Assignment 1: MLPs, CNNs and Backpropagation

Elias Kassapis
University of Amsterdam

Introduction

In this assignment, we have designed and implemented basic neural architectures such as MLPs and CNNs for classification tasks. We have familiarized ourselves with the details of constructing a neural network by analytically deriving backpropagation gradients, manually designing batch normalization, and learned how to implement neural networks with PyTorch. We used the CIFAR-10 dataset and designed such models for 10 way image classification. In this report we present and discuss the design and implementation of the algorithms mentioned above.

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

(a) We have the following gradients to derive

$$\frac{\partial L}{\partial x^{(N)}}, \quad \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}, \quad \frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}, \quad \frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}, \quad \frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}, and \quad \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}$$

Let d_l denote the dimensionality of the input in layer l. Computing the derivatives gives

i. The gradient of the Cross Entropy module is given by

$$\begin{aligned}\frac{\partial L}{\partial x_j^{(N)}} &= \frac{\partial}{\partial x_j^{(N)}}\left(-\sum_i t_i \log x_i^{(N)}\right) \\ &= -\frac{t_j}{x_j^{(N)}} \in \mathbb{R}\end{aligned}$$

Hence

$$\begin{aligned}\frac{\partial L}{\partial x^{(N)}} &= \begin{bmatrix} -\frac{t_1}{x_1^{(N)}} & \cdots & -\frac{t_{d_N}}{x_{d_N}^{(N)}} \end{bmatrix} \\ &= \left(-\frac{t_j}{x^{(N)}}\right)_{j=1}^{d_N} \in \mathbb{R}^{1\times d_N}\end{aligned}$$

Since t is a one-hot vector, we get the following

$$\frac{\partial L}{\partial x_j^{(N)}} = \begin{cases} -\frac{1}{x_j^{(N)}} & \text{if } t_j = \text{argmax } t \\ 0 & \text{otherwise} \end{cases}$$

Submitted to the Spring 2019 Amsterdam Deep Learning course

ii. The gradient of the Softmax module is given by

$$\begin{aligned}\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} &= \frac{\partial}{\partial \tilde{x}_j^{(N)}} \left(\frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)} \right) \\ &= \frac{\partial}{\partial \tilde{x}_j^{(N)}} \left(\exp\left(\tilde{x}_i^{(N)}\right) \left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right) \right)^{-1} \right)\end{aligned}$$

Now if $i = j$, we get

$$\begin{aligned}\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} &= \exp\left(\tilde{x_i}^{(N)}\right) \left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right) \right)^{-1} - \exp\left(\tilde{x}_i^{(N)}\right) \left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right) \right)^{-2} \exp\left(\tilde{x}_j^{(N)}\right) \\ &= \frac{\exp\left(\tilde{x_i}^{(N)}\right)}{\sum_k \exp\left(\tilde{x_k}^{(N)}\right)} - \frac{\exp\left(\tilde{x_i}^{(N)}\right)}{\sum_k \exp\left(\tilde{x_k}^{(N)}\right)} \frac{\exp\left(\tilde{x_j}^{(N)}\right)}{\sum_k \exp\left(\tilde{x_k}^{(N)}\right)} \\ &= \frac{\exp\left(\tilde{x_i}^{(N)}\right)}{\sum_k \exp\left(\tilde{x_k}^{(N)}\right)} \left(1 - \frac{\exp\left(\tilde{x_j}^{(N)}\right)}{\sum_k \exp\left(\tilde{x_k}^{(N)}\right)} \right) \\ &= x_i^{(N)} \left(1 - x_j^{(N)} \right)\end{aligned}$$

If $i \neq j$, we get

$$\begin{aligned}\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} &= -\exp\left(\tilde{x}_i^{(N)}\right) \left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right) \right)^{-2} \exp\left(\tilde{x}_j^{(N)}\right) \\ &= -\frac{\exp\left(\tilde{x_i}^{(N)}\right)}{\sum_k \exp\left(\tilde{x_k}^{(N)}\right)} \frac{\exp\left(\tilde{x_j}^{(N)}\right)}{\sum_k \exp\left(\tilde{x_k}^{(N)}\right)} \\ &= -x_i^{(N)} x_j^{(N)}\end{aligned}$$

Therefore,

$$\begin{aligned}\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} &= \begin{bmatrix} x_1^{(N)}\left(1 - x_1^{(N)}\right) & -x_1^{(N)} x_2^{(N)} & \dots & -x_1^{(N)} x_{d_N}^{(N)} \\ -x_2^{(N)} x_1^{(N)} & x_2^{(N)}\left(1 - x_2^{(N)}\right) & \dots & -x_2^{(N)} x_{d_N}^{(N)} \\ \vdots & \ddots & \ddots & \vdots \\ -x_{d_N}^{(N)} x_1^{(N)} & \dots & \dots & x_{d_N}^{(N)}\left(1 - x_{d_N}^{(N)}\right) \end{bmatrix} \\ &= \left(x_i^{(N)} \left(\mathbb{I}_{i,j} - x_j^{(N)} \right) \right)_{i,j=1}^{d_N, d_N} \in \mathbb{R}^{d_N \times d_N}\end{aligned}$$

where

$$\mathbb{I}_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

iii. The gradient for the $ReLU$ module for layer $l < N$ is given by

$$\frac{\partial x_i^{(l)}}{\partial \tilde{x}_j^{(l)}} = \frac{\partial}{\partial \tilde{x}^{(l)}} \left(\text{ReLU} \left(\tilde{x}_i^{(l)} \right) \right)$$

where

$$\text{ReLU} \left(\tilde{x}_i^{(l)} \right) = \max \left(0, \tilde{x}_i^{(l)} \right) = \begin{cases} 0 & \text{if } x \leq 0 \\ \tilde{x}_i^{(l)} & \text{if } x > 0 \end{cases}$$

Therefore

$$\frac{\partial x_i^{(l)}}{\partial \tilde{x}_j^{(l)}} = \begin{cases} 0 & \text{if } \tilde{x}_j^{(l)} < 0 \text{ or } i \neq j \\ 1 & \text{if } \tilde{x}_j^{(l)} > 0 \text{ and } i = j \\ \text{None} & \text{if } \tilde{x}_j^{(l)} = 0 \end{cases}$$

Now, let

$$R' \left(\tilde{x}_j^{(l)} \right) = \begin{cases} 0 & \text{if } \tilde{x}_j^{(l)} < 0 \\ 1 & \text{if } \tilde{x}_j^{(l)} > 0 \end{cases}$$

then

$$\begin{aligned} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} &= \begin{bmatrix} R' \left(\tilde{x}_1^{(l)} \right) \mathbb{I}_{1,1} & 0 & \dots & 0 \\ 0 & R' \left(\tilde{x}_2^{(l)} \right) \mathbb{I}_{2,2} & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & R' \left(\tilde{x}_{d_l}^{(l)} \right) \mathbb{I}_{d_l,d_l} \end{bmatrix} \\ &= \mathbb{I} \left(R' \left(\tilde{x}^{(l)} \right) \right) \in \mathbb{R}^{d_l \times d_l} \end{aligned}$$

where $\mathbb{I} \in \mathbb{R}^{d_l \times d_l}$ is the identity matrix, and

$$\mathbb{I}_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

iv. The gradient of the Linear module for layer l, with respect to $x_j^{(l-1)}$ is given by

$$\begin{aligned} \frac{\partial \tilde{x}_i^{(l)}}{\partial x_j^{(l-1)}} &= \frac{\partial}{\partial x_j^{(l-1)}} \left(W_i^{(l)} x_i^{(l-1)} + b_i^{(l)} \right) \\ &= \frac{\partial \tilde{x}_i^{(l)}}{\partial x_j^{(l-1)}} = W_{i,j}^{(l)} \in \mathbb{R} \end{aligned}$$

Hence

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

v. The gradient of the Linear module for layer l, with respect to $W_{j,k}$ is given by

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{j,k}^{(l)}} = \frac{\partial}{\partial W_{j,k}^{(l)}} \left(W_i^{(l)} x_i^{(l-1)} + b_i^{(l)} \right)$$

This gives

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{j,k}^{(l)}} = \begin{cases} x_i^{(l-1)} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Therefore

$$\frac{\partial \tilde{x}^{(l)}}{\partial W_{j,k}^{(l)}} = \begin{bmatrix} \frac{\partial \tilde{x}_1}{\partial W_{j,k}} \\ \vdots \\ \frac{\partial \tilde{x}_{d_l}}{\partial W_{j,k}} \end{bmatrix} \in \mathbb{R}^{(d_l \times 1)}$$

were

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{j,k}^{(l)}} = \begin{cases} x_i^{(l-1)} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

It follows that

$$\frac{\partial \tilde{x}^{(l)}}{\partial W_j^{(l)}} = \begin{bmatrix} \frac{\partial \tilde{x}_1}{\partial W_{j,1}} & \cdots & \frac{\partial \tilde{x}_{d_{(l-1)}}}{\partial W_{j,1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \tilde{x}_1}{\partial W_{j,d_l}} & \cdots & \frac{\partial \tilde{x}_{d_{(l-1)}}}{\partial W_{j,d_l}} \end{bmatrix} \in \mathbb{R}^{d_l \times d_{(l-1)}}$$

Hence

$$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \begin{bmatrix} \begin{bmatrix} \frac{\partial \tilde{x}_1}{\partial W_{1,1}} & \cdots & \frac{\partial \tilde{x}_{d_{(l-1)}}}{\partial W_{1,1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \tilde{x}_1}{\partial W_{1,d_l}} & \cdots & \frac{\partial \tilde{x}_{d_{(l-1)}}}{\partial W_{1,d_l}} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \frac{\partial \tilde{x}_1}{\partial W_{d_l,1}} & \cdots & \frac{\partial \tilde{x}_{d_{(l-1)}}}{\partial W_{d_l,1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \tilde{x}_1}{\partial W_{d_l,d_l}} & \cdots & \frac{\partial \tilde{x}_{d_{(l-1)}}}{\partial W_{d_l,d_l}} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} x_1^{(l-1)} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(l-1)} & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \cdots & 0 & x_{d_{(l-1)}}^{(l-1)} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & x_{d_{(l-1)}}^{(l-1)} \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{d_l \times d_l \times d_{(l-1)}}$$

were we assume that d_l is the dimensionality as d_l, purely for graphical reasons (to allow us to concisely illustrate the resulting tensor above)

vi. Finally, he gradient of the Linear module for layer l, with respect to b_j is given by

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial b_j^{(l)}} = \frac{\partial}{\partial b_j^{(l)}} \left(W_i^{(l)} x_i^{(l-1)} + b_i^{(l)} \right)$$

This gives

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial b_j^{(l)}} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Hence

$$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \mathbb{I} \in \mathbb{R}^{d_l \times d_l}$$

where $\mathbb{I} \in \mathbb{R}^{d_l \times d_l}$ is the identity matrix.

(b) We want to use the gradients derived above to compute the following

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

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

Computing these gives

i. Backpropagating the output gradient to the input parameters of the Softmax module gives

$$\begin{aligned}
\frac{\partial L}{\partial \tilde{x}^{(N)}} &= \frac{\partial L}{\partial x^{(N)}} \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} \\
&= \frac{\partial L}{\partial x^{(N)}} = \left(-\frac{t_j}{x^{(N)}} \right)_{j=1}^{d_N} \left(x_i^{(N)} \left(\mathbb{I}_{i,j} - x_j^{(N)} \right) \right)_{i,j=1}^{d_N, d_N} \\
&= \begin{bmatrix} -\frac{t_1}{x_1^{(N)}} & \cdots & -\frac{t_{d_N}}{x_{d_N}^{(N)}} \end{bmatrix} \begin{bmatrix} x_1^{(N)} \left(1 - x_1^{(N)}\right) & -x_2^{(N)} x_1^{(N)} & \cdots & -x_{d_N}^{(N)} x_1^{(N)} \\ -x_1^{(N)} x_2^{(N)} & x_2^{(N)} \left(1 - x_2^{(N)}\right) & \cdots & -x_{d_N}^{(N)} x_2^{(N)} \\ \vdots & \ddots & \ddots & \vdots \\ -x_1^{(N)} x_{d_N}^{(N)} & \cdots & \cdots & x_{d_N}^{(N)} \left(1 - x_{d_N}^{(N)}\right) \end{bmatrix} \\
&= \begin{bmatrix} -x_1^{(N)} \sum_i^{d_N} \left(\frac{t_i}{x_i} \left(\mathbb{I}_{i,1} - x_i \right) \right) & \cdots & -x_{d_N}^{(N)} \sum_i^{d_N} \left(\frac{t_i}{x_i} \left(\mathbb{I}_{i,d_N} - x_i \right) \right) \end{bmatrix} \\
&= \left(-x_j^{(N)} \sum_{i=1}^{d_N} \frac{t_i}{x_i} \left(\mathbb{I}_{i,j} - x_i^{(N)} \right) \right)_{j=1}^{d_N} \in 1 \times \mathbb{R}^{d_N}
\end{aligned}$$

where

$$t_i = \begin{cases} 1 & \text{if } t_i = \text{argmax } t \\ 0 & \text{otherwise} \end{cases}$$

ii. Backpropagating the output gradient to the input parameters of the $ReLU$ module of layer $l < N$ gives

$$\begin{aligned}\frac{\partial L}{\partial \tilde{x}^{(l<N)}} &= \frac{\partial L}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} \\ &= \frac{\partial L}{\partial x^{(l)}} \left(\mathbb{I} \left(R' \left(\tilde{x}^{(l)} \right) \right) \right) \\ &= \frac{\partial L}{\partial x^{(l)}} \begin{bmatrix} R' \left(\tilde{x}_1^{(l)} \right) \mathbb{I}_{1,1} & 0 & \dots & 0 \\ 0 & R' \left(\tilde{x}_2^{(l)} \right) \mathbb{I}_{2,2} & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & R' \left(\tilde{x}_{d_l}^{(l)} \right) \mathbb{I}_{d_l,d_l} \end{bmatrix}\end{aligned}$$

were

$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} \in \mathbb{R}^{1 \times d_{(l<N)}}$$

iii. Backpropagating the output gradient to the input parameter $x^{(l)}$ of the Linear module of layer $l+1$

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

Now if $N = l+1$, we get

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

This gives

$$\frac{\partial L}{\partial x^{(l)}} = \left(-x_j^{(N)} \sum_{i=1}^{d_N} \frac{t_i}{x_i} \left(\mathbb{I}_{i,j} - x_i^{(N)} \right) \right)_{j=1}^{d_N} \times W^{(N)}$$

Therefore

$$\begin{aligned}\frac{\partial L}{\partial \tilde{x}^{(l<N)}} &= \left(-x_j^{(N)} \sum_{i=1}^{d_N} \frac{t_i}{x_i} \left(\mathbb{I}_{i,j} - x_i^{(N)} \right) \right)_{j=1}^{d_N} \times W^{(N)} \times \left(\mathbb{I} \left(R' \left(\tilde{x}^{(l)} \right) \right) \right) \\ &= \left[-x_1^{(N)} \sum_i^{d_N} \left(\tfrac{t_i}{x_i} \left(\mathbb{I}_{i,1} - x_i \right) \right) \quad \dots \quad -x_{d_N}^{(N)} \sum_i^{d_N} \left(\tfrac{t_i}{x_i} \left(\mathbb{I}_{i,d_N} - x_i \right) \right) \right] \begin{bmatrix} W_{1,1} & \dots & W1, d_{(N-1)} \\ \vdots & \ddots & \vdots \\ W_{d_N,1} & \dots & W(d_N, d_{(N-1)} \end{bmatrix} \\ &\quad \begin{bmatrix} R' \left(\tilde{x}_1^{(l)} \right) \mathbb{I}_{1,1} & 0 & \dots & 0 \\ 0 & R' \left(\tilde{x}_2^{(l)} \right) \mathbb{I}_{2,2} & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & R' \left(\tilde{x}_{d_l}^{(l)} \right) \mathbb{I}_{d_l,d_l} \end{bmatrix}\end{aligned}$$

If $l < N-1$ and $l+n = N$, we get

$$\frac{\partial L}{\partial x^{(l)}} = \left(\frac{\partial L}{\partial \tilde{x}^{(N)}}\frac{\partial \tilde{x}^{(N)}}{\partial x^{(N-1)}}\right)\left(\frac{\partial x^{(N-1)}}{\partial \tilde{x}^{(N-1)}}\frac{\partial \tilde{x}^{(N-1)}}{\partial x^{(N-2)}}\right)\cdots\left(\frac{\partial x^{(N-(n-1))}}{\partial \tilde{x}^{(N-(n-1))}}\frac{\partial \tilde{x}^{(N-(n-1))}}{\partial x^{(l)}}\right)$$

were each bracket reflects back-propagating the gradient to the feedforward linear module of the previous layer. We carry out the corresponding matrix multiplications to compute $\frac{\partial L}{\partial \tilde{x}^{(l<N)}}$, were

$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} \in \mathbb{R}^{1\times d_{(l<N)}}$$

iv. Backpropagating the output gradient to the input parameter $W^{(l)}$ of the Linear module of layer $l+1$

$$\begin{aligned}\frac{\partial L}{\partial W^{(l)}} &= \frac{\partial L}{\partial \tilde{x}^{(l)}}\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{x}^{(l)}}\begin{bmatrix}\begin{bmatrix} x_1^{(l-1)} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(l-1)} & 0 & \cdots & 0\end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \cdots & 0 & x_{d_{(l-1)}}^{(l-1)} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & x_{d_{(l-1)}}^{(l-1)}\end{bmatrix}\end{bmatrix}\end{aligned}$$

Now

$$\frac{\partial L}{\partial \tilde{x}^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(N)}}\left(\frac{\partial \tilde{x}^{(N)}}{\partial x^{(N-1)}}\frac{\partial x^{(N-1)}}{\partial \tilde{x}^{(N-1)}}\right)\cdots\left(\frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}}\frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}}\right)$$

were each bracket reflects back-propagating the gradient to the feedforward $ReLU$ module of the previous layer. We carry out the corresponding matrix multiplications to compute $\frac{\partial L}{\partial W^{(l)}}$, were

$$\frac{\partial L}{\partial W^{(l)}} \in \mathbb{R}^{d_l \times d_{(l-1)}}$$

v. Finally, backpropagating the output gradient to the input parameter b of the Linear module of layer $l+1$

$$\begin{aligned}\frac{\partial L}{\partial b^{(l)}} &= \frac{\partial L}{\partial \tilde{x}^{(l)}}\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{x}^{(l)}}\left(\mathbb{I}\right)\end{aligned}$$

were $\frac{\partial L}{\partial \tilde{x}^{(l)}}$ is derived as shown in the previous gradient derivation in question part 1.1(b)(iv), and $\mathbb{I} \in \mathbb{R}^{d_l \times d_l}$ is the identity matrix. Therefore

$$\frac{\partial L}{\partial b^{(l)}} \in \mathbb{R}^{1\times d_l}$$

(c) If a batchsize of $B \neq 1$ is used, the input gets a batch dimension; that is, the input becomes a matrix M where columns represent the dimensions of each input sample and columns represent the different input samples in each batch, therefore $M \in \mathbb{R}^{B \times d_l}$, were B = batch size, and d_l is the dimensionality of the input. This will result in all backpropagation partial derivatives with respect to variables that depend on the input to have an additional batch dimension too.

1.2 NumPy Implementation

For this section I have implemented a multi-layer perceptron (MLP) using purely NumPy routines. I used cross-entropy (CE) as my loss function, stochastic gradient decent (SGD) as my optimization method, and the default parameters provided by the assignment to initialize and train this MLP. These are shown below in Table 1, and the corresponding accuracy and loss curves are illustrated below in Figure 1.

Hidden Layer Units	Learning Rate	Max No. of Steps	Batch Size	Evaluation Freq.
100	2e-3	1500	200	100

Table 1: The default parameter specifications for the NumPy Multi-layer Perceptron

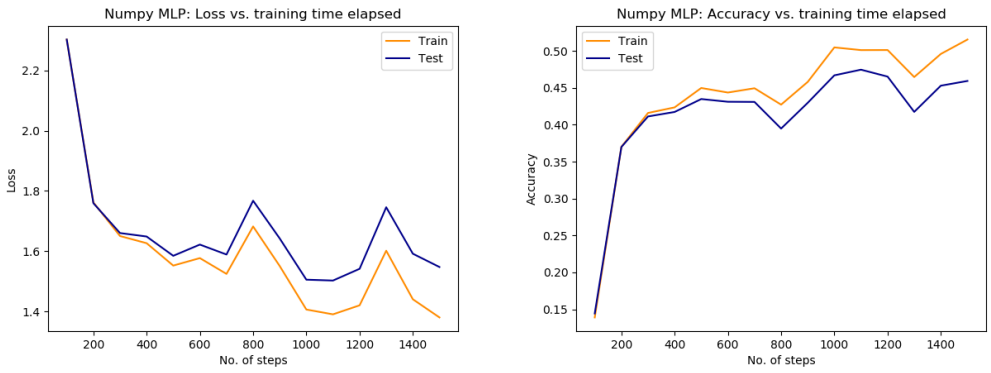

Figure 1: The Loss and Accuracy versus the training time elapsed for the Numpy MLP implementation, using the default parameters

Evaluation of the model was performed every 100 steps, were each step refers to performing SGD on one mini-batch from the training set, on the whole training set and test set. As we can see, improvement in both accuracy and loss follow the exact same pattern for the train set and test set, were the difference in performance of the model on the training set and on test set becoming gradually more pronounced as training time elapses. The model performs better on the training set, indicating that the model slightly overfits on the training set, as expected, since at each consecutive step the model is trained on an increasingly larger part of it. The final training set accuracy was approximately 52%, and test set accuracy was approximately 46%. Correspondingly, the final training set loss was 1.37 (to 2 d.p.), and the final test set loss was 1.54 (to 2 d.p.)

2 PyTorch MLP

For this section, I initially implemented a MLP using PyTorch using the default parameters provided. For this implementation, again I used cross-entropy (CE) as my loss function and stochastic gradient decent (SGD) as my optimization method. The default parameters for this implementation were the same as for the NumPy MLP implementation shown in Table 1. The corresponding accuracy and loss curves are illustrated below in Figure 2.

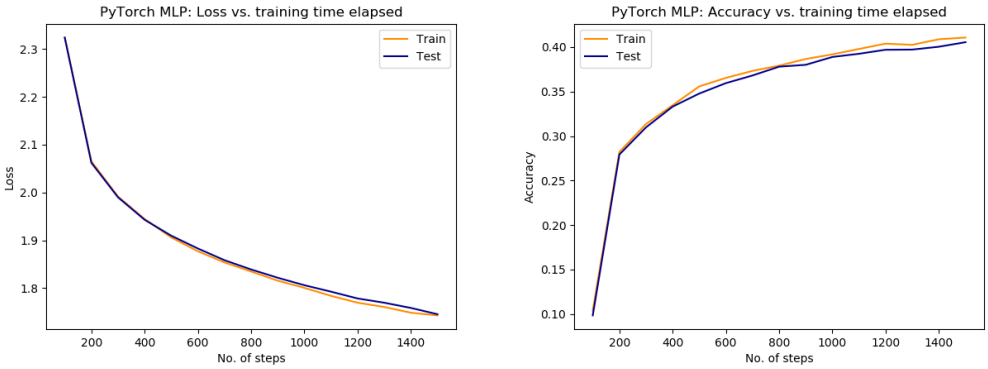

Figure 2: The Loss and Accuracy versus the training time elapsed for the PyTorch MLP implementation using the default parameters

Evaluation was performed in the exact same way as for the NumPy MLP implementation. As we can see in the graphs above, the changes in accuracy and loss show a similar overall pattern as in Figure 1 in the sense that the loss is decreasing, and the accuracy is increasing to similar values, however, there are no conspicuous fluctuations, displaying instead monotonic behaviour. The divergence between the train and test curve described in the previous section can also be detected, but it is much less pronounced. The final training set accuracy and loss were around 41% and 1.74 (to 2 d.p.) respectively, and for the test set the corresponding values were 40% and 1.75 (to 2 d.p.)

Subsequently, we were instructed to test different parameter configurations with to goal to achieve the best accuracy we can. I have tried a number of different combinations of parameter values using a brute force optimization algorithm. These are:

- **No. of hidden layers =** 1, 2, or 3
- **Hidden layer dimensionality =** 128, 256, 384, or 512
- **Learning Rate =** 1e-7, 1e-6 or 1e-5
- **Batch size =** 128, 256, or 384
- **Optimizer =** Adam optimizer, or SGD

I initially kept the maximum number of steps fixed at the default value (1500) and did not use regularization, to reduce the searchspace for the optimization algorithm which simply iterated over all possible combinations. After identifying the best performing combination, I added batch normalization and 1% dropout after each $ReLU$ module of the MLP, and increased the maximum number of steps to 5000. The rest of the resulting best setup parameters are shown below in Table 2. The corresponding accuracy and loss curves are illustrated below in Figure 3.

Hidden Layer Dimensions	Learning Rate	Max Number of Steps	Batch Size	Optimizer
512,128,384	1e-5	5000	384	Adam's

Table 2: The best parameter specifications for the PyTorch Multi-layer Perceptron

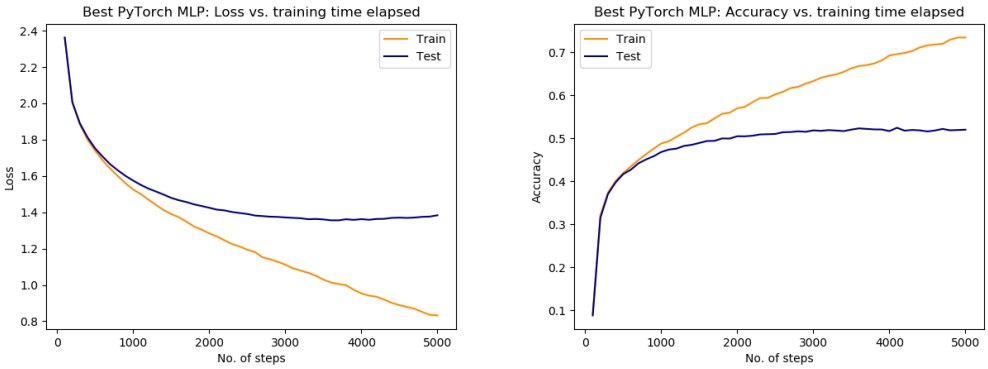

Figure 3: The Loss and Accuracy versus the training time elapsed for the PyTorch MLP implementation using the parameters found to give the best accuracy

Evaluation was performed in the same was as for the previous two MLP implementations. As we can see, the divergence between the performance of the model on the train and test sets increases dramatically as the number of max steps increases, indicating that significant overfitting occurs as more training time elapses. This could also be a result of adding more layers to the MLP, however, batch normalization and dropout should counter the effect to some extend. The behaviour of the curves again appears to be monotonic, and they have a smooth shape. As we can see the model performance appears to plateau after approximately 3000 steps whereas performance on the training set appears to increase in a steady rate after approximately 1500 steps. The final training set accuracy and loss were approximately 73% and 0.83 (to 2 d.p.), and for the test set the accuracy was 52.4 and the loss was 1.38 to (2 d.p.).

3 Custom Module: Batch Normalization

3.1 Automatic differentiation

In this section, we have implemented a custom module for batch normalization using automatic differentiation (Ioffe and Szegedy [2015]).

3.2 Manual implementation of backwards pass

In this section, we manually implemented the backward pass for the batch normalization module. I show the derivations in the section below.

(a) We have the following gradients to derive

$$\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \gamma}, \quad \frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \beta} \quad \text{and} \quad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

Computing these gives

i. We have

$$\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \gamma}$$

were

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

Now

$$\begin{aligned}\frac{\partial y_i^s}{\partial \gamma_j} &= \frac{\partial}{\partial \gamma_j}\left(\gamma_i \hat{x}_i^s + \beta_i\right) \\ &= \hat{x}_i^s \mathbb{I}_{i,j}\end{aligned}$$

were

$$\mathbb{I}_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Therefore

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

ii. We have

$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial \beta}$$

were

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

Now

$$\begin{aligned}\frac{\partial y_i^s}{\partial \beta_j} &= \frac{\partial}{\partial \beta_j}\left(\gamma_i \hat{x}_i^s + \beta_i\right) \\ &= \mathbb{I}_{i,j}\end{aligned}$$

were

$$\mathbb{I}_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Therefore

$$\begin{aligned}\left(\frac{\partial L}{\partial \beta}\right)_j &= \sum_s \sum_i \frac{\partial L}{\partial y} \left(\mathbb{I}_{i,j}\right) \\ &= \sum_s \frac{\partial L}{\partial y_j^s}\end{aligned}$$

iii. We have

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x}$$

were

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

Now

$$\frac{\partial y_i^s}{\partial x_j^r} = \frac{\partial y_i^s}{\partial \hat{x}_j^s} \frac{\partial \hat{x}_j^s}{\partial x_j^r}$$

were

$$\begin{aligned}\frac{\partial y_i^s}{\partial \hat{x}_j^s} &= \frac{\partial}{\partial \hat{x}_j^s}\left(\gamma_i \hat{x}_i^s + \beta_i\right) \\ &= (\mathbb{I}_{i,j})\gamma_i\end{aligned}$$

were

$$\mathbb{I}_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Now

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

were μ_j and σ_j^2 are the mean and variance of x^r across all the samples of the current minibatch, and with ϵ is a constant $\epsilon << 1$ to avoid numerical instability. These are given by

$$\mu_j = \frac{1}{B}\sum_{r=1}^{B} x_j^r \tag{1}$$

and

$$\sigma_j^2 = \frac{1}{B}\sum_{r=1}^{B}\left(x_j^r - \mu_j\right)^2 \tag{2}$$

Now

$$\begin{aligned}\frac{\partial}{\partial x_j^s}\left(\frac{x_j^r}{\sqrt{\sigma_j^2 + \epsilon}}\right) &= \frac{\partial}{\partial x_j^s}\left(x_j^r\left(\sigma_j^2 + \epsilon\right)^{-\frac{1}{2}}\right) \\ &= \mathbb{I}_{r,s}\left(\sigma_j^2 + \epsilon\right)^{-\frac{1}{2}} - \frac{x_j^r}{2}\left(\sigma_j^2 + \epsilon\right)^{-\frac{3}{2}}\left(\frac{\partial}{\partial x_j^s}\left(\sigma_j^2 + \epsilon\right)\right)\end{aligned}$$

Now we have

$$\frac{\partial}{\partial x_j^s}\left(\sigma_j^2+\epsilon\right)$$

Substituting in equation (2) gives

$$`\frac{\partial}{\partial x_j^s}\left(\sigma_j^2+\epsilon\right)=\frac{\partial}{\partial x_j^s}\left(\frac{1}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)^2+\epsilon\right)$$

Substituting in equation (1) gives

$$\begin{aligned}
`\frac{\partial}{\partial x_j^s}\left(\sigma_j^2+\epsilon\right) &= \frac{\partial}{\partial x_j^s}\left(\frac{1}{B}\sum_{r=1}^{B}\left(x_j^r-\frac{1}{B}\sum_{r=1}^{B}x_j^r\right)^2+\epsilon\right)\\
&= \frac{1}{B}\sum_{r=1}^{B}\left(\frac{\partial}{\partial x_j^s}\left(\left(x_j^r-\frac{1}{B}\sum_{r=1}^{B}x_j^r\right)^2+\epsilon\right)\right)\\
&= \frac{1}{B}\sum_{r=1}^{B}\left(2\left(x_j^r-\frac{1}{B}\sum_{r=1}^{B}x_j^r\right)\frac{\partial}{\partial x_j^s}\left(x_j^r-\frac{1}{B}\sum_{r=1}^{B}x_j^r\right)\right)\\
&= \frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\frac{1}{B}\sum_{r=1}^{B}x_j^r\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)\\
&= \frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)
\end{aligned}$$

So

$$\begin{aligned}
\frac{\partial}{\partial x_j^s}\left(\frac{x_j^r}{\sqrt{\sigma_j^2+\epsilon}}\right) &= \left(\sigma_j^2+\epsilon\right)^{-\frac{1}{2}}-\frac{x_j^r}{2}\left(\sigma_j^2+\epsilon\right)^{-\frac{3}{2}}\left(\frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)\right)\\
&= \frac{1}{\sqrt{\sigma_j^2+\epsilon}}-\frac{x_j^r}{2\left(\sigma_j^2+\epsilon\right)^{\frac{3}{2}}}\left(\frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)\right)
\end{aligned}$$

Therefore, at this stage we have

$$\frac{\partial\hat{x}_j^r}{\partial x_j^s}=\left(\frac{1}{\sqrt{\sigma_j^2+\epsilon}}-\frac{x_j^r}{2\left(\sigma_j^2+\epsilon\right)^{\frac{3}{2}}}\left(\frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)\right)\right)-\frac{\partial}{\partial x_j^s}\left(\frac{\mu_j}{\sqrt{\sigma_j^2+\epsilon}}\right)$$

Now

$$\begin{aligned}
\frac{\partial}{\partial x_j^s}\left(\frac{\mu_j}{\sqrt{\sigma_j^2+\epsilon}}\right) &= \frac{\partial}{\partial x_j^s}\left(\mu_j\left(\sigma_j^2+\epsilon\right)^{-\frac{1}{2}}\right)\\
&= \left(\frac{\partial}{\partial x_j^s}\left(\mu_j\right)\right)\left(\sigma_j^2+\epsilon\right)^{-\frac{1}{2}}-\frac{\mu_j}{2}\left(\sigma_j^2+\epsilon\right)^{-\frac{3}{2}}\left(\frac{\partial}{\partial x_j^s}\left(\sigma_j^2+\epsilon\right)\right)
\end{aligned}$$

As we have shown earlier

$$`\frac{\partial}{\partial x_j^s}\left(\sigma_j^2+\epsilon\right)=\frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)$$

So

$$\frac{\partial}{\partial x_j^s}\left(\frac{\mu_j}{\sqrt{\sigma_j^2+\epsilon}}\right)=\left(\frac{\partial}{\partial x_j^s}\left(\mu_j\right)\right)\left(\sigma_j^2+\epsilon\right)^{-\frac{1}{2}}-\frac{\mu_j}{2}\left(\sigma_j^2+\epsilon\right)^{-\frac{3}{2}}\frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)$$

Now

$$\begin{aligned}\frac{\partial}{\partial x_j^s}\left(\mu_j\right)&=\frac{\partial}{\partial x_j^s}\left(\frac{1}{B}\sum_{r=1}^{B}x_j^r\right)\\&=\frac{1}{B}\sum_{r=1}^{B}\frac{\partial}{\partial x_j^s}\left(x_j^r\right)\\&=\frac{1}{B}\sum_{r=1}^{B}\left(\mathbb{I}_{r,s}\right)\\&=\frac{1}{B}\end{aligned}$$

Therefore

$$\begin{aligned}\frac{\partial}{\partial x_j^s}\left(\frac{\mu_j}{\sqrt{\sigma_j^2+\epsilon}}\right)&=\frac{1}{B}\left(\sigma_j^2+\epsilon\right)^{-\frac{1}{2}}-\frac{\mu_j}{2}\left(\sigma_j^2+\epsilon\right)^{-\frac{3}{2}}\frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)\\&=\frac{1}{B\sqrt{\sigma_j^2+\epsilon}}-\frac{\mu_j}{2\left(\sigma_j^2+\epsilon\right)^{\frac{3}{2}}}\left(\frac{2}{B}\sum_{r=1}^{B}\left(x_j^r-\mu_j\right)\left(\mathbb{I}_{r,s}-\frac{1}{B}\right)\right)\end{aligned}$$

Thus

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

This gives

$$\frac{\partial y_i^s}{\partial x_j^r} = \frac{\partial y_i^s}{\partial \hat{x}_j^s}\frac{\partial \hat{x}_j^s}{\partial x_j^r}$$
$$= (\mathbb{I}_{i,j})\gamma_i \left\{ \left(\frac{B-1}{B}\right) \frac{1}{\sqrt{\sigma_j^2+\epsilon}} - \frac{x_j^r - \mu_j}{2\left(\sigma_j^2+\epsilon\right)^{\frac{3}{2}}} \left(\frac{2}{B} \sum_{r=1}^{B} \left(x_j^r - \mu_j\right) \left(\mathbb{I}_{r,s} - \frac{1}{B}\right) \right) \right\}$$

Therefore

$$\left(\frac{\partial L}{\partial x}\right)_j^r = \sum_s \sum_i \left(\frac{\partial L}{\partial y_i^s} (\mathbb{I}_{i,j})\gamma_i \left\{ \left(\frac{B-1}{B}\right) \frac{1}{\sqrt{\sigma_j^2+\epsilon}} - \frac{x_j^r - \mu_j}{2\left(\sigma_j^2+\epsilon\right)^{\frac{3}{2}}} \left(\frac{2}{B} \sum_{r=1}^{B} \left(x_j^r - \mu_j\right) \left(\mathbb{I}_{r,s} - \frac{1}{B}\right) \right) \right\} \right)$$
$$= \sum_s \left(\frac{\partial L}{\partial y_j^s} \gamma_j \left\{ \left(\frac{B-1}{B}\right) \frac{1}{\sqrt{\sigma_j^2+\epsilon}} - \frac{x_j^r - \mu_j}{2\left(\sigma_j^2+\epsilon\right)^{\frac{3}{2}}} \left(\frac{2}{B} \sum_{r=1}^{B} \left(x_j^r - \mu_j\right) \left(\mathbb{I}_{r,s} - \frac{1}{B}\right) \right) \right\} \right)$$

4 PyTorch CNN

Finally, for the last section, I have implemented a convolutional network (ConvNet) using PyTorch which is a small version of the popular VGG network (Simonyan and Zisserman [2015]). The specifications of the ConvNet architecture are shown in Table 1 in the assignment's pdf. Again, I used cross-entropy (CE) as my loss function, but contrary to the perviously implemented models, the default parameter for the optimizer for this neural network was Adam's optimizer. The rest of the parameters used were also the default parameters provided. These are shown below in Table 3, and the corresponding accuracy and loss curves are illustrated below in Figure 4.

Optimizer	**Learning Rate**	**Max No. of Steps**	**Batch Size**	**Evaluation Freq.**
Adam's	1e-4	5000	32	500

Table 3: The default parameter specifications for the PyTorch ConvNet.

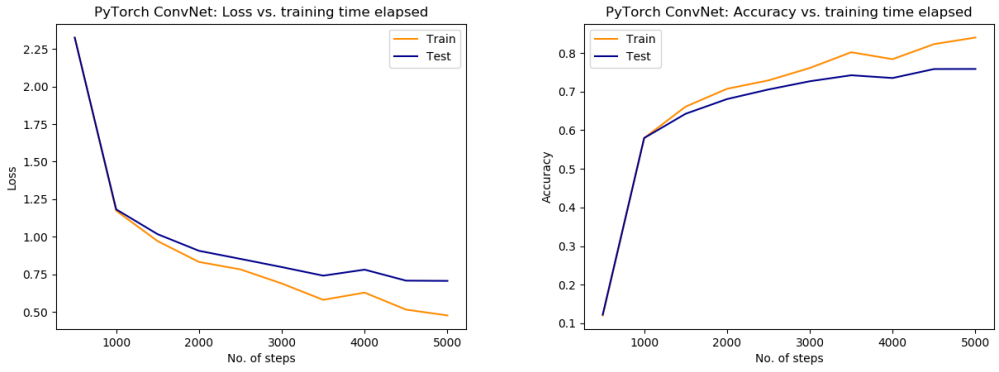

Figure 4: The Loss and Accuracy versus the training time elapsed for the PyTorch ConvNet implementation using the default parameters

Evaluation was performed every 500 training steps for this model, again by evaluating the model on the entire training and test sets. As we have seen in all the models, the curves of the training and

test sets follow the same pattern, however there as training time elapses, a divergence between the performance of the training and test sets increases in a steady rate, which I postulate is a result of overfitting. An important observation to note is that the divergence between the curves in each plot is much less pronounced compared to the PyTorch MLP implementation when using the best parameters found by the brute force optimizer. Increadibly, our final training set accuracy is approximately 84% and the test set accuracy is approximately 75%. The training set loss is 0.48, and test set loss 0.71 to (2 d.p.). We expected the ConvNet model to outperform the MLP models because ConvNets are explicitly constructed so as to maintain spatial information by using localized filters instead of having fully connected layers throughout the neural net which discard any spatial information.

Conclusion

In this assignment we have designed and implemented MLPs and a CNN in a 10-way image classification task. We have found that CNNs outperform the MLP models we used, one of which was optimized using a brute force optimization algorithm to find the best parameters. We have seen that the difference in performance on the testing set between the two architecture families was huge, and be amounted this to the architecture of the CNN which is explicitly designed to maintain spatial information.

References

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015. URL `http://jmlr.org/proceedings/papers/v37/ioffe15.pdf`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.