Assignment 1: MLPs, CNNs and Backpropagation

Alexandra Lindt
alex.lindt@protonmail.com

1 MLP Backprop and NumPy Implementation

1.1 Analytical Derivation of Gradients

The following functions will be used in further computations:

$$diagonal(x) = \begin{bmatrix} x_1 & 0 & \dots & 0 \\ 0 & x_2 & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & x_n \end{bmatrix} \quad \text{with } x \in \mathbb{R}^n \text{ and } diagonal(x) \in \mathbb{R}^{n \times n}$$

$$is\_positive(x) = \begin{bmatrix} (x_1 > 0?) \\ (x_2 > 0?) \\ \vdots \\ (x_n > 0?) \end{bmatrix} \quad \text{with } x \in \mathbb{R}^n \text{ and } is\_positive(x) \in \{0,1\}^n$$

$$\text{Kronecker Delta:} \quad \delta_{jk} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

1.1.a

(1)

$$\begin{aligned} \frac{\partial L}{\partial x^{(N)}} &= \frac{\partial}{\partial x^{(N)}}\big(-\sum_i t_i \log(x_i^{(N)})\big) = \frac{\partial}{\partial x^{(N)}}\big(-t^T \log(x^{(N)})\big) = -t^T \frac{\partial \log(x^{(N)})}{\partial x^{(N)}} \\ &= -t^T \begin{bmatrix} \frac{1}{x_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{x_{d_N}} \end{bmatrix} = -t^T diagonal((x^{(N)})^{-1}) \end{aligned}$$

(2)

$$
\begin{aligned}
\frac{\partial x_k^{(N)}}{\partial \widetilde{x}_j^{(N)}} &= \frac{\partial}{\partial \widetilde{x}_j^{(N)}} \frac{\exp(\widetilde{x}_k^{(N)})}{\sum_{i=1}^{d_N} \exp(\widetilde{x}_i^{(N)})} \\
&= \frac{\partial}{\partial \widetilde{x}_j^{(N)}} \left(\exp(\widetilde{x}_k^{(N)}) \frac{1}{\sum_{i=1}^{d_N} \exp(\widetilde{x}_i^{(N)})} \right) \\
&= \frac{\partial \exp(\widetilde{x}_k^{(N)})}{\partial \widetilde{x}_j^{(N)}} \frac{1}{\sum_{i=1}^{d_N} \exp(\widetilde{x}_i^{(N)})} + \exp(\widetilde{x}_k^{(N)}) \frac{\partial}{\partial \widetilde{x}_j^{(N)}} \left(\sum_{i=1}^{d_N} \exp(\widetilde{x}_i^{(N)}) \right)^{-1} \\
&= \delta_{jk} \frac{\exp(\widetilde{x}_k^{(N)})}{\sum_{i=1}^{d_N} \exp(\widetilde{x}_i^{(N)})} - \exp(\widetilde{x}_k^{(N)}) \frac{\exp(\widetilde{x}_j^{(N)})}{\left(\sum_{i=1}^{d_N} \exp(\widetilde{x}_i^{(N)}) \right)^2} \\
&= \delta_{jk} x_k^{(N)} - x_k^{(N)} x_j^{(N)} \\
&= x_k^{(N)} (\delta_{jk} - x_j^{(N)})
\end{aligned}
$$

$$
\begin{aligned}
\implies \quad \frac{\partial x^{(N)}}{\partial \widetilde{x}^{(N)}} &= \begin{bmatrix} \frac{\partial x_1^{(N)}}{\partial \widetilde{x}_1^{(N)}} & \cdots & \frac{\partial x_1^{(N)}}{\partial \widetilde{x}_{d_N}^{(N)}} \\ \vdots & \ddots & \vdots \\ \frac{x_{d_N}^{(N)}}{\widetilde{x}_1^{(N)}} & \cdots & \frac{x_{d_N}^{(N)}}{\widetilde{x}_{d_N}^{(N)}} \end{bmatrix} = \begin{bmatrix} x_1^{(N)}(1 - x_1^{(N)}) & -x_1^{(N)} x_2^{(N)} & \cdots & -x_1^{(N)} x_{d_N}^{(N)} \\ -x_2^{(N)} x_1^{(N)} & x_2^{(N)}(1 - x_2^{(N)}) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ -x_{d_N}^{(N)} x_1^{(N)} & -x_{d_N}^{(N)} x_2^{(N)} & \cdots & x_{d_N}^{(N)}(1 - x_{d_N}^{(N)}) \end{bmatrix} \\
&= diagonal(x^{(N)}) - x^{(N)} (x^{(N)})^T
\end{aligned}
$$

(3)

$$
\frac{\partial x_k^{(l)}}{\partial \widetilde{x}_j^{(l)}} = \frac{\partial ReLu(\widetilde{x}_k^{(l)})}{\partial \widetilde{x}_j^{(l)}} = \frac{\partial max(0, \widetilde{x}_k^{(l)})}{\partial \widetilde{x}_j^{(l)}} = \begin{cases} \delta_{jk} 1 & \textit{if} \quad \widetilde{x}_j^{(l)} > 0 \\ \delta_{jk} 0 & \textit{otherwise} \end{cases}
$$

$$
\implies \quad \frac{\partial x^{(l)}}{\partial \widetilde{x}^{(l)}} = \begin{bmatrix} \frac{\partial x_1^{(l)}}{\partial \widetilde{x}_1^{(l)}} & \cdots & \frac{\partial x_1^{(l)}}{\partial \widetilde{x}_{d_l}^{(l)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_{d_l}^{(l)}}{\partial \widetilde{x}_1^{(l)}} & \cdots & \frac{\partial x_{d_l}^{(l)}}{\partial \widetilde{x}_{d_l}^{(l)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial x_1^{(l)}}{\partial \widetilde{x}_1^{(l)}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial x_{d_l}^{(l)}}{\partial \widetilde{x}_{d_l}^{(l)}} \end{bmatrix} = diagonal(\textit{is\_positive}(x^{(l)}))
$$

(4)

$$
\frac{\partial \widetilde{x}^{(l)}}{\partial x^{(l-1)}} = \frac{\partial}{\partial x^{(l-1)}} W^{(l)} x^{(l-1)} + b^{(l)} = \frac{\partial}{\partial x^{(l-1)}} W^{(l)} x^{(l-1)} = W^{(l)}
$$

(5)

$$
\frac{\partial \widetilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}} = \frac{\partial}{\partial W_{jk}^{(l)}} \left(W_{i,:} \cdot x^{(l-1)} + b_i \right) = \begin{cases} x_k^{(l-1)} & \text{if } \ i = j \\ 0 & \text{otherwise} \end{cases}
$$

$$\implies \frac{\partial \widetilde{x}^{(l)}}{\partial W^{(l)}} = \begin{bmatrix} \begin{bmatrix}\frac{\partial \widetilde{x}^{(l)}_1}{\partial W^{(l)}}\end{bmatrix} \\ \vdots \\ \begin{bmatrix}\frac{\partial \widetilde{x}^{(l)}_{d_l}}{\partial W^{(l)}}\end{bmatrix}\end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \frac{\partial \widetilde{x}^{(l)}_1}{\partial W^{(l)}_{11}} & \cdots & \frac{\partial \widetilde{x}^{(l)}_1}{\partial W^{(l)}_{1d_{l-1}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \widetilde{x}^{(l)}_1}{\partial W^{(l)}_{d_l 1}} & \cdots & \frac{\partial \widetilde{x}^{(l)}_1}{\partial W^{(l)}_{d_l d_{l-1}}} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \frac{\partial \widetilde{x}^{(l)}_{d_l}}{\partial W^{(l)}_{11}} & \cdots & \frac{\partial \widetilde{x}^{(l)}_{d_l}}{\partial W^{(l)}_{1d_{l-1}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \widetilde{x}^{(l)}_{d_l}}{\partial W^{(l)}_{d_l 1}} & \cdots & \frac{\partial \widetilde{x}^{(l)}_{d_l}}{\partial W^{(l)}_{d_l d_{l-1}}} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} x^{(l-1)}_1 & \cdots & x^{(l-1)}_{d_{l-1}} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ x^{(l-1)}_1 & \cdots & x^{(l-1)}_{d_{l-1}} \end{bmatrix} \end{bmatrix}$$

(6)

$$\frac{\partial \widetilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial}{\partial b^{(l)}} W^{(l)} x^{(l-1)} + b^{(l)} = \frac{\partial}{\partial b^{(l)}} b^{(l)} = I_{d_l}$$

1.1.b

(1)

$$\begin{aligned} \frac{\partial L}{\partial \widetilde{x}^{(N)}} &= \frac{\partial L}{\partial x^{(N)}} \frac{\partial x^{(N)}}{\partial \widetilde{x}^{(N)}} \\ &= -t^T \cdot diagonal((x^{(N)})^{-1}) \cdot \Big(diagonal(x^{(N)}) - x^{(N)} (x^{(N)})^T \Big) \\ &= -t^T \cdot \Big(diagonal(\frac{x^{(N)}}{x^{(N)}}) - \mathbb{1}_{d_n} (x^{(N)})^T \Big) \\ &= -t^T \cdot \big(I_{d_N} - \mathbb{1}_{d_n} (x^{(N)})^T \big) \end{aligned}$$

(2)

$$\frac{\partial L}{\partial \widetilde{x}^{(l<N)}} = \frac{\partial L}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \widetilde{x}^{(l)}} = \frac{\partial L}{\partial x^{(l)}} diagonal(\textit{is\_positive}(x^{(l)}))$$

(3)

$$\frac{\partial L}{\partial x^{(l<N)}} = \frac{\partial L}{\partial \widetilde{x}^{(l+1)}} \frac{\partial \widetilde{x}^{(l+1)}}{\partial x^{(l)}} = \frac{\partial L}{\partial \widetilde{x}^{(l+1)}} W^{(l+1)}$$

(4)

$$\begin{aligned}
\frac{\partial L}{\partial W^{(l)}} &= \frac{\partial L}{\partial \widetilde{x}^{(l)}}\frac{\partial \widetilde{x}^{(l)}}{\partial W^{(l)}} \\
&= \begin{bmatrix} \frac{\partial L}{\partial \widetilde{x}_1^{(l)}} & \cdots & \frac{\partial L}{\partial \widetilde{x}_{d_l}^{(l)}} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} x_1^{(l-1)} & \dots & x_{d_{l-1}}^{(l-1)} \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ x_1^{(l-1)} & \dots & x_{d_{l-1}}^{(l-1)} \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial L}{\partial \widetilde{x}_1^{(l)}} \begin{bmatrix} x_1^{(l-1)} & \dots & x_{d_{l-1}}^{(l-1)} \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} + \dots + \frac{\partial L}{\partial \widetilde{x}_{d_l}^{(l)}} \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ x_1^{(l-1)} & \dots & x_{d_{l-1}}^{(l-1)} \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial \widetilde{x}_1^{(l)}} x_1^{(l-1)} & \dots & \frac{\partial L}{\partial \widetilde{x}_1^{(l)}} x_{d_{l-1}}^{(l-1)} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial \widetilde{x}_{d_l}^{(l)}} x_1^{(l-1)} & \dots & \frac{\partial L}{\partial \widetilde{x}_{d_l}^{(l)}} x_{d_{l-1}}^{(l-1)} \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial \widetilde{x}_1^{(l)}} \\ \vdots \\ \frac{\partial L}{\partial \widetilde{x}_{d_l}^{(l)}} \end{bmatrix} \cdot \begin{bmatrix} x_1^{(l-1)} & \dots & x_{d_{l-1}}^{(l-1)} \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial L}{\partial \widetilde{x}^{(l)}}^T \cdot x^{(l-1)T} \end{bmatrix}
\end{aligned}$$

(5)

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \widetilde{x}^{(l)}}\frac{\partial \widetilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial L}{\partial \widetilde{x}^{(l)}} I_{d_l} = \frac{\partial L}{\partial \widetilde{x}^{(l)}}$$

1.1.c

If a batchsize $B > 1$ is used, we calculate the forward and backward pass for B inputs in parralel. Therefore, the backward pass gradients that we calculated above change their shape such that they can represent all gradients of the respective inputs at the same time.
This means concretely that

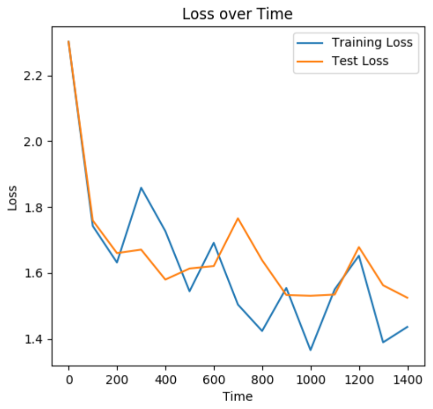
$\frac{\partial L}{\partial \widetilde{x}^{(N)}}$ changes from being $\in \mathbb{R}^{1\times d_N}$ to $\in \mathbb{R}^{B\times d_N}$,

$\frac{\partial L}{\partial \widetilde{x}^{(l<N)}}$, $\frac{\partial L}{\partial x^{(l<N)}}$ and $\frac{\partial L}{\partial b^{(l)}}$ change from being $\in \mathbb{R}^{1\times d_l}$ to $\in \mathbb{R}^{B\times d_l}$ and

$\frac{\partial L}{\partial W^{(l)}}$ changes from being $\mathbb{R}^{1\times d_l \times d_{(l-1)}}$ to $\mathbb{R}^{B\times d_l \times d_{(l-1)}}$.

1.2 NumPy Implementation

With the provided default parameters (i.e. learning rate of 0.002, 1500 timesteps and batch size of 200) the MLP with one hidden layer of 100 neurons achieves an accuracy of 0.466 on the entire test set of Cifar-10 (Krizhevsky and Hinton [2009]). The lowest test loss is 1.525. As expected, the training accuracy is with 0.52 a bit higher than the test accuracy with a respectively lower train loss of 1.365. Figure 1 displays accuracy and loss of the MLP over 1500 timesteps of training.



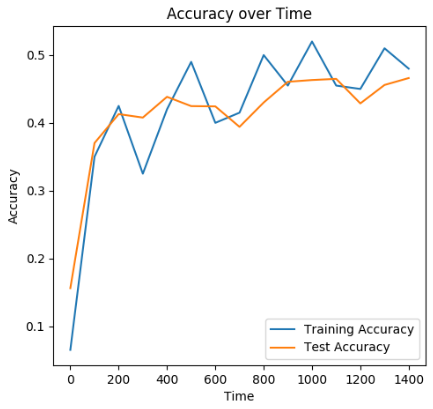


Figure 1: Loss and Accuracy of the MLP with one hidden layer of 100 neurons over 1500 timesteps of training for training and test data. Note that while test loss and accuracy are calculated on the entire test set (10000 samples), training loss and accuracy are calculated for one batch (i.e. 200 samples) and therefore potentially more noisy.

Hidden Units	**Number of Steps**	**Batch Size**	**Accuracy on Test Set**
[256, 256, 256]	**2000**	**200**	0.461
		600	0.484
		1000	0.494
	5000	**200**	0.4540
		600	0.496
		1000	0.498
[512, 256, 128, 64]	**2000**	**200**	0.487
		600	0.487
		1000	0.503
	5000	**200**	0.505
		600	0.512
		1000	0.508
[1024, 512, 256, 128]	**2000**	**200**	0.505
		600	0.521
		1000	0.477
	5000	**200**	0.521
		600	0.527
		1000	0.527

Table 1: MLP accuracy on entire test set for different hyperparameter settings.

2 PyTorch MLP

With the given default parameters, the Pytorch MLP achieves an accuracy of 0.387 when trained with the default Pytorch stochastic gradient optimizer `torch.optim.sgd` on Cifar-10 (Krizhevsky and Hinton [2009]). This is clearly lower than the accuracy that the Numpy MLP achieved with the default parameters. In order to increase the performance of the MLP, it is trained with different hyperparameter settings. A grid-search with the following hyperparameters is performed:

Learning Rate = 0.00002

Hidden Units $\in \{[256, 256, 256], [512, 256, 128, 64], [1024, 512, 256, 128]\}$

Batch Size $\in \{200, 600, 1000\}$

Number of Steps $\in \{2000, 5000\}$

This results in $3 \times 3 \times 1 = 18$ different settings in total. The accuracy of the models on the entire test set is displayed in table 1. I decided to set the learning rate to $2e - 05$ as it gave the best performance for several runs when using the Adam optimizer (Kingma and Ba [2014]) of Pytorch, `torch.optim.adam`, with the default parameters. Loss and accuracy of the model over the training steps are displayed in figure 2. As visualized in table 1, the best accuracy of 0.527 was achieved with hidden units $[1024, 512, 256, 128]$, a batch size of 600 and 5000 training steps. It is apparent that the MLP generally performes the better the more hidden units it has. This indicates that the performance could even be higher for more hidden units than considered here. A higher batch size than the default 200 also clearly benefits the performance of the model, however a batch size of 600 seems to be high enough for the models with hidden units $[1024, 512, 256, 128]$ and $[512, 256, 128, 64]$. The accuracys displayed in table 1 indicate that the model generally benefits a little when it is trained longer. However, when we look at the graphs in 2, we can see that the test accuracy does not really improve anymore after training step 2000. This is also shown in the performance of the model with hidden units $[1024, 512, 256, 128]$, a batch size of 600 and 2000 training steps displayed in table 1. For these settings a similar test accuracy of 0.521 is achieved. Another thing that can be observed is that although the test accuracy does stay at approximately the same level after 2000 steps, the train accuracy keeps on increasing up to step 5000. Although the model keeps adjusting to the training data the test accuracy does not decrease. This might indicate that the model is restricted in overfitting (i.e. might not get much better than 0.8 in training accuracy) and always has to generalize to some extend because of its small size.

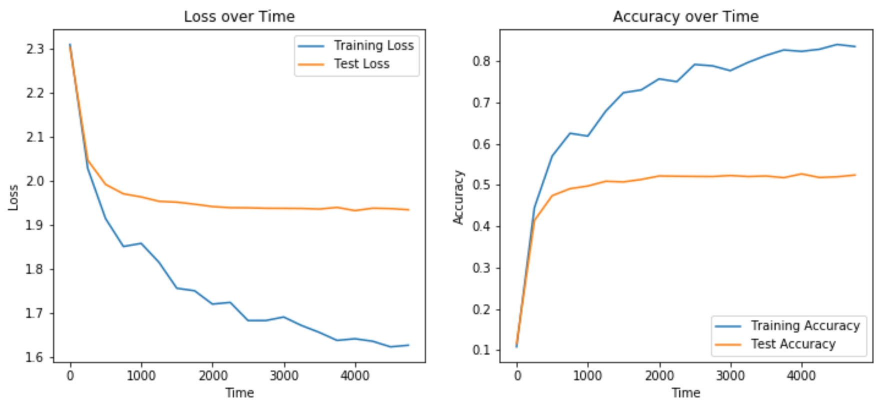


Figure 2: Loss and Accuracy of the MLP with hidden units $[1024, 512, 256, 128]$ and batch size 600 over 5000 timesteps of training for training and test data. Note that while test loss and accuracy are calculated on the entire test set (10000 samples), training loss and accuracy are calculated for one batch (i.e. 600 samples) and therefore potentially more noisy.

3 Custom Module: Batch Normalization

3.1 Automatic Differentiation

Please see the code for a detailed implementation of the batch normalization forward pass using the given formulas.

3.2 Manual Implementation of Backward Pass

3.2.a

(1)

$$\left(\frac{\partial L}{\partial \gamma}\right)_j = \frac{\partial L}{\partial \gamma_j} = \sum_{s=1}^{B}\sum_{i=1}^{C} \frac{\partial L}{\partial y_i^{(s)}} \frac{\partial y_i^{(s)}}{\partial \gamma_j} = \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \frac{\partial y_j^{(s)}}{\partial \gamma_j} = \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \hat{x}_j^{(s)} = \frac{\partial L}{\partial y_j} \cdot \hat{x}_j$$

(2)

$$\left(\frac{\partial L}{\partial \beta}\right)_j = \frac{\partial L}{\partial \beta_j} = \sum_{s=1}^{B}\sum_{i=1}^{C} \frac{\partial L}{\partial y_i^{(s)}} \frac{\partial y_i^{(s)}}{\partial \beta_j} = \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \frac{\partial y_j^{(s)}}{\partial \beta_j} = \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}}$$

(3)

$$\begin{aligned}
\left(\frac{\partial L}{\partial x}\right)_j^r &= \frac{\partial L}{\partial x_j} = \sum_{s=1}^{B}\sum_{i=1}^{C} \frac{\partial L}{\partial y_i^{(s)}} \frac{\partial y_i^{(s)}}{\partial x_j^{(r)}} = \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \frac{\partial y_j^{(s)}}{\partial x_j^{(r)}} \\
&= \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \gamma_j \left(\frac{\partial}{\partial x_j^{(r)}} \frac{x_j^{(s)} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \right) \\
&= \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \gamma_j \left(\frac{\frac{\partial}{\partial x_j^{(r)}}(x_j^{(s)} - \mu_j) \cdot (\sqrt{\sigma_j^2 + \epsilon}) - \frac{\partial}{\partial x_j^{(r)}}(\sqrt{\sigma_j^2 + \epsilon}) \cdot (x_j^{(s)} - \mu_j)}{(\sigma_j^2 + \epsilon)} \right) \\
&= \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \gamma_j \left(\frac{(\delta_{rs} - \frac{1}{B}) \cdot (\sqrt{\sigma_j^2 + \epsilon}) - 0.5(\sigma_j^2 + \epsilon)^{-0.5} \cdot \frac{2}{B}(x_j^{(r)} - \mu_j) \cdot (x_j^{(s)} - \mu_j)}{(\sigma_j^2 + \epsilon)} \right) \\
&= \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \gamma_j \left(\frac{(\delta_{rs} - \frac{1}{B}) - \frac{1}{B} \frac{(x_j^{(r)} - \mu_j)}{\sqrt{\sigma_j^2 + \epsilon}} \frac{(x_j^{(s)} - \mu_j)}{\sqrt{\sigma_j^2 + \epsilon}}}{(\sqrt{\sigma_j^2 + \epsilon})} \right) \\
&= \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \gamma_j \left(\frac{B\delta_{rs} - 1 - \hat{x}_j^{(r)} \hat{x}_j^{(s)}}{B(\sqrt{\sigma_j^2 + \epsilon})} \right) \\
&= \frac{\gamma_j}{B(\sqrt{\sigma_j^2 + \epsilon})} \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \left(B\delta_{rs} - 1 - \hat{x}_j^{(r)} \hat{x}_j^{(s)} \right) \\
&= \frac{\gamma_j}{B(\sqrt{\sigma_j^2 + \epsilon})} \left(\sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} B\delta_{rs} - \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} - \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} \hat{x}_j^{(r)} \hat{x}_j^{(s)} \right) \\
&= \frac{\gamma_j}{B(\sqrt{\sigma_j^2 + \epsilon})} \left(B\frac{\partial L}{\partial y_j^{(r)}} - \sum_{s=1}^{B} \frac{\partial L}{\partial y_j^{(s)}} - \hat{x}_j^{(r)} \frac{\partial L}{\partial y_j} \cdot \hat{x}_j \right)
\end{aligned}$$

Where the following was used:

(3.1)

$$\frac{\partial y_i^{(s)}}{\partial x_i^{(r)}} = \frac{\partial y_i^{(s)}}{\partial \hat{x}_i^{(s)}} \frac{\partial \hat{x}_i^{(s)}}{\partial x_i^{(r)}} = \gamma_i \left(\frac{\partial}{\partial x_i^{(r)}} \frac{x_i^{(s)} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \right)$$

(3.2)

$$\frac{\partial \mu_i}{\partial x_i^{(r)}} = \frac{\partial}{\partial x_i^{(r)}} (\frac{1}{B} \sum_{s=1}^{B} x_i^{(s)}) = \frac{1}{B} \frac{\partial}{\partial x_i^{(r)}} (\sum_{s=1}^{B} x_i^{(s)}) = \frac{1}{B}$$

(3.3)

$$\begin{aligned}\frac{\partial \sigma_i^2}{\partial x_i^{(r)}} &= \frac{\partial}{\partial x_i^{(r)}} (\frac{1}{B} \sum_{s=1}^{B} (x_i^{(s)} - \mu_i)^2) = \frac{1}{B} \sum_{s=1}^{B} \frac{\partial}{\partial x_i^{(r)}} (x_i^{(s)} - \mu_i)^2 \\ &= \frac{1}{B} \sum_{s=1}^{B} 2(x_i^{(s)} - \mu_i) \frac{\partial}{\partial x_i^{(r)}} (x_i^{(s)} - \mu_i) = \frac{2}{B} \sum_{s=1}^{B} (x_i^{(s)} - \mu_i)(\delta_{rs} - \frac{1}{B}) \\ &= \frac{2}{B} \left((x_i^{(r)} - \mu_i) - \sum_{s=1}^{B} \frac{1}{B} (x_i^{(s)} - \mu_i) \right) = \frac{2}{B} (x_i^{(r)} - \mu_i)\end{aligned}$$

3.2.b and 3.2.c

Please see the code for a detailed implementation of the batch normalization backward pass with the gradients derived above.

4 PyTorch CNN

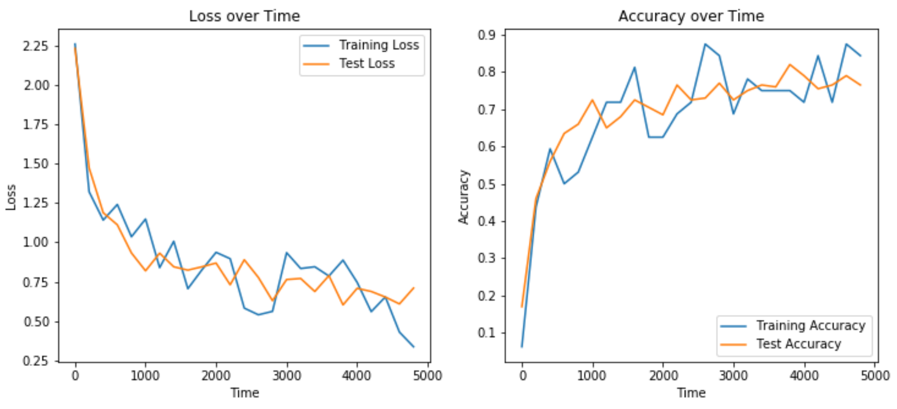


Figure 3: Loss and Accuracy of the trained CNN over 5000 timesteps of training for training and test data. Note that while test loss and accuracy are calculated on 200 samples, training loss and accuracy are calculated for one batch (i.e. 32 samples) and therefore potentially more noisy.

The implemented small version of the VGG-network (Simonyan and Zisserman [2014]) performed as expected. With the default parameters (i.e. learning rate of $1e-4$, batch size of 32, 5000 steps, Adam optimizer (Kingma and Ba [2014])) it achieves a test accuracy of 0.82 and a slightly higher training accuracy of 0.875 on Cifar-10 (Krizhevsky and Hinton [2009]) with respetive lowest training loss of 0.338 and test loss of 0.603. Note that the network was evaluated on only 32 samples (=one batch) for the training accuracy and on 200 samples for the test accuracy, so these numbers might be a little noisy.

References

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.