AI Project

Robin MENEUST

August 2023

Table of contents

Introduction

1 Softmax derivative

Softmax :

$$s_{z_i}(z_1, z_2, ...z_n) = \frac{e^{z_i}}{\sum\limits_{j=1}^{n} e^{z_j}} \tag{1}$$

Derivative :

$$
\begin{aligned}
\frac{\partial s_{z_i}}{\partial z_i} &= \frac{\partial\left(\frac{e^{z_i}}{\sum\limits_{j=1}^{n} e^{z_j}}\right)}{\partial z_i} \\
&= \frac{\partial\left(\frac{e^{z_i}}{k+e^{z_i}}\right)}{\partial z_i}, \quad \text{where } k = \sum_{j=1,\ j\neq i}^{n} e^{z_j} \\
&= \frac{\partial\left(1-\frac{k}{k+e^{z_i}}\right)}{\partial z_i} \\
&= \frac{ke^{z_i}}{(k+e^{z_i})^2} \\
&= \left(\sum_{j=1,\ j\neq i}^{n} e^{z_j}\right) \frac{e^{z_i}}{\left(\left(\sum\limits_{j=1,\ j\neq i}^{n} e^{z_j}\right)+e^{z_i}\right)^2} \\
&= \left(\sum_{j=1,\ j\neq i}^{n} e^{z_j}\right) \frac{e^{z_i}}{\left(\sum\limits_{j=1}^{n} e^{z_j}\right)^2} \\
&= \left(\left(\sum_{j=1}^{n} e^{z_j}\right)-e^{z_i}\right) \frac{e^{z_i}}{\left(\sum\limits_{j=1}^{n} e^{z_j}\right)^2} \\
&= e^{z_i}\left(\frac{1}{\sum\limits_{j=1}^{n} e^{z_j}} - \frac{s_{z_i}^2}{e^{z_i}}\right) \\
&= s_{z_i} - s_{z_i}^2 \\
&= s_{z_i}(1-s_{z_i})
\end{aligned} \tag{2}
$$

And:

$$
\begin{aligned}
\frac{\partial s_{z_k}}{\partial z_{k\neq i}} &= \frac{\partial \left(\frac{e^{z_i}}{\sum\limits_{j=1}^{n} e^{z_j}} \right)}{\partial z_{k\neq i}} \\
&= e^{z_i} \frac{\partial \left(\frac{1}{c+e^{z_k}} \right)}{\partial z_{k\neq i}}, \quad \text{where } c = \sum_{j=1,\ j\neq k}^{n} e^{z_j} \\
&= -e^{z_i} \frac{e^{z_k}}{(c+e^{z_k})^2} \\
&= -\frac{e^{z_i} e^{z_k}}{\left(\left(\sum\limits_{j=1,\ j\neq k}^{n} e^{z_j} \right) + e^{z_k} \right)^2} \\
&= -\frac{e^{z_i} e^{z_k}}{\left(\sum\limits_{j=1}^{n} e^{z_j} \right)^2} \\
&- s_{z_i} s_{z_k}
\end{aligned} \tag{3}
$$

So we have :

$$
\begin{aligned}
\frac{\partial s_{z_k}}{\partial z_i} &= \begin{cases} s_{z_i}(1 - s_{z_i}) & \text{if } i = k \\ -s_{z_i} s_{z_k} & \text{else} \end{cases} \\
&= s_{z_i} \left(\delta_{ik} - s_{z_k} \right)
\end{aligned} \tag{4}
$$

And

$$
\begin{aligned}
\frac{\partial E_i}{\partial z_k^{(n-1)}} &= \frac{\partial E_i}{\partial s_{z_i}} \frac{\partial s_{z_i}}{\partial z_k^{(n-1)}} \\
&= (s_{z_i} - \hat{y}_i)\, s_{z_i} \left(\delta_{ik} - s_{z_k} \right)
\end{aligned} \tag{5}
$$

2 Definitions and standard functions derivatives

Let :

1. C be the total cost function
2. y_i be the output (prediction) i

3. $\hat{y}_i$ be the expected output i

4. C_i be the cost for output i (e.g. $\frac{1}{2}\left(\hat{y}_i - y_i\right)^2$

5. $w_{i,j}^{(l)}$ be the weight of the neuron j of the layer $l-1$ for the neuron i of the layer l

6. $z_i^{(l)}$ be the weighted sum for the neuron i of the layer l (activation function input)

7. $a_i^{(l)} = g^{(l)}(z_i^{(l)})$ be the output of the neuron i of the layer l (activation function output)

8. $b_i^{(l)}$ be the bias of the neuron i of the layer l

9. L be the number of layers and the index of the output layer (layers index goes from 1 to L)

10. n_l be the number of neurons in the layer l

11. The derivative of Sigmoid σ is $\sigma(1-\sigma)$

12. The derivative of Softmax $s_{z_i}(z_i)$ is $s_{z_i}(z_i)\left(1 - s_{z_i}(z_i)\right)$

Back-propagation

1 Output layer L

Here we consider that the activation function of the layer L is Softmax s.

$$\frac{\partial C}{\partial w_{i,j}^{(L)}} = \frac{\partial C}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial w_{i,j}^{(L)}}$$

Where

$$\frac{\partial C}{\partial a_i^{(L)}} = \frac{\partial \frac{1}{n_l} \sum\limits_{k=1}^{n_L} C_k}{\partial a_i^{(L)}} = \frac{1}{n_L} \sum_{k=1}^{n_l} \frac{\partial C_k}{\partial a_i^{(L)}} = \frac{1}{n_L} \frac{\partial C_i}{\partial a_i^{(L)}}$$

$$\frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} = \frac{\partial s_{z_i^{(L)}}}{\partial z_i^{(L)}} = g'^{(L)}(z_i^{(L)})$$

$$\frac{\partial z_i^{(L)}}{\partial w_{i,j}^{(L)}} = \frac{\partial\left(\sum_{k=1}^{n_{L-1}}\left(w_{i,k}^{(L)}a_i^{(L-1)}\right)+b_i^{(L)}\right)}{\partial w_{i,j}^{(L)}} = \sum_{k=1}^{n_{L-1}}\left(\frac{\partial w_{i,k}^{(L)}a_i^{(L-1)}}{\partial w_{i,j}^{(L)}}\right)+\frac{\partial b_i^{(L)}}{\partial w_{i,j}^{(L)}} = a_j^{(L-1)}$$

For the bias it's almost the same equation:

$$\frac{\partial C}{\partial b_i^{(L)}} = \frac{\partial C}{\partial a_i^{(L)}}\frac{\partial a_i^{(L)}}{\partial z_i^{(L)}}\frac{\partial z_i^{(L)}}{\partial b_i^{(L)}}$$

Where

$$\frac{\partial z_i^{(L)}}{\partial b_i^{(L)}} = \frac{\partial\left(\left(\sum_{k=1}^{n_{L-1}}w_{i,k}^{(L)}a_i^{(L-1)}\right)+b_i^{(L)}\right)}{\partial b_i^{(L)}} = \sum_{k=1}^{n_{L-1}}\left(\frac{\partial w_{i,k}^{(L)}a_i^{(L-1)}}{\partial b_i^{(L)}}\right)+\frac{\partial b_i^{(L)}}{\partial b_i^{(L)}} = 1$$

2 Layer L-1

Here we consider that the activation function of the layer $L-1$ and the other ones except L is sigmoid σ.

$$\frac{\partial C}{\partial w_{i,j}^{(L-1)}} = \frac{\partial C}{\partial a_i^{(L-1)}}\frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}}\frac{\partial z_i^{(L-1)}}{\partial w_{i,j}^{(L-1)}}$$

Where

$$\frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} = \frac{\partial \sigma}{\partial z_i^{(L-1)}} = g'^{(L-1)}(z_i^{(L-1)})$$

$$\frac{\partial z_i^{(L-1)}}{\partial w_{i,j}^{(L-1)}} = \frac{\partial\left(\sum_{k=1}^{n_{L-2}}\left(w_{i,k}^{(L-1)}a_i^{(L-2)}\right)+b_i^{(L-1)}\right)}{\partial w_{i,j}^{(L-1)}} = \sum_{k=1}^{n_{L-2}}\left(\frac{\partial w_{i,k}^{(L-1)}a_i^{(L-2)}}{\partial w_{i,j}^{(L-1)}}\right)+\frac{\partial b_i^{(L-1)}}{\partial w_{i,j}^{(L-1)}} = a_j^{(L-2)}$$

$$\frac{\partial C}{\partial a_i^{(L-1)}} = \sum_{k=1}^{n_L}\frac{\partial C}{\partial a_k^{(L)}}\frac{\partial a_k^{(L)}}{\partial z_k^{(L)}}\frac{\partial z_k^{(L)}}{\partial a_i^{(L-1)}}$$

We already calculated the 2 first derivatives in the previous subsection, and for the last one:

$$\frac{\partial z_k^{(L)}}{\partial a_i^{(L-1)}} = \frac{\partial \left(\sum_{p=1}^{n_{L-1}} \left(w_{k,p}^{(L)} a_k^{(L-1)} \right) + b_k^{(L)} \right)}{\partial a_i^{(L-1)}} = \sum_{p=1}^{n_{L-1}} \left(\frac{\partial w_{k,p}^{(L)} a_k^{(L-1)}}{\partial a_i^{(L-1)}} \right) + \frac{\partial b_k^{(L)}}{\partial a_i^{(L-1)}} = w_{k,i}^{(L)}$$

3 Layer $l < L$

$$\frac{\partial C}{\partial w_{i,j}^{(l)}} = \frac{\partial C}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial C}{\partial a_i^{(l)}} g'^{(l)}(z_i^{(l)}) a_j^{(l-1)}$$

Where if $l < L$:

$$\frac{\partial C}{\partial a_i^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial a_i^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial a_k^{(l+1)}} g'^{(l+1)}(z_k^{(l+1)}) w_{k,i}^{(l+1)}$$

Otherwise if $l = L$:

$$\frac{\partial C}{\partial a_i^{(L)}} = \frac{1}{n_L} \frac{\partial C_i}{\partial a_i^{(L)}}$$

4 Algorithm

Step 1: feed-forward and store values

```
# input: input vector, fed to this network
# getWeightedSum(layerIndex, prevLayerValues) (z_i)
# activationFunction(layerIndex, input) (a_i)

outputs = []
weightedSums = []

outputs[0] = getWeightedSum(0, input)
weightedSums[0] = activationFunction(i, outputs[0])

for(i in range(len(layers)):
    weightedSums[i] = getWeightedSum(i, outputs[i-1])
    outputs[i] = activationFunction(i, weightedSums[i])
```

Step 2: Back-propagation

```
# dC/da_k * da_k/dz_k
currentCostDerivatives = getCostDerivatives(outputs[len(layers)-1], expectedOutput) # dC/da_k
for(l in range(len(layers)-1,-1,-1):
    currentCostDerivatives[l] *= (1.0f/getLayerSize(len(layers) - 1) * activationDerivatives[l];


for(l in range(len(layers)-1,-1,-1):
    # Next cost derivatives computation
    if l>0:
        nextCostDerivatives = []
        for(i in range(getLayerSize(l-1))):
            nextCostDerivatives[i] = 0
            for(k in range(getLayerSize(l))):
                nextCostDerivatives[i] += currentCostDerivatives[k] * getWeight(l,k,i) # dC/da_k * da_k/dz_k * dz_k/da_i

    # Adjust the weights and biases of the current layer

    prevLayerOutput = outputs[l-1] if l>0 else input

    for(i in range(getLayerSize(l))):
        for(j in range(len(prevLayerOutput)):
            setWeight(l,i,j) -= lr * currentCostDerivatives[i] * prevLayerOutput[j] # lr = learning rate and we have dC/da_k * da_k/dz_k * dz_k/dw_i,j
            setBias(l,i) -= lr * currentCostDerivatives[i] # dC/da_k * da_k/dz_k

    currentCostDerivatives = nextCostDerivatives
```

Conv2D and Pooling layers

Work in progress