DL homework 1

Daan Smedinga 10560963

Collaborators: Maico Timmerman, Ruben Polak, Aron Hammond, Jasper Bakker.

1 MLP backprop and NumPy implementation

1.1 Question 1.1a

$$\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{(N)}}\right)_j = \frac{\partial - \sum_i t_i \ln x_i^{(N)}}{\partial x_j^{(N)}} \tag{1}$$

$$= \frac{\partial - \ln x_{argmax(\boldsymbol{t})}^{(N)}}{\partial x_j^{(N)}} \tag{2}$$

$$= \begin{cases} \frac{1}{x_j^{(N)}} & \text{if } j = argmax(\boldsymbol{t}) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The following uses $\mathcal{S}$ to refer to the Softmax function.

$$\left(\frac{\partial \boldsymbol{x}^{(N)}}{\partial \widetilde{\boldsymbol{x}}^{(N)}}\right)_{ij} = \frac{\partial \mathcal{S}\left(x_i^{(N)}\right)}{\partial x_j^{(N)}} \tag{4}$$

$$= \frac{\partial \frac{e^{x_i^{(N)}}}{\sum_{k=1}^{d_N} e^{x_k^{(N)}}}}{\partial x_j^{(N)}} \tag{5}$$

In the case of $i = j$ we get: (6)

$$= \frac{e^{x_i^{(N)}} \sum_{k=1}^{d_N} e^{x_k^{(N)}} - e^{x_i^{(N)}} e^{x_j^{(N)}}}{\left(\sum_{k=1}^{d_N} e^{x_k^{(N)}}\right)^2} \tag{7}$$

$$= \frac{e^{x_i^{(N)}}}{\sum_{k=1}^{d_N} e^{x_k^{(N)}}} \frac{\left[\sum_{k=1}^{d_N} e^{x_k^{(N)}}\right] - e^{x_j^{(N)}}}{\sum_{k=1}^{d_N} e^{x_i^{(N)}}} \tag{8}$$

$$= \mathcal{S}\left(x_i^{(N)}\right)\left(1 - \mathcal{S}\left(x_j^{(N)}\right)\right) \tag{9}$$

In the case of $i \neq j$ we get: (10)

$$= \frac{0 \sum_{k=1}^{d_N} e^{x_k^{(N)}} - e^{x_i^{(N)}} e^{x_j^{(N)}}}{\left(\sum_{k=1}^{d_N} e^{x_k^{(N)}}\right)^2} \tag{11}$$

$$= -\mathcal{S}\left(x_i^{(N)}\right) \mathcal{S}\left(x_j^{(N)}\right) \tag{12}$$

For some $l < N$:

$$\left(\frac{\partial \boldsymbol{x}^{(l)}}{\partial \widetilde{\boldsymbol{x}}^{(l)}}\right)_{ij} = \frac{\partial x_i^{(l)}}{\partial \widetilde{x}_j^{(l)}} \tag{13}$$

$$= \frac{\partial \max(0, \widetilde{x}_i^{(l)})}{\partial \widetilde{x}_j^{(l)}} \tag{14}$$

$$= \begin{cases} 1 & \text{if } i = j \wedge \widetilde{x}_j^{(l)} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

$$\frac{\partial \widetilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{x}^{(l-1)}} = \frac{\partial \left(W^{(l)} \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}\right)}{\partial \boldsymbol{x}^{(l-1)}} = \left(W^{(l)}\right)^T \tag{16}$$

$$\frac{\partial \widetilde{\boldsymbol{x}}^{(l)}}{\partial W^{(l)}} = \frac{\partial \left(W^{(l)} \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}\right)}{\partial W^{(l)}} = \left(\boldsymbol{x}^{(l-1)}\right)^T \tag{17}$$

$$\frac{\partial \widetilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \left(W^{(l)} \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}\right)}{\partial \boldsymbol{b}^{(l)}} = \mathcal{I} \tag{18}$$

1.2 Question 1.1b

Let

$$M = \begin{bmatrix} | & d_N & | \\ \mathcal{S}\left(\widetilde{\boldsymbol{x}}^{(N)}\right) & \dots\dots\dots & \mathcal{S}\left(\widetilde{\boldsymbol{x}}^{(N)}\right) \\ | & & | \end{bmatrix} \tag{19}$$

such that M is a $d_N \times d_N$ matrix. Then:

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(N)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{(N)}} \frac{\partial \boldsymbol{x}^{(N)}}{\partial \widetilde{\boldsymbol{x}}^{(N)}} \tag{20}$$

$$= -\frac{1}{x_{argmax(\boldsymbol{t})}^{(N)}} M \odot \left(\mathcal{I} - M^T\right) \tag{21}$$

Where $\odot$ is the element-wise multiplication operator.
For some l where $l < N$:

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(N)}} \frac{\partial \widetilde{\boldsymbol{x}}^{(N)}}{\partial \boldsymbol{x}^{(N-1)}} \frac{\partial \boldsymbol{x}^{(N-1)}}{\partial \widetilde{\boldsymbol{x}}^{(N-1)}} \frac{\partial \widetilde{\boldsymbol{x}}^{(N-1)}}{\partial \boldsymbol{x}^{(N-2)}} \frac{\partial \boldsymbol{x}^{(N-2)}}{\partial \widetilde{\boldsymbol{x}}^{(N-2)}} \cdots \frac{\partial \widetilde{\boldsymbol{x}}^{(l+1)}}{\partial \boldsymbol{x}^{(l)}} \frac{\partial \boldsymbol{x}^{(l)}}{\partial \widetilde{\boldsymbol{x}}^{(l)}} \tag{22}$$

$$= \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(N)}} \left[\prod_{i=1}^{N-l} \frac{\partial \widetilde{\boldsymbol{x}}^{(N-i+1)}}{\partial \boldsymbol{x}^{(N-i)}} \frac{\partial \boldsymbol{x}^{(N-i)}}{\partial \widetilde{\boldsymbol{x}}^{(N-i)}}\right] \tag{23}$$

$$= \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(N)}} \left[\prod_{i=1}^{N-l} \left(W^{(N-i+1)}\right)^T \operatorname{diag}\left(\Theta\left(\widetilde{\boldsymbol{x}}^{(N-i)}\right)\right)\right] \tag{24}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(l+1)}} \frac{\partial \widetilde{\boldsymbol{x}}^{(l+1)}}{\partial \boldsymbol{x}^{(l)}} \tag{25}$$

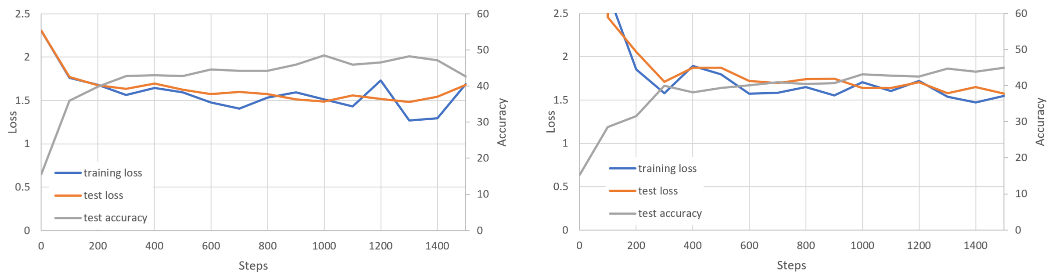
$$= \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(l+1)}} \left(W^{(l+1)}\right)^T \tag{26}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(l)}} \frac{\partial \widetilde{\boldsymbol{x}}^{(l)}}{\partial W^{(l)}} \tag{27}$$

$$= \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(l)}} \left(\boldsymbol{x}^{(l-1)}\right)^T \tag{28}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(l)}} \frac{\partial \widetilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \tag{29}$$

$$= \frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{x}}^{(l)}} \mathcal{I} \tag{30}$$



(a) NumPy implementation.

(b) PyTorch implementation.

Figure 1: Different implementation with the same (default) parameters.

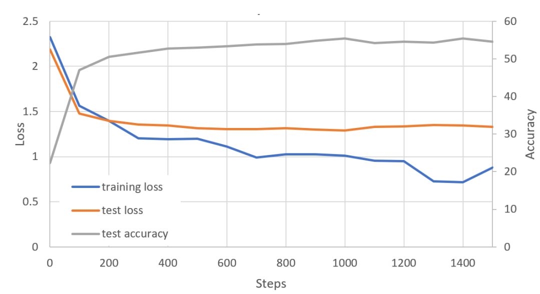


Figure 2: Training of the best MLP found (4 hidden layers of 500 units).

1.3 Question 1.1c

The equations don't change much. The $\boldsymbol{x}$ becomes effectively a matrix, but the operations are applied in such a way that the same operations for a single example are applied equally to each element across the batch dimension. The implementation changes a little bit because all of the data and gradients going through the network have an extra batch dimension. The batch dimension of the gradients is summed or averaged out before updating the weights.

1.4 Question 1.2

Figure 1(a) shows the training curve for the manual implementation of the MLP network in NumPy. The implementation was done completely with matrix operations, and the training time is equal to the PyTorch implementation (about 10 seconds on cpu). When adding a second hidden layer, the training process is only effective when the weights are initialized with 1e-2 variance instead of 1e-4.

2 PyTorch MLP

Constructing and training the same architecture in PyTorch was very easy. With default parameters, it led to a similar accuracy in similar training time (CPU), as figure 1(b) shows. Table 1 shows the successive experiments that were run to optimize the hyperparameters. The thought process went as follows: From default parameters, first attempt to increase training time. Add extra hidden layer and train for twice as long. Then experiment with different shapes (wide or deep) while keeping number of parameters similar (320k). Try much longer training time (9000 steps). Reached 49,9%! Try adding momentum. Try RMSprop, but it completely fails to learn with even high learning rates. Try Adam, extremely effective! It actually overfits in long runs, so add some weight decay. Try shallower networks again with more nodes. Try 4 layers of 1000 nodes. Takes a long time but reaches 53% accuracy! Add batch normalization layers after each hidden layer (red line crossed). Same network jumps to 55% accuracy, but also reaches it much more quickly. Try reducing complexity to see how it affects training time and accuracy. Eventually, a good balance was found with 4 layers of 500 units. On an nVidia GTX 970, it reaches 55% accuracy in 10 seconds! See figure 2 for the training curves.

hidden layers	optimizer	learning rate	momentum	weight decay	batch size	max steps	eval freq	max accuracy	time taken
100	SGD	0.002	0	0	200	1500	100	0.46	2.774
100	SGD	0.002	0	0	400	1500	100	0.4669	3.847
100,100	SGD	0.002	0	0	200	1500	100	0.4629	2.773
100,100	SGD	0.002	0	0	200	3000	100	0.4743	5.343
100,50,50	SGD	0.002	0	0	200	3000	100	0.4821	5.769
100,40,40,40,40	SGD	0.002	0	0	200	3000	100	0.4736	6.37
100,30,30,30,30,30,30,30,30,30	SGD	0.002	0	0	200	3000	100	0.1021	8.239
100,30,30,30,30,30,30,30,30,30	SGD	0.02	0	0	200	3000	100	0.3705	8.101
100,30,30,30,30,30,30,30,30,30	SGD	0.02	0	0	800	3000	100	0.378854	15.519
100,30,30,30,30,30,30,30,30,30	SGD	0.05	0	0	200	9000	100	0.499	24.222
400,100	SGD	0.002	0	0	200	3000	100	0.486	6.637
100,30,30,30,30,30,30,30,30,30	SGD	0.002	0.9	0	200	3000	100	0.3677	8.68
100,30,30,30,30,30,30,30,30,30	SGD	0.002	0.9	0	200	9000	100	0.4901	25.82
100,30,30,30,30,30,30,30,30,30	SGD	0.002	0.9	0	400	9000	100	0.4624	32.678
100,30,30,30,30,30,30,30,30,30	RMSprop	0.2	0	0	400	3000	100	0.1	11.8
100,30,30,30,30,30,30,30,30,30	RMSprop	0.2	0.9	0	400	3000	100	0.1	12.409
100,30,30,30,30,30,30,30,30,30	Adam	0.002	0	0	400	3000	100	0.511	12.824
100,30,30,30,30,30,30,30,30,30	Adam	0.001	0	0	400	9000	100	0.5051	37.233
100,30,30,30,30,30,30,30,30,30	Adam	0.001	0	0.001	400	9000	100	0.5183	39.899
100,30,30,30,30,30,30,30,30,30	Adam	0.001	0	0.003	400	9000	100	0.5137	38.719
100,30,30,30,30,30,30,30,30,30	Adam	0.001	0	0.002	400	9000	100	0.5136	39.938
400,100	Adam	0.001	0	0.002	400	1500	100	0.4687	5.85
400,100	Adam	0.001	0	0.002	400	300	100	0.271	1.256
400,100	Adam	0.001	0	0.002	400	3000	100	0.5031	11.687
100,100,100	Adam	0.001	0	0.002	400	3000	100	0.5051	9.331
100,100,100	Adam	0.001	0	0.002	400	1500	100	0.5009	4.683
400,400,400	Adam	0.001	0	0.002	400	1500	100	0.5173	6.908
1000,1000,1000,1000	Adam	0.001	0	0.002	400	1500	100	0.5187	17.853
1000,1000,1000,1000	Adam	0.001	0	0.002	400	9000	100	0.5342	106.6
1000,1000,1000,1000	Adam	0.001	0	0.002	400	9000	100	0.5534	120.333
1000,1000,1000,1000	Adam	0.002	0	0.002	400	1500	100	0.5215	19.681
1000,1000,1000,1000	Adam	0.001	0	0.002	400	3000	100	0.5534	38.921
1000,1000,1000,1000	Adam	0.0005	0	0.002	400	3000	100	0.5598	38.933
100,30,30,30,30,30,30,30,30,30	Adam	0.0005	0	0.002	400	3000	100	0.5065	19.176
400,400,400	Adam	0.0005	0	0.002	400	3000	100	0.5493	15.903
200,100	Adam	0.0005	0	0.002	400	3000	100	0.5458	10.271
100,50	Adam	0.0005	0	0.002	400	3000	100	0.5331	10.323
100,100,100	Adam	0.0005	0	0.002	400	1500	100	0.5354	6.027
50,50,50	Adam	0.0005	0	0.002	400	1500	100	0.5107	5.912
150,80	Adam	0.0005	0	0.002	400	1500	100	0.5279	5.37
3000	Adam	0.0005	0	0.002	400	1500	100	0.5265	22.112
200,50	Adam	0.001	0	0.002	400	1500	100	0.5256	5.308
200,100	Adam	0.001	0	0.002	400	1500	100	0.5351	5.306
300,300	Adam	0.001	0	0.002	400	1500	100	0.5367	6.322
80,80,80,80	Adam	0.001	0	0.002	400	1500	100	0.5298	6.391
200,100,100	Adam	0.001	0	0.002	400	1500	100	0.5346	6.051
100,100,100	Adam	0.001	0	0.002	400	1500	100	0.5307	5.968
200,100,50	Adam	0.001	0	0.002	400	1500	100	0.536	5.95
200,100,50	Adam	0.0005	0	0.002	400	3000	100	0.544	11.85
500,500,500,500	Adam	0.0005	0	0.002	400	3000	100	0.5541	19.809
500,500,500,500	Adam	0.0005	0	0.002	400	1500	100	0.5541	10.102
500,500,500,500	Adam	0.0002	0	0.002	400	1500	100	0.5494	10.062
500,500,500,500,500	Adam	0.0005	0	0.002	400	1500	100	0.5559	11.266
400,400,400,400,400	Adam	0.0005	0	0.002	400	1500	100	0.549	10.084
1000,1000,1000,1000,1000,1000,	Adam	0.0001	0	0.002	400	3000	100	0.5431	58.182
1000,1000,1000,1000,1000,1000,	Adam	0.0001	0	0.004	400	3000	100	0.542	58.482

Table 1: Results of experiments. Highlighted in green is the network chosen as best. Networks under red line use added batchnorm layers.

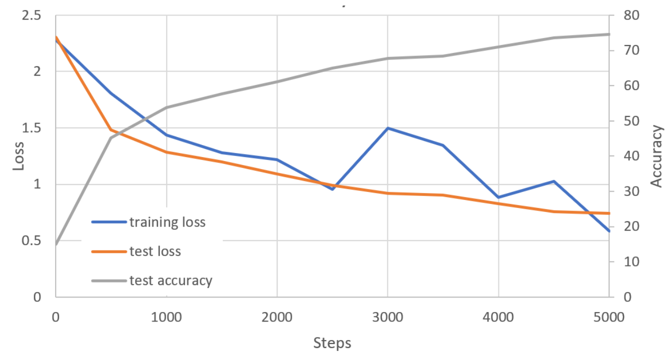


Figure 3: Training the convolutional neural network with default settings.

3 Custom Module: Batch Normalization

3.1 Question 3.1

I managed to implement the forward pass and satisfy the unit test.

3.2 Question 3.2

Unfortunately, I ran out of time to commit to batch norm.

4 PyTorch CNN

Implementing the convolutional neural network in PyTorch was very easy. Figure 3 shows the training curves with default parameters, reaching 75% accuracy as expected in about 170 seconds. It was mentioned that batch size is typically set to whatever fits in VRAM. With a batch size of 800 instead of 32, the VRAM was fully used and the network reached 75% accuracy in 95 seconds. After 170 seconds, it reached 79% accuracy. A significant improvement!