Deep Learning Assignment 1

Yke Rusticus
Student nr.: 11306386
University of Amsterdam
yke.rusticus@student.uva.nl
15/11/2019

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

Question 1.1a

$\frac{\partial L}{\partial x^{(N)}}$:

$$\begin{aligned}
L &= -\sum_i t_i \log x_i^{(N)} && (1)\\
\left(\frac{\partial L}{\partial x^{(N)}}\right)_i &= -t_i / x_i^{(N)} && (2)\\
\frac{\partial L}{\partial x^{(N)}} &= -\begin{bmatrix} t_1/x_1^{(N)} & t_2/x_2^{(N)} & \cdots & t_{d_N}/x_{d_N}^{(N)} \end{bmatrix} && (3)\\
&= -[t \oslash x]^\top \qquad \text{element-wise division} && (4)
\end{aligned}$$

$\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}$:

$$\begin{aligned}
x^{(N)} &= \text{softmax}(\tilde{x}^{(N)}) = \frac{\exp(\tilde{x}^{(N)})}{\sum_{i=1}^{d_N} \exp(\tilde{x}_i^{(N)})} && (5)\\
\left(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\right)_{ij} &= \frac{\partial}{\partial \tilde{x}_j^{(N)}} \frac{\exp(\tilde{x_i}^{(N)})}{\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)})} && (6)\\
&= \frac{\delta_{ij} \exp(\tilde{x}_j^{(N)}) \sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}) - \exp(\tilde{x}_i^{(N)}) \exp(\tilde{x}_j^{(N)})}{\left(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)})\right)^2} && (7)\\
&= \text{softmax}(\tilde{x}^{(N)})_j (\delta_{ij} - \text{softmax}(\tilde{x}^{(N)})_i) && (8)\\
\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} &= \text{diag}\big(\text{softmax}(\tilde{x}^{(N)})\big) - \text{softmax}(\tilde{x}^{(N)}) \text{softmax}(\tilde{x}^{(N)})^\top && (9)
\end{aligned}$$

Here the function diag$(\cdot)$ is defined such that, given a vector $x \in \mathbb{R}^N$:

$$\text{diag}(x) = \begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & x_N \end{bmatrix} \tag{10}$$

$\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}$:

$$x^{(l)} = \text{LeakyReLU}(\tilde{x}^{(l)}) = \max(0, \tilde{x}^{(l)}) + a\min(0, \tilde{x}^{(l)}), \qquad l < N \tag{11}$$

$$\left(\frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}}\right)_{ij} = \frac{\partial}{\partial \tilde{x}_j^{(l)}}\big(\max(0, \tilde{x}_i^{(l)}) + a\min(0, \tilde{x}_i^{(l)})\big) \tag{12}$$

$$= \begin{cases} \delta_{ij} & \text{if } \tilde{x}_i^{(l)} > 0 \\ a\delta_{ij} & \text{if } \tilde{x}_i^{(l)} \leq 0 \end{cases} \tag{13}$$

$$\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} = \begin{bmatrix} \begin{cases} 1 & \text{if } \tilde{x}_1^{(l)} > 0 \\ a & \text{if } \tilde{x}_1^{(l)} \leq 0 \end{cases} & 0 & \cdots & 0 \\ 0 & \begin{cases} 1 & \text{if } \tilde{x}_2^{(l)} > 0 \\ a & \text{if } \tilde{x}_2^{(l)} \leq 0 \end{cases} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \begin{cases} 1 & \text{if } \tilde{x}_{d_l}^{(l)} > 0 \\ a & \text{if } \tilde{x}_{d_l}^{(l)} \leq 0 \end{cases} \end{bmatrix} \tag{14}$$

$\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}$:

$$\tilde{x}^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)}, \quad l = 1, \ldots, N \tag{15}$$

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}\right)_{ij} = \frac{\partial}{\partial x_j^{(l-1)}}\big(w_i^{(l)}x^{(l-1)} + b_i^{(l)}\big), \qquad w_i^{(l)} \text{ is the } i\text{'th row of } W^{(l)} \tag{16}$$

$$= W_{ij}^{(l)} \tag{17}$$

$$\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} = W^{(l)} \tag{18}$$

$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}$:

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}\right)_{ijk} = \frac{\partial}{\partial W_{jk}^{(l)}}\big(w_i^{(l)}x^{(l-1)} + b_i^{(l)}\big) \tag{19}$$

$$= \delta_{ij}x_k^{(l-1)} \tag{20}$$

$$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \text{diag}^*(x^{(l-1)}) \tag{21}$$

In this case $\text{diag}^*(\cdot)$ is defined a little bit different than in the previous case (hence the asterisk), i.e. for any x:

$$\text{diag}^*(x) = \begin{bmatrix} x & 0 & \cdots & 0 \\ 0 & x & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & x \end{bmatrix} \tag{22}$$

$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}$:

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}\right)_{ij} = \frac{\partial}{\partial b_j^{(l)}} b_i^{(l)} = \delta_{ij} \tag{23}$$

$$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = I \in \mathbb{R}^{d_l \times d_l}, \qquad I \text{ is the identity matrix} \tag{24}$$

Question 1.1b

We have now derived the following derivatives (see 1.1a):

$$\underbrace{\frac{\partial L}{\partial x^{(N)}}}_{(a)}, \quad \underbrace{\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}}_{(b)}, \quad \underbrace{\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}}_{(c)}, \quad \underbrace{\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}}_{(d)}, \quad \underbrace{\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}}_{(e)}, \quad \underbrace{\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}}_{(f)}, \tag{25}$$

to which we will refer in the following with their given subscripts (a), (b), etc.
Then we get for this question:

$$(i) \quad \frac{\partial L}{\partial \tilde{x}^{(N)}} = \underbrace{\frac{\partial L}{\partial x^{(N)}}}_{(a)} \underbrace{\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}}_{(b)} \tag{26}$$

$$(ii) \quad \frac{\partial L}{\partial \tilde{x}^{(l<N)}} = \frac{\partial L}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} = \underbrace{\frac{\partial L}{\partial \tilde{x}^{(l+1)}}}_{\substack{(i) \text{ if } l=N-1, \\ \text{otherwise prev.} \\ \text{result of } (ii)}} \underbrace{\frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}}}_{\substack{(d) \\ \text{rewritten}}} \underbrace{\frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}}}_{(c)} \tag{27}$$

$$(iii) \quad \frac{\partial L}{\partial x^{(l<N)}} = \underbrace{\frac{\partial L}{\partial \tilde{x}^{(l+1)}}}_{\substack{(i) \text{ if } l=N-1, \\ \text{otherwise prev.} \\ \text{result of } (ii)}} \underbrace{\frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}}}_{\substack{(d) \\ \text{rewritten}}} \tag{28}$$

$$(iv) \quad \frac{\partial L}{\partial W^{(l)}} = \underbrace{\frac{\partial L}{\partial \tilde{x}^{(l)}}}_{\substack{(i) \text{ if } l=N, \\ \text{otherwise } (ii)}} \underbrace{\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}}_{(e)} \tag{29}$$

$$(v) \quad \frac{\partial L}{\partial b^{(l)}} = \underbrace{\frac{\partial L}{\partial \tilde{x}^{(l)}}}_{\substack{(i) \text{ if } l=N, \\ \text{otherwise } (ii)}} \underbrace{\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}}_{(f)} \tag{30}$$

(d): $\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} = W^{(l)}$, so (d) rewritten: $\frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}} = W^{(l+1)}$.
We see that the equations $(i), \ldots, (v)$ are fully determined by themselves (recursively) and by the derivatives obtained in question 1.1a. The given answers are purposely not written out fully in order to maintain the overview, as they are already defined in matrix form in question 1.1a.

Question 1.1c

In the case of $B = 1$, the gradients (or backprop equations) depend only on one data sample. If a batchsize $B \neq 1$ is used, then the backprop equations become dependent on B data samples. In that case the gradients are approximated as the mean of all individual gradients depending on one of the B samples, similar to how the total loss is defined as the mean over all the individual losses.

1.2 NumPy implementation

Question 1.2

The NumPy implementation is provided in the code. Results for default parameters are given in Figure 1. As we can see from the figure, the test accuracy model fluctuates between 0.4 and 0.5, corresponding to the provided aim of 0.46.

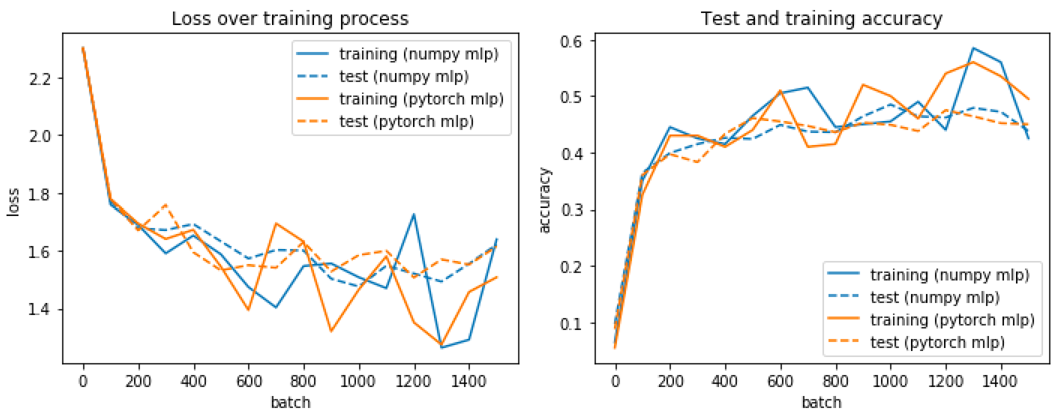

Figure 1: Model performance of the numpy and pytorch MLP (Multi-Layer-Perceptron) implementation, using default parameters.

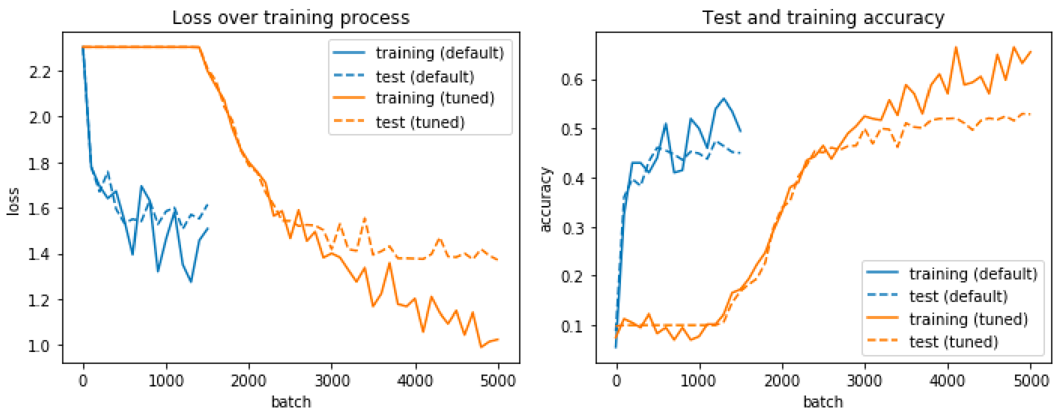

Figure 2: Model performance of the pytorch MLP, shown for the dafault parameters and the tuned parameters. The parameter values are given in Table 1

Parameter	Default	Tuned
learning_rate	0.002	0.008
neg_slope	0.02	0.05
batch_size	200	400
max_steps	1500	5000
dnn_hidden_units	100	100, 100

Table 1: Parameter values used in the default and tuned model.

2 PyTorch MLP

Question 2

Figure 1 also shows results of the pytorch implementation. We can see that the results using the default parameters are (as expected) similar to the NumPy implementation. After parameter tuning, we have found a set of parameters for which the model performs slightly better than using the default parameters. The results for this parameter setting, compared to the default setting are shown in Figure 2. Over the last 5 test evaluations, the tuned model has an accuracy of 0.524. In the figure we see that the tuned model needs some iterations to start improving. This might be due to a local minimum, where the model needs to escape from at the beginning. For this reason, a learning rate of 0.008 was chosen (compared to the smaller default learning rate of 0.002), as this was just enough not to get stuck at the beginning, while still converging to a decent minimum in the end. The batch size of 400 was chosen twice as big as the default, in order to reduce large noisy jumps in the weight updates. The addition of an extra layer, and the increased negative slope for the LeakyReLU module, were chosen based on an educated guess and resulted in the final performance as shown in Figure 2. The parameters are given in Table 1.

3 Custom Module: Batch Normalization

3.1 Automatic differentiation

Question 3.1

See code.

3.2 Manual implementation of backward pass

Question 3.2a

Given are:

$$y_i^s = \gamma_i \hat{x}_i^s + \beta_i \tag{31}$$

$$\hat{x}_i^s = \frac{x_i^s - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{32}$$

$$\sigma_i^2 = \frac{1}{B}\sum_{s=1}^{B}(x_i^s - \mu_i)^2 \tag{33}$$

$$\mu_i = \frac{1}{B}\sum_{s=1}^{B} x_i^s \tag{34}$$

Derivation of the derivatives:

$$\left(\frac{\partial L}{\partial \gamma}\right)_j = \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \gamma_j} \tag{35}$$

$$= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\delta_{ij}\hat{x}_i^s \tag{36}$$

$$\left(\frac{\partial L}{\partial \beta}\right)_j = \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \beta_j} \tag{37}$$

$$= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\delta_{ij} \tag{38}$$

The derivation of the third derivative $\left(\frac{\partial L}{\partial x}\right)_j^r$ is a bit longer, so we will do this in parts.

$$\begin{aligned}
\frac{\partial}{\partial x_j^r}(x_i^s - \mu_i) &= \delta_{rs}\delta_{ij} - \frac{\partial}{\partial x_j^r}\mu_i && (39)\\
&= \delta_{rs}\delta_{ij} - \delta_{ij}\frac{1}{B} && (40)\\
&= \delta_{ij}\big(\delta_{rs} - \frac{1}{B}\big) && (41)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial x_j^r}\sqrt{\sigma_i^2+\epsilon} &= \frac{1}{2\sqrt{\sigma_i^2+\epsilon}}\frac{\partial}{\partial x_j^r}\sigma_i^2 && (42)\\
&= \frac{1}{2}\underbrace{\frac{1}{\sqrt{\sigma_i^2+\epsilon}}}_{:=g}\frac{\partial}{\partial x_j^r}\frac{1}{B}\sum_{s=1}^{B}(x_i^s-\mu_i)^2 && (43)\\
&= \frac{g}{2}\frac{1}{B}\sum_{s=1}^{B}\frac{\partial}{\partial x_j^r}(x_i^s-\mu_i)^2 && (44)\\
&= \frac{g}{2}\frac{1}{B}\sum_{s=1}^{B}2(x_i^s-\mu_i)\frac{\partial}{\partial x_j^r}(x_i^s-\mu_i) && (45)\\
&= g\frac{1}{B}\sum_{s=1}^{B}(x_i^s-\mu_i)\delta_{ij}(\delta_{rs}-\frac{1}{B}) && (46)\\
&= g\frac{1}{B}\Big[\sum_{s=1}^{B}(x_i^s-\mu_i)\delta_{ij}\delta_{rs} - \sum_{s=1}^{B}(x_i^s-\mu_i)\delta_{ij}\frac{1}{B}\Big] && (47)\\
&= g\frac{\delta_{ij}}{B}\Big[(x_i^r-\mu_i) - \underbrace{\frac{1}{B}\sum_{s=1}^{B}(x_i^s-\mu_i)}_{=\mu_i-\mu_i=0}\Big] && (48)\\
&= \frac{\delta_{ij}(x_i^r-\mu_i)}{B\sqrt{\sigma_i^2+\epsilon}} && (49)
\end{aligned}$$

For the full derivative we then get:

$$\begin{aligned}
\left(\frac{\partial L}{\partial x}\right)_j^r &= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial x_j^r} && (50)\\
&= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \hat{x}_i^s}\frac{\partial \hat{x}_i^s}{\partial x_j^r} && (51)\\
&= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\frac{\partial}{\partial x_j^r}\frac{x_i^s-\mu_i}{\sqrt{\sigma_i^2+\epsilon}} && (52)\\
&= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\frac{[\frac{\partial}{\partial x_j^r}(x_i^s-\mu_i)]\sqrt{\sigma_i^2+\epsilon} - (x_i^s-\mu_i)\frac{\partial}{\partial x_j^r}\sqrt{\sigma_i^2+\epsilon}}{\sigma_i^2+\epsilon} && (53)\\
&= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\frac{\delta_{ij}\big(\delta_{rs}-\frac{1}{B}\big)\sqrt{\sigma_i^2+\epsilon} - (x_i^s-\mu_i)\frac{\delta_{ij}(x_i^r-\mu_i)}{B\sqrt{\sigma_i^2+\epsilon}}}{\sigma_i^2+\epsilon} && (54)\\
&= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\delta_{ij}\frac{\big(\delta_{rs}-\frac{1}{B}\big)\sqrt{\sigma_i^2+\epsilon} - (x_i^s-\mu_i)\frac{(x_i^r-\mu_i)}{B\sqrt{\sigma_i^2+\epsilon}}}{\sigma_i^2+\epsilon} && (55)
\end{aligned}$$

Question 3.2b

Not implemented.

Question 3.2c

Not implemented.

4 PyTorch CNN

Question 4

Using the given ConvNet architechture, we get an accuracy of 0.76 (over the last 5 evaluations) using the default parameters. The results are shown in Figure 3. From the curves we see that the test accuracy gradually increases, while the training accuracy fluctuates. At around 4000 batches, test accuracy settles, while training accuracy is still changing. The training loss keeps going down, as expected, however it would have been interesting to compare it with the test loss. The test loss is not included, as the memory size of the computer system would not allow it (and the lack of time to solve this problem.) For example, if we would see that at some point the test loss goes up while training loss goes down, we could identify a possible start of overfitting. However, given the accuracy curves, this does not seem to be the case.

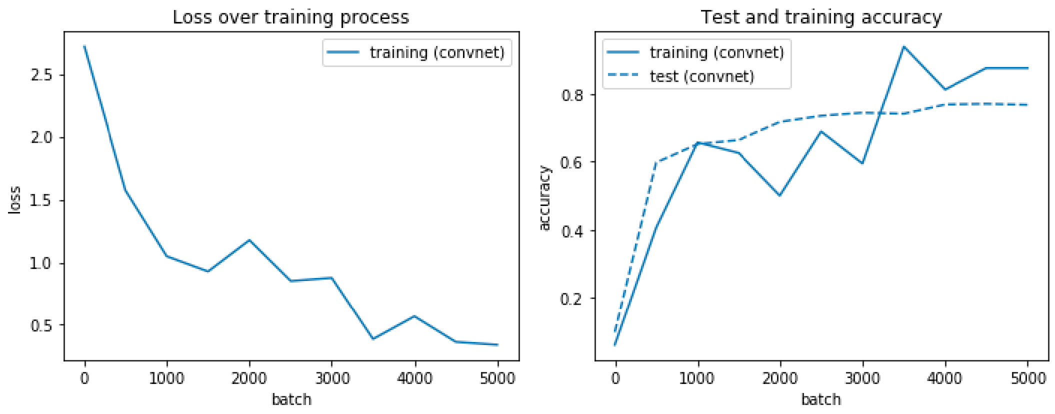

Figure 3: Loss and accuracy of the CNN using default parameters.