Assignment 1: MLPs, CNNs and Backpropagation

Kai Liang
University of Amsterdam
kai.liang@student.uva.nl

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

a) It is assumed for the following questions that the gradient of a scalar with respect to a column vector is a row vector.

$$\begin{aligned}
\left(\frac{\partial L}{\partial x^{(N)}}\right)_i &= \left(\frac{\partial L}{\partial x_i^{(N)}}\right) \\
&= \frac{\partial\left(-\sum_i t_i \log x_i^{(N)}\right)}{\partial x_i^{(N)}} \\
&= -\frac{t_i}{x_i^{(N)}} \\
\left(\frac{\partial L}{\partial x^{(N)}}\right) &= \left(-t \circ (x^{(N)})^{-1}\right)^T \\
\left(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\right)_{ij} &= \left(\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}}\right) \\
&= \frac{\partial\left(\frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}_k^{(N)}\right)}\right)}{\partial \tilde{x}_j^{(N)}} \\
&= \begin{cases} \frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}_k^{(N)}\right)} - \frac{\left(\exp\left(\tilde{x}_i^{(N)}\right)\right)^2}{\left(\sum_{k=1}^{d_N}\exp\left(\tilde{x}_k^{(N)}\right)\right)^2} & \text{if } i = j \\ -\frac{\exp\left(\tilde{x}_i^{(N)}\right)\exp\left(\tilde{x}_j^{(N)}\right)}{\left(\sum_{k=1}^{d_N}\exp\left(\tilde{x}_k^{(N)}\right)\right)^2} & \text{if } i \neq j \end{cases} \\
&= \begin{cases} \left(1 - S_j\right) S_i & \text{if } i = j \\ -S_j S_i & \text{if } i \neq j \end{cases} \\
&= \left(\delta_{ij} - S_j\right) S_i \\
\left(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\right) &= \operatorname{diag}\left(x^{(N)}\right) - x^{(N)} x^{(N)^T} \\
\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} &= \frac{\partial \max\left(0, \tilde{x}^{(l<N)}\right)}{\partial \tilde{x}^{(l<N)}} \\
&= \operatorname{diag}\left(\mathbb{1}_{\tilde{x}^{(l<N)}>0}\right)
\end{aligned}$$

$$\begin{aligned}\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} &= \frac{\partial \left(W^{(l)} x^{(l-1)} + b^{(l)}\right)}{\partial x^{(l-1)}} \\ &= W^{(l)}\end{aligned}$$

$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}$ is a tensor where the ijk-th element is:

$$\begin{aligned}\left(\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}\right)_{ijk} &= \left(\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}}\right) \\ &= \frac{\partial \left(W_{i,:}^{(l)} x^{(l-1)} + b_i^{(l)}\right)}{\partial W_{jk}^{(l)}} \\ &= \begin{cases} x_k^{(l-1)} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \\ &= x_k^{(l-1)} \delta_{ij} \\ \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} &= \frac{\partial (W^{(l)} x^{(l-1)} + b^{(l)})}{\partial b^{(l)}} \\ &= \mathbb{I}\end{aligned}$$

b)

$$\begin{aligned}\frac{\partial L}{\partial \tilde{x}^{(N)}} &= \frac{\partial L}{\partial x^{(N)}} \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} \\ &= \frac{\partial L}{\partial x^{(N)}} \left(\text{diag}\left(x^{(N)}\right) - x^{(N)} x^{(N)^T}\right) \\ \frac{\partial L}{\partial \tilde{x}^{(l<N)}} &= \frac{\partial L}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} \\ &= \frac{\partial L}{\partial x^{(l)}} \text{diag}\left(\mathbb{1}_{\tilde{x}^{(l)}>0}\right) \\ \frac{\partial L}{\partial x^{(l<N)}} &= \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{x}^{(l+1)}} W^{(l+1)} \\ \frac{\partial L}{\partial W^{(l)}} &= \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} \\ &= x^{(l-1)} \frac{\partial L}{\partial \tilde{x}^{(l)}} \\ \frac{\partial L}{\partial b^{(l)}} &= \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{x}^{(l)}}\end{aligned}$$

c) If a batch size $B \neq 1$ is used, the gradient of the loss with respect to the final layer (i.e. $\frac{\partial L_{total}}{\partial \mathbf{x}^{(N)}}$) needs to be divided by the number of samples in the batch during backpropagation. Besides, for previous layers, the batch dimension should also be taken into account separately, except for the gradient with respect to the bias in the linear layer where the batch dimension needs to be summed up.

1.2 NumPy Implementation

The detailed implementation of MLP with NumPy routines can be found in `modules.py` and `mlp_numpy.py`. With this implementation, matrix multiplications are thoroughly used rather than iterating over samples in the batch or weight rows/columns. Using the default parameters, the result as shown in Figure 1 is obtained. Specifically, the model gets a final accuracy of 0.47 on the entire test set.

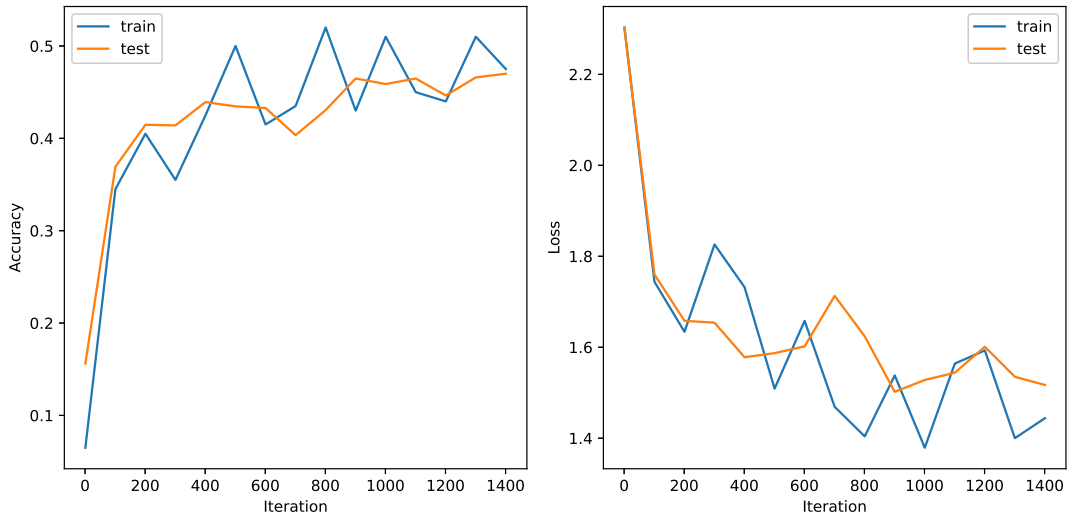

Figure 1: *Loss & accuracy curves of NumPy MLP on the training and test set with default parameters.*

2 PyTorch MLP

The detailed implementation of the MLP with PyTorch can be found in `mlp_pytorch.py` and `train_mlp_pytorch.py`. Using the default parameters, the model gets the result as shown in Figure 2. Specifically, the final accuracy on the entire test set is 0.4394.

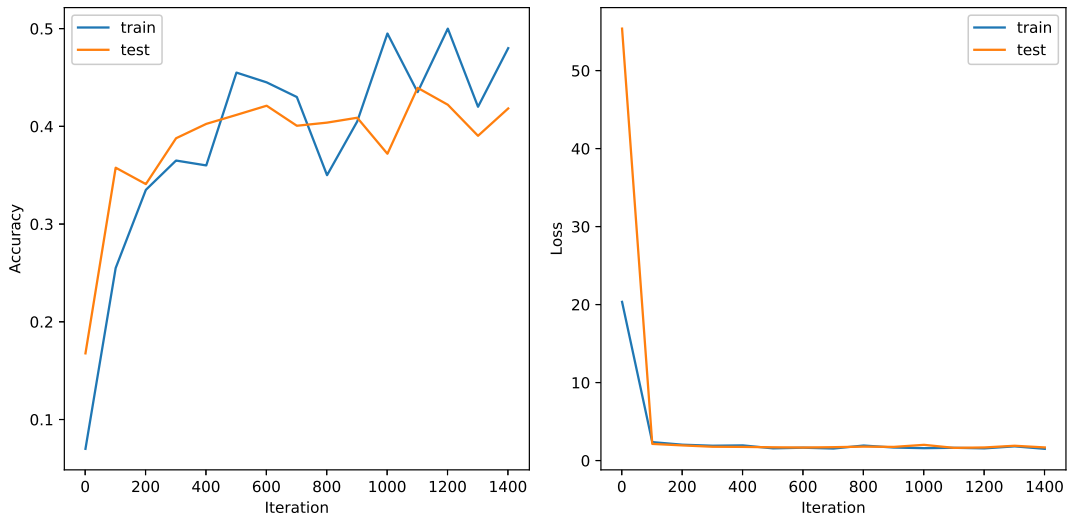

Figure 2: *Loss & accuracy curves of PyTorch MLP on the training and test set with default parameters.*

To obtain better accuracies on the test set, a number of experiments are conducted for different parameter settings, which can be found in Table 1.

Table 1: *Modifications of parameters and the resulting test accuracies.*

	Hidden Units	Learning Rate	Max Steps	Optimizer	Weight Decay	Test Accuracy
0	100	2e-3	1500	SGD	0	0.4394
1	200,200,200,100,100	2e-3	1500	SGD	0	0.4564
2	200,200,200,100,100	2e-3	7000	SGD	0	0.5147
3	200,200,200,100,100	2e-3	7000	Adam	0	0.5327
4	**200,200,200,100,100**	**1.5e-3**	**7000**	**Adam**	**3e-4**	**0.5350**

The ideas I choose new modifications from the initial setting (i.e. 0) to the fine-tuned setting (i.e. 4) is as follows. On the one hand, empirically speaking, deepening the network could almost always increase the performance of the network on the classification task compared to use only one hidden layer, which is also the case for increasing the number of training epochs. On the other hand, doing so will inevitably bring in the problem of overfitting. Hence, it is important to choose a smarter optimizer during parameter updates, and adapt the value of learning rate during training. Using the last row of parameter settings (i.e. best model) in the above table, the accuracy and loss curves on the training set and test set are shown in Figure 3.

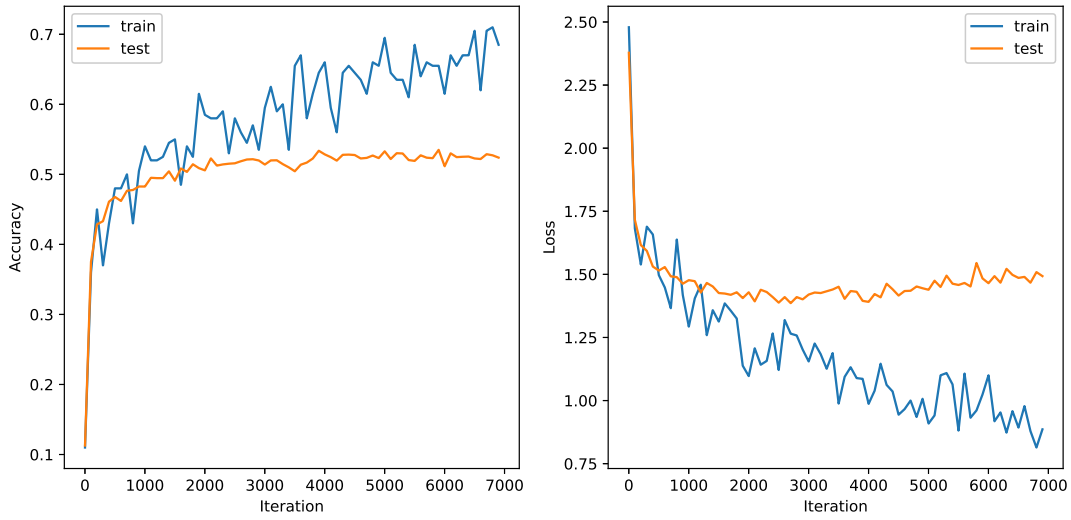

Figure 3: *Loss & accuracy curves of PyTorch MLP on the training and test set with best parameters.*

As can be inferred from Figure 3, the best model performs better than the one under the default setting in terms of both test accuracy and data fitting.

3 Custom Module: Batch Normalization

3.1 Automatic differentiation

The detailed implementation of 1D Batch Normalization operation as an `nn.Module` can be found in `custom_batchnorm.py`. For initialization, γ is initialized to be a vector of 1s with the size equals to the number of neurons, and β is initialized to be a vector of 0s with the size equals to the number of neurons.

3.2 Manual implementation of backwards pass

a) $\dfrac{\partial L}{\partial \gamma}$ is a row vector of size $1 \times C$, where the j-th element is:

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

$\frac{\partial L}{\partial \beta}$ is a row vector of size $1 \times C$, where the j-th element is:

$$\begin{aligned}\left(\frac{\partial L}{\partial \beta}\right)_j &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \frac{\partial y_i^s}{\partial \beta_j} \\ &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \frac{\partial \left(\gamma_i \hat{x}_i^s + \beta_i\right)}{\partial \beta_j} \\ &= \sum_s \frac{\partial L}{\partial y_j^s}\end{aligned}$$

$\frac{\partial L}{\partial x}$ is a matrix of size $C \times B$, where the rj-th element is:

$$\begin{aligned}\left(\frac{\partial L}{\partial x}\right)_j^r &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \frac{\partial y_i^s}{\partial x_j^r} \\ &= \frac{\partial L}{\partial \hat{x}_j^r} \frac{\partial \hat{x}_j^r}{\partial x_j^r} + \sum_s \frac{\partial L}{\partial \hat{x}_j^s} \frac{\partial \hat{x}_j^s}{\partial \mu_j} \frac{\partial \mu_j}{\partial x_j^r} + \sum_s \frac{\partial L}{\partial \hat{x}_j^s} \frac{\partial \hat{x}_j^s}{\partial \sigma_j^2} \left(\frac{\partial \sigma_j^2}{\partial x_j^r} + \frac{\partial \sigma_j^2}{\partial \mu_j} \frac{\partial \mu_j}{\partial x_j^r}\right)\end{aligned}$$

where:

$$\begin{aligned}\left(\frac{\partial L}{\partial \hat{x}}\right)_j^r &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \frac{\partial y_i^s}{\partial \hat{x}_j^r} \\ &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \frac{\partial \left(\gamma_i \hat{x}_i^s + \beta_i\right)}{\partial \hat{x}_j^r} \\ &= \frac{\partial L}{\partial y_j^r} \gamma_j \\ \frac{\partial \hat{x}_j^r}{\partial x_j^r} &= \frac{\partial \left(\frac{x_j^r - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}\right)}{\partial x_j^r} \\ &= \frac{1}{\sqrt{\sigma_j^2 + \epsilon}} \\ \frac{\partial \hat{x}_j^r}{\partial \mu_j} &= \frac{\partial \left(\frac{x_j^r - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}\right)}{\partial \mu_j} \\ &= -\frac{1}{\sqrt{\sigma_j^2 + \epsilon}}\end{aligned}$$

$$
\begin{aligned}
\frac{\partial \hat{x}_j^r}{\partial \sigma_j^2} &= \frac{\partial \left(\frac{x_j^r - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \right)}{\partial \sigma_j^2} \\
&= -\frac{1}{2} \left(x_j^r - \mu_j \right) \left(\sigma_j^2 + \epsilon \right)^{-\frac{3}{2}} \\
\frac{\partial \mu_j}{\partial x_j^r} &= \frac{\partial \left(\frac{1}{B} \sum_s x_j^s \right)}{\partial x_j^r} \\
&= \frac{1}{B} \\
\frac{\partial \sigma_j^2}{\partial x_j^r} &= \frac{\partial \left(\frac{1}{B} \sum_s \left(x_j^s - \mu_j \right)^2 \right)}{\partial x_j^r} \\
&= \frac{2}{B} \left(x_j^r - \mu_j \right) \\
\frac{\partial \sigma_j^2}{\partial \mu_j} &= \frac{\partial \left(\frac{1}{B} \sum_s \left(x_j^s - \mu_j \right)^2 \right)}{\partial \mu_j} \\
&= -\frac{2}{B} \sum_s \left(x_j^s - \mu_j \right) \\
&= \frac{2}{B} B \mu_j - 2\mu_j \\
&= 0
\end{aligned}
$$

Hence,

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

b) The detailed implementation of the Batch Norm as a `torch.autograd.Function` can be found in `custom_batchnorm.py`.

c) The detailed implementation of creating γ and β as `nn.Parameters` can be found in `custom_batchnorm.py`.

4 PyTorch CNN

The detailed implementation of CNN with PyTorch can be found in `convnet_pytorch.py` and `train_convnet_pytorch.py`. With the default parameters, the model gets the result as shown in Figure 4. Specifically, the final accuracy on the entire test set is 0.7837.

As can be inferred from Figure 4, the test loss and accuracy decrease over the entire experiment, when indicates that the model does not overfit. Also, it is clear that CNN performs significantly better than MLP for this task.

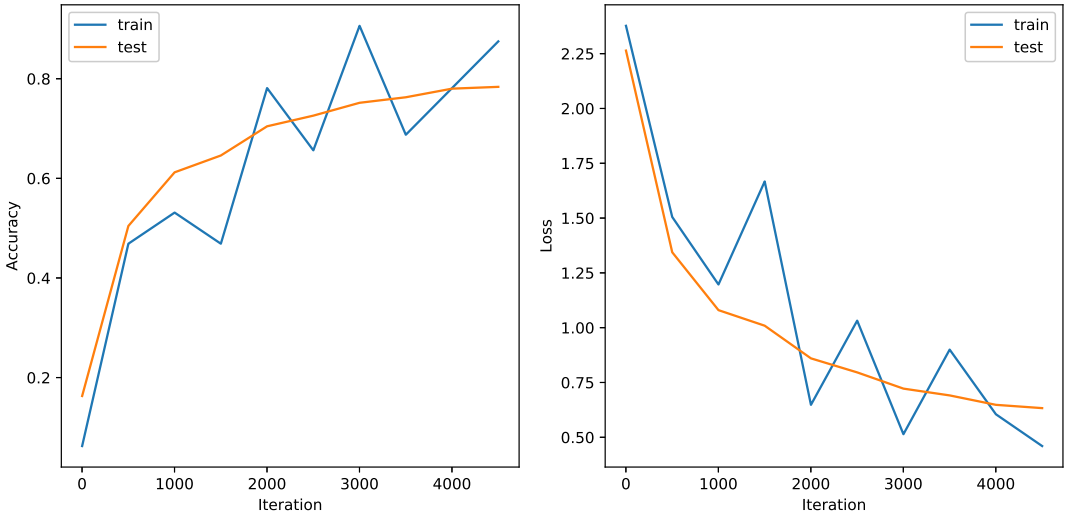

Figure 4: *Loss & accuracy curves of PyTorch CNN on the training and test set with default parameters.*