Assignment 1. MLPs, CNNs and Backpropagation

David Biertimpel
david.biertimpel@student.uva.nl
uva-id:12324418

1 MLP backprop and NumPy implementation

Note: I follow the convention that derivatives are row vectors.

1.1 Analytical derivation of gradients

Question 1.1 a)

$$\frac{\partial \mathcal{L}}{\partial x_i^{(N)}} = -\frac{\partial}{\partial x_i^{(N)}} \sum_i t_i \log x_i^{(N)} = -\frac{t_i}{x_i^{(N)}}$$

$$\implies \frac{\partial \mathcal{L}}{\partial x^{(N)}} = -\frac{t}{x^{(N)}} = -\big[\dots, \frac{t_i}{x_i^{(N)}}, \dots\big] \in \mathbb{R}^{1 \times d_N}$$

$$\begin{aligned}
\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} &= \frac{\partial}{\partial \tilde{x}_j^{(N)}} \left(\frac{\exp(\tilde{x}_i^{(N)})}{\sum_k^{d_N} \exp(\tilde{x}_k^{(N)})} \right) \\
&= \frac{\frac{\partial}{\partial \tilde{x}_j^{(N)}} \big(\exp(\tilde{x}_i^{(N)}) \big) \big(\sum_k^{d_N} \exp(\tilde{x}_k^{(N)}) \big) - \exp(\tilde{x}_i^{(N)}) \frac{\partial}{\partial \tilde{x}_j^{(N)}} \big(\sum_k^{d_N} \exp(\tilde{x}_k^{(N)}) \big)}{\big(\sum_k^{d_N} \exp(\tilde{x}_k^{(N)}) \big)^2} \\
&= \frac{\exp(\tilde{x}_i^{(N)}) \delta_{ij} \big(\sum_k^{d_N} \exp(\tilde{x}_k^{(N)}) \big) - \exp(\tilde{x}_i^{(N)}) \exp(\tilde{x}_j^{(N)})}{\big(\sum_k^{d_N} \exp(\tilde{x}_k^{(N)}) \big)^2} \\
&= \frac{\exp(\tilde{x}_i^{(N)})}{\sum_k^{d_N} \exp(\tilde{x}_k^{(N)})} \frac{\delta_{ij} \sum_k^{d_N} \exp(\tilde{x}_k^{(N)}) - \exp(\tilde{x}_j^{(N)})}{\sum_k^{d_N} \exp(\tilde{x}_k^{(N)})} \\
&= x_i^{(N)} \big(\delta_{ij} - x_j^{(N)} \big) \\
\implies \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} &= \begin{bmatrix}
x_1^{(N)}\big(1 - x_1^{(N)}\big) & -x_1^{(N)} x_2^{(N)} & \dots & \dots & -x_1^{(N)} x_{d_N}^{(N)} \\
-x_2^{(N)} x_1^{(N)} & \ddots & \dots & \dots & \vdots \\
\vdots & \dots & x_i^{(N)}\big(\delta_{ij} - x_j^{(N)}\big) & \dots & \vdots \\
\dots & \dots & \dots & \ddots & \dots \\
-x_{d_N}^{(N)} x_1^{(N)} & \dots & \dots & \dots & x_{d_N}^{(N)}\big(1 - x_{d_N}^{(N)}\big)
\end{bmatrix} \\
&= \mathrm{Diag}\big(x^{(N)}\big) - x^{(N)} x^{(N)T} \in \mathbb{R}^{d_N \times d_N}
\end{aligned}$$

where δ is the kronecker delta.

$$\begin{aligned}
\frac{\partial x_i^{(l<N)}}{\partial \tilde{x}_i^{(l<N)}} &= \frac{\partial}{\partial \tilde{x}_i^{(N)}}\Bigg(\max(0, x_i^{(l<N)}) + a \cdot \min(0, x_i^{(l<N)}) \Bigg) \\
&= \frac{\partial}{\partial \tilde{x}_i^{(N)}}\Bigg(\max(0, x_i^{(l<N)}) \Bigg) + a \cdot \frac{\partial}{\partial \tilde{x}_i^{(N)}}\Bigg(\min(0, x_i^{(l<N)}) \Bigg) \\
&= \begin{cases} 1 & \text{if} \quad \tilde{x}_i^{(N)} > 0 \\ a & \text{elif} \quad \tilde{x}_i^{(N)} < 0 \\ undef. & \text{else } \tilde{x}_i^{(N)} == 0 \end{cases} \\
\implies \frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} &= \text{Diag}\Bigg(\Big[\dots, \begin{cases} 1 & \text{if} \quad \tilde{x}_i^{(N)} > 0 \\ a & \text{elif} \quad \tilde{x}_i^{(N)} < 0 \\ undef. & \text{else } \tilde{x}_i^{(N)} == 0 \end{cases}, \dots \Big] \Bigg) \in \mathbb{R}^{d_l \times d_l}
\end{aligned}$$

Note that we will not populate the Jacobian with *undefined* if $\tilde{x}_i^{(N)} == 0$ but will treat this edge case adequately. In the following Numpy implementation we solve this by considering the derivative to be equal to a if $\tilde{x}_i^{(N)} <= 0$.

$$\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} = \frac{\partial}{\partial x^{(l-1)}}\Bigg(W^{(l)} x^{(l-1)} + b^{(l)} \Bigg) = W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$$

$$\begin{aligned}
\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} &= \frac{\partial}{\partial W^{(l)}}\Bigg(W^{(l)} x^{(l-1)} + b^{(l)} \Bigg) = \frac{\partial}{\partial W^{(l)}}\Bigg(W^l x^{l-1} \Bigg) \\
&= \begin{bmatrix} \frac{\partial \tilde{x}_1^{(l)}}{\partial W^{(l)}} \\ \vdots \\ \frac{\partial \tilde{x}_i^{(l)}}{\partial W^{(l)}} \\ \vdots \\ \frac{\partial \tilde{x}_{d_l}^{(l)}}{\partial W^{(l)}} \end{bmatrix} \in \mathbb{R}^{d_l \times (d_l \times d_{l-1})}
\end{aligned}$$

where,

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W^{(l)}} = \begin{bmatrix} \mathbf{0}^T \\ \vdots \\ x^{(l-1)T} \\ \vdots \\ \mathbf{0}^T \end{bmatrix} \in \mathbb{R}^{1 \times (d_l \times d_{l-1})}$$

since,

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{i,:}^{(l)}} = x^{(l-1)T} \quad \text{and} \quad \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{j \neq i,:}^{(l)}} = 0^T$$

$$
\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial}{\partial b^{(l)}}\left(W^{(l)}x^{(l-1)} + b^{(l)}\right) = \frac{\partial}{\partial b^{(l)}}\left(b^{(l)}\right)
$$
$$
= \text{Diag}\left(\left[1, \dots, 1\right]\right) \in \mathbb{R}^{d_l \times d_l}
$$

Question 1.1 b)

$$
\frac{\partial \mathcal{L}}{\partial \tilde{x}^{(N)}} = \frac{\partial \mathcal{L}}{\partial x^{(N)}} \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} = \frac{\partial \mathcal{L}}{\partial x^{(N)}}\left(\text{Diag}\big(x^{(N)}\big) - x^{(N)}x^{(N)T}\right)
$$

Check dimensions: $(1 \times d_N) \times (d_N \times d_N) = (1 \times d_N)$

$$
\frac{\partial \mathcal{L}}{\partial \tilde{x}^{(l<N)}} = \frac{\partial \mathcal{L}}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} = \frac{\partial \mathcal{L}}{\partial x^{(l)}} \text{Diag}\left(\left[\dots, \begin{cases} 1 & \text{if} \quad \tilde{x}_i^{(N)} > 0 \\ a & \text{elif} \quad \tilde{x}_i^{(N)} < 0 \\ undef. & \text{else } \tilde{x}_i^{(N)} == 0 \end{cases}, \dots\right]\right)
$$

Check dimensions: $(1 \times d_l) \times (d_l \times d_l) = (1 \times d_l)$

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

Check dimensions: $(1 \times d_{l+1}) \times (d_{l+1} \times d_l) = (1 \times d_l)$

$$
\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial \tilde{x}^{(l)}} \begin{bmatrix} \frac{\partial \tilde{x}_1^{(l)}}{\partial W^{(l)}} \\ \vdots \\ \frac{\partial \tilde{x}_i^{(l)}}{\partial W^{(l)}} \\ \vdots \\ \frac{\partial \tilde{x}_{d_l}^{(l)}}{\partial W^{(l)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \tilde{x}_1^{(l)}} \frac{\partial \tilde{x}_1^{(l)}}{\partial W^{(l)}} + & \dots & + \frac{\partial \mathcal{L}}{\partial \tilde{x}_i^{(l)}} \frac{\partial \tilde{x}_i^{(l)}}{\partial W^{(l)}} + & \dots & + \frac{\partial \mathcal{L}}{\partial \tilde{x}_{d_l}^{(l)}} \frac{\partial \tilde{x}_{d_l}^{(l)}}{\partial W^{(l)}} \end{bmatrix}
$$
$$
= \left[\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \tilde{x}_1^{(l)}} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \tilde{x}_i^{(l)}} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \tilde{x}_{d_l}^{(l)}} \end{bmatrix} \begin{bmatrix} x_1^{(l-1)} & \dots & x_i^{(l-1)} & \dots & x_{d_{l-1}}^{(l-1)} \end{bmatrix} \right] = \left[\frac{\partial \mathcal{L}}{\partial \tilde{x}^{(l)}}^T x^{(l-1)T} \right] \qquad \text{(outer product)}
$$

Check dimensions: $(1 \times d_l) \times (d_l \times (d_l \times d_{l-1})) = (d_l \times d_{l-1})$

$$
\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \frac{\partial \mathcal{L}}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial \mathcal{L}}{\partial \tilde{x}^{(l)}} \text{Diag}\left(\left[1, \dots, 1\right]\right) = \frac{\partial \mathcal{L}}{\partial \tilde{x}^{(l)}}
$$

Check dimensions: $(1 \times d_l) \times (d_l \times d_l) = (1 \times d_l)$

Question 1.1 c)

If we use a batchsize of $B > 1$, the above equations will not change fundamentally, except that the dimensionality of the gradients increases by means of the batchsize. This makes sense as we now forward propagate a matrix of examples $X \in \mathbb{R}^{B \times d_0}$ and consequently backpropagate tensors of gradients of size:

$$
\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial x^{(N)}}, \frac{\partial \mathcal{L}}{\partial \tilde{x}^{(N)}} \in \mathbb{R}^{(B \times d_N)} \\
&\frac{\partial \mathcal{L}}{\partial x^{(l)}}, \frac{\partial \mathcal{L}}{\partial \tilde{x}^{(l)}}, \frac{\partial \mathcal{L}}{\partial b^{(l)}} \in \mathbb{R}^{(B \times d_l)} \\
&\frac{\partial \mathcal{L}}{\partial W^{(l)}} \in \mathbb{R}^{(B \times d_l \times d_{d-1})}
\end{aligned}
$$

1.2 NumPy implementation

The source code can be found in the Python files `modules.py`, `mlp_numpy.py` and `train_mlp_numpy.py`. The performance achieved with the default parameters is presented in Figure 1.

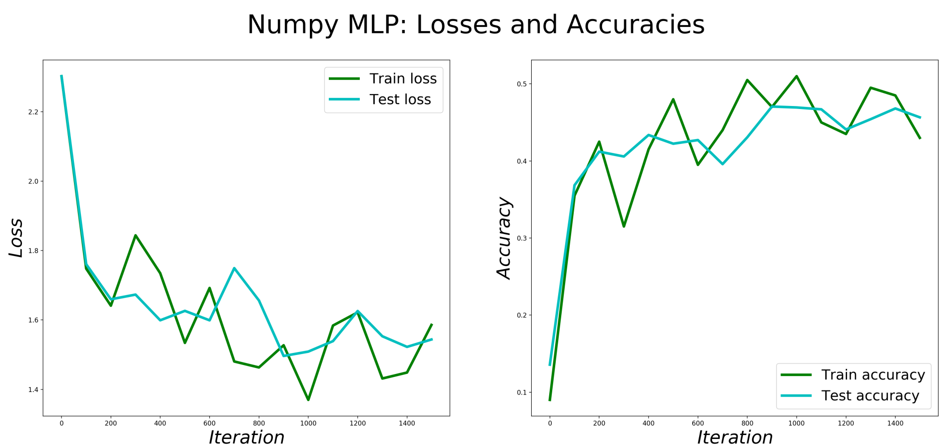

Figure 1: Training and test loss and accuracy during training achieved by the Numpy MLP initialized with the default parameters.

2 PyTorch MLP

The results achieved with the Pytorch MLP across different hyperparameter settings can be observed in Table 1. The training and test loss and accuracy of the best performing model is shown in Figure 2. When looking at Table 1 it is evident that the most significant performance improvements can be attributed to the chosen optimizer. While the classic Stochastic Gradient Descent (SGD) is not able to notably surpass 40% test accuracy, RMSprop and ADAM perform considerably better with the same settings. We also get the impression that three hidden layer $(500, 300, 100)$ perform better than two $(500, 300)$ although this does not hold for the SGD case. We see that ADAM is performing better than RMSprop, however, both are first not capable of achieving the aimed 52% test accuracy. For this we chose to add l_2-regularization (*weight-decay* parameter) to the model using the ADAM optimizer and increase the batchsize from 500 to 750. Lastly, when looking at the accuracy and loss curves in Figure 2, we see that the best performing MLP is clearly overfitting on the training data.

Based on the size of the hyperparameter space, it is obvious that meaningful comparisons are only possible when performing a large-scale grid search. But even with this small sample, we can infer that choosing the right optimizer is crucial to the performance of the MLP.

The corresponding code can be found in the Python files `mlp_pytorch.py` and `train_mlp_pytorch.py`.

Performance of PyTorch MLP

Hidden Units	Batchsize	Learning Rate	Optimizer	Final Train Accuracy	Final Test Accuracy
(*default row:*) 100	200	0.002	SGD	0.405	0.3976
500, 300	500	2e-4	SGD	0.478	0.4033
500, 300, 100	500	2e-4	SGD	0.382	0.3855
500, 300	500	2e-4	RMSprop	0.704	0.4767
500, 300, 100	500	2e-4	RMSprop	0.732	0.4978
500, 300	500	2e-4	ADAM	0.806	0.4959
500, 300, 100	500	2e-4	ADAM	0.812	0.5103
500, 300	750	2e-4 ($wd = 0.02$)	ADAM	0.884	0.5003
500, 300, 100	750	2e-4 ($wd = 0.02$)	ADAM	0.7986	**0.5236**

Table 1: Results achieved with the PyTorch MLP across different hyperparameter settings. The hyperparameters not listed are set to the default values which can be found in the corresponding Python files. The abbreviation wd stands for weight-decay and shows our used parameter for the l_2-regularization.

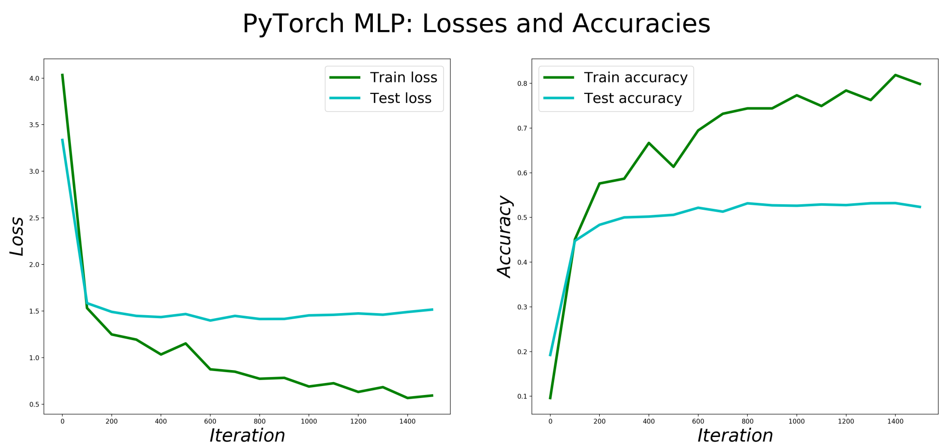

Figure 2: Loss and accuracy of the best performing PyTorch MLP (training and test). See last row in Table 1

3 Custom Module: Batch Normalization

3.1 Automatic differentiation

See code in the Python file `custom_batchnorm.py`.

3.2 Manual implementation of backward pass

Question 3.2 a)

$$\frac{\partial \mathcal{L}}{\partial \gamma_j} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial \gamma_j} = \sum_s \sum_i \frac{\partial \mathcal{L}}{\partial y_i^s}\frac{\partial y_i^s}{\partial \gamma_j} = \sum_s \sum_i \frac{\partial \mathcal{L}}{\partial y_i^s}\frac{\partial}{\partial \gamma_j}\left(\gamma_i \hat{x}_i^s + \beta_i\right) = \sum_s \frac{\partial \mathcal{L}}{\partial y_j^s}\hat{x}_j^s$$

$$\implies \sum_s \frac{\partial \mathcal{L}}{\partial y^s}\hat{x}^s \in \mathbb{R}^{1\times C}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_j} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial \beta_j} = \sum_s \sum_i \frac{\partial \mathcal{L}}{\partial y_i^s}\frac{\partial y_i^s}{\partial \beta_j} = \sum_s \sum_i \frac{\partial \mathcal{L}}{\partial y_i^s}\frac{\partial}{\partial \beta_j}\left(\gamma_i \hat{x}_i^s + \beta_i\right) = \sum_s \frac{\partial \mathcal{L}}{\partial y_j^s}$$
$$\implies \sum_s \frac{\partial \mathcal{L}}{\partial y^s} \in \mathbb{R}^{1\times C}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_j^r} &= \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial x_j^r} = \sum_s \sum_i \frac{\partial \mathcal{L}}{\partial y_i^s}\frac{\partial y_i^s}{\partial x_j^r} = \sum_s \sum_i \frac{\partial \mathcal{L}}{\partial y_i^s}\frac{\partial}{\partial x_j^r}\left(\gamma_i \hat{x}_i^s + \beta_i\right) \\
&= \gamma_j \sum_s \frac{\partial \mathcal{L}}{\partial y_i^s}\frac{\partial}{\partial x_j^r}\left(\frac{x_j^s - \mu_j}{\sqrt{\sigma_j^2+\epsilon}}\right) = \gamma_j \sum_s \frac{\partial \mathcal{L}}{\partial y_i^s}\left(\frac{\frac{\partial}{\partial x_j^r}\left(x_j^s-\mu_j\right)\left(\sqrt{\sigma_j^2+\epsilon}\right) - \left(x_j^s-\mu_j\right)\frac{\partial}{\partial x_j^r}\left(\sqrt{\sigma_j^2+\epsilon}\right)}{\sigma_j^2+\epsilon}\right) \\
&= \gamma_j \sum_s \frac{\partial \mathcal{L}}{\partial y_i^s}\left(\frac{\left(\delta_{sr} - \frac{\partial}{\partial x_j^r}\left(\frac{1}{B}\sum_s x_j^s\right)\right)\left(\sqrt{\sigma_j^2+\epsilon}\right) - \left(x_j^s-\mu_j\right)\frac{1}{2}\left(\sigma_j^2+\epsilon\right)^{-\frac{1}{2}}\frac{\partial}{\partial x_j^r}\left(\frac{1}{B}\sum_s (x_j^s-\mu_j)^2\right)}{\sigma_j^2+\epsilon}\right) \\
&= \gamma_j \sum_s \frac{\partial \mathcal{L}}{\partial y_i^s}\left(\frac{\left(\delta_{sr} - \frac{1}{B}\right)\left(\sqrt{\sigma_j^2+\epsilon}\right) - \left(x_j^s-\mu_j\right)\frac{1}{2}\left(\sigma_j^2+\epsilon\right)^{-\frac{1}{2}}\frac{2}{B}\sum_s (x_j^s-\mu_j)\left(\delta_{sr}-\frac{1}{B}\right)}{\sigma_j^2+\epsilon}\right) \\
&= \gamma_j \sum_s \frac{\partial \mathcal{L}}{\partial y_i^s}\left(\frac{\left(\delta_{sr} - \frac{1}{B}\right)\left(\sqrt{\sigma_j^2+\epsilon}\right) - \left(x_j^s-\mu_j\right)\frac{1}{2}\left(\sigma_j^2+\epsilon\right)^{-\frac{1}{2}}\frac{2}{B}\left(x_j^r-\mu_j\right)}{\sigma_j^2+\epsilon}\right) \\
&= \gamma_j \sum_s \frac{\partial \mathcal{L}}{\partial y_i^s}\left(\frac{\left(\delta_{sr} - \frac{1}{B}\right) - \frac{1}{B}\left(\frac{x_j^s-\mu_j}{\sqrt{\sigma_j^2+\epsilon}}\frac{x_j^r-\mu_j}{\sqrt{\sigma_j^2+\epsilon}}\right)}{\sqrt{\sigma_j^2+\epsilon}}\right) \\
&= \gamma_j \sum_s \frac{\partial \mathcal{L}}{\partial y_i^s}\left(\frac{B\delta_{sr} - 1 - \hat{x}_j^s\hat{x}_j^r}{B\sqrt{\sigma_j^2+\epsilon}}\right) = \frac{\gamma_j}{B\sqrt{\sigma_j^2+\epsilon}}\sum_s \frac{\partial \mathcal{L}}{\partial y_i^s}\left(B\delta_{sr} - 1 - \hat{x}_j^s\hat{x}_j^r\right) \in \mathbb{R}^{B\times C}
\end{aligned}$$

Question 3.2 b)

See code in the Python file `custom_batchnorm.py`.

Question 3.2 c)

See code in the Python file `custom_batchnorm.py`.

4 PyTorch CNN

The implemented CNN is trained with the default hyperparameters which can be observed in Table 2. In Figure 3 we see the respective loss and accuracy curves over the training. When comparing these plots with those of the best performing PyTorch MLP, we see that the training and test loss and accuracy do not diverge significantly. This suggests that the CNN does not overfit as strongly as the MLP.

# Epochs	Batchsize	Learning Rate	Optimizer	Final Training Accuracy	Final Test Accuracy
5000	32	1e-4	ADAM	0.9063	0.7688

Table 2: The hyperparameters and performance of the VGG network.

The corresponding code can be found in the Python files `convnet_pytorch.py` and `train_convnet_pytorch.py`.

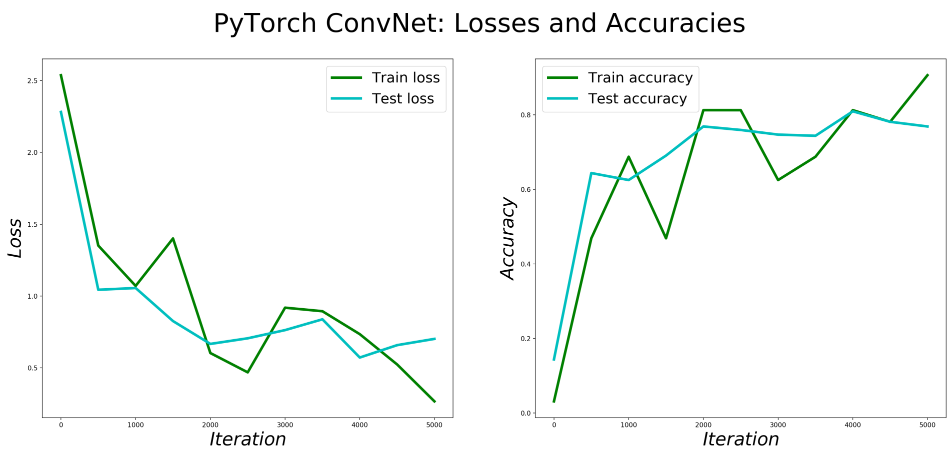

Figure 3: Loss and accuracy of the VGG network initialized with the default parameters.