# Assignment 1: MLPs, CNNs and Backpropagation

**Steven van de Graaf**
12234036
steven.vandegraaf@student.uva.nl

## 1 MLP backprop and NumPy implementation

### 1.1 Analytical derivation of gradients

**Question 1.1 a)**

1. For this gradient, we consider:

$$\frac{\partial L}{\partial x^{(N)}} = \frac{\partial}{\partial x^{(N)}} \left( -\sum_i t_i \log x_i^{(N)} \right), \tag{1}$$

where

$$\left( \frac{\partial}{\partial x^{(N)}} \left( -\sum_i t_i \log x_i^{(N)} \right) \right)_j = \frac{\partial}{\partial x_j^{(N)}} \left( -\sum_i t_i \log x_i^{(N)} \right) = -\frac{t_j}{x_j^{(N)}}, \tag{2}$$

as for all $i \neq j$, $x_i^{(N)}$ does not depend on $x_j^{(N)}$, and, as such, its derivative with respect to $x_j^{(N)}$ will be zero. As such, the gradient is given by:

$$\frac{\partial L}{\partial x^{(N)}} = \left[ -\tfrac{t_i}{x_i^{(N)}} \right]_{i=1}^{d_N} = \begin{bmatrix} -\tfrac{t_1}{x_1^{(N)}} & -\tfrac{t_2}{x_2^{(N)}} & \cdots & -\tfrac{t_{d_N}}{x_{d_N}^{(N)}} \end{bmatrix} \in \mathbb{R}^{1 \times d_N} \tag{3}$$

2. For this gradient, we consider:

$$\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}, \tag{4}$$

where

$$\left( \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} \right)_{i,j} = \frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} = \frac{\partial}{\partial \tilde{x}_j^{(N)}} \left( \frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)} \right). \tag{5}$$

For $i = j$, we get:

$$\frac{\partial}{\partial \tilde{x}_j^{(N)}} = \frac{\exp\left(\tilde{x}_i^{(N)}\right) \cdot \left( \sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right) \right) - \exp\left(\tilde{x}_i^{(N)}\right) \cdot \frac{\partial}{\partial \tilde{x}_j^{(N)}} \left( \sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right) \right)}{\left( \sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right) \right)^2}, \tag{6}$$

As $\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)$ does not depend on $\tilde{x}_j^{(N)}$ for $j \neq k$, we get:

$$\begin{aligned}
\frac{\partial}{\partial \tilde{x}_j^{(N)}} &= \frac{\exp\left(\tilde{x}_i^{(N)}\right) \cdot \left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)\right) - \exp\left(\tilde{x}_i^{(N)}\right) \exp\left(\tilde{x}_j^{(N)}\right)}{\left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)\right)^2} \\
&= \frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)} - \frac{\exp\left(\tilde{x}_i^{(N)}\right) \exp\left(\tilde{x}_j^{(N)}\right)}{\left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)\right)^2} \\
&= x_i^{(N)} - \frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)} \frac{\exp\left(\tilde{x}_j^{(N)}\right)}{\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)} \\
&= x_i^{(N)} - x_i^{(N)} x_j^{(N)} = x_i^{(N)} \cdot \left(1 - x_j^{(N)}\right). \qquad (7)
\end{aligned}$$

For $i \neq j$, we get:

$$\begin{aligned}
\frac{\partial}{\partial \tilde{x}_j^{(N)}} &= \frac{-\exp\left(\tilde{x}_i^{(N)}\right) \cdot \frac{\partial}{\partial \tilde{x}_j^{(N)}} \left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)\right)}{\left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)\right)^2} \\
&= -\frac{\exp\left(\tilde{x}_i^{(N)}\right) \cdot \exp\left(\tilde{x}_j^{(N)}\right)}{\left(\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)\right)^2} \\
&= -\frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)} \cdot \frac{\exp\left(\tilde{x}_j^{(N)}\right)}{\sum_{k=1}^{d_N} \exp\left(\tilde{x}_k^{(N)}\right)} = -x_i^{(N)} \cdot x_j^{(N)} \qquad (8)
\end{aligned}$$

Combining the results from Eqns. 7 and 8, we get:

$$\left(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\right)_{i,j} = x_i^{(N)} \cdot \left(\mathbb{1}(i = j) - x_j^{(N)}\right). \qquad (9)$$

Finally, we get:

$$\begin{aligned}
\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} &= \left[x_i^{(N)} \cdot \left(\mathbb{1}(i = j) - x_j^{(N)}\right)\right]_{i,j=1}^{d_N, d_N} \\
&= \begin{bmatrix}
x_1^{(N)} \cdot \left(1 - x_1^{(N)}\right) & x_1^{(N)} \cdot \left(-x_2^{(N)}\right) & \cdots & x_1^{(N)} \cdot \left(-x_{d_N}^{(N)}\right) \\
x_2^{(N)} \cdot \left(-x_1^{(N)}\right) & x_2^{(N)} \cdot \left(1 - x_2^{(N)}\right) & \cdots & x_2^{(N)} \cdot \left(-x_{d_N}^{(N)}\right) \\
\vdots & \vdots & \ddots & \vdots \\
x_{d_N}^{(N)} \cdot \left(-x_1^{(N)}\right) & x_{d_N}^{(N)} \cdot \left(-x_2^{(N)}\right) & \cdots & x_{d_N}^{(N)} \cdot \left(1 - x_{d_N}^{(N)}\right)
\end{bmatrix} \in \mathbb{R}^{d_N \times d_N}
\end{aligned} \qquad (10)$$

3. For this gradient, we consider:

$$\begin{aligned}
\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} &= \frac{\partial}{\partial \tilde{x}^{(l<N)}} \left(\max\left(0, \tilde{x}^{(l<N)}\right)\right) = \begin{cases} 1 & \text{if } \tilde{x}^{(l<N)} > 0 \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{bmatrix}
\mathbb{1}\left(\tilde{x}_1^{(l<N)} > 0\right) & 0 & \cdots & 0 \\
0 & \mathbb{1}\left(\tilde{x}_2^{(l<N)} > 0\right) & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \mathbb{1}\left(\tilde{x}_{d_l}^{(l<N)} > 0\right)
\end{bmatrix} \\
&= \operatorname{diag}\left(\mathbb{1}\left(\tilde{x}^{(l<N)} > 0\right)\right) \in \mathbb{R}^{d_l \times d_l}. \qquad (11)
\end{aligned}$$

Note that the derivative is not defined for $\tilde{x}^{(l<N)} = 0$, but we set it to $0$.

4. For this gradient, we consider:

$$\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} = \frac{\partial}{\partial x^{(l-1)}} \left( W^{(l)} x^{(l-1)} + b^{(l)} \right) = W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}. \tag{12}$$

5. For this gradient, we consider:

$$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \frac{\partial}{\partial W^{(l)}} \left( W^{(l)} x^{(l-1)} + b^{(l)} \right) = \begin{bmatrix} \frac{\partial \tilde{x}_1^{(l)}}{\partial W^{(l)}} \\ \frac{\partial \tilde{x}_2^{(l)}}{\partial W^{(l)}} \\ \vdots \\ \frac{\partial \tilde{x}_{d_l}^{(l)}}{\partial W^{(l)}} \end{bmatrix}, \tag{13}$$

where

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W^{(l)}} = \begin{bmatrix} \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{1,1}^{(l)}} & \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{1,2}^{(l)}} & \cdots & \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{1,d_{l-1}}^{(l)}} \\ \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{2,1}^{(l)}} & \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{2,2}^{(l)}} & \cdots & \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{2,d_{l-1}}^{(l)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{d_l,1}^{(l)}} & \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{d_l,2}^{(l)}} & \cdots & \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{d_l,d_{l-1}}^{(l)}} \end{bmatrix}, \tag{14}$$

where

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{j,k}^{(l)}} = \frac{\partial}{\partial W_{j,k}^{(l)}} \left( \sum_{m=1}^{d_l-1} W_{i,m} x_m^{(l-1)} + b_i^{(l)} \right) = \begin{cases} x_k^{(l-1)} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}. \tag{15}$$

As such, we get:

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W^{(l)}} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(l-1)} & x_2^{(l-1)} & \cdots & x_{d_{l-1}}^{(l-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \tag{16}$$

and

$$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \begin{bmatrix} \begin{bmatrix} x_1^{(l-1)} & x_2^{(l-1)} & \cdots & x_{d_{l-1}}^{(l-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(l-1)} & x_2^{(l-1)} & \cdots & x_{d_{l-1}}^{(l-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(l-1)} & x_2^{(l-1)} & \cdots & x_{d_{l-1}}^{(l-1)} \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{d_l \times (d_l \times d_{l-1})}. \tag{17}$$

6. For this gradient, we consider:

$$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial}{\partial b^{(l)}} \left( W^{(l)} x^{(l-1)} + b^{(l)} \right) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{d_l \times d_l}. \tag{18}$$

**Question 1.1 b)**

1. For this gradient, using the results of Eqns. 3 and 10, we consider:

$$
\begin{aligned}
\frac{\partial L}{\partial \tilde{x}^{(N)}} &= \frac{\partial L}{\partial x^{(N)}} \frac{\partial^{(N)}}{\partial \tilde{x}^{(N)}} \\
&= \begin{bmatrix} -\frac{t_1}{x_1^{(N)}} \\ -\frac{t_2}{x_2^{(N)}} \\ \vdots \\ -\frac{t_{d_N}}{x_{d_N}^{(N)}} \end{bmatrix}^T \cdot \begin{bmatrix} x_1^{(N)} \cdot \left(1 - x_1^{(N)}\right) & x_1^{(N)} \cdot \left(-x_2^{(N)}\right) & \cdots & x_1^{(N)} \cdot \left(-x_{d_N}^{(N)}\right) \\ x_2^{(N)} \cdot \left(-x_1^{(N)}\right) & x_2^{(N)} \cdot \left(1 - x_2^{(N)}\right) & \cdots & x_2^{(N)} \cdot \left(-x_{d_N}^{(N)}\right) \\ \vdots & \vdots & \ddots & \vdots \\ x_{d_N}^{(N)} \cdot \left(-x_1^{(N)}\right) & x_{d_N}^{(N)} \cdot \left(-x_2^{(N)}\right) & \cdots & x_{d_N}^{(N)} \cdot \left(1 - x_{d_N}^{(N)}\right) \end{bmatrix} \\
&= \begin{bmatrix} -\frac{t_1}{x_1^{(N)}} \cdot \left(x_1^{(N)} \cdot \left(1 - x_1^{(N)}\right) + x_1^{(N)} \cdot \left(-x_2^{(N)}\right) + \cdots + x_1^{(N)} \cdot \left(-x_{d_N}^{(N)}\right)\right) \\ -\frac{t_2}{x_2^{(N)}} \cdot \left(x_2^{(N)} \cdot \left(-x_1^{(N)}\right) + x_2^{(N)} \cdot \left(1 - x_2^{(N)}\right) + \cdots + x_2^{(N)} \cdot \left(-x_{d_N}^{(N)}\right)\right) \\ \vdots \\ \frac{t_{d_N}}{x_{d_N}^{(N)}} \cdot \left(x_{d_N}^{(N)} \cdot \left(-x_1^{(N)}\right) + x_{d_N}^{(N)} \cdot \left(-x_2^{(N)}\right) + \cdots + x_{d_N}^{(N)} \cdot \left(1 - x_{d_N}^{(N)}\right)\right) \end{bmatrix}^T \\
&= \begin{bmatrix} -\frac{t_1}{x_1^{(N)}} \cdot x_1^{(N)} \cdot \left(\left(1 - x_1^{(N)}\right) + \left(-x_2^{(N)}\right) + \cdots + \left(-x_{d_N}^{(N)}\right)\right) \\ -\frac{t_2}{x_2^{(N)}} \cdot x_2^{(N)} \cdot \left(\left(-x_1^{(N)}\right) + \left(1 - x_2^{(N)}\right) + \cdots + \left(-x_{d_N}^{(N)}\right)\right) \\ \vdots \\ -\frac{t_{d_N}}{x_{d_N}^{(N)}} \cdot x_{d_N}^{(N)} \cdot \left(\left(-x_1^{(N)}\right) + \left(-x_2^{(N)}\right) + \cdots + \left(1 - x_{d_N}^{(N)}\right)\right) \end{bmatrix}^T \\
&= \begin{bmatrix} -t_1 \cdot \left(1 - x_1^{(N)} - x_2^{(N)} - \cdots - x_{d_N}^{(N)}\right) \\ -t_2 \cdot \left(1 - x_1^{(N)} - x_2^{(N)} - \cdots - x_{d_N}^{(N)}\right) \\ \vdots \\ -t_{d_N} \cdot \left(1 - x_1^{(N)} - x_2^{(N)} - \cdots - x_{d_N}^{(N)}\right) \end{bmatrix}^T \\
&= \begin{bmatrix} -t_1 \cdot \left(1 - \sum_{i=1}^{d_N} x_i^{(N)}\right) \\ -t_2 \cdot \left(1 - \sum_{i=1}^{d_N} x_i^{(N)}\right) \\ \vdots \\ -t_{d_N} \cdot \left(1 - \sum_{i=1}^{d_N} x_i^{(N)}\right) \end{bmatrix}^T \in \mathbb{R}^{1 \times d_N} \qquad (19)
\end{aligned}
$$

2. For this gradient, using the results of Eqn. 11, we consider:

$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} = \frac{\partial L}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} = \frac{\partial L}{\partial x^{(l)}} \cdot \operatorname{diag}\left(\mathbb{1}\left(\tilde{x}^{(l)} > 0\right)\right) \in \mathbb{R}^{d_l \times d_l} \qquad (20)$$

3. For this gradient, using the results of Eqn. 12, we consider:

$$\frac{\partial L}{\partial x^{(l<N)}} = \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \cdot W^{(l+1)} \in \mathbb{R}^{1 \times d_l}. \qquad (21)$$

4. For this gradient, using the results of Eqn. 17, we consider:

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}}\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \begin{bmatrix} \frac{\partial L}{\partial \tilde{x}_1^{(l)}} \\ \frac{\partial L}{\partial \tilde{x}_2^{(l)}} \\ \vdots \\ \frac{\partial L}{\partial \tilde{x}_{d_l}^{(l)}} \end{bmatrix}^T \cdot \begin{bmatrix} \begin{bmatrix} x_1^{(l-1)} & x_2^{(l-1)} & \cdots & x_{d_{l-1}}^{(l-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(l-1)} & x_2^{(l-1)} & \cdots & x_{d_{l-1}}^{(l-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(l-1)} & x_2^{(l-1)} & \cdots & x_{d_{l-1}}^{(l-1)} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial \tilde{x}_1^{(l)}} \cdot x_1^{(l-1)} & \frac{\partial L}{\partial \tilde{x}_1^{(l)}} \cdot x_2^{(l-1)} & \cdots & \frac{\partial L}{\partial \tilde{x}_1^{(l)}} \cdot x_{d_{l-1}}^{(l-1)} \\ \frac{\partial L}{\partial \tilde{x}_2^{(l)}} \cdot x_1^{(l-1)} & \frac{\partial L}{\partial \tilde{x}_2^{(l)}} \cdot x_2^{(l-1)} & \cdots & \frac{\partial L}{\partial \tilde{x}_2^{(l)}} \cdot x_{d_{l-1}}^{(l-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial \tilde{x}_{d_l}^{(l)}} \cdot x_1^{(l-1)} & \frac{\partial L}{\partial \tilde{x}_{d_l}^{(l)}} \cdot x_2^{(l-1)} & \cdots & \frac{\partial L}{\partial \tilde{x}_{d_l}^{(l)}} \cdot x_{d_{l-1}}^{(l-1)} \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{1 \times d_l \times d_{l-1}} \quad (22)$$

5. For this gradient, using the results of Eqn. 18, we consider:

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}}\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \begin{bmatrix} \frac{\partial L}{\partial \tilde{x}_1^{(l)}} \\ \frac{\partial L}{\partial \tilde{x}_2^{(l)}} \\ \vdots \\ \frac{\partial L}{\partial \tilde{x}_{d_l}^{(l)}} \end{bmatrix}^T \cdot \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial L}{\partial \tilde{x}_1^{(l)}} \\ \frac{\partial L}{\partial \tilde{x}_2^{(l)}} \\ \vdots \\ \frac{\partial L}{\partial \tilde{x}_{d_l}^{(l)}} \end{bmatrix}^T = \frac{\partial L}{\partial \tilde{x}^{(l)}} \in \mathbb{R}^{1 \times d_l} \quad (23)$$

**Question 1.1 c)**

If a batchsize $B \neq 1$ is used, the backpropagation equations derived above change such that all gradients with respect to $x^{(l)}$, $\tilde{x}^{(l)}$ and $b^{(l)}$ are extended with an extra dimension of magnitude $B$, such that:

$$\frac{\partial L}{\partial \tilde{x}^{(N)}} \in \mathbb{R}^{1 \times d_l \times B},$$
$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} \in \mathbb{R}^{d_l \times d_l \times B},$$
$$\frac{\partial L}{\partial x^{(l<N)}} \in \mathbb{R}^{1 \times d_l \times B},$$
$$\text{and } \frac{\partial L}{\partial b^{(l)}} \in \mathbb{R}^{1 \times d_l \times B}.$$

As such, the losses of the $B$ input samples in the batch are stacked on top of each other.

### 1.2 NumPy Implementation

#### Question 1.2

Below, in Fig. 1, you'll find a plot of the accuracy-loss curves obtained for my implementation of the *Multi-Layer Perceptron* (MLP) in `Python`'s `NumPy`.

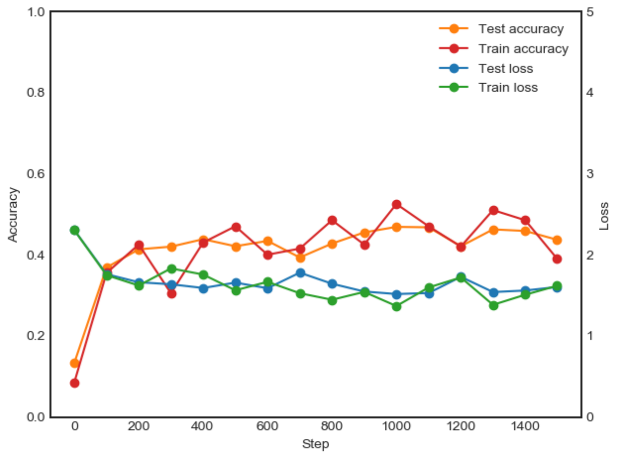

Figure 1: Plot of the accuracy-loss curves for the MLP as implemented in `NumPy`

These results were obtained using the *default* parameters, namely:

```
DNN_HIDDEN_UNITS_DEFAULT = '100'
LEARNING_RATE_DEFAULT = 2e-3
MAX_STEPS_DEFAULT = 1500
BATCH_SIZE_DEFAULT = 200
EVAL_FREQ_DEFAULT = 100
```

Below, in table 1, you'll find the best and last accuracies and losses obtained during training for my implementation of the MLP in `NumPy`:

## 2 PyTorch MLP

With my implementation of the MLP in `PyTorch`, in order to achieve the highest possible accuracy on the test, I tuned the hyperparameters by means of an exhaustive grid search, such that different combinations of settings for these hyperparameters were tested. For my initial exhaustive grid search, you will find the hyperparameters and their possible settings used below:

| Dataset | Metric | Best | Last |
|---|---|---|---|
| *Train* | Accuracy | 0.525 | 0.39 |
| | Loss | 1.367 | 1.616 |
| *Test* | Accuracy | 0.469 | 0.437 |
| | Loss | 1.515 | 1.602 |

Table 1: Results obtained during training for the MLP as implemented in `NumPy`

```
learning_rate = [2e-2, 2e-3, 2e-4]
max_steps = [1500, 3000]
batch_size = [100, 200, 400]
dnn_hidden_units = ['100', '200', '400', '100,100', '200,200', '400,400',
                    '100,100,100', '200,200,200', '400,400,400']
optimizer = ['SGD', 'Adam', 'Adadelta']
```

As such, in total $3 \cdot 2 \cdot 3 \cdot 9 \cdot 3 = 486$ different combinations of settings were tried and tested. Plots of the accuracies of various combinations of these settings can be found in appendix A, Fig. 6.

When comparing the three optimizers tested, it is interesting to note that both the `SGD` and `Adam` (except for when a single hidden layer is used) optimizers perform very poorly when a "large" learning rate of $0.02$ is used, but not `Adadelta`. Conversely, when a "small" learning rate of $0.0002$ is used, `Adadelta` performs relatively poorly, when compared to both `SGD` and `Adam`.

When comparing the three learning rates tested, it is interesting to note that for both the `SGD` and `Adam` optimizers, the smaller learning rates (i.e. $0.002$ and $0.0002$ perform better) than the larger learning rate, but not for `Adadelta`. The highest accuracies seem to be ontained when using the smallest learning rate, however, for both `SGD` and `Adam`.

When comparing the various compositions of hidden layers, the models that have more layers with more neurons per layer consistently outperform those with less layers or less neurons per layer.

Out of all of the $486$ models tested, the highest accuracy obtained on the test set was $51.92\%$, with the following combination of hyperparameter values:

```
learning_rate = 2e-4
max_steps = 3000
batch_size = 400
dnn_hidden_units = '400,400,400'
optimizer = 'Adam'
```

A plot of the accuracy-loss curves of this "best" model can be found below in Fig. 2.

As observed previously, for the various combinations of settings tried for these hyperparameters, there seems to be a positive correlation with the complexity of the model (in terms of its depth and width) and the performance obtained on the test set. As such, an additional grid search was performed to investigate whether even more complex (i.e. deeper) models would yield an even higher performance. As such, the following hyperparameter values were tried:

```
learning_rate = 2e-4
max_steps = 3000
batch_size = 400
dnn_hidden_units = ['400,400,400', '400,400,400,400', '400,400,400,400,400']
optimizer = 'Adam'
```

A plot of the accuracies obtained on the test set for the three models tried can be found below in Fig. 3

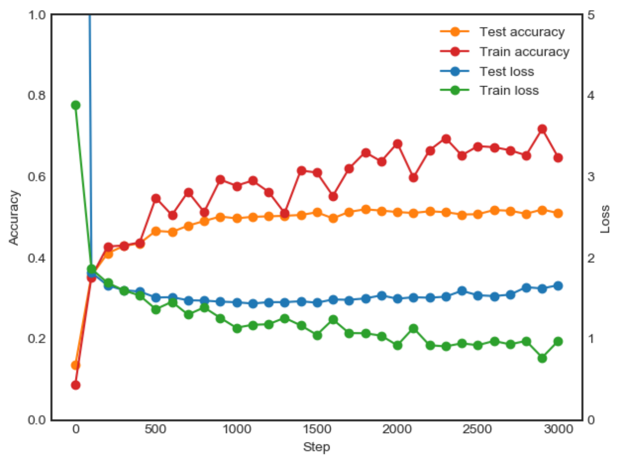

Figure 2: Plot of the accuracy-loss curves for the best-performing model (in terms of highest test accuracy obtained) of the initial exhaustive grid search

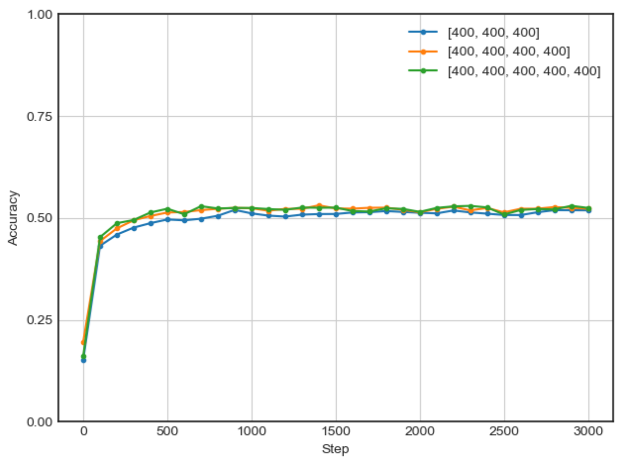

Figure 3: Plot of the accuracies for the models in the additional grid search

While the deepest model (i.e. `dnn_hidden_units = '400,400,400,400'`) achieved the highest end-of-training test set accuracy ($52.47\%$), the model which uses `dnn_hidden_units = '400,400,400'` achieves the highest test set accuracy overall during training ($53.11\%$). A plot of the accuracy-loss curves of this new "best" model can be found below in Fig. 4.

As can be observed from Fig. 4, this model does seem to overfit quite a bit, as the accuracy on the training data comes close to $100\%$ and the loss comes close to $0$. As such, the loss obtained on the test data increases (instead of decreases) for most of the training process.

## 3 Custom Module: Batch Normalization

### 3.1 Automatic differentiation

#### Question 3.1

Please see my `custom_batchnorm.py` for my implementation of the Batch Normalization operation.

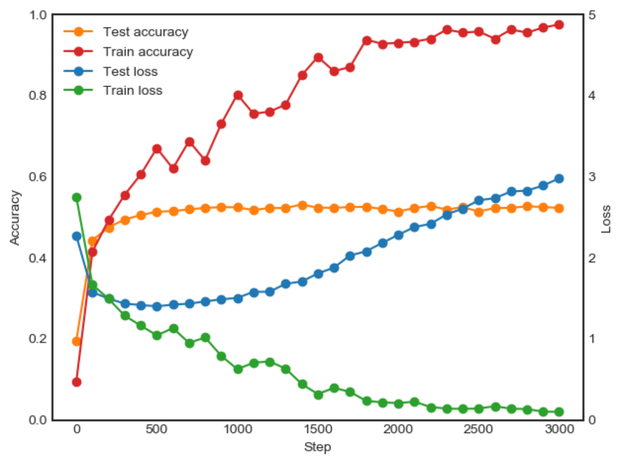

Figure 4: Plot of the accuracy-loss curves for the best-performing model (in terms of highest test accuracy obtained) of the additional grid search

### 3.2 Manual implementation of backwards pass

**Question 3.2 a)**

For the sake of my own sanity, I have not (yet) computed the backpropagation equations for the batch normalization operation.

**Question 3.2 b)**

Please see my `custom_batchnorm.py` for my implementation of the Batch Normalization autograd function. For the implementation of the backwards pass, I made use of this resource.

**Question 3.2 c)**

Please see my `custom_batchnorm.py` for my implementation of the Batch Normalization module.

## 4 PyTorch CNN

Below, in Fig. 5, you'll find a plot of the accuracy-loss curves obtained for my implementation of the *Convolutional Neural Network* (CNN) in `PyTorch`.

These results were obtained using the *default* parameters, namely:

```
LEARNING_RATE_DEFAULT = 1e-4
BATCH_SIZE_DEFAULT = 32
MAX_STEPS_DEFAULT = 5000
EVAL_FREQ_DEFAULT = 500
OPTIMIZER_DEFAULT = 'ADAM'
```

Below, in table 2, you'll find the best and last accuracies and losses obtained during training for my implementation of the CNN in `PyTorch`:

As can be reviewed from both Fig. 5 and table 2, this CNN clearly performs best out of all models and implementations tried and tested thus far, as expected.

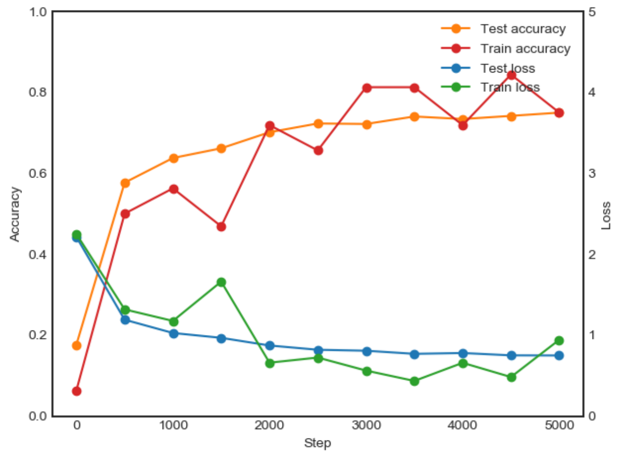

Figure 5: Plot of the accuracy-loss curves for the CNN as implemented in `PyTorch`

| **Dataset** | **Metric** | **Best** | **Last** |
|---|---|---|---|
| *Train* | Accuracy | 0.844 | 0.750 |
| | Loss | 0.433 | 0.941 |
| *Test* | Accuracy | 0.750 | 0.750 |
| | Loss | 0.748 | 0.748 |

Table 2: Results obtained during training for the CNN as implemented in `PyTorch`

## A Grid search plots

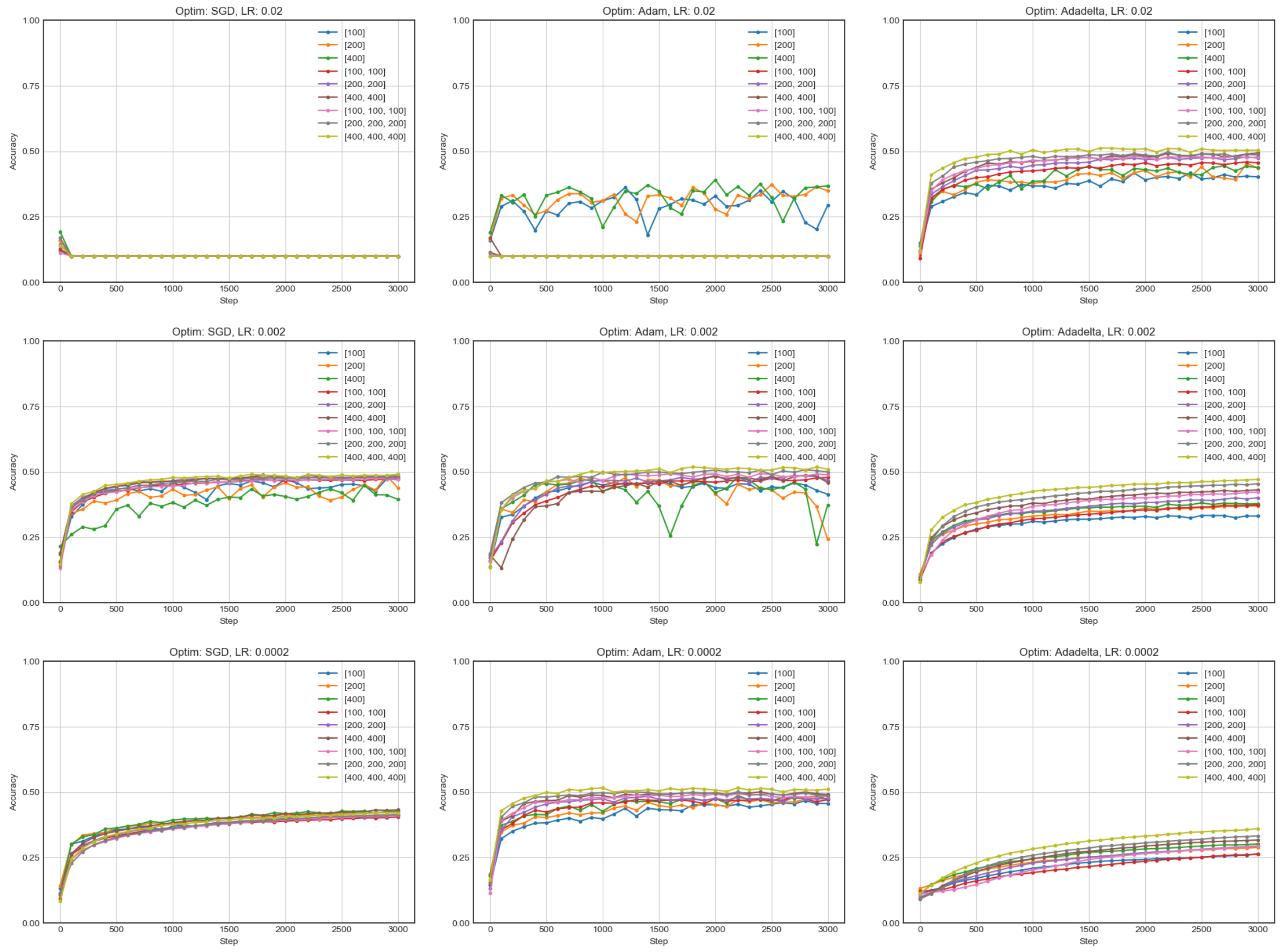

Figure 6: Plots of the accuracies obtained for various combinations of settings during the training of the MLP as implemented in `PyTorch`. For all plots, `max_steps = 3000`, and `batch_size = 400` were used.