My Notes

Softmax classification with cross-entropy (2/2)

This tutorial will describe the softmax function used to model multiclass classification problems. We will provide derivations of the gradients used for optimizing any parameters with regards to the cross-entropy.

The previous section described how to represent classification of 2 classes with the help of the logistic function. For multiclass classification there exists an extension of this logistic function, called the softmax function, which is used in multinomial logistic regression. What follows will explain the softmax function and how to derive it.

This is the second part of a 2-part tutorial on classification models trained by cross-entropy:

- Part 1: Logistic classification with cross-entropy
- Part 2: Softmax classification with cross-entropy (this)

```
# Python imports
```

Softmax function

The logistic output function described in the previous section can only be used for the classification between two target classes $t = 1$ and $t = 0$. This logistic function can be generalized to output a multiclass categorical probability distribution by the softmax function. This softmax function ς takes as input a C-dimensional vector $\mathbf{z}$ and outputs a C-dimensional vector $\mathbf{y}$ of real values between 0 and 1. This function is a normalized exponential and is defined as:

$$y_c = \varsigma(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{d=1}^{C} e^{z_d}} \quad \text{for } c = 1 \cdots C$$

The denominator $\sum_{d=1}^{C} e^{z_d}$ acts as a regularizer to make sure that $\sum_{c=1}^{C} y_c = 1$. As the output layer of a neural network, the softmax function can be represented graphically as a layer with C neurons.

We can write the probabilities that the class is $t = c$ for $c = 1 \ldots C$ given input $\mathbf{z}$ as:

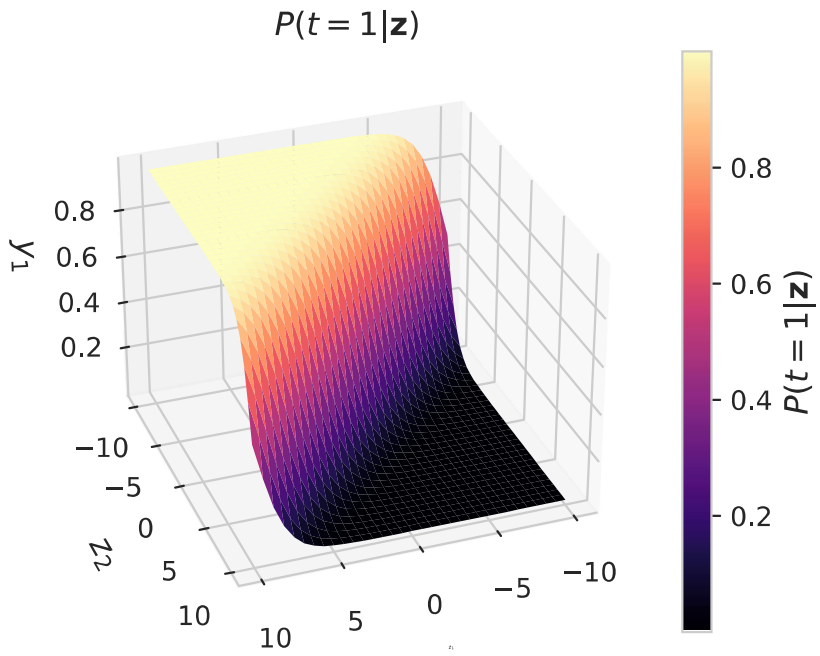
$$\begin{bmatrix} P(t=1|\mathbf{z}) \\ \vdots \\ P(t=C|\mathbf{z}) \end{bmatrix} = \begin{bmatrix} \varsigma(\mathbf{z})_1 \\ \vdots \\ \varsigma(\mathbf{z})_C \end{bmatrix} = \frac{1}{\sum_{d=1}^{C} e^{z_d}} \begin{bmatrix} e^{z_1} \\ \vdots \\ e^{z_C} \end{bmatrix}$$

Where $P(t = c|\mathbf{z})$ is thus the probability that that the class is c given the input $\mathbf{z}$.

These probabilities of the output $P(t = 1|\mathbf{z})$ for an example system with 2 classes ($t = 1$, $t = 2$) and input $\mathbf{z} = [z_1, z_2]$ are shown in the figure below. The other probability $P(t = 2|\mathbf{z})$ will be complementary.

```
def softmax(z):
    """Softmax function"""
    return np.exp(z) / np.sum(np.exp(z))
```

```
# Plot the softmax output for 2 dimensions for both classes
```



Derivative of the softmax function

To use the softmax function in neural networks, we need to compute its derivative. If we define $\Sigma_C = \sum_{d=1}^{C} e^{z_d}$ for $c = 1 \cdots C$ so that $y_c = e^{z_c}/\Sigma_C$, then this derivative $\partial y_i / \partial z_j$ of the output $\mathbf{y}$ of the softmax function with respect to its input $\mathbf{z}$ can be calculated as:

$$\text{if } i = j : \frac{\partial y_i}{\partial z_i} = \frac{\partial \frac{e^{z_i}}{\Sigma_C}}{\partial z_i} = \frac{e^{z_i}\Sigma_C - e^{z_i}e^{z_i}}{\Sigma_C^2} = \frac{e^{z_i}}{\Sigma_C}\frac{\Sigma_C - e^{z_i}}{\Sigma_C} = \frac{e^{z_i}}{\Sigma_C}(1 - \frac{e^{z_i}}{\Sigma_C}) = y_i(1 - y_i)$$

$$\text{if } i \neq j : \frac{\partial y_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\Sigma_C}}{\partial z_j} = \frac{0 - e^{z_i}e^{z_j}}{\Sigma_C^2} = -\frac{e^{z_i}}{\Sigma_C}\frac{e^{z_j}}{\Sigma_C} = -y_i y_j$$

Note that if $i = j$ this derivative is similar to the derivative of the logistic function.

Cross-entropy loss function for the softmax function

To derive the loss function for the softmax function we start out from the likelihood function that a given set of parameters θ of the model can result in prediction of the correct class of each input sample, as in the derivation for the logistic loss function. The maximization of this likelihood can be written as:

$$\underset{\theta}{\operatorname{argmax}}\ \mathcal{L}(\theta|\mathbf{t},\mathbf{z})$$

The likelihood $\mathcal{L}(\theta|\mathbf{t},\mathbf{z})$ can be rewritten as the joint probability of generating $\mathbf{t}$ and $\mathbf{z}$ given the parameters θ: $P(\mathbf{t},\mathbf{z}|\theta)$. Which can be decomposed as a conditional distribution and a marginal:

$$P(\mathbf{t},\mathbf{z}|\theta) = P(\mathbf{t}|\mathbf{z},\theta)P(\mathbf{z}|\theta)$$

Since we are not interested in the probability of $\mathbf{z}$ we can reduce this to: $\mathcal{L}(\theta|\mathbf{t},\mathbf{z}) = P(\mathbf{t}|\mathbf{z},\theta)$. Which can be written as $P(\mathbf{t}|\mathbf{z})$ for fixed θ. Since each t_c is dependent on the full $\mathbf{z}$, and only 1 class can be activated in the $\mathbf{t}$ we can write

$$P(\mathbf{t}|\mathbf{z}) = \prod_{i=c}^{C} P(t_c|\mathbf{z})^{t_c} = \prod_{c=1}^{C} \varsigma(\mathbf{z})_c^{t_c} = \prod_{c=1}^{C} y_c^{t_c}$$

As was noted during the derivation of the loss function of the logistic function, maximizing this likelihood can also be done by minimizing the negative log-likelihood:

$$-\log \mathcal{L}(\theta|\mathbf{t},\mathbf{z}) = \xi(\mathbf{t},\mathbf{z}) = -\log \prod_{c=1}^{C} y_c^{t_c} = -\sum_{c=1}^{C} t_c \cdot \log(y_c)$$

Which is the cross-entropy error function ξ. Note that for a 2 class system output $t_2 = 1 - t_1$ and this results in the same error function as for logistic regression:
$\xi(\mathbf{t},\mathbf{y}) = -t_c \log(y_c) - (1 - t_c)\log(1 - y_c)$.

The cross-entropy error function over a batch of multiple samples of size n can be calculated as:

$$\xi(T,Y) = \sum_{i=1}^{n} \xi(\mathbf{t}_i,\mathbf{y}_i) = -\sum_{i=1}^{n}\sum_{c=1}^{C} t_{ic} \cdot \log(y_{ic})$$

Where t_{ic} is 1 if and only if sample i belongs to class c, and y_{ic} is the output probability that sample i belongs to class c.

Derivative of the cross-entropy loss function for the softmax function

The derivative $\partial \xi / \partial z_i$ of the loss function with respect to the softmax input z_i can be calculated as:

$$\begin{aligned}
\frac{\partial \xi}{\partial z_i} &= -\sum_{j=1}^{C} \frac{\partial t_j \log(y_j)}{\partial z_i} = -\sum_{j=1}^{C} t_j \frac{\partial \log(y_j)}{\partial z_i} = -\sum_{j=1}^{C} t_j \frac{1}{y_j} \frac{\partial y_j}{\partial z_i} \\
&= -\frac{t_i}{y_i} \frac{\partial y_i}{\partial z_i} - \sum_{j \neq i}^{C} \frac{t_j}{y_j} \frac{\partial y_j}{\partial z_i} = -\frac{t_i}{y_i} y_i (1 - y_i) - \sum_{j \neq i}^{C} \frac{t_j}{y_j} (-y_j y_i) \\
&= -t_i + t_i y_i + \sum_{j \neq i}^{C} t_j y_i = -t_i + \sum_{j=1}^{C} t_j y_i = -t_i + y_i \sum_{j=1}^{C} t_j \\
&= y_i - t_i
\end{aligned}$$

Note that we already derived $\partial y_j / \partial z_i$ for $i = j$ and $i \neq j$ above.

The result that $\partial \xi / \partial z_i = y_i - t_i$ for all $i \in C$ is the same as the derivative of the cross-entropy for the logistic function which had only one output node.

This is the second part of a 2-part tutorial on classification models trained by cross-entropy:

- Part 1: Logistic classification with cross-entropy
- Part 2: Softmax classification with cross-entropy (this)

To see the softmax function in action on a minimal neural network, please read part 4 of this series on how to implement a neural network in NumPy.

```
# Versions used
```

```
Python implementation: CPython
Python version       : 3.9.4
IPython version      : 7.23.1

seaborn    : 0.11.1
numpy      : 1.20.2
matplotlib: 3.4.2
```

This post at peterroelants.github.io is generated from an IPython notebook file. Link to the full IPython notebook file

Softmax | Logistic Regression | Machine Learning | Cross-Entropy | Classification | Gradient Descent | Neural Networks | Notebook