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 :

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

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

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

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

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

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

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

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

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

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, Max-pooling and flatten layers

1 Conv2D Layer

Here we will look at the following example :

1. Input: 3x3 matrix $w_I = 3$
2. Filter: 2x2 matrix $w_F = 2$
3. Number of output matrices: 3
4. Padding p (Number of lines and columns of zeros added). We want in this example to get an output with the same dimensions as the input. Additionally, if the input matrix length and width are equal we need $w_I + p - w_F + 1$ iterations to read a full row, this number will be output matrix width. Hence we have: $w_I + p - w_F + 1 = w_I \Leftrightarrow p = w_F - 1$. Here $p = 1$
5. Activation function = ReLU : We use it to keep positive values for the next layers

Feed-forward

For each filter Fn :

$$Fn = \begin{pmatrix} Fn_{1,1} & Fn_{1,2} \\ Fn_{2,1} & Fn_{2,2} \end{pmatrix} \tag{6}$$

$$input = I = \begin{pmatrix} I_{1,1} & I_{1,2} & I_{1,3} \\ I_{2,1} & I_{2,2} & I_{2,3} \\ I_{3,1} & I_{3,2} & I_{3,3} \end{pmatrix} \tag{7}$$

We add padding, if p is odd then we add more one more row and column of zero on the last row and column, otherwise there is the same padding on every sides. In this example $p = 1$ is odd and $p/2 = 0$, thus we don't add zeros on the upper and left sides (because $p/2 = 0$), but we add one row and one columns on the lower and right sides.

$$input = I_{pad} = \begin{pmatrix} I_{1,1} & I_{1,2} & I_{1,3} & 0 \\ I_{2,1} & I_{2,2} & I_{2,3} & 0 \\ I_{3,1} & I_{3,2} & I_{3,3} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{8}$$

$$output = On = Conv\left(\begin{pmatrix} I_{1,1} & I_{1,2} & I_{1,3} & 0 \\ I_{2,1} & I_{2,2} & I_{2,3} & 0 \\ I_{3,1} & I_{3,2} & I_{3,3} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \left(\boxed{\begin{matrix} Fn_{1,1} & Fn_{1,2} \\ Fn_{2,1} & Fn_{2,2} \end{matrix}}\right)\right) \tag{9}$$

$$= \begin{pmatrix} On_{1,1} & On_{1,2} & On_{1,3} \\ \boxed{On_{2,1}} & On_{2,2} & On_{2,3} \\ On_{3,1} & On_{3,2} & On_{3,3} \end{pmatrix}$$

Here, for instance, $On_{2,1} = ReLU(I_{2,1} \times Fn_{1,1} + I_{2,2} \times Fn_{1,2} + I_{3,1} \times Fn_{2,1} + I_{3,2} \times Fn_{2,2})$

Back-propagation

We need to calculate two derivatives for each input matrices :

1. One to continue the back-propagation in the previous layers
2. Another one to adjust the filters values (similarly to the weights of the Dense layers)

Let's begin with the first one:

$$
\begin{aligned}
&\forall a, b \in [\![1, w_O]\!]^2, \forall i, j \in [\![a, a + w_F]\!] \times [\![b, b + w_F]\!],\\
&\frac{\partial O_{a,b}}{\partial I_{i,j}} = \frac{\partial \sum\limits_{k,l \in [\![0, w_F - 1]\!] \times [\![0, w_F - 1]\!]} h(a + k, b + l) \times F_{k+1,l+1}}{\partial I_{i,j}}\\
&\text{where } h(k, l) = \begin{cases} I_{k,l} & \text{if } k, l \in [\![1 + \frac{p}{2}, w_I]\!]^2 \\ 0 & \text{else} \end{cases}
\end{aligned}
\tag{10}
$$

So we get

$$
\frac{\partial O_{a,b}}{\partial I_{i,j}} = \begin{cases} F_{i-a+1,j-b+1} & \text{if } (i - a), (j - b) \in [\![1, w_O]\!]^2 \\ 0 & \text{else} \end{cases}
\tag{11}
$$

IN PROGRESS

Then we calculate the second one:

$$
\begin{aligned}
&\forall a, b \in [\![1, w_O]\!]^2, \forall i, j \in [\![1, w_F]\!]^2,\\
&\frac{\partial O_{a,b}}{\partial F_{i,j}} = \frac{\partial \sum\limits_{k,l \in [\![0, w_F - 1]\!] \times [\![0, w_F - 1]\!]} h(a + k, b + l) \times F_{k+1,l+1}}{\partial F_{i,j}}\\
&\frac{\partial O_{a,b}}{\partial F_{i,j}} = h(a + i - 1, b + j - 1)\\
&\frac{\partial O_{a,b}}{\partial F_{i,j}} = \begin{cases} I_{a+i-1,b+j-1} & \text{if } (a + i - 1), (b + j - 1) \in [\![1 + \frac{p}{2}, w_I]\!]^2 \\ 0 & \text{else} \end{cases}
\end{aligned}
\tag{12}
$$

IN PROGRESS

2 Max-pooling Layer

Feed-forward

Here we will look at the following example :

1. Input: 3x3 matrix $w_I = 3$
2. Filter: 2x2 matrix $w_F = 2$ because we want to divide the resolution by 2 here
3. Output: 2x2 matrix $w_O = 2$
4. Stride: $s = w_F = 2$ because we don't want to re-read values here. (s is the width of the step between 2 iterations when we move the filter (e.g. here we move the filter matrix of 2 cells to the right for each iteration, and of 2 cells to the bottom when we reached the last cell of the row)).
5. Padding: p (Number of lines and columns of zeros added). Here, we want to satisfy the previous conditions while minimizing the padding, so we get: $w_I \equiv p \mod w_F$ (i.e. in *C++*: p = w_I % w_F). Here we get $p = 1$.
6. We don't need an activation function here, since the max of a positive value is always positive, there are no weights (that could be negative for Conv2D filters or Dense layers) involved here.

If we have multiple input matrices, we just apply it to each one, and we get as many output matrices as we got in the input.

We add padding the same way as before.

$$input = I_{pad} = \begin{pmatrix} I_{1,1} & I_{1,2} & I_{1,3} & 0 \\ I_{2,1} & I_{2,2} & I_{2,3} & 0 \\ I_{3,1} & I_{3,2} & I_{3,3} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{13}$$

$$MaxPool_{2x2}\left(\begin{pmatrix} I_{1,1} & I_{1,2} & I_{1,3} & 0 \\ I_{2,1} & I_{2,2} & I_{2,3} & 0 \\ I_{3,1} & I_{3,2} & I_{3,3} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\right) = \begin{pmatrix} \boxed{O_{1,1}} & O_{1,2} \\ O_{2,1} & \boxed{O_{2,2}} \end{pmatrix} \tag{14}$$

For example, $O_{1,1} = \max(I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2})$

Back-propagation

Here, there is no parameter to adjust, so we just compute the derivative used to continue the back-propagation for each input matrices. It's similar to the section with the Dense layers and it will be used in the for loop of the back-propagation. We have $O_{a,b}^{(l)} = a_p^{(l)} = a_k^{(l)}$, it's just that we have 2 dimensions instead of one, but if we consider it flatten then it's the same.

$$\forall a, b \in [\![1, w_O]\!]^2, \forall i, j \in [\![(a-1) \times s + 1, a \times s]\!] \times [\![(b-1) \times s + 1, b \times s]\!],\ \frac{\partial O_{a,b}}{\partial I_{i,j}}$$

$$= \begin{cases} 1 & \text{if } i, j = \underset{k,l \in [\![(a-1)\times s+1, a\times s]\!] \times [\![(b-1)\times s+1, b\times s]\!]}{\arg\max} (I_{k,l}) \\ 0 & \text{else} \end{cases} \tag{15}$$

Thus, we have defined above $\frac{\partial z_k^{(l)}}{\partial a_i^{(l-1)}}$ and we have $\frac{\partial a_k^{(l)}}{\partial z_k^{(l)}} = 1$

3 Flatten Layer

This layer take one or multiple input matrices and flatten and concatenate them to get only one vector (dim = 1)

Feed-forward

$$flatten(I) = flatten\left(\begin{pmatrix} I1_{1,1} & I1_{1,2} & I1_{1,3} \\ I1_{2,1} & I1_{2,2} & I1_{2,3} \\ I1_{3,1} & I1_{3,2} & I1_{3,3} \end{pmatrix}, \begin{pmatrix} I2_{1,1} & I2_{1,2} & I2_{1,3} \\ I2_{2,1} & I2_{2,2} & I2_{2,3} \\ I2_{3,1} & I2_{3,2} & I2_{3,3} \end{pmatrix}\right) = \tag{16}$$

$(I1_{1,1},\ I1_{1,2},\ I1_{1,3},\ I1_{2,1},\ I1_{2,2},\ I1_{2,3},\ I1_{3,1},\ I1_{3,2},\ I1_{3,3},\ I2_{1,1},\ I2_{1,2},\ I2_{1,3},\ I2_{2,1}, I2_{2,2}, I2_{2,3}, I2_{3,1}, I2_{3,2}, I2_{3,3})$

Back-propagation

There is no derivative involved, we simply revert the flatten transformation, so we need to store the dimensions before the transformation:

$(I1_{1,1}, I1_{1,2}, I1_{1,3}, I1_{2,1}, I1_{2,2}, I1_{2,3}, I1_{3,1}, I1_{3,2}, I1_{3,3}, I2_{1,1}, I2_{1,2}, I2_{1,3}, I2_{2,1}, I2_{2,2}, I2_{2,3}, I2_{3,1}, I2_{3,2}, I2_{3,3})$

$$\mapsto \left(\begin{pmatrix} I1_{1,1} & I1_{1,2} & I1_{1,3} \\ I1_{2,1} & I1_{2,2} & I1_{2,3} \\ I1_{3,1} & I1_{3,2} & I1_{3,3} \end{pmatrix}, \begin{pmatrix} I2_{1,1} & I2_{1,2} & I2_{1,3} \\ I2_{2,1} & I2_{2,2} & I2_{2,3} \\ I2_{3,1} & I2_{3,2} & I2_{3,3} \end{pmatrix}\right) \tag{17}$$