Deep Learning practical 1

Jochem Soons
MSc Artificial Intelligence
University of Amsterdam
UvA-id: 11327030
Date: 15-11-2020
`jochem.soons@student.uva.nl`

1 MLP backprop and NumPy Implementation

1.1 Evaluating the Gradients

Question 1.1a)

1) The derivative w.r.t. $\boldsymbol{W}$:

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{W}} \rightarrow \left[\frac{\delta L}{\delta \boldsymbol{W}}\right]_{ij} = \frac{\delta L}{\delta W_{ij}} & \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta Y_{mn}}{\delta W_{ij}} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta \sum_k X_{mk} W^T_{kn} + B_{mn}}{\delta W_{ij}} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \sum_k X_{mk} \delta_{in} \delta_{kj} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \delta_{in} X_{mj} \\
&= \sum_{m} \frac{\delta L}{\delta Y_{mi}} X_{mj} \\
&= \sum_{m} \left(\frac{\delta L}{\delta Y_{im}}\right)^T X_{mj} \\
&= \left[\left(\frac{\delta L}{\delta \boldsymbol{Y}}\right)^T \boldsymbol{X}\right]_{ij} \\
\frac{\delta L}{\delta \boldsymbol{W}} = \left(\frac{\delta L}{\delta \boldsymbol{Y}}\right)^T \boldsymbol{X} &
\end{aligned}
$$

2) The derivative w.r.t. $\boldsymbol{b}$:

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{b}} \rightarrow \Big[\frac{\delta L}{\delta \boldsymbol{b}}\Big]_i = \frac{\delta L}{\delta b_i}& \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta Y_{mn}}{\delta b_i} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta \sum_k X_{mk} W^T_{kn} + B_{mn}}{\delta b_i} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta B_{mn}}{\delta b_i} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta b_n}{\delta b_i} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \delta_{in} \\
&= \sum_{m} \frac{\delta L}{\delta Y_{mi}} \\
\frac{\delta L}{\delta \boldsymbol{b}} = \mathbf{1}^T \Big(\frac{\delta L}{\delta \boldsymbol{Y}}\Big)&
\end{aligned}
$$

Where $\mathbf{1}^T$ is the ones row-vector ($\mathbf{1}^T \in R^{1\times S}$).

3) The derivative w.r.t. $\boldsymbol{X}$:

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{X}} \rightarrow \Big[\frac{\delta L}{\delta \boldsymbol{X}}\Big]_{ij} = \frac{\delta L}{\delta X_{ij}}& \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta Y_{mn}}{\delta X_{ij}} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta \sum_k X_{mk} W^T_{kn} + B_{mn}}{\delta X_{ij}} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \sum_k \delta_{im}\delta_{jk} W_{nk} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \delta_{im} W_{nj} \\
&= \sum_{n} \frac{\delta L}{\delta Y_{in}} W_{nj} \\
&= \Big[\Big(\frac{\delta L}{\delta \boldsymbol{Y}}\Big)\boldsymbol{W}\Big]_{ij} \\
\frac{\delta L}{\delta \boldsymbol{X}} = \Big(\frac{\delta L}{\delta \boldsymbol{Y}}\Big)\boldsymbol{W}&
\end{aligned}
$$

Question 1.1b)

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{X}} \rightarrow \left[\frac{\delta L}{\delta \boldsymbol{X}}\right]_{ij} = \frac{\delta L}{\delta X_{ij}} & \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta Y_{mn}}{\delta X_{ij}} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta h(X_{mn})}{\delta X_{ij}} \delta_{im}\delta_{jn} \\
&= \frac{\delta L}{\delta Y_{ij}} \frac{\delta h(X_{ij})}{\delta X_{ij}} \\
&= \left[\Big(\frac{\delta L}{\delta \boldsymbol{Y}}\Big) \circ \frac{\delta h(\boldsymbol{X})}{\delta \boldsymbol{X}}\right]_{ij} \\
\frac{\delta L}{\delta \boldsymbol{X}} = \Big(\frac{\delta L}{\delta \boldsymbol{Y}}\Big) \circ \frac{\delta h(\boldsymbol{X})}{\delta \boldsymbol{X}} &
\end{aligned}
$$

Question 1.1c)

1) The Softmax derivative:

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{X}} \rightarrow \left[\frac{\delta L}{\delta \boldsymbol{X}}\right]_{ij} = \frac{\delta L}{\delta X_{ij}} & \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta Y_{mn}}{\delta X_{ij}} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta}{\delta X_{ij}} \frac{\exp(X_{mn})}{\sum_k \exp(X_{mk})} \\
&= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\exp(X_{mn})}{\sum_k \exp(X_{mk})} \Big(\delta_{jn} - \frac{\exp(X_{mj})}{\sum_k \exp(X_{mk})}\Big)\delta_{mi} \\
&= \sum_{n} \frac{\delta L}{\delta Y_{in}} \frac{\exp(X_{in})}{\sum_k \exp(X_{ik})} \Big(\delta_{jn} - \frac{\exp(X_{ij})}{\sum_k \exp(X_{ik})}\Big) \\
&= \sum_{n} \frac{\delta L}{\delta Y_{in}} Y_{in}(\delta_{jn} - Y_{ij})
\end{aligned}
$$

2) The Cross-Entropy derivative:

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{X}} \rightarrow \left[\frac{\delta L}{\delta \boldsymbol{X}}\right]_{ij} = \frac{\delta L}{\delta X_{ij}} & \\
&= \frac{\delta}{\delta X_{ij}} - \frac{1}{S} T_{sk} \log(X_{sk}) \\
&= -\frac{1}{S} \sum_{s,k} \delta_{is}\delta_{jk} \frac{T_{sk}}{X_{sk}} \\
&= -\frac{1}{S} \frac{T_{ij}}{X_{ij}} \\
&= -\frac{1}{S} \Big[\boldsymbol{T} \oslash \boldsymbol{X}\Big]_{ij} \\
\frac{\delta L}{\delta \boldsymbol{X}} = -\frac{1}{S}\Big(\boldsymbol{T} \oslash \boldsymbol{X}\Big) &
\end{aligned}
$$

Where $\oslash$ denotes the Hadamard division (element-wise divison) of matrices $\boldsymbol{T}$ and $\boldsymbol{X}$.

Question 1.1d)

1)

$$Y_{21} = X_{21}K_{11} + X_{22}K_{12} + X_{31}K_{21} + X_{32}K_{22}$$

2)

$$\begin{aligned}\frac{\delta L}{\delta K_{11}} &= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta Y_{mn}}{\delta K_{11}} \\ &= \frac{\delta L}{\delta Y_{11}} \frac{\delta Y_{11}}{\delta K_{11}} + \frac{\delta L}{\delta Y_{12}} \frac{\delta Y_{12}}{\delta K_{11}} + \frac{\delta L}{\delta Y_{21}} \frac{\delta Y_{21}}{\delta K_{11}} + \frac{\delta L}{\delta Y_{22}} \frac{\delta Y_{22}}{\delta K_{11}} \\ &= \frac{\delta L}{\delta Y_{11}} X_{11} + \frac{\delta L}{\delta Y_{12}} X_{12} + \frac{\delta L}{\delta Y_{21}} X_{21} + \frac{\delta L}{\delta Y_{22}} X_{22} \\ &= \sum_{m=1}^{2} \sum_{n=1}^{2} \frac{\delta L}{\delta Y_{mn}} X_{mn}\end{aligned}$$

3)

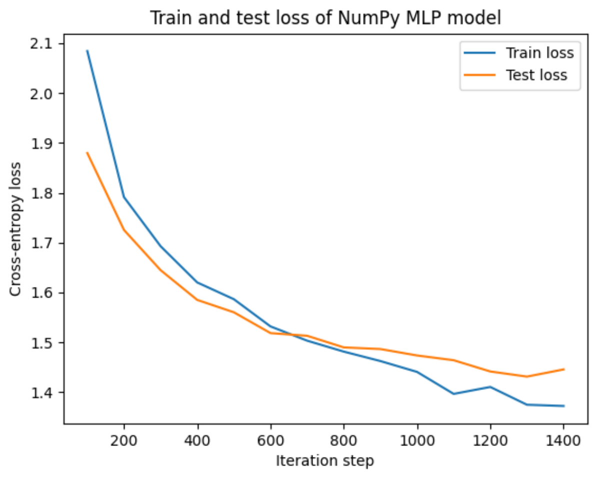
$$\begin{aligned}\frac{\delta L}{\delta X_{12}} &= \sum_{m,n} \frac{\delta L}{\delta Y_{mn}} \frac{\delta Y_{mn}}{\delta X_{12}} \\ &= \frac{\delta L}{\delta Y_{11}} \frac{\delta Y_{11}}{\delta X_{12}} + \frac{\delta L}{\delta Y_{12}} \frac{\delta Y_{12}}{\delta X_{12}} + \frac{\delta L}{\delta Y_{21}} \frac{\delta Y_{21}}{\delta X_{12}} + \frac{\delta L}{\delta Y_{22}} \frac{\delta Y_{22}}{\delta X_{12}} \\ &= \frac{\delta L}{\delta Y_{11}} K_{12} + \frac{\delta L}{\delta Y_{12}} K_{11} \\ &= \sum_{n=1}^{2} \frac{\delta L}{\delta Y_{1n}} K_{1n}\end{aligned}$$

1.2 NumPy implementation

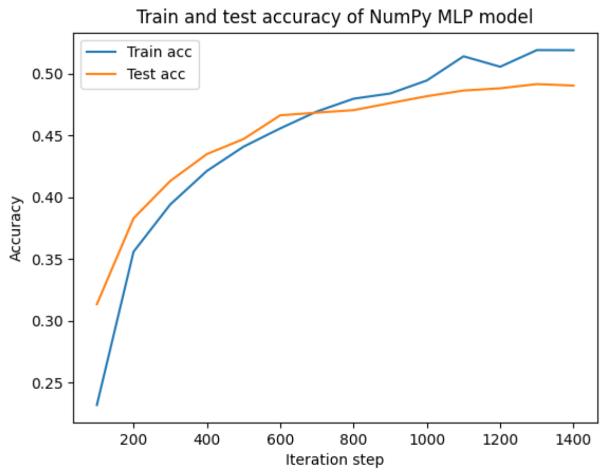
Question 1.2

The loss and accuracy plots of the NumPy MLP model, consisting of one hidden layer of 100 neurons, are displayed in figure 1. These plots were obtained using the default parameters (*batch size=200, learning rate=1e-3, max steps=1400*). Training for 1400 steps produced a test accuracy of 0.491.

One note I would like to make here, is that I did not choose to plot the training loss and accuracy at every training step, because then the plots tended to get pretty messy (with a lot of oscillations). Therefore, to smoothen the plot, I plotted the average training loss and accuracy between the previous evaluation step and the current evaluation step at each evaluation interval. I took this smoothing approach for *all subsequent plots in this assignment*. The plots in figure 1, unsurprisingly, basically just illustrate the training process of the MLP model, in which the train and test loss decreases over the number of iterations, and the train and test accuracy increases over the number of iterations.



(a) Train and test loss evaluated throughout training.



(b) Train and test accuracy evaluated throughout training.

Figure 1: Loss **(a)** and accuracy **(b)** plots throughout the training process of the NumPy MLP model, with evaluation frequency set at every 100th step.

2 Pytorch MLP

Question 2.1

After implementing the PyTorch version of the MLP, I performed multiple experiments, in which I applied different modifications to study how they would affect performance. An overview of these different setups can be found in table 1. From table 1, it becomes clear that using the default parameters of the previous question (ID 1), I obtained a final test accuracy of 0.464. The first thing I tried in order to improve results was applying a Batch normalization layer before each linear layer (ID 2), which already showed a lot of improvement in performance. Thereafter, in setups 3, 4 and 5 I tried adding more layers to the network as well as increasing the number of steps, which also benefitted performance. In all these setups (2-6), I also increased the learning rate because it seemed that using a learning rate of 1e-3 resulted in too slow convergence for these setups. In setup 6 I noticed that using the Adam optimizer instead of SGD with a slightly smaller batch size and a decreased learning rate and number of training steps would result in even better performance. This setup produced a final test accuracy of 0.560. After this optimal result, I also tried some other modifications, such as using Layer normalization instead of Batch normalization and applying other activation functions. But this did not improve results, and therefore my optimal performance was achieved using setup 6.

The loss and accuracy plots of the best performing model (ID 6) are displayed in figure 2. The plots once again visualize the learning process of the MLP model: the train and test loss decreases over the number of iterations, and the train and test accuracy increases over the number of iterations. Moreover, we do see that the gap between the training and testing loss/accuracy starts to increase at the final stages of training, which shows that there the model starts with slightly overfitting on the training data. Therefore, it would not be wise to increase the number of iterations steps to more than 2000 for this specific model setup.

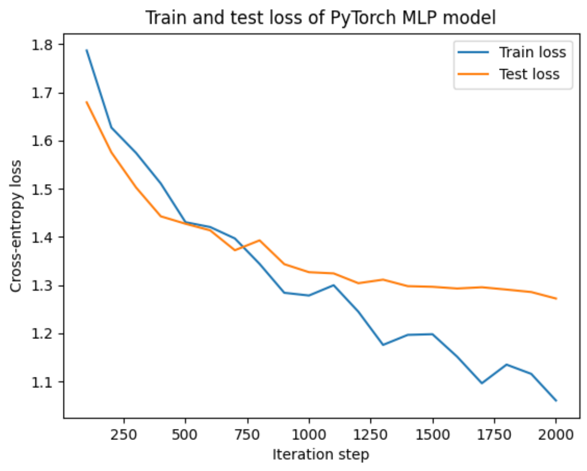
Question 2.2

It could be that the hyperbolic tangent function has an easier job approximating a non-linear classification function than the ELU function, which may yield higher performance and/or faster training convergence. So empirical testing of difference in performance between ELU and Tanh might always be a good idea, as this difference in performance is highly dependent on the dataset we have and the specific model architecture. I tested the difference in performance however, and as can be concluded from table 1, and using Tanh did not provide any benefits in performance over using ELU. So empirically, we already see that ELU outperformed the Tanh function, both in accuracy score and speed of convergence.

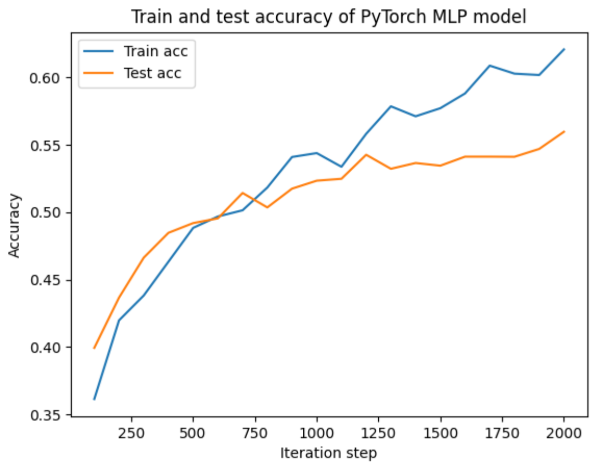
Moreover, by looking at the question from a theoretical perspective, there is a possible explanation of the evident disadvantage of using the Tanh function instead of ELU: this is the problem of vanishing gradients. Because the derivative of the Tanh function produces relatively low values compared to the derivate of ELU, gradients of the Tanh layers that are computed in the backpropagation process are generally small numbers. These small values in turn are used as factors in computing the other gradients of the network as we move towards in the input layer, causing the computed gradients to keep on decreasing within the network. This in turn produces difficulty in optimizing the network, as the low gradients result in a network that refuses to learn further or being simply too slow to reach optimal performance.

ID	**Setup**							**Test acc.**
	Layers	LR	Batch size	Steps	Norm./Reg.	Optim.	Act.func.	
1	(100)	1e-3	200	1400	-	SGD	ELU	0.464
2	(100)	1e-2	200	10000	Batch norm.	SGD	ELU	0.529
3	(100, 100)	1e-2	200	10000	Batch norm.	SGD	ELU	0.524
4	(1024, 512)	1e-2	200	10000	Batch norm.	SGD	ELU	0.539
5	(1024, 512, 256, 64)	1e-2	128	10000	Batch norm.	SGD	ELU	0.557
6	(1024, 512, 256, 64)	1e-3	128	2000	Batch norm.	Adam	ELU	**0.560**
7	(1024, 512, 256, 64)	1e-3	128	2000	Layer norm.	Adam	ELU	0.540
8	(1024, 512, 256, 64)	1e-3	128	2000	Batch norm.	Adam	ReLU	0.552
9	(1024, 512, 256, 64)	1e-3	128	5000	Batch norm.	Adam	Tanh	0.501

Table 1: Overview of different model setups and their corresponding test accuracy scores. The top performing accuracy is displayed in **bold**.



(a) Train and test loss evaluated throughout training.



(b) Train and test accuracy evaluated throughout training.

Figure 2: Loss **(a)** and accuracy **(b)** plots throughout the training process of the best performing PyTorch MLP model (setup 6 in table 1), with evaluation frequency set at every 100th step.

3 Custom Module: Layer Normalization

3.1 Automatic differentiation

3.2 Manual implementation of backward pass

Question 3.2 a)

1) the derivative w.r.t. to $\boldsymbol{\gamma}$:

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{\gamma}} \rightarrow \left[\frac{\delta L}{\delta \boldsymbol{\gamma}}\right]_i = \frac{\delta L}{\delta \gamma_i} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \frac{\delta Y_{sj}}{\delta \gamma_i} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \frac{\delta \gamma_j \hat{X}_{sj} + \beta_j}{\delta \gamma_i} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \delta_{ij} \hat{X}_{sj} \\
&= \sum_{s} \frac{\delta L}{\delta Y_{si}} \hat{X}_{si} \\
\frac{\delta L}{\delta \boldsymbol{\gamma}} = \mathbf{1}^T \left(\frac{\delta L}{\delta \boldsymbol{Y}} \circ \hat{\boldsymbol{X}}\right)
\end{aligned}
$$

Where $\mathbf{1}^T$ is the ones row-vector ($\mathbf{1}^T \in R^{1\times S}$).

2) the derivative w.r.t. to $\boldsymbol{\beta}$:

$$
\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{\beta}} \rightarrow \left[\frac{\delta L}{\delta \boldsymbol{\beta}}\right]_i = \frac{\delta L}{\delta \beta_i} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \frac{\delta Y_{sj}}{\delta \beta_i} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \frac{\delta \gamma_j \hat{X}_{sj} + \beta_j}{\delta \beta_i} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \delta_{ij} \\
&= \sum_{s} \frac{\delta L}{\delta Y_{si}} \\
\frac{\delta L}{\delta \boldsymbol{\beta}} = \mathbf{1}^T \left(\frac{\delta L}{\delta \boldsymbol{Y}}\right)
\end{aligned}
$$

Where $\mathbf{1}^T$ is again the ones row-vector ($\mathbf{1}^T \in R^{1\times S}$).

3) the derivative w.r.t. to $\boldsymbol{X}$:

Before we can derive our expression for $\frac{\delta L}{\delta \boldsymbol{X}}$, we first derive some other partial derivatives that we can use in obtaining our final expression:

$$\frac{\delta \mu_s}{\delta X_{ri}}$$
$$= \frac{\delta}{\delta X_{ri}} \frac{1}{M} \sum_{m=1}^{M} (X_{sm} - \mu_s)$$
$$= \frac{1}{M} \sum_{m=1}^{M} \frac{\delta}{\delta X_{ri}} (X_{sm} - \mu_s)$$
$$= \frac{1}{M} \delta_{sr}$$

$$\frac{\delta \sigma_s}{\delta X_{ri}}$$
$$= \frac{\delta}{\delta X_{ri}} \frac{1}{M} \sum_{m=1}^{M} (X_{sm} - \mu_s)^2$$
$$= \frac{1}{M} \sum_{m=1}^{M} \frac{\delta}{\delta X_{ri}} (X_{sm} - \mu_s)^2$$
$$= \frac{1}{M} 2\delta_{sr}(X_{si} - \mu_s)$$

Using $\frac{\delta \mu_s}{\delta X_{ri}}$ and $\frac{\delta \sigma_s}{\delta X_{ri}}$ we can derive $\frac{\delta \hat{X}_{sj}}{\delta X_{ri}}$:

$$\frac{\delta \hat{X}_{sj}}{\delta X_{ri}}$$
$$= \frac{\delta}{\delta X_{ri}} \frac{X_{sj} - \mu_s}{\sqrt{\sigma_s^2 + \epsilon}}$$

Using the product rule:

$$= \frac{1}{\sqrt{\sigma_s^2 + \epsilon}} \cdot \frac{\delta}{\delta X_{ri}} (X_{sj} - \mu_s) + (X_{sj} - \mu_s) \cdot \frac{\delta}{\delta X_{ri}} \frac{1}{\sqrt{\sigma_s^2 + \epsilon}}$$
$$= \frac{1}{\sqrt{\sigma_s^2 + \epsilon}} (\delta_{sr}\delta_{ij} - \frac{1}{M}\delta_{sr}) + (X_{sj} - \mu_s) \cdot - \frac{\frac{1}{M} 2\delta_{sr}(X_{si} - \mu_s)}{2(\sigma_s^2 + \epsilon)\sqrt{\sigma_s^2 + \epsilon}}$$
$$= \frac{\delta_{sr}\delta_{ij} - \frac{1}{M}\delta_{sr}}{\sqrt{\sigma_s^2 + \epsilon}} - \frac{(X_{sj} - \mu_s) \cdot \frac{1}{M}\delta_{sr}(X_{si} - \mu_s)}{(\sigma_s^2 + \epsilon)\sqrt{\sigma_s^2 + \epsilon}}$$
$$= \frac{\delta_{sr}}{\sqrt{\sigma_s^2 + \epsilon}} \Big(\delta_{ij} - \frac{1}{M} - \frac{\frac{1}{M}(X_{sj} - \mu_s)(X_{si} - \mu_s)}{\sigma_s^2 + \epsilon} \Big)$$
$$= \frac{\delta_{sr}}{\sqrt{\sigma_s^2 + \epsilon}} \Big(\delta_{ij} - \frac{1}{M} - \frac{1}{M} \frac{(X_{sj} - \mu_s)}{\sqrt{\sigma_s^2 + \epsilon}} \frac{(X_{si} - \mu_s)}{\sqrt{\sigma_s^2 + \epsilon}} \Big)$$
$$= \frac{\delta_{sr}}{\sqrt{\sigma_s^2 + \epsilon}} \Big(\delta_{ij} - \frac{1}{M} - \frac{1}{M} \hat{X}_{sj}\hat{X}_{si} \Big)$$

And with this we can derive our final derivative $\frac{\delta L}{\delta \boldsymbol{X}}$:

$$\begin{aligned}
\frac{\delta L}{\delta \boldsymbol{X}} \rightarrow \left[\tfrac{\delta L}{\delta X}\right]_{ri} = \frac{\delta L}{\delta X_{ri}} &= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \frac{\delta Y_{sj}}{X_{ri}} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \frac{\delta Y_{sj}}{\delta \hat{X}_{sj}} \frac{\delta \hat{X}_{sj}}{X_{ri}} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \gamma_j \frac{\delta \hat{X}_{sj}}{X_{ri}} \\
&= \sum_{s,j} \frac{\delta L}{\delta Y_{sj}} \gamma_j \left(\frac{\delta_{sr}}{\sqrt{\sigma_s^2 + \epsilon}} \Big(\delta_{ij} - \frac{1}{M} - \frac{1}{M} \hat{X}_{sj} \hat{X}_{si} \Big) \right) \\
&= \sum_{j=1}^{M} \frac{\delta L}{\delta Y_{rj}} \gamma_j \left(\frac{1}{\sqrt{\sigma_r^2 + \epsilon}} \Big(\delta_{ij} - \frac{1}{M} - \frac{1}{M} \hat{X}_{rj} \hat{X}_{ri} \Big) \right) \\
&= \frac{1}{\sqrt{\sigma_r^2 + \epsilon}} \Big(\sum_{j=1}^{M} \delta_{ij} \frac{\delta L}{\delta Y_{rj}} \gamma_j - \frac{1}{M} \sum_{j=1}^{M} \frac{\delta L}{\delta Y_{rj}} \gamma_j - \frac{1}{M} \hat{X}_{ri} \sum_{j=1}^{M} \hat{X}_{rj} \frac{\delta L}{\delta Y_{rj}} \gamma_j \Big) \\
&= \frac{1}{\sqrt{\sigma_r^2 + \epsilon}} \Big(\frac{\delta L}{\delta Y_{ri}} \gamma_i - \frac{1}{M} \sum_{j=1}^{M} \frac{\delta L}{\delta Y_{rj}} \gamma_j - \frac{1}{M} \hat{X}_{ri} \sum_{j=1}^{M} \hat{X}_{rj} \frac{\delta L}{\delta Y_{rj}} \gamma_j \Big)
\end{aligned}$$

And we can simplify the expression a bit further by noting that $\frac{\delta L}{\delta Y_{ri}} \gamma_i = \frac{\delta L}{\delta \hat{X}_{ri}}$ and $\frac{\delta L}{\delta Y_{rj}} \gamma_j = \frac{\delta L}{\delta \hat{X}_{rj}}$:

$$= \frac{1}{\sqrt{\sigma_r^2 + \epsilon}} \Big(\frac{\delta L}{\delta \hat{X}_{ri}} - \frac{1}{M} \sum_{j=1}^{M} \frac{\delta L}{\delta \hat{X}_{rj}} - \frac{1}{M} \hat{X}_{ri} \sum_{j=1}^{M} \hat{X}_{rj} \frac{\delta L}{\delta \hat{X}_{rj}} \Big)$$

Question 3.2b)

-

Question 3.2c)

-

Question 3.2d)

Both Batch Normalization and Layer Normalization address the problem of Covariate Shift within networks. Covariate Shift can be defined as the change in the distribution of network activations due to the change in network parameters during training [1]. This change of distribution of each layer's inputs during training complicate training of Deep Neural Networks by requiring lower learning rates and careful parameter initialization. Moreover, training can be improved by reducing the Covariate Shift within networks [1].

Now, Batch normalization works by computing statistics across the batch (i.e. the batch mean and variance), so it normalizes inputs across the batch dimension. This way, Batch Normalization decreases the Covariate Shift, thereby allowing usage of higher learning rates and being less careful about initialization. It also introduces regularization in the network, which can eliminate the need for other regularization techniques such as dropout [1]. However, the essence of Batch Normalization lies in the fact that we normalize inputs for each mini-batch, which in turn means that the estimated mean and variance that are used to normalize the inputs will differ for each mini-batch. This has its limitations. The main limitation is that the normalization process is batch size dependent. Moreover, Batch Normalization is sensitive to different batch sizes and will not work well when batch sizes become too small (so taking a more purely stochastic approach is not possible) because then the batch mean and variance are not longer good estimates for the dataset mean and variance. Because small batch sizes cannot be used in Batch Normalization, it is generally also not a technique that can be used in online learning settings, i.e. real-time settings in which batch sizes must be small (or the batch size is one). An other limitation of Batch Normalization is that it is difficult to apply to recurrent connections in RNNs [2].

To overcome these limitations of Batch normalization, Layer Normalization was introduced [2]. The essence of Layer Normalization is that inputs are not normalized across the batch dimension but across the input features (so in a sense, it is the transposed operation of Batch Normalization). Because in Layer Normalization the statistics (i.e. mean and variance) are calculated over the feature dimension, the normalization process itself is batch size independent. So in contrast to Batch Normalization, there is no dependence between input samples. This means that in Layer Normalization we can use arbitrary batch sizes (also small ones), whereas with Batch Normalization there is a lower bound of the batch size (we cannot use small ones). Moreover, Layer Normalization has been experimentally proven to work well when applied to RNNs. More specifically, it proved to improve both training time and generalization performance of several existing RNN models [2]. An obvious disadvantage of using Layer Normalization however, is that it will not work well when the number of features per input is small.

So put together, when we compare the effect of different batch sizes on both normalization techniques, we can conclude that Batch Normalization is highly batch size dependent, where it will not work well on small batch sizes, whereas Layer Normalization is batch size independent, meaning performance will not be affected by different batch sizes.

4 PyTorch CNN

Question 4a)

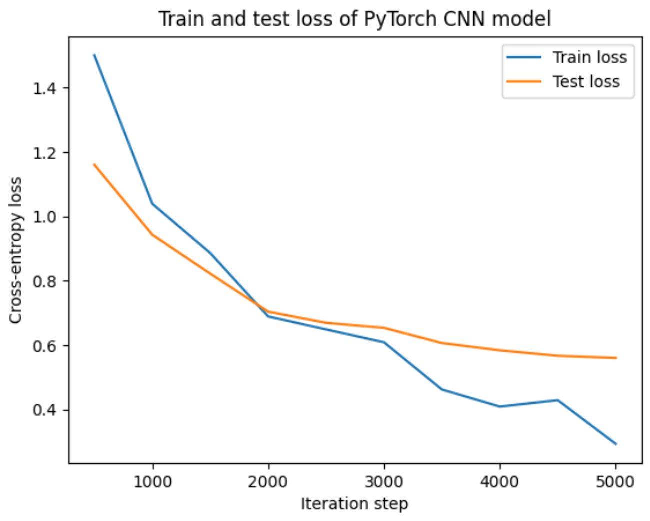
Using the default parameters when running the CNN using the pre-activation ResNet blocks, a test accuracy of 0.812 was obtained (*batch size=32, learning rate=1e-4, optimizer=Adam, max steps=5000*). The loss and accuracy curves corresponding to the training process are displayed in figure 3. Once again, the plots illustrate the learning process of the CNN model: the train and test loss decreases over the number of iterations, and the train and test accuracy increases over the number of iterations.

Question 4b) Transfer Learning [1]

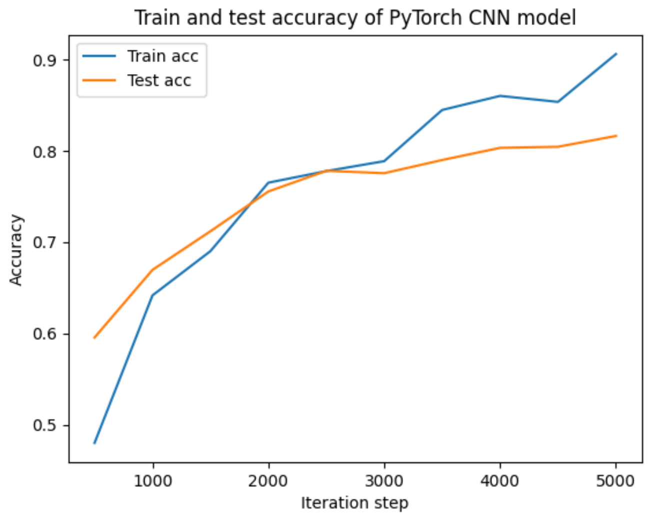
For this task, I used the Torchvision model library to load the pretrained 'DenseNet121' model [3]. The proposed technique of DenseNets essentially boils down to the idea a network in which each layer is connected to every other layer in a feed-forward fashion. When introduced in 2017, its creators demonstrated that usage of DenseNet helps to counteract the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters [3].

After I loaded the pretrained DenseNet model, I chose to retrain/fine-tune every layer of the network, instead of freezing some layers and e.g. only fine-tune the final layer (in early testing fine-tuning the whole network proved to be computational feasible). After trying some different setups of hyperparameters, I obtained a test accuracy of 0.854 using the same hyperparameters as in Question 4a (*Batch size=32, learning rate=1e-4, optimizer=Adam, max steps=5000*). So the DenseNet model outperformed our simpler CNN model by roughly 4% accuracy. The corresponding loss and accuracy plots are shown in figure 4. The plots illustrate the same learning process as all of the previous plots. However, it is noticeable that the starting values of the train/test loss and accuracy of the pretrained DenseNet model are lower and higher, respectively, when compared to the CNN model in question 4.a) that was naturally not initialized using pretrained weights. This already shows the capacity of a transfer learning approach to obtain better performance earlier on in the training process when compared to e.g. random initialization.

[1] I implemented this question in the file transfer_learning.py. You can run this specific setup by running: transfer_learning.py –model='DenseNet'

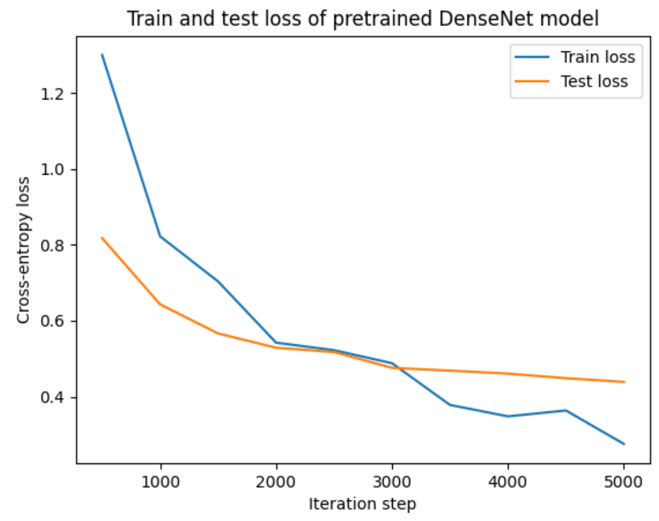


(a) Train and test loss evaluated throughout training.

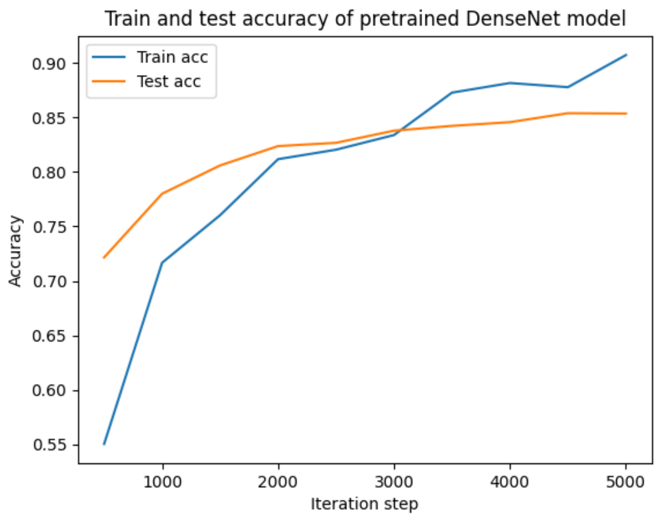


(b) Train and test accuracy evaluated throughout training.

Figure 3: Loss **(a)** and accuracy **(b)** plots throughout the training process of the PyTorch CNN model using pre-activation ResNet blocks, with evaluation frequency set at every 500th step.



(a) Train and test loss evaluated throughout training.



(b) Train and test accuracy evaluated throughout training.

Figure 4: Loss **(a)** and accuracy **(b)** plots throughout the training process of the pretrained DenseNet 121 model, with evaluation frequency set at every 500th step.

References

[1] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.