Deep Learning
Assignment 1

Nil Stolt Ansó (12371149)

1 MLP backprop and NumPy implementation

Question 1.1 a)

i)

$$\frac{\partial L}{\partial x^{(N)}} = -t^T \frac{\partial log(x^{(N)})}{\partial x^{(N)}} = -t^T diag(1/x^{(N)})$$

where t is a one-hot vector of targets.

ii)

$$\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} = \frac{\partial softmax(\tilde{x}^{(N)})_i}{\partial \tilde{x}_j^{(N)}} = \frac{exp(\tilde{x}_i^{(N)}) \sum_j exp(\tilde{x}_j^{(N)}) - exp(\tilde{x}_i^{(N)}) exp(\tilde{x}_j^{(N)})}{\left(\sum_j exp(\tilde{x}_j^{(N)})\right)^2}$$

if $i = j$:

$$\frac{exp(\tilde{x}_i^{(N)}) \sum_j exp(\tilde{x}_j^{(N)}) - exp(\tilde{x}_i^{(N)}) exp(\tilde{x}_j^{(N)})}{\left(\sum_j exp(\tilde{x}_j^{(N)})\right)^2} = \frac{exp(\tilde{x}_i^{(N)})}{\sum_j exp(\tilde{x}_j^{(N)})} \frac{\left(\sum_j exp(\tilde{x}_j^{(N)}) - exp(\tilde{x}_j^{(N)})\right)}{\sum_j exp(\tilde{x}_j^{(N)})}$$

$$= x_i^{(N)} \left(1 - x_j^{(N)}\right)$$

if $i \neq j$:

$$\frac{exp(\tilde{x}_i^{(N)}) \sum_j exp(\tilde{x}_j^{(N)}) - exp(\tilde{x}_i^{(N)}) exp(\tilde{x}_j^{(N)})}{\left(\sum_j exp(\tilde{x}_j^{(N)})\right)^2} = \frac{-exp(\tilde{x}_i^{(N)})}{\sum_j exp(\tilde{x}_j^{(N)})} \frac{exp(\tilde{x}_j^{(N)})}{\sum_j exp(\tilde{x}_j^{(N)})} = -x_i^{(N)} x_j^{(N)}$$

This can be condensed through the Kronecker delta function into:

$$\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} = x_i^{(N)} \left(\delta_{ij} - x_j^{(N)}\right), \text{ where } \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if} i \neq j \end{cases}$$

The matrix notation would thus be:

$$\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} = diag(x^{(N)}) - x^{(N)} (x^{(N)})^T$$

iii)

$$\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} = \frac{\partial ReLU(\tilde{x})}{\partial \tilde{x}^{(l<N)}} = \begin{cases} 1, & \text{if } \tilde{x}_i^{(l<N)} > 0 \\ 0, & \text{otherwise} \end{cases}$$

iv)

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

v)

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

vi)

$$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial x^{(l)} W^l + b^{(l)}}{\partial b^{(l)}} = \mathbb{I}$$

where $\mathbb{I}$ is the identity matrix.

Question 1.1 b)

i)

$$\frac{\partial L}{\partial \tilde{x}^{(N)}} = \frac{\partial L}{\partial x^{(N)}} \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} = \frac{\partial L}{\partial x^{(N)}} \left[\text{diag}\left(x^{(N)}\right) - x^{(N)} \cdot \left(x^{(N)}\right)^T \right]$$

ii)

$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} = \frac{\partial L}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} = \frac{\partial L}{\partial x^{(l)}} \text{diag}\left(\mathbb{1}\left\{\tilde{x}^{(l)} > 0\right\}\right)$$

iii)

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

iv)

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}$$

$$= \frac{\partial L}{\partial \tilde{x}^{(l)}} \begin{bmatrix} \begin{bmatrix} x_1^{(l-1)} & \dots & x_n^{(l-1)} \\ \vdots & \vdots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \vdots & \vdots \\ x_1^{(l-1)} & \dots & x_n^{(l-1)} \end{bmatrix} \end{bmatrix}$$

Where $\frac{\partial L}{\partial \tilde{x}^{(l)}}$ is a $1 \times d_l$ vector, and $\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}$ is a $d_l \times (d_l \times d_{l-1})$ tensor

v)

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \mathbb{I} = \frac{\partial L}{\partial \tilde{x}^{(l)}}$$

Question 1.1 c)

Since the input in batch form will have an extra dimension for each element in the batch, the output loss will be a vector of losses. This loss vector will be averaged to provide the total loss used for backpropagation. Since we have activations for each input element of the batch, all resulting derivative matrices will have an extra dimension, where each element i of that dimension contains the derivatives corresponding to the batch's input element at index i. When updating a layer's weights $W^{(l)}$, the derivative used to update the weight $W_{ij}^{(l)}$ will be the average derivative across the B dimension of $\frac{\partial L}{\partial W^{(l)}}$, which is a 3 dimensional tensor of dimensions $(d_l \times d_l) \times B$, where B is the batch size.

Question 1.2

At every 100 steps (*eval freq*), the model is evaluated using the testing set. In figure 1, the loss and accuracy curves are plotted for testing and training procedures of the numpy implementation. These curves are for the default parameters. The best accuracy during a testing evaluation was after 900 training steps, which achieved the following results:

	Train		Test	
	Loss	Accuracy	Loss	Accuracy
Mean value	1.565	0.415	1.477	0.486

Table 1: MLP numpy implementation's mean values for loss and accuracy for both the testing set and training batches. These values were obtained after 900 training steps.

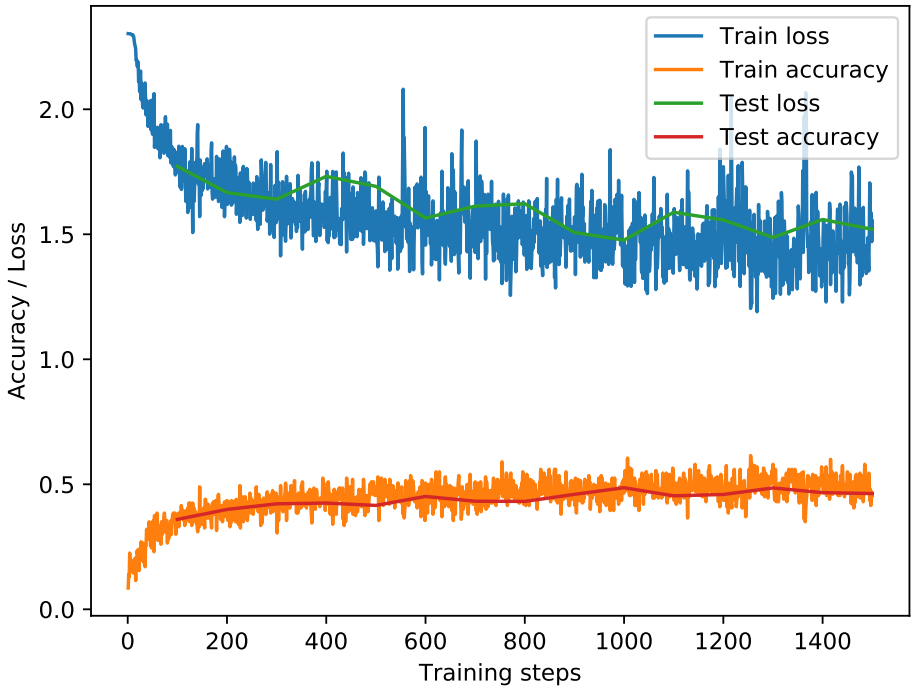

Figure 1: MLP numpy implementation's curves for loss and accuracy. Curves for both the testing set and training batches are displayed. Adam optimizer was used.

2 Pytorch MLP

Question 2

At every 100 steps (*eval freq*), the model is evaluated using the testing set. In figure 1, the loss and accuracy curves are plotted for testing and training procedures of the numpy implementation. These curves are for the parameters listed below in table 2. Furthermore, the optimizer used is 'Adam'.

The best accuracy during a testing evaluation was after 2800 training steps, which achieved the following results:

Parameter name	Value
dnn_hidden_units	4 hidden layers, 300 neurons each
learning_rate	7e-4
max_steps	3000
batch_size	200

	Train		Test	
	Loss	Accuracy	Loss	Accuracy
Mean value	0.835	0.675	1.541	0.539

Table 2: MLP Pytorch implementation's mean values for loss and accuracy for both the testing set and training batches. These values were obtained after 2800 training steps.

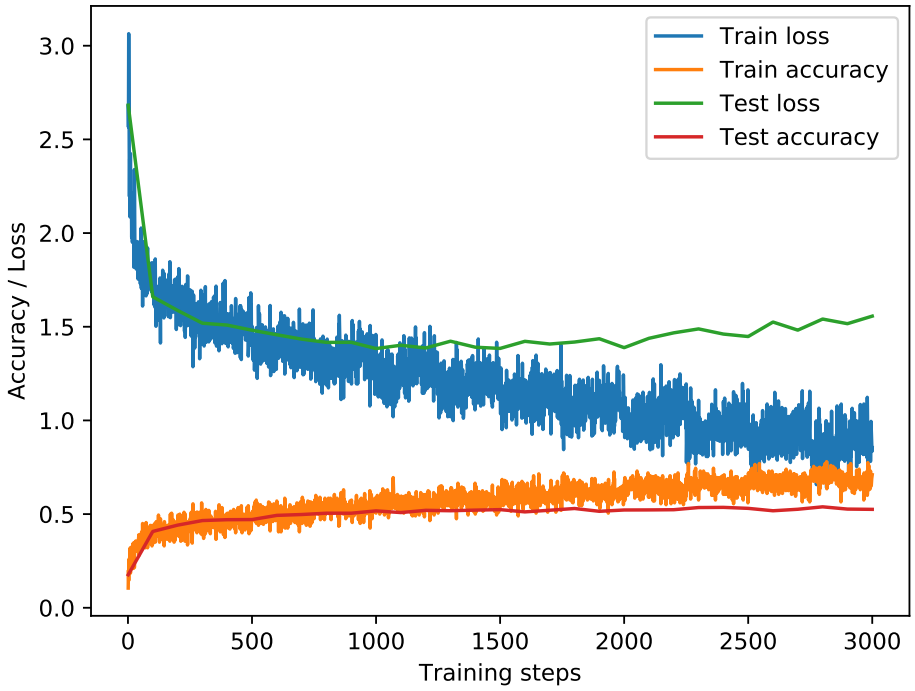

Figure 2: MLP Pytorch implementation's curves for loss and accuracy. Curves for both the testing set and training batches are displayed. Adam optimizer was used.

3 Custom Module: Batch Normalization

Question 3.1

The implementation can be found in the attached script *custom_batchnorm.py*.

After implementing batch normalization, an instance of the module was added after every ReLU module. The same parameters as in table 2 were used. Adam optimizer was used by the model. Doing these things boosted the results to the ones seen in Table 4 below and the curves seen in figure 3. Highest performance was obtained after 2800 training steps.

	Train		Test	
	Loss	Accuracy	Loss	Accuracy
Mean value	0.472	0.812	1.471	0.557

Table 3: MLP Pytorch implementation's mean values for loss and accuracy for both the testing set and training batches after adding batch normalization modules. Highest performance was obtained after 2800 training steps.

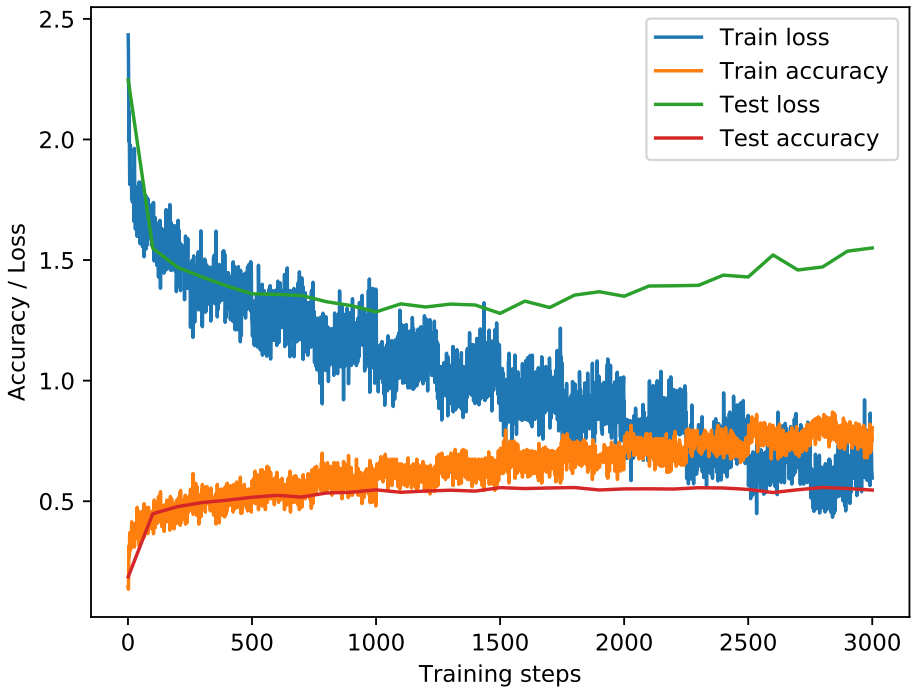

Figure 3: MLP Pytorch implementation's curves for loss and accuracy. Curves for both the testing set and training batches are displayed.

Question 3.2 a)

i)

$$\frac{\partial y_i^s}{\partial \gamma_j} = \frac{\partial(\gamma_i \hat{x}_i^s + \beta_i)}{\partial \gamma_j} = \delta_{ij}\hat{x}_i^s$$

Where δ_{ij} is the Kronecker delta function.

$$\left(\frac{\partial L}{\partial \gamma}\right)_j = \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \gamma_j} = \sum_s \frac{\partial L}{\partial y_j^s}\hat{x}_j^s$$

Since he derivative $\frac{\partial y_i^s}{\partial \gamma_j}$ is only non-zero for the element at index j.

ii)

$$\frac{\partial y_i^s}{\partial \beta_j} = \frac{\partial(\gamma_i \hat{x}_i^s + \beta_i)}{\partial \beta_j} = \mathbb{1}(i=j)$$

$$\left(\frac{\partial L}{\partial \beta}\right)_j = \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \beta_j} = \sum_s \frac{\partial L}{\partial y_j^s}$$

iii)

$$\left(\frac{\partial L}{\partial x}\right)_j^r = \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial x_j^r}$$

$$\frac{\partial y_i^s}{\partial x_j^r} = \frac{\partial\left(\gamma_i \hat{x}_i^s + \beta_i\right)}{\partial x_j^r} = \gamma_i \frac{\partial \hat{x}_i^s}{\partial x_j^r}$$

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

$$= \frac{\left(\delta_{ij}\delta_{rs} - \frac{\partial \mu_i}{\partial x_j^r}\right)\sqrt{\sigma_i^2+\epsilon} - (x_i^s - \mu_i)\frac{1}{2\sqrt{\sigma_i^2+\epsilon}}\frac{\partial(\sigma_i^2+\epsilon)}{\partial x_j^r}}{\sqrt{\sigma_i^2+\epsilon}^2}$$

$$= \frac{\left(\delta_{ij}\delta_{rs} - \frac{\partial \mu_i}{\partial x_j^r}\right)\sqrt{\sigma_i^2+\epsilon} - \frac{x_i^s-\mu_i}{2\sqrt{\sigma_i^2+\epsilon}}\frac{\partial \sigma_i^2}{\partial x_j^r}}{\sigma_i^2+\epsilon}$$

Where δ_{xy} is the Kronecker delta function.
The derivative of the the sample mean wrt the input:

$$\frac{\partial \mu_i}{\partial x_j^r} = \frac{\partial\left(\frac{1}{B}\sum_{s=1}^B x_i^s\right)}{\partial x_j^r} = \frac{1}{B}\sum_{s=1}^B \frac{\partial\left(x_i^s\right)}{\partial x_j^r} = \frac{1}{B}\frac{\partial\left(x_i^r\right)}{\partial x_j^r} = \frac{1}{B}\delta_{ij} = \frac{\delta_{ij}}{B}$$

The derivative of the the sample variance wrt the input:

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

$$= \frac{2\delta_{ij}}{B}\sum_{s=1}^B \left(x_i^s-\mu_i\right)\left(\delta_{rs}-\frac{1}{B}\right) = \frac{2\delta_{ij}}{B}\left(\left(x_i^r-\mu_i\right) - \frac{1}{B}\sum_{s=1}^B\left(x_i^s-\mu_i\right)\right)$$

$$= \frac{2\delta_{ij}}{B}\left(\left(x_i^r-\mu_i\right) - \frac{1}{B}0\right) = \frac{2\delta_{ij}}{B}\left(x_i^r-\mu_i\right)$$

Putting everything together:

$$\left(\frac{\partial L}{\partial x}\right)_j^r = \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial x_j^r} = \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i \frac{\left(\delta_{ij}\delta_{rs} - \frac{\delta_{ij}}{B}\right)\sqrt{\sigma_i^2+\epsilon} - \frac{x_i^s-\mu_i}{2\sqrt{\sigma_i^2+\epsilon}}\frac{2\delta_{ij}}{B}\left(x_i^r-\mu_i\right)}{\sigma_i^2+\epsilon}$$

$$= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\delta_{ij}\frac{\left(\delta_{rs}-\frac{1}{B}\right)\sqrt{\sigma_i^2+\epsilon} - \frac{x_i^s-\mu_i}{\sqrt{\sigma_i^2+\epsilon}}\frac{1}{B}\left(x_i^r-\mu_i\right)}{\sigma_i^2+\epsilon}$$

$$= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\delta_{ij}\frac{\frac{1}{B}\left(B\delta_{rs}-1\right)\sqrt{\sigma_i^2+\epsilon} - \frac{x_i^s-\mu_i}{\sqrt{\sigma_i^2+\epsilon}}\frac{1}{B}\left(x_i^r-\mu_i\right)}{\sigma_i^2+\epsilon}$$

$$= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\delta_{ij}\frac{\left(B\delta_{rs}-1\right)\sqrt{\sigma_i^2+\epsilon} - \frac{x_i^s-\mu_i}{\sqrt{\sigma_i^2+\epsilon}}\left(x_i^r-\mu_i\right)}{B(\sigma_i^2+\epsilon)}$$

$$= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\delta_{ij}\frac{\left(B\delta_{rs}-1\right)\sqrt{\sigma_i^2+\epsilon}}{B(\sigma_i^2+\epsilon)} - \frac{\frac{x_i^s-\mu_i}{\sqrt{\sigma_i^2+\epsilon}}\left(x_i^r-\mu_i\right)}{B(\sigma_i^2+\epsilon)}$$

$$= \sum_s\sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\delta_{ij}\frac{\left(B\delta_{rs}-1\right)}{B\sqrt{\sigma_i^2+\epsilon}} - \frac{\frac{x_i^s-\mu_i}{\sqrt{\sigma_i^2+\epsilon}}\left(x_i^r-\mu_i\right)}{B\sqrt{(\sigma_i^2+\epsilon)}\sqrt{(\sigma_i^2+\epsilon)}}$$

$$= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \gamma_i \delta_{ij} \frac{1}{B\sqrt{\sigma_i^2 + \epsilon}} \left((B\delta_{rs} - 1) - \frac{(x_i^s - \mu_i)(x_i^r - \mu_i)}{\sqrt{(\sigma_i^2 + \epsilon)}\sqrt{(\sigma_i^2 + \epsilon)}} \right)$$

$$= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \gamma_i \delta_{ij} \frac{1}{B\sqrt{\sigma_i^2 + \epsilon}} (B\delta_{rs} - 1 - \hat{x}_i^s \hat{x}_i^r)$$

$$= \sum_s \frac{\partial L}{\partial y_j^s} \gamma_j \frac{1}{B\sqrt{\sigma_j^2 + \epsilon}} (B\delta_{rs} - 1 - \hat{x}_j^s \hat{x}_j^r)$$

$$= \frac{\gamma_j}{B\sqrt{\sigma_j^2 + \epsilon}} \sum_s \frac{\partial L}{\partial y_j^s} (B\delta_{rs} - 1 - \hat{x}_j^s \hat{x}_j^r)$$

$$= \frac{\gamma_j}{B\sqrt{\sigma_j^2 + \epsilon}} \left(B \frac{\partial L}{\partial y_j^s} - \frac{\partial L}{\partial \beta_j^s} - \hat{x}_j^r \frac{\partial L}{\partial \gamma_j^s} \right)$$

Question 3.2 b)

See implementation in attached script *custom_ batchnorm.py*

Question 3.2 c)

See implementation in attached script *custom_ batchnorm.py*

4 Pytorch CNN

Question 4

The provided convnet architecture was implemented as seen in script *convnet_ pytorch.py*.

Default parameters were used alongside Adam and batch normalization. After the default 5000 training steps, the model seems to not have fully converged. The best performance was obtained at the 5000 training step mark as seen in table 4. This can also be seen from figure 4. Unlike in the plots for the MLP implementations, the test loss curve for the Convnet has not started to go up after reaching a minimum (as it has not reached a minimum yet after 5000 steps). Similarly, the test accuracy curve has not began going down. This means the model has not began to overfit yet by the end of the training run.

	Train		Test	
	Loss	Accuracy	Loss	Accuracy
Mean value	0.286	0.969	0.702	0.770

Table 4: Convnet Pytorch implementation's mean values for loss and accuracy for both the testing set and training batches. Batch normalization and Adam were used with the provided default parameters. These values were achieved after the model was trained for 5000 steps.

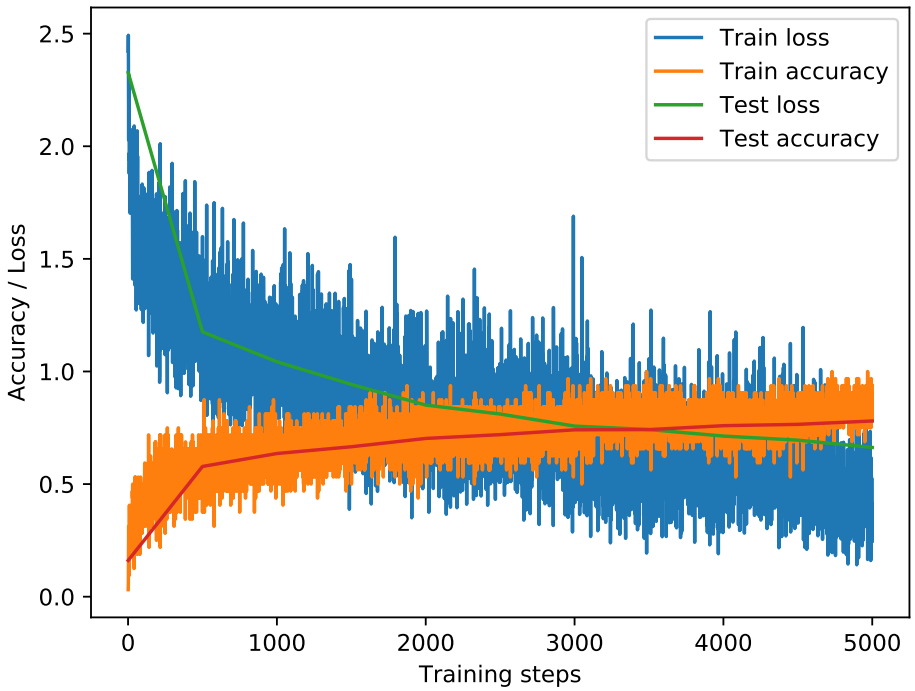

Figure 4: Convnet Pytorch implementation's curves for loss and accuracy. Curves for both the testing set and training batches are displayed. Batch normalization and Adam were used with the provided default parameters.