Deep Learning - Assignment 1

T. Castiglione Ferrari - 2673807

November 11, 2020

Exercise 1

Let's calculate the local derivatives of the Softmax activation function and the Cross Entropy loss function:

- Softmax activation function:

$$f_i(x) = p(y|x; w) = \frac{e^{x_i}}{\sum_{i=1}^{J} e^{x_j}}$$

for i = 1, ..., J.
Remembering the chain rule: For input -> b -> c -> output the derivative is

$$\frac{\partial output}{\partial input} = \frac{\partial output}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial input}$$

Then, the local derivative of softmax is the derivative of the output with regards to the input. Considering an arbitrary i and j:

$$\frac{\partial f_i}{\partial x_j} = \frac{\partial \frac{e^{x_i}}{\sum_{i=1}^{J} e^{x_j}}}{\partial x_j}$$

Now we can consider at two cases:

  - If i == j:

$$\frac{\partial \frac{e^{x_i}}{\sum_{i=1}^{J} e^{x_j}}}{\partial x_j} = \frac{e^{a_i}\sum -e^{a_j}e^{a_i}}{(\sum_{k=1}^{N} e^{a_k})^2} = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}\frac{\sum -e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} = f_i(x)(1 - f_j(x))$$

  - If i != j, the same concept goes:

$$\frac{\partial \frac{e^{x_i}}{\sum_{i=1}^{J} e^{x_j}}}{\partial x_j} = \frac{0 - e^{a_j}e^{a_i}}{(\sum_{k=1}^{N} e^{a_k})^2} = -\frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}}\frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} = -f_i(x)f_j(x)$$

In conclusion, considering

$$\sigma_{ij} = \begin{cases} 0 & \text{if i == j} \\ 1 & \text{if i != j} \end{cases}$$

we have:

$$\frac{\partial f_i}{\partial x_j} = f_i(x)(\sigma_{ij} - f_j(x))$$

- Remembering the chain rule and the results obtained above, we can analyze the Cross Entropy loss function, considering o the output, t the target and

$$t_j^x, o_j^x = [0, 1]$$

the target and obtained values of x for output neurons j:

$$E_x = -\sum_n (\log(p(y_n|x_n; w)) = -\sum_k (t_k^x \log o_k^x + (1 - t_k^x) log(1 - o_k^x))$$

Now we can compute the derivative of the Cross Entropy loss function as:

$$\frac{\partial E_x}{\partial o_j^x} = \frac{\partial(-\sum_k (t_k^x \log o_k^x + (1 - t_k^x) \log(1 - o_k^x)))}{\partial o_j^x} = \frac{\partial(t_j^x \log o_j^x)}{\partial o_j^x} - \frac{\partial(1 - t_j^x) \log(1 - o_j^x}{\partial o_j^x} =$$

$$\frac{\partial E_x}{\partial o_j^x} = \frac{t_j^x}{o_j^x} + \frac{1 - t_j^x}{1 - o_j^x}$$

Furthermore, if the layer before had Softmax activation function, this can be simplified:

$$\frac{\partial E_x}{\partial x_j} = \frac{\partial E_x}{\partial o_j^x} \frac{\partial o_j^x}{\partial x_j} = \frac{t_j^x}{o_j^x} + \frac{1 - t_j^x}{1 - o_j^x} (o_j^x (1 - o_j^x) = t_j^x + t_j^x o_j^x + o_j^x - t_j^x o_j^x =$$

$$\frac{\partial E_x}{\partial x_j} = -t_j^x + o_j^x$$

Considering the last result, it appears that

$$\frac{\partial E_x}{\partial o_j^x}$$

is not strictly required for computing the back backward pass, if the last layer (before the loss function) is passed through the Softmax activation function.

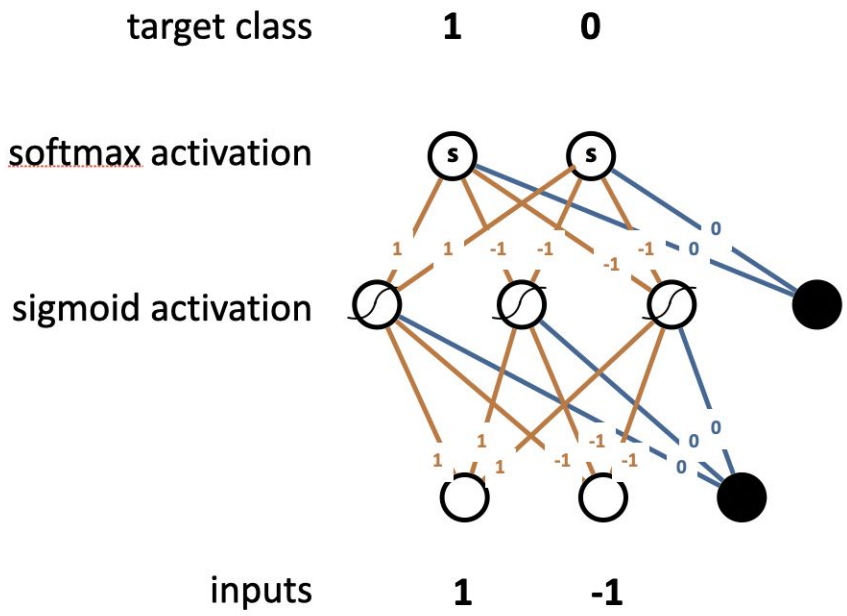

Figure 1: Multi Layer Perceptron

Exercise 2

Considering the given network structure shown above - and assuming that the loss function is Cross Entropy loss function, the results obtained are the following:

- $$\frac{\partial O}{\partial V} = \frac{\partial O}{\partial \text{CrossEntropy}} \cdot \frac{\partial \text{CrossEntropy}}{\partial \text{Softmax}} \cdot \frac{\partial \text{Softmax}}{\partial V} = (t - a) \cdot a_2^\top =$$
$$= [[0.11009963, 0.11009963, 0.11009963][0.11009963, 0.11009963, 0.11009963]]$$
where a_2 is the output of the Sigmoid layer, t is the target output and o is the actual output.

- $$\frac{\partial O}{\partial C} = \frac{\partial O}{\partial \text{Softmax}} \cdot \frac{\partial \text{CrossEntropy}}{\partial \text{Softmax}} \cdot \frac{\partial \text{Softmax}}{\partial} = (t - a) =$$
$$= [[0.125], [0.125]]$$

- $$\frac{\partial O}{\partial W} = \frac{\partial O}{\partial \text{CrossEntropy}} \cdot \frac{\partial \text{CrossEntropy}}{\partial \text{Softmax}} \cdot \frac{\partial \text{Softmax}}{\partial V} \cdot \frac{\partial V}{\partial \text{Sigmoid}} \frac{\partial \text{Sigmoid}}{\partial W} = a_2 \cdot (1 - a_2) \cdot V^\top \cdot (t - a) \cdot a_1^\top =$$
$$= [[0.0262484, -0.0262484][-0.0262484, 0.0262484][-0.0262484, 0.0262484]]$$
where a_1 is the input data

- $$\frac{\partial O}{\partial B} = \frac{\partial O}{\partial \text{CrossEntropy}} \cdot \frac{\partial \text{CrossEntropy}}{\partial \text{Softmax}} \cdot \frac{\partial \text{Softmax}}{\partial V} \cdot \frac{\partial V}{\partial \text{Sigmoid}} \frac{\partial \text{Sigmoid}}{\partial B} = a_2 \cdot (1 - a_2) \cdot V^\top \cdot (t - a) =$$

$$= [[0.0262484][-0.0262484][-0.0262484]]$$

Exercise 3

Because of an unknown bug, which I tried to solve for several days, I am unable to present satisfactory data, as, even if the system works as intended, the lack of convergence/divergence behaviour and the overall results thus obtained suggest that there is a conceptual error in the math used - given also the different results in the point above then the one suggested by you -. As I tried hard, it is even harder to admit that it seems that my project of "recreate" a neural network framework - such as Tensorflow and Pytorch - backfired. This being said, I am going to keep trying solving this issue, as the system developed by me, even if somehow chaotic - perhaps caused by my desire to over accomplish in such a short amount of time -, is modular and extensible.
This explanation does not want to be an excuse, just an informative note.