First Assignment: Deep Neural Network / Backpropagation / Multi-layer perceptron

Daniel Jiménez - 1939216
Juan Mata Naranjo - 1939671
Alessandro Quattrociocchi - 1609286
Tansel Simsek - 1942297

Advanced Machine Learning
La Sapienza University of Rome
October 27, 2021

1 Question 2: Backpropagation

1.1 a

We start out by estimating the gradient of the loss function with respect to the vector of scores $z_i^{(3)}$

$$\begin{aligned}\frac{\partial J(\theta, \{x_i, y_i\}_{i=1}^N}{\partial z_i^{(3)}} &= \frac{\partial}{\partial z_i^{(3)}} \frac{1}{N} \sum_{i=1}^{N} -\log\left[\frac{\exp(z_i^{(3)})_{y_i}}{\sum_{j=1}^{k} \exp(z_i^{(3)})_j}\right] = \frac{1}{N} \frac{\partial}{\partial z_i^{(3)}} \left(-\log\left[\frac{\exp(z_i^{(3)})_{y_i}}{\sum_{j=1}^{k} \exp(z_i^{(3)})_j}\right]\right) = \\ &= \frac{1}{N} \frac{\partial}{\partial z_i^{(3)}} \left(-\log(\psi(z_i^{(3)}))\right) = -\frac{1}{N} \frac{1}{\psi(z_i^{(3)})} \frac{\partial \psi(z_i^{(3)})}{\partial z_i^{(3)}}\end{aligned} \tag{1}$$

where $\psi(\cdot)$ represents the softmax, a function that takes as input a vector and returns a vector of the same input dimension. The next step to complete the proof is to compute the gradient of $\psi(z_i^{(3)})$ with respect to $z_i^{(3)}$. This will be a vector, so for simplicity we will start by computing a single index of this gradient and then apply it over all the indexes (β represents a specific index of the $z_i^{(3)}$ vector, i.e. $\beta \in [1, k]$):

if $\mathbf{y_i} = \beta$:

$$\frac{\partial \psi(z_i^{(3)})_{y_i}}{\partial (z_i^{(3)})_\beta} = \frac{\exp(z_i^{(3)})_{y_i} \sum_{j=1}^k \exp(z_i^{(3)})_j - \exp(z_i^{(3)})_{y_i} \exp(z_i^{(3)})_\beta}{\left(\sum_{j=1}^k \exp(z_i^{(3)})_j\right)^2} =$$
$$= \frac{\exp(z_i^{(3)})_{y_i} [\sum_{j=1}^k \exp(z_i^{(3)})_j - \exp(z_i^{(3)})_\beta]}{\left(\sum_{j=1}^k \exp(z_i^{(3)})_j\right)^2} = \tag{2}$$
$$= \psi(z_i^{(3)})_{y_i} (1 - \psi(z_i^{(3)})_{y_i})$$

if $\mathbf{y_i} \neq \beta$:

$$\frac{\partial \psi(z_i^{(3)})_{y_i}}{\partial (z_i^{(3)})_\beta} = \frac{-\exp(z_i^{(3)})_{y_i} \exp(z_i^{(3)})_\beta}{\left(\sum_{j=1}^k \exp(z_i^{(3)})_j\right)^2} = \tag{3}$$
$$= -\psi(z_i^{(3)})_{y_i} \psi(z_i^{(3)})_\beta$$

Where $\psi(z_i^{(3)})_\beta$ denotes the softmax evaluated on the index β.

The key point here is to realize that:

$$\frac{\partial (\exp(z_i^{(3)})_{y_i})}{\partial (z_i^{(3)})_\beta} = \begin{cases} \exp(z_i^{(3)})_{y_i} & \text{if } y_i = \beta \\ 0 & \text{if } y_i \neq \beta \end{cases} \tag{4}$$

Combining both results together we get that:

$$\frac{\partial \psi(z_i^{(3)})_{y_i}}{\partial z_i^{(3)}{}_\beta} = \psi(z_i^{(3)})_{y_i} (\Delta_{y_i \beta} - \psi(z_i^{(3)})_\beta) \tag{5}$$

where $\Delta_{y_i,\beta}$ is the Kronecker delta.

Applying these results for the full gradient we get:

$$\frac{\partial J(\theta, \{x_i, y_i\}_{i=1}^N)}{\partial z_i^{(3)}} = -\frac{1}{N} \frac{1}{\psi(z_i^{(3)})} \frac{\partial \psi(z_i^{(3)})}{\partial z_i^{(3)}} = \frac{1}{N} \frac{1}{\psi(z_i^{(3)})} \psi(z_i^{(3)})(\psi(z_i^{(3)}) - \Delta_i) =$$
$$= \frac{1}{N} (\psi(z_i^{(3)}) - \Delta_i) \tag{6}$$

1.2 B

Following the same approach as before we will start by computing the scalar derivatives to later generalize to higher dimensions:

$$\frac{\partial J(\theta, \{x_i, y_i\}_{i=1}^N)}{\partial W^{(2)}} = \sum_{i=1}^N \frac{\partial J}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial W^{(2)}} = \sum_{i=1}^N \frac{1}{N} (\psi(z_i^{(3)}) - \Delta_i) \frac{\partial z_i^{(3)}}{\partial W^{(2)}} \tag{7}$$

The unknown term is the last partial derivative. What we know is that:

$$z^{(3)} = a^{(2)}W^{(2)} + b^{(2)} \tag{8}$$

$$\frac{\partial z^{(3)}}{\partial W^{(2)}} = \frac{\partial (a^{(2)}W^{(2)})}{\partial W^{(2)}} \tag{9}$$

Using β once again as a index holder of the vector $z^{(3)}$, assuming $a^{(2)}$ is a vector of length m, and using l, j as indexes to get the scalars that form the weight matrix we can derive the following expressions:

$$z_\beta^{(3)} = \sum_{k=1}^{m} a_k^{(2)} W_{\beta k}^{(2)} \tag{10}$$

$$\frac{\partial z_\beta^{(3)}}{\partial W_{lj}^{(2)}} = \sum_{k=1}^{m} a_k^{(2)} \frac{\partial W_{\beta k}^{(2)}}{\partial W_{lj}^{(2)}} = a_j^{(2)} \tag{11}$$

$$\frac{\partial J}{\partial W_{lj}^{(2)}} = \frac{\partial J}{\partial (z_i^{(3)})_\beta} \frac{\partial (z_i^{(3)})_\beta}{\partial W_{lj}^{(2)}} = \frac{\partial L}{\partial (z_i^{(3)})_\beta} a_j^{(2)} \tag{12}$$

In matrix form, the previous equation can be expressed in the following manner:

$$\frac{\partial J}{\partial W^{(2)}} = \begin{pmatrix} \frac{\partial J}{\partial (z_i^{(3)})_1} a_1^{(2)} & \frac{\partial J}{\partial (z_i^{(3)})_1} a_2^{(2)} & \cdots & \frac{\partial J}{\partial (z_i^{(3)})_1} a_m^{(2)} \\ \frac{\partial J}{\partial (z_i^{(3)})_2} a_1^{(2)} & \frac{\partial J}{\partial (z_i^{(3)})_2} a_2^{(2)} & \cdots & \frac{\partial J}{\partial (z_i^{(3)})_2} a_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial (z_i^{(3)})_k} a_1^{(2)} & \frac{\partial J}{\partial (z_i^{(3)})_k} a_2^{(2)} & \cdots & \frac{\partial J}{\partial (z_i^{(3)})_k} a_m^{(2)} \end{pmatrix} = \frac{\partial J}{\partial z_i^{(3)}} (a_i^{(2)})^T \tag{13}$$

This directly allows us to confirm that:

$$\frac{\partial J(\theta, \{x_i, y_i\}_{i=1}^N)}{\partial W^{(2)}} = \sum_{i=1}^{N} \frac{1}{N} (\psi(z_i^{(3)}) - \Delta_i)(a_i^{(2)})^T \tag{14}$$

If we include the regularization term we simply need to add the following term:

$$\frac{\partial}{\partial W^{(2)}} \lambda (||W^{(1)}||_2^2 + ||W^{(2)}||_2^2) = \frac{\partial}{\partial W^{(2)}} \lambda ||W^{(2)}||_2^2 \tag{15}$$

We compute this derivative by looking once again at the individual components of the derivative and then generalizing it to the final jacobian:

$$\frac{\partial ||W^{(2)}||_2^2}{\partial W_{ij}^{(2)}} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \frac{\partial}{\partial W_{ij}^{(2)}} (W_{pq}^{(2)})^2 = 2W_{ij}^{(2)} \tag{16}$$

Therefore, putting everything together we obtain:

$$\frac{\partial \tilde{J}}{\partial W^{(2)}} = \sum_{i=1}^{N} \frac{1}{N} (\psi(z_i^{(3)}) - \Delta_i)(a_i^{(2)})^T + 2\lambda W^{(2)} \tag{17}$$

1.3 C

In this section we will be using the chain rule to compute the remaining partial derivatives which will allow us to complete the back propagation of our neural network.

1.3.1 With respect to $b^{(2)}$

$$\begin{aligned}\frac{\partial \tilde{J}(\theta,\{x_i,y_i\}_{i=1}^N)}{\partial b^{(2)}} &= \frac{\partial J(\theta,\{x_i,y_i\}_{i=1}^N)}{\partial b^{(2)}} = \sum_{i=1}^N \frac{\partial J}{\partial z_i^{(3)}}\frac{\partial z_i^{(3)}}{\partial b^{(2)}} = \\ &= \sum_{i=1}^N \frac{\partial J}{\partial z_i^{(3)}}\frac{\partial}{\partial b^{(2)}}(W^{(2)}a^{(2)}+b^{(2)}) = \sum_{i=1}^N \frac{\partial J}{\partial z_i^{(3)}}\frac{\partial}{\partial b^{(2)}}(b^{(2)}) = \sum_{i=1}^N \frac{\partial J}{\partial z_i^{(3)}} = \\ &= \sum_{i=1}^N \frac{1}{N}(\psi(z_i^{(3)})-\Delta_i)\end{aligned} \tag{18}$$

Where we have used:

$$\frac{\partial b^{(2)}}{\partial b^{(2)}} = \left(\frac{\partial b^{(2)}}{\partial b^{(2)}}\right)_{ij} = \left(\frac{\partial b_i^{(2)}}{\partial b_j^{(2)}}\right) = \mathbb{1} \tag{19}$$

1.3.2 With respect to $W^{(1)}$

$$\frac{\partial \tilde{J}(\theta,\{x_i,y_i\}_{i=1}^N)}{\partial W^{(1)}} = \frac{\partial}{\partial W^{(1)}}\frac{1}{N}\sum_{i=1}^N -log(\Psi(z_i^{(3)})) + \frac{\partial}{\partial W^{(1)}}\lambda(||W^{(1)}||_2^2 + ||W^{(2)}||_2^2) = \frac{\partial J}{\partial W^{(1)}} + 2\lambda W^{(1)} \tag{20}$$

The first term is an unknown derivative which we need to estimate more carefully, however the second part is similar to what has been done in (15).

$$\frac{\partial J}{\partial W^{(1)}} = \frac{\partial J}{\partial z_i^{(2)}}\frac{\partial z_i^{(2)}}{\partial W^{(1)}} = \underbrace{\frac{\partial J}{\partial z_i^{(3)}}}_{1}\underbrace{\frac{\partial z_i^{(3)}}{\partial a_i^{(2)}}}_{2}\underbrace{\frac{\partial a_i^{(2)}}{\partial z_i^{(2)}}}_{3}\underbrace{\frac{\partial z_i^{(2)}}{\partial W^{(1)}}}_{4} \tag{21}$$

1:

$$\frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} = \frac{1}{N}(\psi(z_i^{(3)})-\Delta_i) \tag{22}$$

2:

$$\frac{\partial z_i^{(3)}}{\partial a_i^{(2)}} = \frac{\partial}{\partial a_i^{(2)}}(a^{(2)}W^{(2)}) \tag{23}$$

We know we can express the value of a single index of $z_i^{(3)}$ as:

$$(z_i^{(3)})_\beta = \sum_{j=1}^m a_j^{(2)}W_{j\beta}^{(2)} \tag{24}$$

$$\left(\frac{\partial z_i^{(3)}}{\partial a_i^{(2)}}\right)_{st} = \frac{\partial (z_i^{(3)})_s}{\partial (a_i^{(2)})_t} = \frac{\partial}{\partial (a_i^{(2)})_t} \sum_{j=1}^{m} (a_i^{(2)})_j W_{js}^{(2)} = \\ = \sum_{j=1}^{m} \frac{\partial (a_i^{(2)})_j}{\partial (a_i^{(2)})_t} W_{js}^{(2)} = W_{ts}^{(2)} \tag{25}$$

$$\frac{\partial z_i^{(3)}}{\partial a_i^{(2)}} = (W^{(2)})^T \tag{26}$$

3:

Similarly to previous steps we will compute the derivative of the individual elements and then generalize to the full matrix:

$$\left(\frac{\partial a_i^{(2)}}{\partial z_i^{(2)}}\right)_{st} = \frac{\partial (a_i^{(2)})_s}{\partial (z_i^{(2)})_t} = \frac{\partial}{\partial (z_i^{(2)})_t} \phi((z_i^{(2)})_s) = \\ = \begin{cases} \frac{\partial}{\partial (z_i^{(2)})_t} \phi((z_i^{(2)})_s) = \phi'((z_i^{(2)})_j) & \text{if } t = s \\ 0 & \text{if } t \neq s \end{cases} \tag{27}$$

So we can say that:

$$\frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} = \text{diag}(\phi'((z_i^{(2)}))) \tag{28}$$

where:

$$\phi'((z_i^{(2)}))_j = \begin{cases} 1 & \text{if } (z_i^{(2)})_j > 0 \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

4:

In combination with the previous part we can already assume to know:

$$\frac{\partial z_i^{(2)}}{\partial W^{(1)}} = (a_i^{(1)})^T \tag{30}$$

We finally conclude that:

$$\frac{\partial J}{\partial W^{(1)}} = \sum_{i=1}^{N} \frac{1}{N} (\psi(z_i^{(3)}) - \Delta_i)(W^{(2)})^T \text{diag}(\phi'(z_i^{(2)}))(a_i^{(1)})^T \tag{31}$$

and therefore:

$$\frac{\partial \tilde{J}}{\partial W^{(1)}} = \sum_{i=1}^{N} \frac{1}{N} (\psi(z_i^{(3)}) - \Delta_i)(W^{(2)})^T \text{diag}(\phi'(z_i^{(2)}))(a_i^{(1)})^T + 2\lambda W^{(1)} \tag{32}$$

In order to make the dimensions match we need to slightly change the order of operations:

$$\frac{\partial \tilde{J}}{\partial W^{(1)}} = \sum_{i=1}^{N} (a_i^{(1)})^T \frac{1}{N} (\psi(z_i^{(3)}) - \Delta_i)(W^{(2)})^T \text{diag}(\phi'(z_i^{(2)})) + 2\lambda W^{(1)} \tag{33}$$

It's also important to notice that this equation implies we need to construct an diagonal matrix from $\phi'(z_i^{(2)})$. In practice however this will implemented as a element-wise multiplication (see code for more details).

1.3.3 With respect to $b^{(1)}$

$$\begin{aligned} \frac{\partial \tilde{J}(\theta, \{x_i, y_i\}_{i=1}^N)}{\partial b^{(1)}} &= \frac{\partial}{\partial b^{(1)}} \frac{1}{N} \sum_{i=1}^{N} -log(\Psi(z_i^{(3)})) = \frac{\partial J(\theta, \{x_i, y_i\}_{i=1}^N)}{\partial b^{(1)}} = \\ &= \frac{\partial J}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial b^{(1)}} = \frac{\partial J}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} \mathbb{1} = \\ &= \sum_{i=1}^{N} \frac{1}{N} (\psi(z_i^{(3)}) - \Delta_i)(W^{(3)})^T \text{diag}(\phi'(z_i^{(2)})) \end{aligned} \tag{34}$$

2 Question 3: Stochastic gradient descent training

After implementing the Stochastic Gradient Descent (SGD) algorithm on the toy data set, we obtained the graph shown in Figure 1. As expected, the loss decreases along with the iterations. Approximately after 50 iterations, the loss is stable and very close to 0. The final loss is 0.0156 in the training data set.

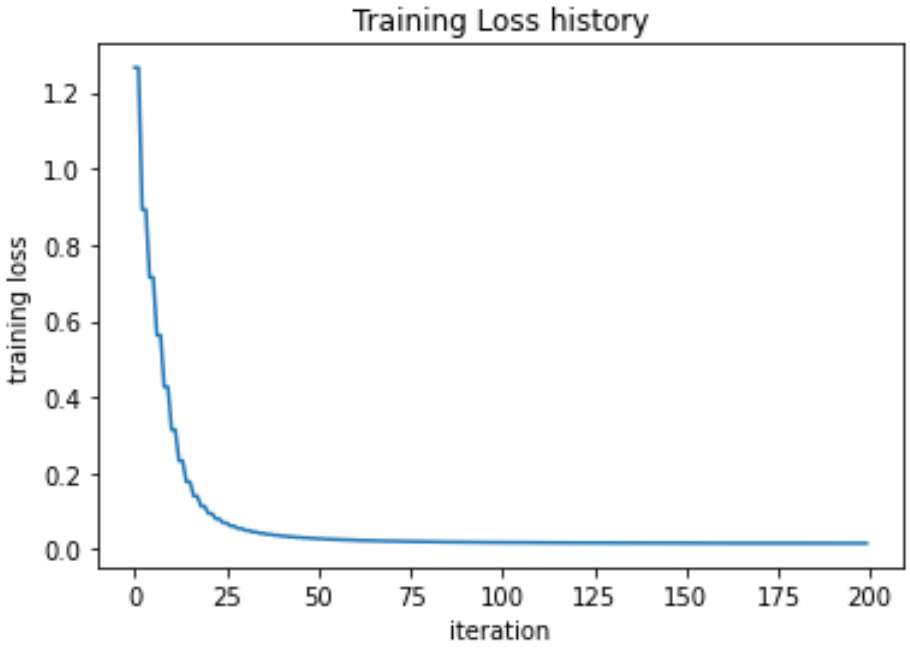


Figure 1: Training loss in gradient descent history toy data set

Later we applied the SGD to the CIFAR-10 data set. In the upper chart of Figure 2, we show the loss history. It goes decreasing, and finally, it arrived at 1.962184 after 2000 iterations. Nevertheless, the pace of how it decreases seems linear. The latter may indicate that the learning rate is too low.

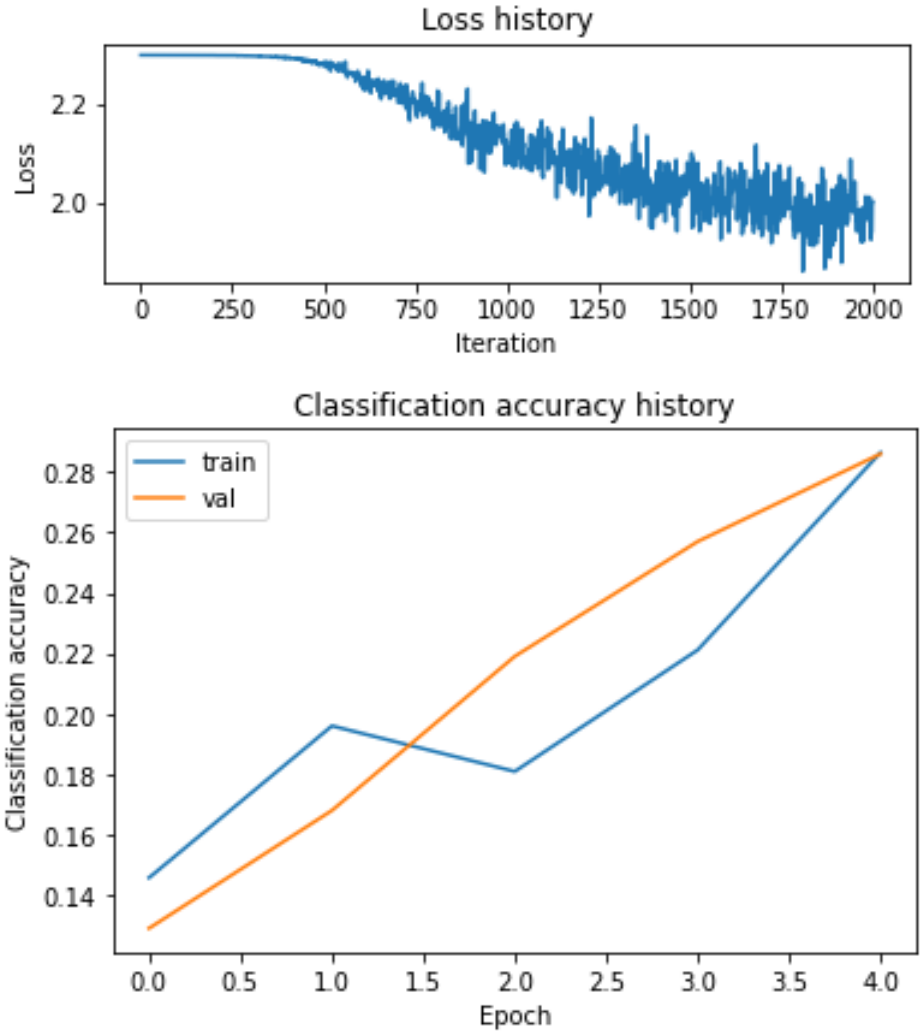


Figure 2: Training loss in gradient descent history / Accuracy across epochs

The lower part of Figure 2 demonstrates that the accuracy from train to validation is close. That also makes us change the size of the model.

The Accuracy in the validation data set that we obtained was 0.29. To improve that metric, we followed the next approach:

A Use Principal Components Analysis (PCA) to reduce the dimensionality of the data set. With this approach, we reduced the training time by taking only the variables that account for 95% of the variability. In our case, we finished working with 216 features (instead of 3072).

B Define ranges for the hyper-parameters. In our case, we defined the ranges and then, randomly, we drew 20 values for each hyper-parameters. The next table shows the ranges used for each hyper-parameter:

Hyper-parameter	Min	Max
Learning rate	-7	-2
Regularization Strength	-7	-2
Hidden Size	32	400
Training Epochs	100	500

C Define the complete grid with the hyper-parameters. It means to do the Cartesian product of all the items selected to get all the possible combinations of them. In the end, we obtained 160,000 combinations.

D Define a random grid by taking only 100 random combinations from the complete grid.

E Train the 100 neural networks with each combination of hyper-parameters and calculate the validation accuracy.

F Define the best neural network as the one that got the highest accuracy in the validation data set.

The previous process took approximately 6 hours to run. Moreover, the hyper-parameters that generated the best performance are shown in the next table:

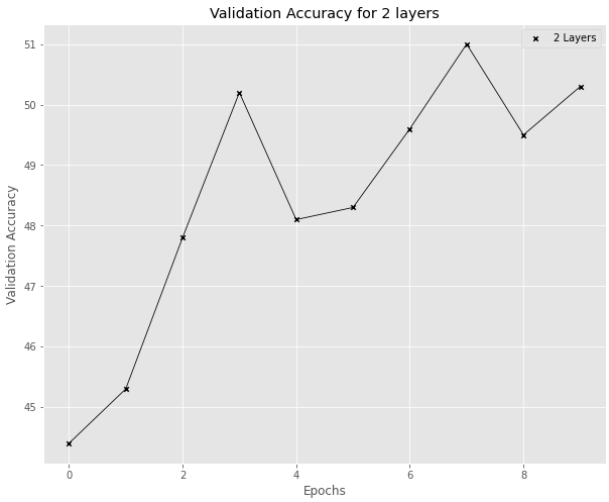
Hyper-parameter	Value
Learning rate	0.002239034272168369
Regularization Strength	0.00015834527427829734
Hidden Size	330
Training Epochs	412

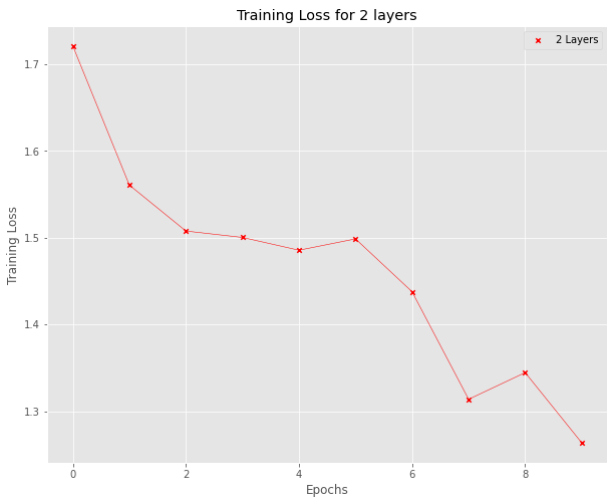
In the end, the performance of the best neural network is shown in the following table:

Metric	Training	Validation	Test
Accuracy	0.62718	0.551	0.551

3 Question 4: Implement multi-layer perceptron using PyTorch library

In this part, we report some of the results obtained by using the PyTorch package.
In particular, for the two-layers-network, we have obtained a validation accuracy of **50.3%** and test accuracy of **50.6%**.





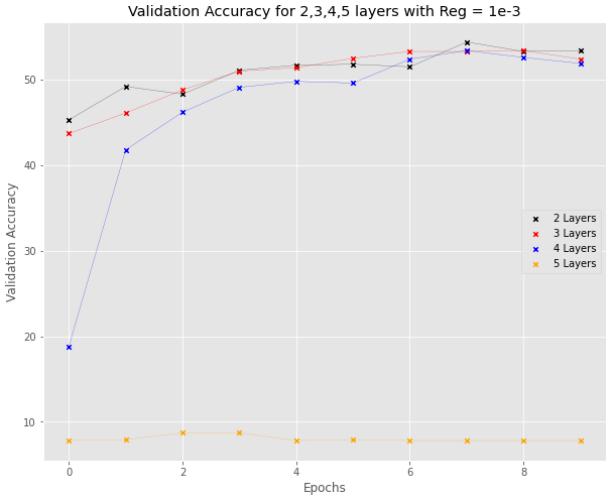
After verifying that the obtained accuracy met the condition [>48%], we extended the depth of the neural network by considering more layers and adding more neurons. Indeed, increasing the number of parameters guarantees that validation accuracy and test accuracy grow up to a certain value, while the training loss decreases the deeper the model is. The largest network we tested consists of five layers [200,150,100,80].

As can be seen from the table below, the accuracy value increases until we add the last layer, where the network stops learning and stops at the initial training loss value. The latter shows that a more complex and deeper network

does not always return good results, in general. That way, we made the network affected by gradient vanishing (widening it even more) or by overfitting.

In this example, as the number of parameters to be inferred increases, the network overfits our validation and test set data, so it is necessary to set a penalty score (regularisation parameter) that does not encourage the model to prefer more complex solutions.

Numbers of Layers	Validation Accuracy	Test Accuracy
2	53.4%	54.9%
3	52.4%	53.6%
4	51.5%	54.2%
5	7.8%	9%

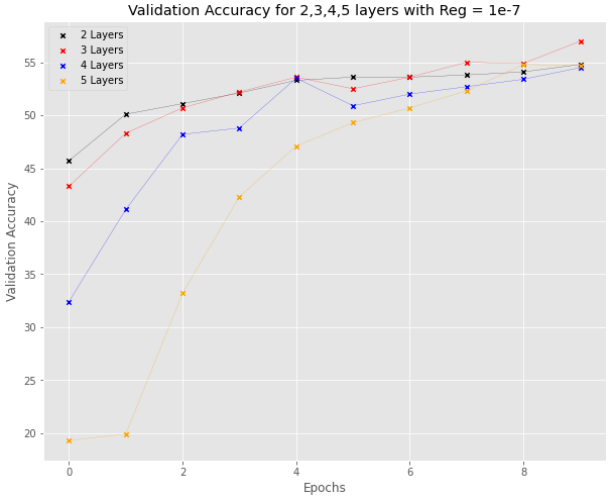


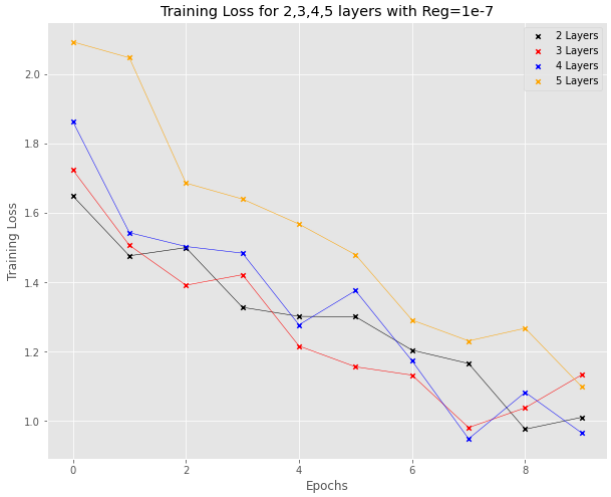
Looking at the network stops learning with 5 levels, we were presented with several possible ways forward:

- Change the number of neurons for each level.
- Increase the number of epochs.
- Decrease the network regularisation factor.
- Prune the network to maintain only the levels really necessary for the network's performance (NN pruning).

We decided to try one more regularisation parameter, which is lower than the one before. In this case, we were expecting to see lower validation and test accuracy. The latter because decreasing the penalisation factor (regularisation parameter) may elevate the complexity of the model. While the training accuracy gets higher, the test and validation accuracy can decrease (ie. overfitting). However, in this case, we can observe that the validation and test accuracy are still high which the model may not overfit the test and validation sets. By considering possible future data sets, we decided to keep the regularisation parameter as 1e-3, which will make the model more general.

Numbers of Layers	Validation Accuracy	Test Accuracy
2	54.8%	54.8%
3	57.0%	55.7%
4	54.5%	56.7%
5	54.7%	52.4%





In the worst-case scenario, setting reg = 0, the model would not suffer from any penalty factor, and it would prefer increasing the complexity of the model. For this reason, we decided to identify as the best model the one composed of 4 layers with 200, 150, 100 neurons. Since this model suffers from the initialization of the weight matrix, we saved it in the "model.ckpt" file for easier reproducibility of the result (setting the hidden_layer list equal to [200,150,100]). It can be verified by setting *train= False*, while *train= True* tests the basic model with only 2 hidden layers.