Deep Learning Assignment 1

Nils Lehmann
12868175
`nils.lehmann@student.uva.nl`

1 MLP Backprop and NumPy Implementation

1.1 Evaluating the Gradients

1.1.a Linear Module

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{W}}\right]_{ij} &= \frac{\partial L}{\partial W_{ij}} = \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial W_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial}{\partial W_{ij}} \sum_{p} X_{mp} W^T_{pn} + B_{mn} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial}{\partial W_{ij}} \sum_{p} X_{mp} W_{np} + B_{mn} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \sum_{p} X_{mp} \frac{\partial W_{np}}{\partial W_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \sum_{p} X_{mp} \delta_{ni} \delta_{pj} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} X_{mj} \delta_{ni} \delta_{jj} \\
&= \sum_{m} \frac{\partial L}{\partial Y_{mi}} X_{mj} \delta_{ii} \delta_{jj} \\
&= \sum_{m} \left[\frac{\partial L}{\partial Y}\right]_{mi} X_{mj} \\
&= \sum_{m} \left[\frac{\partial L}{\partial Y}\right]^T_{im} X_{mj} \\
&= \left[\left[\frac{\partial L}{\partial \mathbf{Y}}\right]^T \mathbf{X}\right]_{ij}
\end{aligned}
$$

$$\begin{aligned}
\left[\frac{\partial L}{\partial \boldsymbol{b}}\right]_i &= \frac{\partial L}{\partial b_i} = \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial}{\partial b_i} \sum_p X_{mp} W^T_{pn} + B_{mn} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial B_{mn}}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \delta_{ni} \\
&= \sum_{m} \frac{\partial L}{\partial Y_{mi}} \\
&= \left[1^T \frac{\partial L}{\partial \boldsymbol{Y}}\right]_i
\end{aligned}$$

$$\begin{aligned}
\left[\frac{\partial L}{\partial \boldsymbol{X}}\right]_{ij} &= \frac{\partial L}{\partial X_{ij}} = \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial}{\partial X_{ij}} \sum_p X_{mp} W^T_{pn} + B_{mn} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \sum_p \frac{\partial X_{mp}}{\partial X_{ij}} W_{np} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \sum_p \delta_{mi} \delta_{pj} W_{np} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \delta_{mi} \delta_{jj} W_{nj} \\
&= \sum_{n} \frac{\partial L}{\partial Y_{in}} \delta_{ii} \delta_{jj} W_{nj} \\
&= \sum_{n} \frac{\partial L}{\partial Y_{in}} W_{nj} \\
&= \sum_{n} \left[\frac{\partial L}{\partial Y}\right]_{in} W_{nj} = \left[\frac{\partial L}{\partial \boldsymbol{Y}} \boldsymbol{W}\right]_{ij}
\end{aligned}$$

1.1.b Activation Module

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \boldsymbol{X}}\right]_{ij} &= \frac{\partial L}{\partial X_{ij}} = \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} = \frac{\partial L}{\partial Y_{mn}} \frac{\partial}{\partial X_{ij}} h\left(X_{mn}\right) \\
&= \sum_{m,n} = \frac{\partial L}{\partial Y_{mn}} h'\left(X_{mn}\right) \frac{\partial X_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} = \frac{\partial L}{\partial Y_{mn}} h'\left(X_{mn}\right) \delta_{mi} \delta_{nj} \\
&= \frac{\partial L}{\partial Y_{ij}} h'\left(X_{ij}\right) \\
&= \left[\frac{\partial L}{\partial \boldsymbol{Y}} \circ h'\left(\boldsymbol{X}\right)\right]_{ij}
\end{aligned}
$$

1.1.c Softmax

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \boldsymbol{X}}\right]_{ij} &= \frac{\partial L}{\partial X_{ij}} = \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial}{\partial X_{ij}} \frac{\exp X_{mn}}{\sum_k \exp X_{mk}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \left(\exp X_{mn} \frac{\partial X_{mn}}{\partial X_{ij}} \left(\sum_k \exp X_{mk} \right)^{-1} + \exp\left(X_{mn}\right)(-1) \left(\sum_k \exp X_{mk} \right)^{-2} \left(\sum_k \frac{\partial}{\partial X_{ij}} \exp X_{mk} \right) \right. \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \left(\delta_{mi}\delta_{nj} \frac{\exp X_{mn}}{\sum_k \exp X_{mk}} - \frac{\exp X_{mn}}{\left(\sum_k \exp X_{mk}\right)^2} \sum_k \exp X_{mk} \frac{\partial X_{mk}}{\partial X_{ij}} \right) \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \left(\delta_{mi}\delta_{nj} \frac{\exp X_{mn}}{\sum_k \exp X_{mk}} - \frac{\exp X_{mn}}{\left(\sum_k \exp X_{mk}\right)^2} \sum_k \exp\left(X_{mk}\right) \delta_{mi}\delta_{kj} \right) \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \left(\delta_{mi}\delta_{nj} \frac{\exp X_{mn}}{\sum_k \exp X_{mk}} - \frac{\exp X_{mn}}{\left(\sum_k \exp X_{mk}\right)^2} \exp\left(X_{mj}\right) \delta_{mi}\delta_{jj} \right) \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \delta_{mi}\delta_{nj} \frac{\exp X_{mn}}{\sum_k \exp X_{mk}} - \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\exp X_{mn}}{\left(\sum_k \exp X_{mk}\right)^2} \exp\left(X_{mj}\right) \delta_{mi}\delta_{jj} \\
&= \sum_{n} \frac{\partial L}{\partial Y_{in}} \delta_{ii}\delta_{nj} \frac{\exp X_{in}}{\sum_k \exp X_{ik}} - \sum_{n} \frac{\partial L}{\partial Y_{in}} \frac{\exp X_{in}}{\left(\sum_k \exp X_{ik}\right)^2} \exp\left(X_{ij}\right) \delta_{ii} \\
&= \frac{\partial L}{\partial Y_{ij}} \delta_{jj} \frac{\exp X_{ij}}{\sum_k \exp X_{ik}} - \sum_{n} \frac{\partial L}{\partial Y_{in}} \frac{\exp X_{in}}{\left(\sum_k \exp X_{ik}\right)^2} \exp\left(X_{ij}\right) \\
&= \frac{\partial L}{\partial Y_{ij}} \delta_{jj} \frac{\exp X_{ij}}{\sum_k \exp X_{ik}} - \sum_{n} \frac{\partial L}{\partial Y_{in}} \frac{\exp X_{in}}{\sum_k \exp X_{ik}} \frac{\exp X_{ij}}{\sum_k \exp X_{ik}} \\
&= \frac{\partial L}{\partial Y_{ij}} Y_{ij} - \sum_{n} \frac{\partial L}{\partial Y_{in}} Y_{in} Y_{ij} = \left(\frac{\partial L}{\partial \boldsymbol{Y}} \boldsymbol{Y} \right)_{ij} - \sum_{n} \left(\frac{\partial L}{\partial \boldsymbol{Y}} \boldsymbol{Y} \right)_{in} Y_{ij}
\end{aligned}
$$

1.1.d Loss Module

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \boldsymbol{X}}\right]_{ij} &= \frac{\partial L}{\partial X_{ij}} \\
&= \frac{\partial}{\partial X_{ij}}\left(-\frac{1}{S}\sum_{pq} T_{pq}\log(X_{pq})\right) \\
&= -\frac{1}{S}\sum_{pq} T_{pq}\frac{\partial}{\partial X_{ij}}\log(X_{pq}) \\
&= -\frac{1}{S}\sum_{pq} T_{pq}\frac{1}{X_{ij}}\frac{\partial X_{pq}}{\partial X_{ij}} \\
&= -\frac{1}{S}\sum_{pq} T_{pq}\frac{1}{X_{ij}}\delta_{pi}\delta_{qj} \\
&= -\frac{1}{S}\sum_{p} T_{pj}\frac{1}{X_{ij}}\delta_{pi}\delta_{jj} \\
&= -\frac{1}{S}T_{ij}\frac{1}{X_{ij}}\delta_{ii} \\
&= -\frac{1}{S}\left[\boldsymbol{T}\circ\boldsymbol{X'}\right]_{ij}\text{, where X' hold the reciprocal of each element of X.}
\end{aligned}
$$

1.2 NumPy Implementation

The NumPy implementation can be found in modules.py, mlp_numpy.py, and train_mlp_numpy.py. The training loss and accuracy were calculated in a "running" manner as done in the pytorch tutorial Chilamkurthy. At the specified evaluation frequency step, loss and accuracy are calculated over the entire testset. The results can be seen in Figure 1. Both accuracy and loss improve steadily, while there could probably arise overfitting when training more.

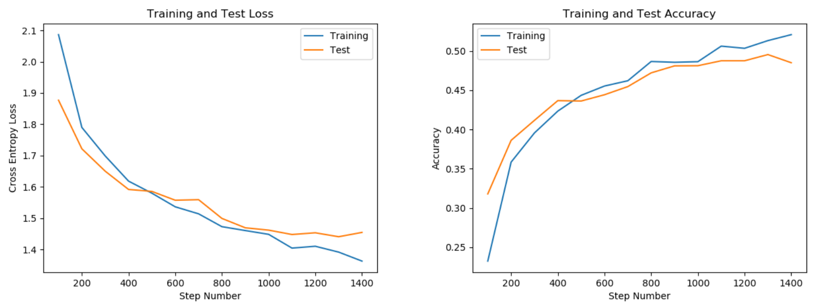


Figure 1: Loss and accuracy results for default parameters with NumPy Implementation

2 PyTorch MLP

2.1 Implementation

The PyTorch Implementation can be found in mlp_pytorch.py and train_mlp_pytorch.py. Both the training and evaluation at specified frequency are done with the same batch size. The results with default parameter settings are shown in Figure 2. Comparing, the results with the NumPy implementation, it can be observed that the accuracy scores of training and test set in the pytorch implementation begin to diverge at around step 800, while they stay fairly close to each other in the NumPy one.

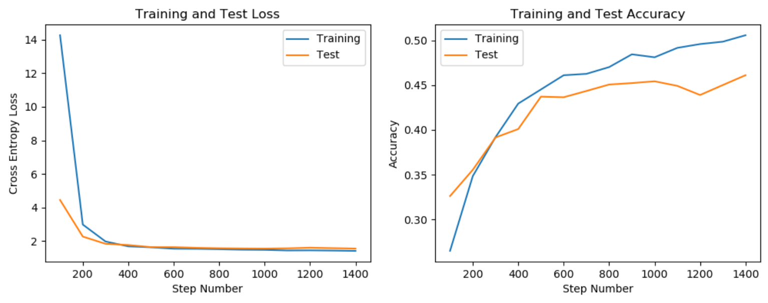


Figure 2: Loss and accuracy results for model with default parameters.

In order to improve accuracy, several steps were taken. Figures that demonstrate the effect of the implemented changes are shown in the appendix. A summary of experiments with corresponding accuracy is given in table 1. First, I changed the optimizer to the Adam optimizer, as it adjusts learning rates adaptively. With respect to activation functions, a ReLU implementation often performed slightly worse than ELU so the activation function was not changed for the remaining experiments. However, that itself did not increase accuracy. The most significant performance boost came from the addition of batch normalization, as accuracy increased six percentage points. In order to push the accuracy above the 52% threshold, the model was trained for 2000 instead of 1400. Here, overfitting noticably increases. As a counter measure to overfitting an experiment with a a deeper but shallower network with five layers and 25 hidden nodes each was run as a form of regularization. This indeed decreases overfitting significantly as Figure 7d shows. However, this architectue decreases the accuracy performance by close to 5%. In contrast, a single layer network with 150 hidden units, yields a higher accuracy score but also overfits much more as Figure 7e demonstrates.

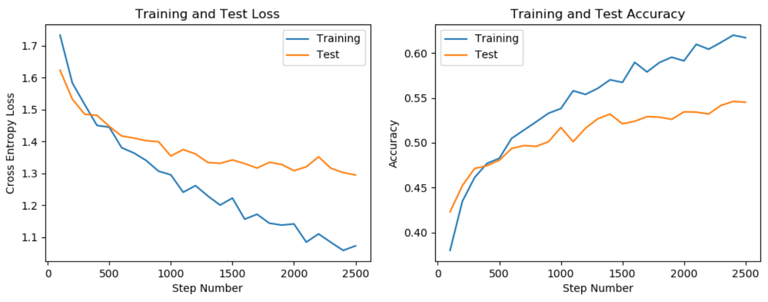


Figure 3: Loss and accuracy results for model with best results.

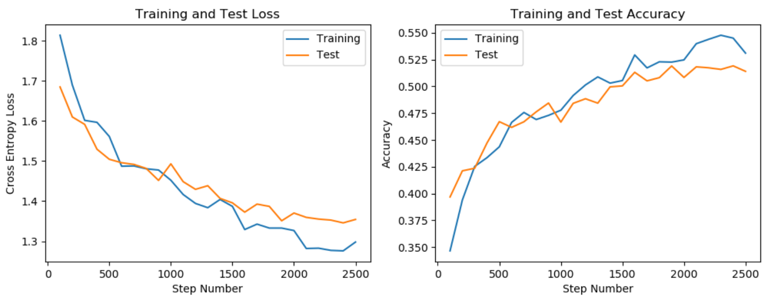


Figure 4: Loss and accuracy results for model with best results but batch size of 100.

Subsequently, a more complex model with 5 layers and 150 hidden units was run for 2500 iterations, which increased the accuracy to 54.5%. However, overfitting also becomes a greater issue. In a last experiment to tackle the overfitting problem, a simple but very effective change was made. By

Optimizer	Activation	Batch Normalization	Hidden Units	Max Steps	Accuracy
SGD	ELU	No	100	1400	46.11
Adam	ELU	No	100	1400	45.89
Adam	ReLU	No	100	1400	45.52
Adam	ELU	Yes	100	1400	51.57
Adam	ELU	Yes	100	2000	**52.25**
Adam	ELU	Yes	25,25,25,25,25	2000	47.85
Adam	ELU	Yes	150	2000	**52.49**
Adam	ELU	Yes	150,150,150,150,150	2500	**54.16**

Table 1: Pytorch MLP experiment results. The learning rate remained at default value for all experiments.

reducing the batch size to 100, instead of 200, the overfitting gap is largly diminished, albeit there is a slight trade off with accuracy as figure 4 shows.

2.2 Discussion Activation Function

The Tanh function has the nice property that it it zero centered. However, it has a significant problem of saturating gradients towards both the positive and negative tail. This means that we would have to be more careful with proper weight initialization, and are more likely to run into gradient issues. In contrast, the ELU function has non-saturating gradients for all positive inputs and even to some extent for negative inputs in proximity to zero. This means we are less likely to encounter vanishing gradients compared to using Tanh. However, in theory ELU also introduces another hyperparamater for the slope that we have to choose. From experiments run with the pytorch MLP, it also seems that using ELU lets the model converge faster than using a Tanh activation function.

3 Custom Module: Layer Normalization

3.1 Layer Normalization Implementation

Implementation can be found in custom_layernorm.py.

3.2 Manual implementation of backward pass

3.2.a

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

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

For the derivative with respect to input X, first compute, $\left[\frac{\partial \mu}{\partial X}\right]_{ri}$, $\left[\frac{\partial \sigma^2}{\partial X}\right]_{ri}$, $\left[\frac{\partial \hat{X}}{\partial X}\right]_{ri}$, and $\left[\frac{\partial Y}{\partial X}\right]_{ri}$ in order to subsequently compute $\left[\frac{\partial L}{\partial X}\right]_{ri}$.

$$
\begin{aligned}
\left[\frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{X}}\right]_{ri} &= \frac{\partial \mu_s}{\partial X_{ri}} \\
&= \frac{\partial}{\partial X_{ri}} \frac{1}{M} \sum_{j}^{M} X_{sj} \\
&= \frac{1}{M} \sum_{i}^{M} \delta_{rs} \delta_{ij} \\
&= \frac{1}{M} \delta_{rs}
\end{aligned}
$$

$$
\begin{aligned}
\left[\frac{\partial \boldsymbol{\sigma^2}}{\partial \boldsymbol{X}}\right]_{ri} &= \frac{\partial \sigma_s^2}{\partial X_{ri}} \\
&= \frac{\partial}{\partial X_{ri}} \frac{1}{M} \sum_j^M (X_{sj} - \mu_s)^2 \\
&= \frac{1}{M} \sum_j^M 2(X_{sj} - \mu_s)(\frac{\partial X_{sj}}{\partial X_{ri}} - \frac{\partial \mu_s}{\partial X_{ri}}) \\
&= \frac{1}{M} \sum_j^M 2(X_{sj} - \mu_s)(\delta_{sr}\delta_{ji} - \frac{1}{M}\delta_{sr}) \\
&= \frac{2}{M} \left(\sum_j^M (X_{sj} - \mu_s)\delta_{sr}\delta_{ji} - \frac{1}{M} \sum_j^M \delta_{sr}(X_{sj} - \mu_s) \right) \\
&= \frac{2}{M} \left(\sum_j^M (X_{sj} - \mu_s)\delta_{sr}\delta_{ji} - \delta_{sr} \frac{1}{M} \sum_j^M X_{sj} - \mu_s \right) \\
&= \frac{2}{M} \left(\sum_j^M (X_{sj} - \mu_s)\delta_{sr}\delta_{ji} - \delta_{sr}(\mu_s - \mu_s) \right) \\
&= \frac{2}{M} \sum_j^M (X_{sj} - \mu_s)\delta_{sr}\delta_{ji} \\
&= \frac{2}{M}(X_{si} - \mu_s)\delta_{sr}
\end{aligned}
$$

$$
\begin{aligned}
\left[\frac{\partial \boldsymbol{\hat{X}}}{\partial \boldsymbol{X}}\right]_{ri} &= \frac{\partial \hat{X}_{sj}}{\partial X_{ri}} \\
&= \frac{\partial}{\partial X_{ri}} \frac{X_{sj} - \mu_s}{\sqrt{\sigma_s^2 + \epsilon}} = \frac{\partial}{\partial X_{ri}} (X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-\frac{1}{2}} \\
&= \left(\frac{\partial X_{sj}}{\partial X_{ri}} - \frac{\partial \mu_s}{\partial X_{ri}} \right) (\sigma_s^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{2}(X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-\frac{3}{2}} \frac{\partial \sigma_s^2}{\partial X_{ri}} \\
&= \left(\delta_{sr}\delta_{ji} - \frac{1}{M}\delta_{rs} \right) (\sigma_s^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{2}(X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-\frac{3}{2}} \frac{2}{M}(X_{si} - \mu_s)\delta_{sr} \\
&= \left(\delta_{sr}\delta_{ji} - \frac{1}{M}\delta_{rs} \right) (\sigma_s^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M}(X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-\frac{3}{2}} (X_{si} - \mu_s)\delta_{sr}
\end{aligned}
$$

$$
\begin{aligned}
\left[\frac{\partial Y}{\partial \boldsymbol{X}}\right]_{ri} &= \frac{\partial Y_{sj}}{\partial X_{ri}} \\
&= \frac{\partial}{\partial X_{ri}} \left(\gamma_j \hat{X}_{sj} + \beta_i \right) \\
&= \gamma_j \frac{\partial \hat{X}_{sj}}{\partial X_{ri}}
\end{aligned}
$$

$$
\begin{aligned}
\left[\frac{\partial L}{\partial \boldsymbol{X}}\right]_{ri} &= \frac{\partial L}{\partial X_{ri}} = \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial X_{ri}} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j \frac{\partial \hat{X}_{sj}}{\partial X_{ri}} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j \left(\left(\delta_{sr}\delta_{ji} - \frac{1}{M}\delta_{rs} \right) (\sigma_s^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M}(X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-\frac{3}{2}}(X_{si} - \mu_s)\delta_{sr} \right) \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j (\delta_{sr}\delta_j i - \frac{1}{M}\delta_{rs})(\sigma_s^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M} \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j (X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-\frac{3}{2}}(X_{si} - \mu_s)\delta_{sr} \\
&= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j \delta_{sr}\delta_{ji} (\sigma_s^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M} \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j \delta_{rs} (\sigma_s^2 + \epsilon)^{-\frac{1}{2}} - \\
&\qquad \frac{1}{M} \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \gamma_j (X_{sj} - \mu_s)(\sigma_s^2 + \epsilon)^{-\frac{3}{2}}(X_{si} - \mu_s)\delta_{sr} \\
&= \sum_{s} \frac{\partial L}{\partial Y_{si}} \gamma_i \delta_{sr}\delta_{ii} (\sigma_s^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M} \sum_{j} \frac{\partial L}{\partial Y_{rj}} \gamma_j \delta_{rr} (\sigma_r^2 + \epsilon)^{-\frac{1}{2}} - \\
&\qquad \frac{1}{M} \sum_{j} \frac{\partial L}{\partial Y_{rj}} \gamma_j (X_{rj} - \mu_r)(\sigma_r^2 + \epsilon)^{-\frac{3}{2}}(X_{ri} - \mu_r)\delta_{rr} \\
&= \frac{\partial L}{\partial Y_{ri}} \gamma_i (\sigma_r^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M} \sum_{j} \frac{\partial L}{\partial Y_{rj}} \gamma_j (\sigma_r^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M} \sum_{j} \frac{\partial L}{\partial Y_{rj}} \gamma_j (X_{rj} - \mu_r)(\sigma_r^2 + \epsilon)^{-\frac{3}{2}}(X_{ri} - \mu_r) \\
&= \frac{\partial L}{\partial Y_{ri}} \gamma_i (\sigma_r^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M} \sum_{j} \frac{\partial L}{\partial Y_{rj}} \gamma_j (\sigma_r^2 + \epsilon)^{-\frac{1}{2}} - \frac{1}{M} \sum_{j} \frac{\partial L}{\partial Y_{rj}} \gamma_j (\sigma_r^2 + \epsilon)^{-\frac{3}{2}}(X_{ri} - \mu_r)(X_{rj} - \mu_r) \\
&= \left[\frac{\partial L}{\partial \boldsymbol{Y}} \circ (\boldsymbol{\sigma}^{-\frac{1}{2}} \boldsymbol{\gamma^T})\right]_{ri} - \frac{1}{M} \sum_{j} \left[\frac{\partial L}{\partial \boldsymbol{Y}} \circ (\boldsymbol{\sigma}^{-\frac{1}{2}} \boldsymbol{\gamma^T})\right]_{rj} - \frac{1}{M} \left[\boldsymbol{X} - \boldsymbol{\mu}\right]_{ri} \sum_{j} \left[\frac{\partial L}{\partial \boldsymbol{Y}} \circ (\boldsymbol{\sigma}^{-\frac{3}{2}} \boldsymbol{\gamma^T}) \circ [\boldsymbol{X} - \boldsymbol{\mu}]\right]_{rj}
\end{aligned}
$$

3.2.b Implementation torch.autograd.Function

The implementation can be found in custom_layernorm.py.

3.2.c Module Implementation

The implementation can be found in custom_layernorm.py.

3.2.d Layer and Batch Normalization

An important step in data preprocessing is zero centering the data with unit variance. In neural networks this is also important, as the activation functions are ideally centered around zero. While there is no issue processing the data before training, we would also like to have data of such form inside the network in order to leverage constant behavior of the activation functions. Batch Normalization aims to do exactly that as it computes the mean and variance across a mini-batch and normalizes the entire batch accordingly. These normalizations can be added inside the network and have learnable parameters for shifting and scaling the data. While there are several benefits, there are also limitations, mainly that it requires large mini-batch sizes, because smaller batch sizes will lead to more noisy estimates of the true underlying mean and variance of the data. Layer Normalization aims to omit this limitation by normalizing the data across the features instead. Hence, Layer Normalization is independent of the batch-size. This makes Layer Normalization much more useful for RNNs, whereas the effectiveness is limited for image classification. However, it remains unclear why exactly either of the two normalization methods work. With respect to Batch Normalization two hypothesis are that it either removes covariate shifts of data distributions or simplifies learning by suppressing higher order interactions.

4 Pytorch CNN

4.1 Implementation

The loss and accury curves for the trained model with default parameters can be seen in Figure 5. The final test accuracy reaches 81.9%. After 5000 steps, it seems as if both accuracy and loss could improve by training more. However, after step 3000 and more significantly after step 4500 overfitting occurs.

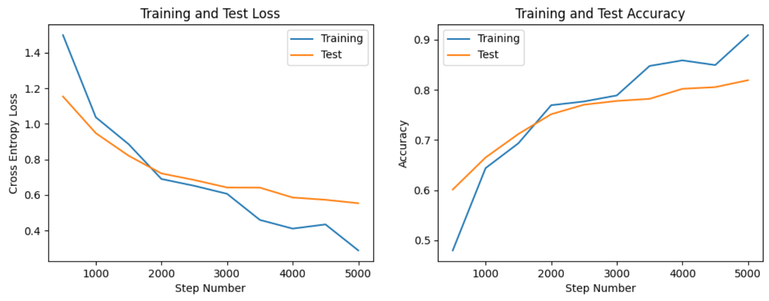


Figure 5: Loss and accuracy results with default parameters CNN implementation

4.2 Transfer Learning

As a comparison to the CNN implementation in the previous section, a ResNet34 architecture was chosen. Instead of defining a CNN model from scratch, the model was downlaoded with pretrained weights from PyTorch and implemented in pytorchTransfer.py . The model was fine-tuned with the same parameter settings as the CNN model: learning rate of 0.001, batch size 32, max steps 5000, evaluation frequency of 500 and an Adam optimizer. The only addition was a learning rate scheduler, that decreased the learning rate by a factor of 10 every 1500 steps. The ResNet implementation then achieves a similar testset accuracy of 81.75%. When comparing the two plots, it is noticeable, that ResNet testset loss and accuracy flatten already after 2000 steps, while it seems as if the default CNN implementation is still on an improving trend after 5000 iterations. Additionally, overfitting also begins to occur at step 2000 but that gap remains fairly constant between 3500 and 5000 steps. One possible reason why the ResNet implementation does not outperform the default CNN implementation, could be that it expects 224x224 pixel images, while the CIFAR10 dataset only includes images that are 32x32 [He et al., 2016]. With respect to the number of trainable parameters, ResNet34 has 21,289,802 parameters, while the default CNN implementation has only about half as many: 11,138,122.

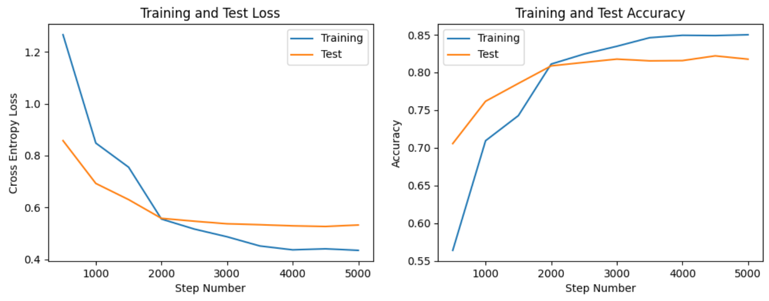


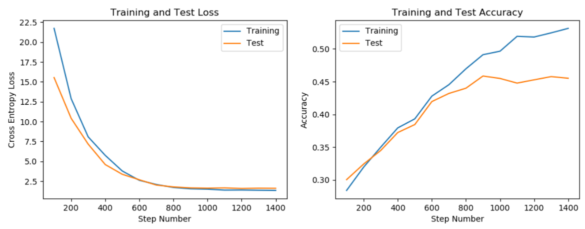
Figure 6: Loss and accuracy results same parameter settings with pretrained ResNet 34 Network

References

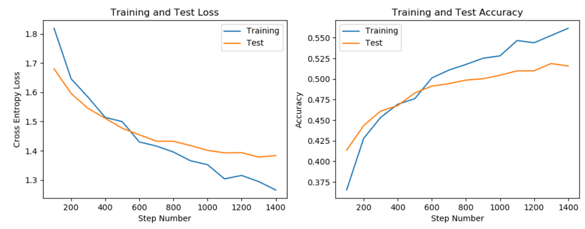
S. Chilamkurthy. URL `https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition.*, 2016.

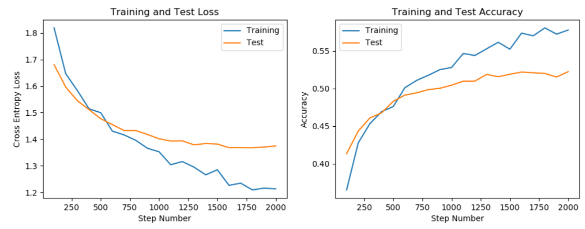
Appendix



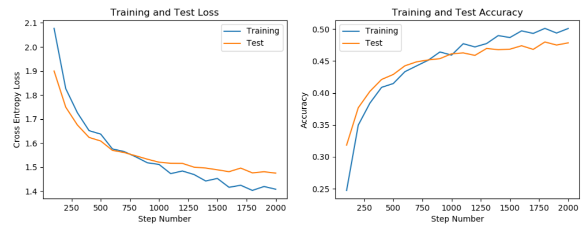
(a) Adam Optimizer with ELU activations, 1400 iterations, default learning rate



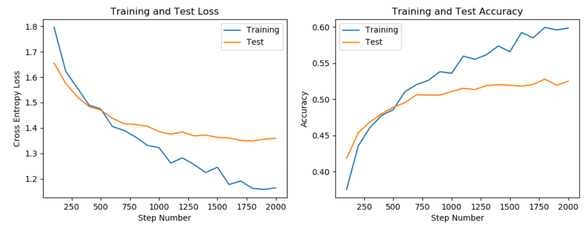
(b) Adam Optimizer with ELU activations, Batch Normalization, 1400 iterations, default learning rate



(c) Adam Optimizer with ELU activations, Batch Normalization, 2000 iterations, default learning rate



(d) Adam Optimizer with ELU activations, Batch Normalization, 2000 iterations, 5 layers with 25 hidden units each, default learning rate



(e) Adam Optimizer with ELU activations, Batch Normalization, 1400 iterations, 150 hidden units, default learning rate