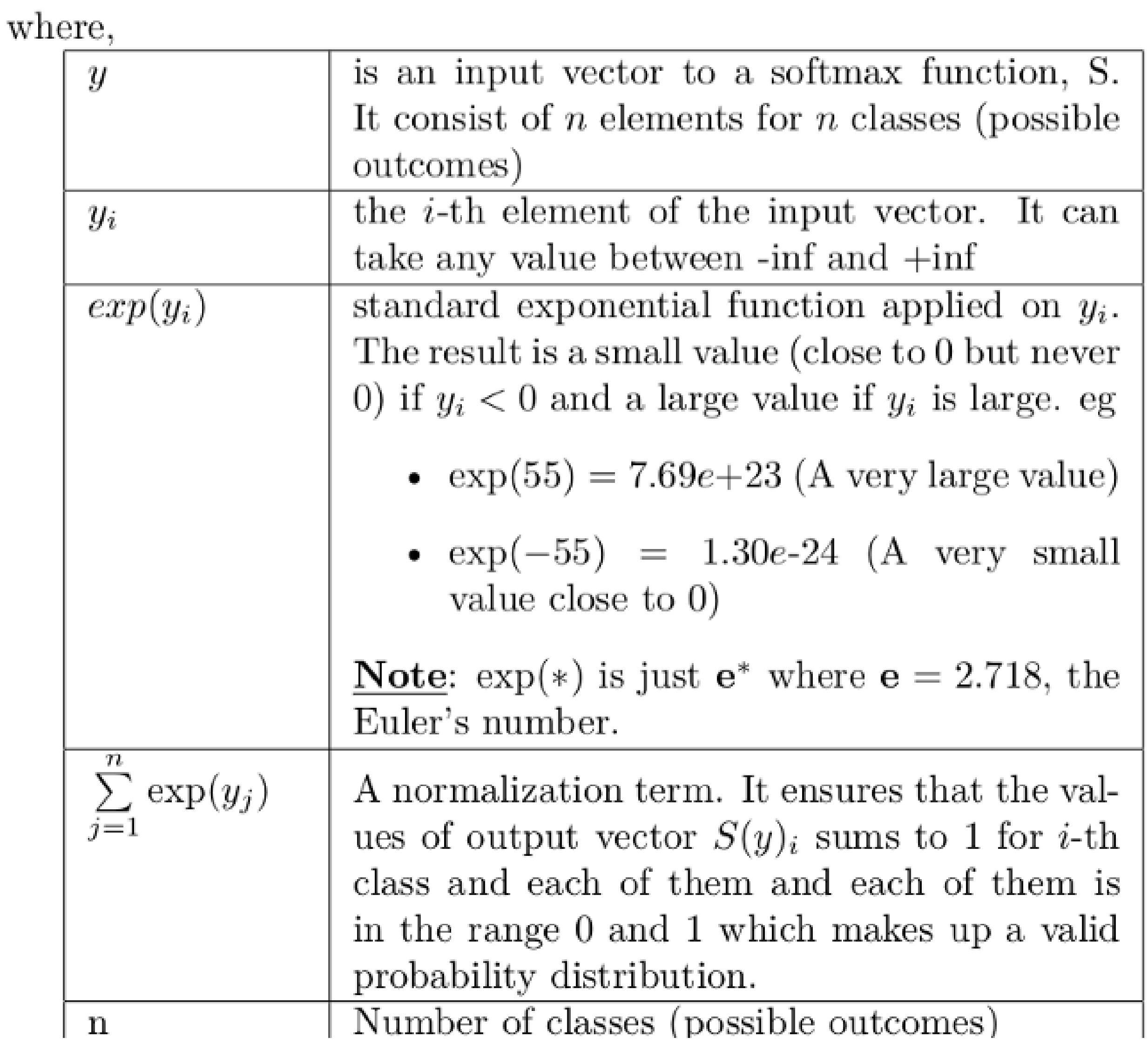
the deep learning model.

> *It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes. — Wikipedia [link]*

Softmax is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say `v`) with probabilities of each possible outcome. The probabilities in vector `v` sums to one for all possible outcomes or classes.

Mathematically, Softmax is defined as,

$$S(y)_i = \frac{\exp(y_i)}{\sum\limits_{j=1}^{n} \exp(y_j)}$$

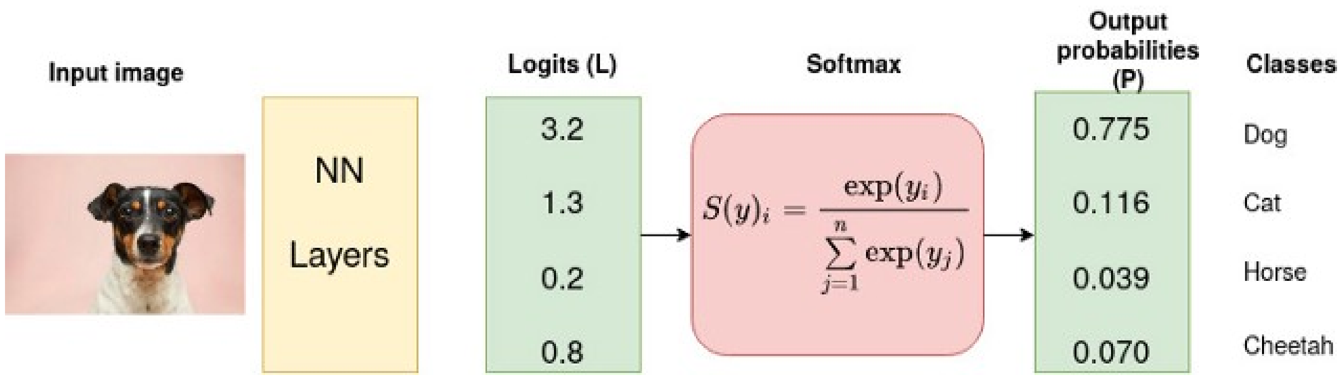


where,

y	is an input vector to a softmax function, S. It consist of n elements for n classes (possible outcomes)
y_i	the i-th element of the input vector. It can take any value between -inf and +inf
$exp(y_i)$	standard exponential function applied on y_i. The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg • $\exp(55) = 7.69e{+}23$ (A very large value) • $\exp(-55) = 1.30e\text{-}24$ (A very small value close to 0) **Note**: $\exp(*)$ is just $\mathbf{e}^*$ where $\mathbf{e} = 2.718$, the Euler's number.
$\sum\limits_{j=1}^{n} \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sums to 1 for i-th class and each of them and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

Example

Consider a CNN model which aims at classifying an image as either a dog, cat, horse or cheetah (4 possible outcomes/classes). The last (fully-connected) layer of the CNN outputs a vector of logits, L, that is passed through a Softmax layer that transforms the logits into probabilities, P. These probabilities are the model predictions for each of the 4 classes.



Input image source: Photo by Victor Grabarczyk on Unsplash . Diagram by author.

Let us calculate the probability generated by the first logit after Softmax is applied

$$\begin{aligned}\exp(3.2) &= 24.5325\\ \exp(1.3) &= 3.6693\\ \exp(0.2) &= 1.2214\\ \exp(0.8) &= 2.2255\end{aligned}$$

and therefore,

$$S(3.2) = \frac{\exp(3.2)}{\exp(3.2)+\exp(1.3)+\exp(0.2)+\exp(0.8)}$$

$$= 0.775$$

You can calculate the other values in the same manner.

In python, we can implement Softmax as follows

```
from math import exp

def softmax(input_vector):
    # Calculate the exponent of each element in the input vector
    exponents = [exp(j) for j in input_vector]

    # divide the exponent of each value by the sum of the
    # exponents and round of to 3 decimal places
    p = [round(exp(i)/sum(exponents),3) for i in input_vector]

    return p

print(softmax([3.2,1.3,0.2,0.8]))
```

Output:

```
[0.775, 0.116, 0.039, 0.07]
```

Notation: We can represent all the logits as a vector, **v**, and apply the activation function, **S**, on this vector to output the probabilities vector, **p**, and represent the operation as follows

$$S(\mathbf{v}) = \mathbf{p}$$

$$\text{for example, in our case, we will have}$$

$$\left(\begin{bmatrix} 3.2 \\ 1.3 \end{bmatrix} \right) \quad \begin{bmatrix} 0.775 \\ 0.116 \end{bmatrix}$$

Note that the labels: dog, cat, horse and cheetah are in string format. We need to define a way in which we represent these values as numerical values.

Categorical Data into Numerical Data

The truth labels are categorical data: any particular image can be categorized into one of these groups: dog, cat, horse or cheetah. The computer however does not understand this kind of data and therefore we need to convert them into numerical data. There are two ways to do so:

1. Integer encoding
2. One-hot encoding

Integer Encoding (Also called Label Encoding)

In this kind of encoding, labels are assigned unique integer values. For example in our case, we will have,

0 for “dog”, **1** for “cat”, **2** for “horse” and **3** for “cheetah”.

When to use integer encoding: It is used when the labels are ordinal in nature, that is, labels with some order, for example, consider a classification problem where we want to classify a service as either poor, neutral or good, then we can encode these classes as follows

0 for “poor”, **1** for “neutral” and **2** for “good”.

Clearly, the labels have some order and the labels gives the weights to the labels accordingly.

problem where we have 3 classes: Iris setosa, Iris versicolor and Iris virginica,

0 for "Iris setosa", **1** for "Iris versicolor" and **2** for "Iris virginica".

The model may take a natural ordering of the labels (2>1>0) and give more weight to one class over another when in fact these are just labels with no specific ordering implied.

One-hot encoding

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough. One-hot encoding is preferred.

In one-hot encoding, the labels are represented by a binary variable (1 and 0s) such that for a given class a binary variable with 1 for position corresponding to that specific class and 0 elsewhere is generated, for example, in our case we will have the following labels for our 4 classes

[1,0,0,0] for "dog", **[0,1,0,0]** for "cat", **[0,0,1,0]** for "horse" and **[0,0,0,1]** for "cheetah".

Remark: Despite the fact that we have answered the question of "when to use which type of encoding system?". There is a way of actually using any kind of encoding method. For TensorFlow and Keras, this depends on how you define your loss function. We will get to this later.

Recall: The denominator of Softmax function is a normalization term. It ensures that the output of the function is a value between 0 and 1.

$$S(y)_i = \frac{\exp(y_i)}{\sum\limits_{j=1}^{n} \exp(y_j)}$$

divide it by the sum of the all logits to get the probabilities? Why take the exponents? Here are some two reasons.

- Softmax normalization reacts to small and large variation/change differently but standard normalization does not differentiate the stimulus by intensity so longest the proportion is the same, for example,

# Softmax normalization

softmax([2,4]) = [0.119, 0.881]

softmax([4,8]) = [0.018, 0.982]

# Standard normalization

```
def std_norm(input_vector):
    p = [round(i/sum(input_vector),3) for i in input_vector]
    return p
```

std_norm([2,4]) = [0.333, 0.667]

std_norm([4,8]) = [0.333, 0.667]

Notice the difference? for standard normalization, a vector and the same vector scaled by a scalar yields the same output. For above case, the first vector [2,4] was multiplied by 2 to yield [4,8] and both of them yield the same output. With the same reasoning, the following pairs will yield the same output: {[8,24], [2.4, 7.199]} for scale factor of 0.3. In fact, any vector scaled by a factor yields the same output as the original vector.

- Another problem arises when there are negative values in the logits. In that case, you will end up with negative probabilities in the output. The Softmax is not affected with negative values because exponent of any value (positive or negative) is always a positive value.

I hope after reading this you now have a clearer understanding of how Softmax activation function actually works.

Cross-Entropy Loss Function

A loss function used in most classification problems to optimize machine learning model...

towardsdatascience.com

On Object Detection Metrics With Worked Example

AP, mAP, AP50 among other metrics explained with an example.

towardsdatascience.com

End to End Machine Learning Project: Reviews Classification

A project to classify a review as either positive or negative

towardsdatascience.com

Thank you for reading 😊

Deep Learning Machine Learning Data Science Neural Networks Softmax