Assignment 1. MLPs, CNNs and Backpropagation

Xiaoxiao Wen
12320323
xiaoxiao.wen@student.uva.nl

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

a) Index notation:

$$\left(\frac{\partial L}{\partial x^{(N)}}\right)_i = -\frac{\partial \sum_m t_m \log(x_m^{(N)})}{\partial x_i^{(N)}} = -\frac{1}{x_i}\delta_{iargmax(t)}$$

$$\begin{aligned}\left(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\right)_{ij} &= \frac{\partial \frac{exp(\tilde{x}_i^{(N)})}{\sum_{m=1}^{d_N} exp(\tilde{x}_m^{(N)})}}{\partial \tilde{x}_j^{(N)}} = \frac{\delta_{ij} exp(\tilde{x}_i^{(N)}) \sum_{m=1}^{d_N} exp(\tilde{x}_m^{(N)}) - exp(\tilde{x}_j^{(N)}) exp(\tilde{x}_i^{(N)})}{[\sum_{m=1}^{d_N} exp(\tilde{x}_m^{(N)})]^2} \\ &= \delta_{ij}\frac{exp(\tilde{x}_i^{(N)})}{\sum_{m=1}^{d_N} exp(\tilde{x}_m^{(N)})} - \frac{exp(\tilde{x}_j^{(N)}) exp(\tilde{x}_i^{(N)})}{[\sum_{m=1}^{d_N} exp(\tilde{x}_m^{(N)})]^2} \\ &= \delta_{ij}\text{softmax}(\tilde{x}_i^{(N)}) - \text{softmax}(\tilde{x}_j^{(N)})\text{softmax}(\tilde{x}_i^{(N)}) \\ &= (\delta_{ij} - \text{softmax}(\tilde{x}_j^{(N)}))\text{softmax}(\tilde{x}_i^{(N)})\end{aligned}$$

$$\left(\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}\right)_{ij} = \delta_{ij}\mathbf{1}_{x_j} \qquad \mathbf{1} \text{ - unit step function}$$

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}\right)_{ij} = W_{ij}$$

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}\right)_{ijk} = x_k^{(l-1)}\delta_{ij}$$

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}\right)_{ij} = \delta_{ij}$$

b) **Assumption: all vectors are column vectors** and if a $\in \mathbb{R}$ and $A \in \mathbb{R}^{n\times m}$, then $\left[\frac{\partial a}{\partial A}\right]_{ij} = \frac{\partial a}{\partial A ji}$, so $\frac{\partial a}{\partial A} \in \mathbb{R}^{m\times n}$.

Note: the following derivations are for theoretical study. The real implementations in the code are slightly modified (with broadcasting, element-wise operations etc.) to adjust to the batch dimension and Numpy conventions.

Index notation:

Preprint. Work in progress.

$$\left(\frac{\partial L}{\partial \tilde{x}^{(N)}}\right)_i = \sum_j \frac{\partial L}{\partial x_j^{(N)}} \frac{\partial x_j^{(N)}}{\partial \tilde{x}_i^{(N)}}$$
$$= \sum_j \frac{\partial L}{\partial x_j^{(N)}} (\delta_{ji} - \text{softmax}(\tilde{x}_i^{(N)})) \text{softmax}(\tilde{x}_j^{(N)})$$

Matrix notation:

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

Index notation:

$$\left(\frac{\partial L}{\partial \tilde{x}^{(l<N)}}\right)_i = \sum_j \frac{\partial L}{\partial x_j^{(l)}} \frac{\partial x_j^{(l)}}{\partial \tilde{x}_i^{(l)}}$$
$$= \sum_j \frac{\partial L}{\partial x_j^{(l)}} \delta_{ij} \mathbf{1}_{x_j}$$
$$= \frac{\partial L}{\partial x_i^{(l)}} \mathbf{1}_{x_i}$$

Matrix notation:

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

Index notation:

$$\left(\frac{\partial L}{\partial x^{(l<N)}}\right)_i = \sum_j \frac{\partial L}{\partial \tilde{x}_j^{(l+1)}} \frac{\partial \tilde{x}_j^{(l+1)}}{\partial x_i^{(l)}}$$
$$= \sum_j \frac{\partial L}{\partial \tilde{x}_j^{(l+1)}} W_{ji}$$

Matrix notation:

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

Index notation:

$$\left(\frac{\partial L}{\partial W^{(l)}}\right)_{ij} = \sum_k \frac{\partial L}{\partial \tilde{x}_k^{(l)}} \frac{\partial \tilde{x}_k^{(l)}}{\partial W_{ij}^{(l)}}$$
$$= \sum_k \frac{\partial L}{\partial \tilde{x}_k^{(l)}} x_j^{(l-1)} \delta_{ki}$$
$$= \frac{\partial L}{\partial \tilde{x}_i^{(l)}} x_j^{(l-1)}$$

Matrix notation:

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

Index notation:

$$\begin{aligned}\left(\frac{\partial L}{\partial b^{(l)}}\right)_i &= \sum_j \frac{\partial L}{\partial \tilde{x}_j^{(l)}} \frac{\partial \tilde{x}_j^{(l)}}{\partial b_i^{(l)}} \\ &= \sum_j \frac{\partial L}{\partial \tilde{x}_j^{(l)}} \delta_{ji} \\ &= \frac{\partial L}{\partial \tilde{x}_i^{(l)}}\end{aligned}$$

Matrix notation:

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

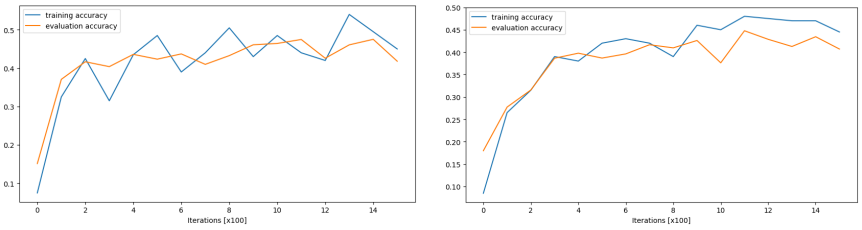
c) The above derived backpropagation equations do not change, because the batch operation only affect the cross entropy loss where we have

$$\left(\frac{\partial L_{total}}{\partial L_{individual}}\right)_s = \frac{1}{B}$$

so during the backpropagation of the cross entropy loss, only a fraction $\frac{1}{B}$ of L_{total} is propagated back.

1.2 NumPy implementation

The accuracy and loss curves for the default values of parameters are shown in Figure 1a and Figure 2a respectively.



(a) **(b)**

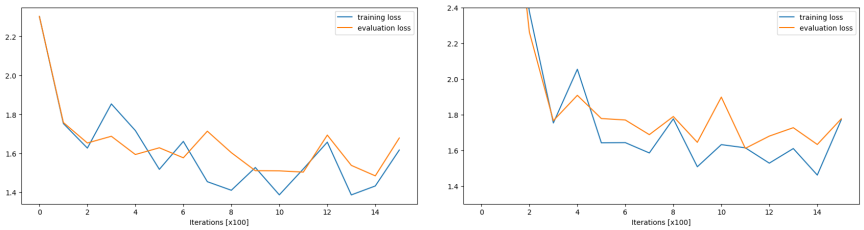
Figure 1: *Training and evaluation accuracies for the Numpy implementation (left) and the PyTorch implementation (right).*

2 PyTorch MLP

Using the default parameters, the comparisons between the Numpy implementation and the PyTorch implementation are shown in Figure 1 and Figure 2 for accuracy and loss respectively.

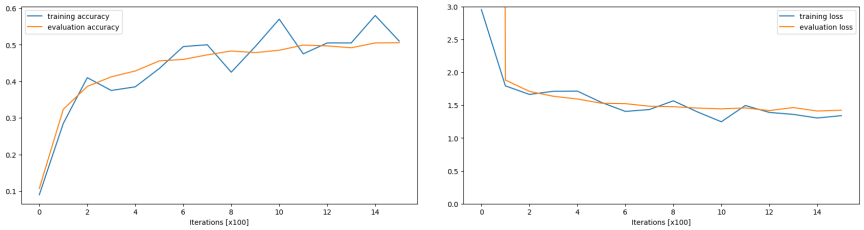
As shown by the accuracy and loss curves, there is no overfitting issue with the current configuration, so there exists some margin for increasing the model complexity in order to increase the performance.

The first improvement on the MLP model is to empirically increase the hidden layers both in depth and width and use Adam optimizer instead of SGD to relax the need for learning rate adjustment. The model complexity is increased by using 4 linear layers with hidden units 1024, 2048, 2048, 4096



(a) **(b)**

Figure 2: *Training and evaluation losses for the Numpy implementation (left) and the PyTorch implementation (right).*



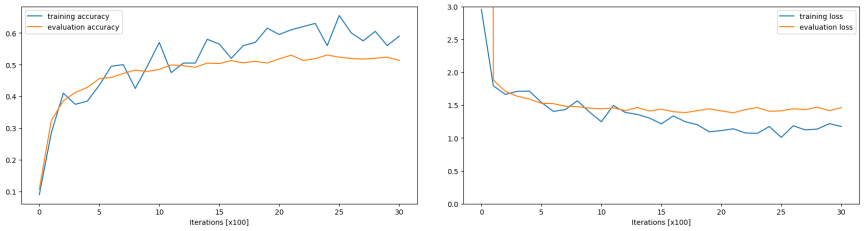
(a) **(b)**

Figure 3: *Training and evaluation accuracy (left) and loss (right) for the MLP implementation with the first improvement.*

respectively and the Adam optimizer is used with the default parameters. As shown in Figure 3, the test accuracy steadily increases to around 0.5, which is a marginal improvement compared to the initial MLP model.

As shown by the training loss curve in Figure 3b, with the current limited maximum number of steps, the training loss does not converge. So the next step is to tune the maximum number of training steps to assure convergence of the training loss.

The second improvement is to increase the maximum number of training steps from 1500 to 3000 and observe whether the loss and accuracy converges to further.



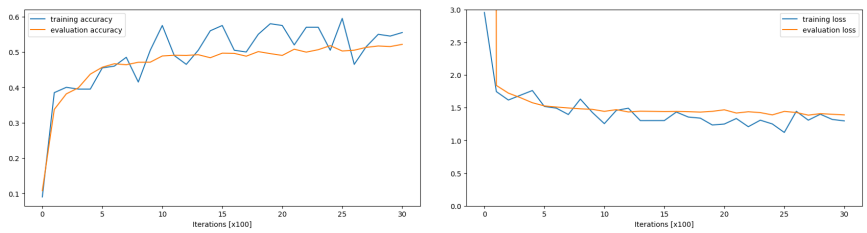
(a) **(b)**

Figure 4: *Training and evaluation accuracy (left) and loss (right) for the MLP implementation with the second improvement.*

As can be seen in Figure 4, compared to the first improvement, with increased number of training steps, the performance increases to some extent, but the loss and accuracy hit the plateau and do not converge further. As the training loss/accuracy starts to diverge obviously from the evaluation loss/accuracy, the problem of overfitting starts to occurs. In order to gain better performance, the next step is to add regularization when training the weights.

The third improvement includes adding L2-regularization/weight decay on the Adam optimizer to regularize the learned weights and have a better ability of generalization of the model. The

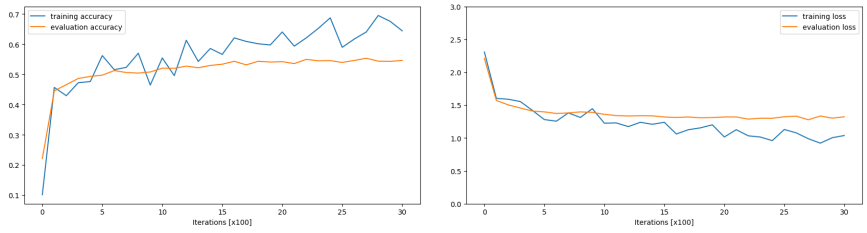
performance is shown in Figure 5. The test accuracy is further increased to around 0.52 which shows that the improvement is effective.



(a) **(b)**

Figure 5: *Training and evaluation accuracy (left) and loss (right) for the MLP implementation with the third improvement.*

Furthermore, a fourth improvement is also made for the MLP model. Intuitively, with the aforementioned aspects already tuned, one remaining improvement is using batch normalization for each linear layer for better training performance. Hence, the fourth improvement only introduces batch normalization layers for each linear layer before the ReLU activation. As shown in Figure 6, the introduction of the batch normalization layers further helps to increase the training performance, and the test accuracy is around 0.54.



(a) **(b)**

Figure 6: *Training and evaluation accuracy (left) and loss (right) for the MLP implementation with the fourth improvement.*

3 Custom Module: Batch Normalization

3.1 Automatic differentiation

For module initialization, the number of channels and the slack ϵ are stored and the parameters γ and β are associated with the module. Then for the forward propagation, the four steps of the batch normalization are implemented accordingly. Detailed implementation can be seen in the codes.

3.2 Manual implementation of backward pass

a) **Assumption: all vectors are column vectors** and if a $\in \mathbb{R}$ and $A \in \mathbb{R}^{n\times m}$, then $\left[\frac{\partial a}{\partial A}\right]_{ij} = \frac{\partial a}{\partial A j i}$, so $\frac{\partial a}{\partial A} \in \mathbb{R}^{m\times n}$.

Note: the following derivations are for theoretical study. The real implementations in the code are slightly modified (with broadcasting, element-wise operations etc.) to adjust to the batch dimension and Numpy conventions.

Index notation:

$$\begin{aligned}\left(\frac{\partial L}{\partial \gamma}\right)_j &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \frac{\partial y_i^s}{\partial \gamma_j} \\ &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \hat{x}_j^s \delta_{ij} \\ &= \sum_s \frac{\partial L}{\partial y_j^s} \hat{x}_j^s\end{aligned}$$

Matrix notation:

$$\frac{\partial L}{\partial \gamma} = \left(\text{diag}\left(\frac{\partial L}{\partial y}\hat{x}\right)\right)^T$$

Index notation:

$$\begin{aligned}\left(\frac{\partial L}{\partial \beta}\right)_j &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \frac{\partial y_i^s}{\partial \beta_j} \\ &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \delta_{ij} \\ &= \sum_s \frac{\partial L}{\partial y_j^s}\end{aligned}$$

Matrix notation:

$$\frac{\partial L}{\partial \beta} = [1...1]^{1\times B}\left(\frac{\partial L}{\partial y}\right)^T$$

Index notation:

$$
\begin{aligned}
\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} \sum_t \sum_k \frac{\partial y_i^s}{\partial \hat{x}_k^t}\frac{\partial \hat{x}_k^t}{\partial x_j^r} \\
&= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \sum_t \sum_k \gamma_k \delta_{ik}\delta_{st}\frac{\partial \hat{x}_k^t}{\partial x_j^r} \\
&= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\gamma_i \frac{\partial \hat{x}_i^s}{\partial x_j^r} \\
&= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\gamma_i \delta_{ij}\frac{\partial \hat{x}_i^s}{\partial x_j^r} \\
&= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{\partial \hat{x}_j^s}{\partial x_j^r} \\
&= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sqrt{\frac{1}{B}\sum_{n=1}^B (x_j^n - \frac{1}{B}\sum_{b=1}^B x_j^b)^2 + \epsilon}} \\
&\cdot [(\delta_{rs} - \frac{1}{B})\sqrt{\frac{1}{B}\sum_{n=1}^B (x_j^n - \frac{1}{B}\sum_{b=1}^B x_j^b)^2 + \epsilon} \\
&- (x_j^s - \frac{1}{B}\sum_{m=1}^B x_j^m)\frac{\partial \sqrt{\frac{1}{B}\sum_{n=1}^B (x_j^n - \frac{1}{B}\sum_{b=1}^B x_j^b)^2 + \epsilon}}{\partial x_j^r}] \\
&= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sigma_j^2 + \epsilon}[(\delta_{rs} - \frac{1}{B})\sqrt{\sigma_j^2 + \epsilon} - (x_j^s - \mu_j)\frac{\partial \sqrt{\frac{1}{B}\sum_{n=1}^B (x_j^n - \frac{1}{B}\sum_{b=1}^B x_j^b)^2 + \epsilon}}{\partial (\frac{1}{B}\sum_{n=1}^B (x_j^n - \frac{1}{B}\sum_{b=1}^B x_j^b)^2 + \epsilon)} \\
&\cdot \frac{\partial (\frac{1}{B}\sum_{n=1}^B (x_j^n - \frac{1}{B}\sum_{b=1}^B x_j^b)^2 + \epsilon)}{\partial x_j^r}] \\
&= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sigma_j^2 + \epsilon}[(\delta_{rs} - \frac{1}{B})\sqrt{\sigma_j^2 + \epsilon} - (x_j^s - \mu_j)\frac{1}{2}\frac{1}{\sigma_j^2 + \epsilon} \\
&\cdot \sum_{l=1}^B \frac{\partial (\frac{1}{B}\sum_{n=1}^B (x_j^n - \frac{1}{B}\sum_{b=1}^B x_j^b)^2 + \epsilon)}{\partial (x_j^l - \frac{1}{B}\sum_{b=1}^B x_j^b)^2}\frac{\partial (x_j^l - \frac{1}{B}\sum_{b=1}^B x_j^b)^2}{\partial (x_j^l - \frac{1}{B}\sum_{b=1}^B x_j^b)}\frac{(x_j^l - \frac{1}{B}\sum_{b=1}^B x_j^b)}{x_j^r} \\
&= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sigma_j^2 + \epsilon}[(\delta_{rs} - \frac{1}{B})\sqrt{\sigma_j^2 + \epsilon} - (x_j^s - \mu_j)\frac{1}{B}\frac{1}{\sqrt{\sigma_j^2 + \epsilon}}\sum_{l=1}^B (x_j^l - \frac{1}{B}\sum_{b=1}^B x_j^b)(\delta_{rl} - \frac{1}{B})] \\
&= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sigma_j^2 + \epsilon}[(\delta_{rs} - \frac{1}{B})\sqrt{\sigma_j^2 + \epsilon} - \frac{1}{B}(x_j^s - \mu_j)\frac{1}{\sqrt{\sigma_j^2 + \epsilon}}\sum_{l=1}^B (x_j^l\delta_{rl} - \frac{1}{B}x_j^l - \mu_j\delta_{rl} + \frac{1}{B}\mu_j)] \\
&= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sigma_j^2 + \epsilon}[(\delta_{rs} - \frac{1}{B})\sqrt{\sigma_j^2 + \epsilon} - \frac{1}{B}(x_j^s - \mu_j)\frac{1}{\sqrt{\sigma_j^2 + \epsilon}}(x_j^r - \mu_j - \mu_j + \mu_j)] \\
&= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sqrt{(\sigma_j^2 + \epsilon)^3}}[(\delta_{rs} - \frac{1}{B})(\sigma_j^2 + \epsilon) - \frac{1}{B}(x_j^s - \mu_j)(x_j^r - \mu_j)]
\end{aligned}
$$

Then given that $\left(\frac{\partial L}{\partial \gamma}\right)_j = \sum_s \frac{\partial L}{\partial y_j^s}\hat{x}_j^s$, we know $\left(\frac{\partial L}{\partial \hat{x}}\right)_j^s = \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j$. Subsequently, we can do some rearranging on the above derived equation:

$$\left(\frac{\partial L}{\partial x}\right)_j^r = \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sqrt{(\sigma_j^2+\epsilon)^3}}[(\delta_{rs}-\frac{1}{B})(\sigma_j^2+\epsilon)-\frac{1}{B}(x_j^s-\mu_j)(x_j^r-\mu_j)]$$
$$= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{1}{\sqrt{\sigma_j^2+\epsilon}}\frac{1}{B}[B\delta_{rs}-1-\frac{1}{\sqrt{\sigma_j^2+\epsilon}}\frac{1}{\sqrt{\sigma_j^2+\epsilon}}(x_j^s-\mu_j)(x_j^r-\mu_j)]$$
$$= \frac{1}{\sqrt{\sigma_j^2+\epsilon}}\frac{1}{B}[B\frac{\partial L}{\partial y_j^r}\gamma_j - \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j - \frac{x_j^r-\mu_j}{\sqrt{\sigma_j^2+\epsilon}}\sum_s \frac{\partial L}{\partial y_j^s}\gamma_j \frac{x_j^s-\mu_j}{\sqrt{\sigma_j^2+\epsilon}}]$$
$$= \frac{1}{\sqrt{\sigma_j^2+\epsilon}}\frac{1}{B}[B\frac{\partial L}{\partial \hat{x}_j^r} - \sum_s \frac{\partial L}{\partial \hat{x}_j^s} - \hat{x}_j^r \sum_s \frac{\partial L}{\partial \hat{x}_j^s}\hat{x}_j^s]$$

Matrix notation:

$$\left(\frac{\partial L}{\partial x}\right) = \frac{1}{\sqrt{\sigma_j^2+\epsilon}}\frac{1}{B}[B\frac{\partial L}{\partial \hat{x}} - \frac{\partial L}{\partial \hat{x}}[1...1]^{B\times B} - \hat{x}^T \odot (\text{diag}\left(\frac{\partial L}{\partial \hat{x}}\hat{x}\right)[1...1]^{1\times B})]$$

b) The `forward()` method of the autograd function is implemented as described in the steps where the tensors like $\hat{x}$, γ and β are stored for the backpropagation as derived above.

The `backward()` method of the autograd function is implemented as derived above with slight modifications.

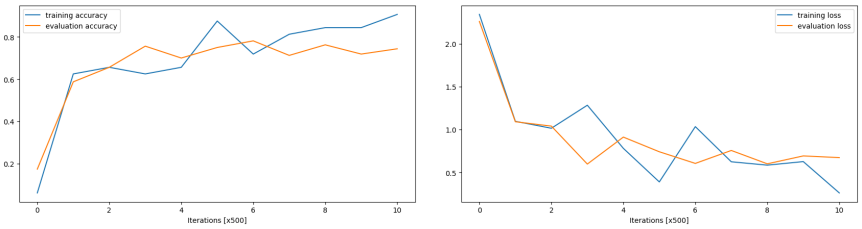
Detailed implementation can be seen in the code.

c) The newly created `CustomBatchNormManual` module is similarly initialized as `CustomBatchNormAutograd` and in its `forward()` method, the the autograd function class is initialized and invoked with `apply()` where the forward and backward propagations are properly implemented.

Detailed implementation can be seen in the code.

4 PyTorch CNN

As shown in Figure 7, using the default parameters and configurations as required, the ConvNet has a performance around 0.75 accuracy on the test set, which fulfills the requirement.



(a) **(b)**

Figure 7: *Training and evaluation accuracy (left) and loss (right) for the ConvNet implementation.*