Deep Learning - 2023/2024
Homework 1

Instituto Superior Técnico

Group 10
97124 - Jiaqi Yu
99320 - Rodrigo Lopes

The coding exercises were done by both students, which were then compared. The best implementation was selected through collaboration on testing and bug-solving. Question number 3 was solved together. Both students contributed equally.

Question 1

1.

(a) Train 20 epochs of the perceptron on the training set and report its performance on the training, validation and test sets.

- **Training accuracy:** 0.4654; **Validation accuracy:** 0.461; **Test accuracy:** 0.3422

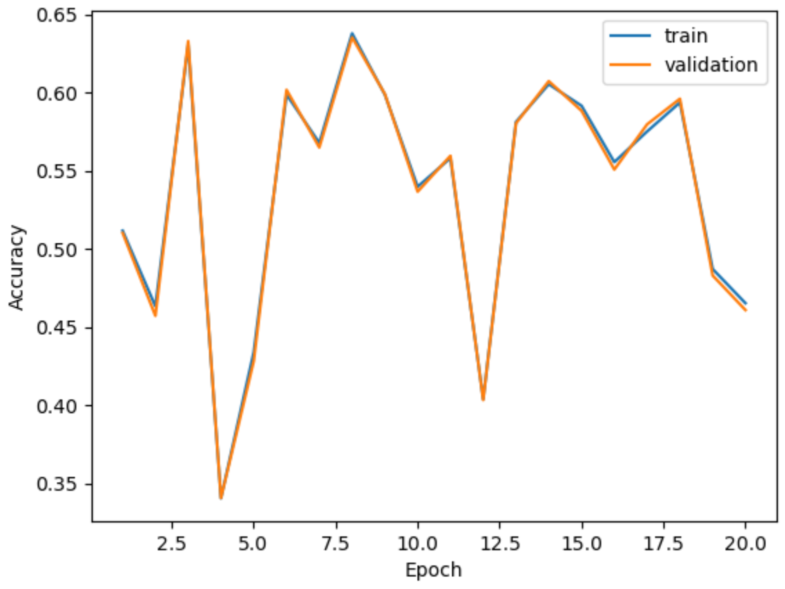

Figure 1: Plot of the train and validation accuracies

(b) Train 20 epochs of the Logistic regression, the models obtained using two different learning rates of $\eta = 0.01$ and $\eta = 0.001$.

- **Test accuracy (η = 0.01):** 0.5784

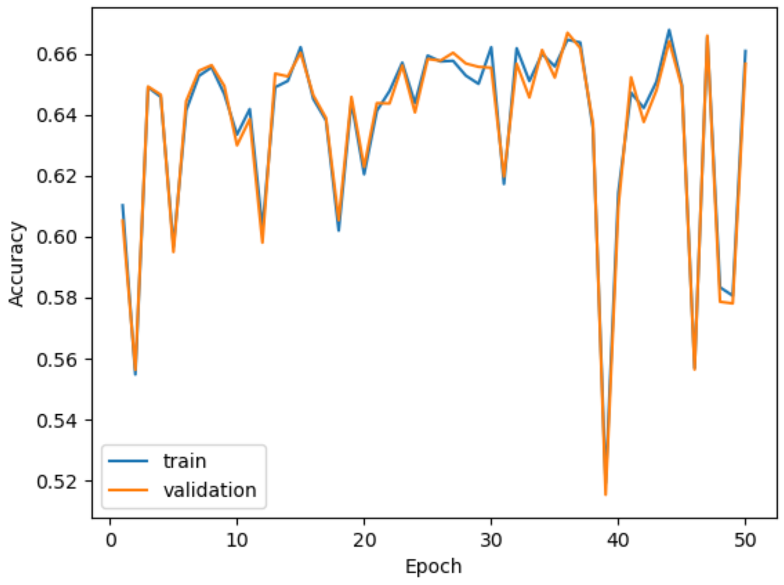

Figure 2: Plot of the train and validation accuracies using learning rate of $\eta = 0.01$

- **Test accuracy (η = 0.001):** 0.5936

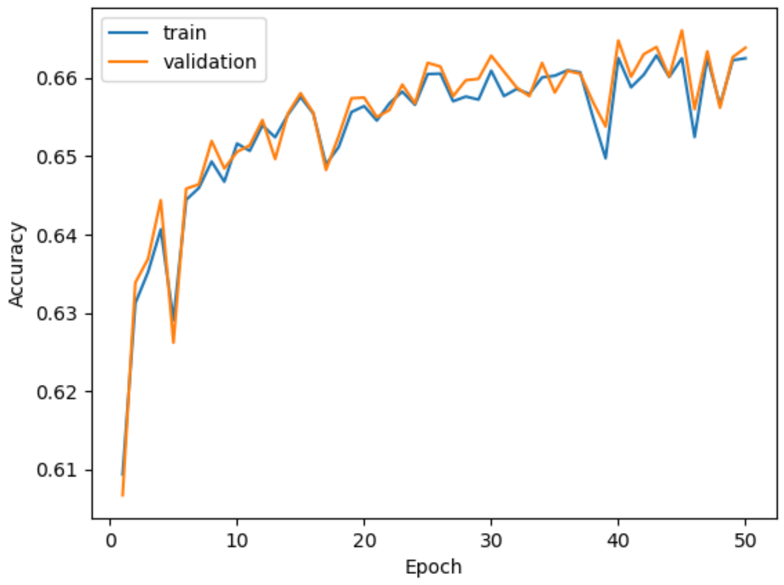

Figure 3: Plot of the train and validation accuracies using learning rate of $\eta = 0.001$

2.

(a) Comment the following claim and justify.

Is this true of false? This is true.

Logistic Regression excels at learning linear models. When utilizing pixel values as features, it can only establish linear decision boundaries, treating each pixel independently and missing out on intricate pixel relationships. The model's simplicity lies in its convex optimization nature, guaranteeing a single global minimum. Training is straightforward, making it an efficient choice for simpler tasks.

On the other hand, an MLP, especially with non-linear ReLU activations, is adept at learning internal representations. By employing ReLU at the end of each layer, the model captures intricate patterns and dependencies among pixels, allowing for more expressive representations. However, training an MLP involves non-convex optimization, introducing the challenge of multiple local minima. This complexity requires advanced techniques like gradient descent variants and careful initialization for successful convergence.

The key strength of an MLP with non-linear activations lies in its ability to discern complex internal representations. Unlike logistic regression, it can capture hierarchical features and correlations between input pixels, enabling it to make more sophisticated decisions.

In summary, while logistic regression is advantageous for its simplicity and convex optimization, it falls short in capturing complex relationships. The MLP, with the power of non-linear activations, excels in learning intricate patterns but requires more intricate training procedures due to its non-convex optimization landscape.

(b) Use 200 hidden units, a relu activation function for the hidden layers, and a multinomial logistic loss in the output layer. Train the model for 20 epochs with stochastic gradient descent with a learning rate of 0.001.

- **Test accuracy:** 0.7637

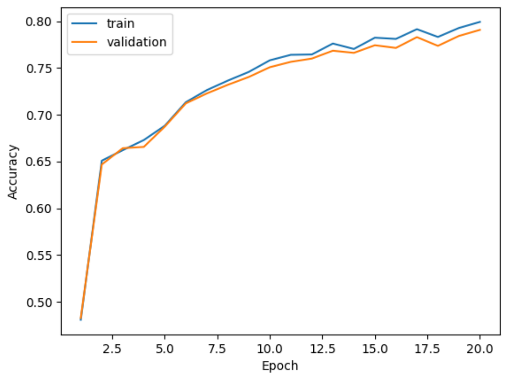

Figure 4: Plot of the train and validation accuracy

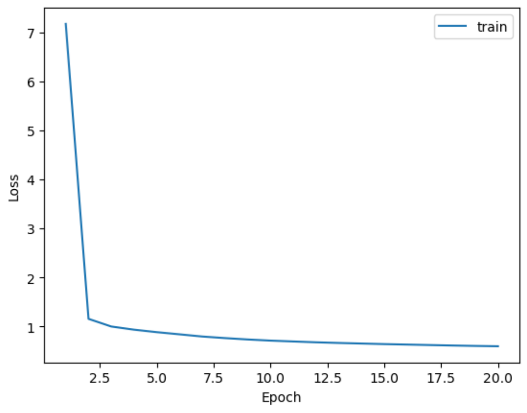

Figure 5: Plot of the train loss

Question 2

1.

(a) Train your model for 20 epochs and tune the learning rate on your validation data, using the following values: 0.001, 0.01, 0.1 and use a batch size of 16.

- η **= 0.1 Training accuracy:** 0.9687; **Validation accuracy:** 0.6224

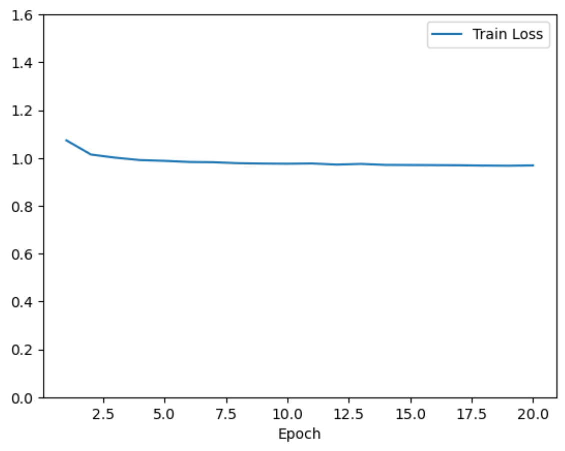

Figure 6: Plot of the training loss using learning rate of $\eta = 0.1$

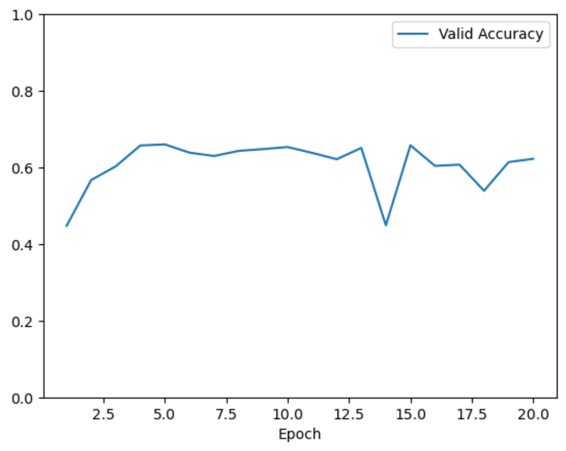

Figure 7: Plot of the validation accuracy using learning rate of $\eta = 0.1$

- η **= 0.01 Training accuracy:** 0.9370 ; **Validation accuracy:** 0.6535

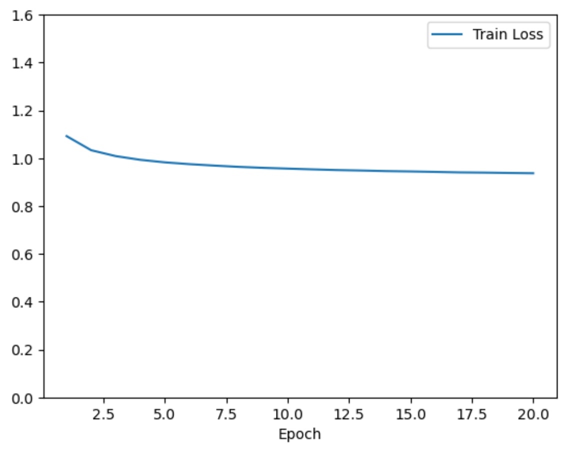

Figure 8: Plot of the training loss using learning rate of $\eta = 0.01$

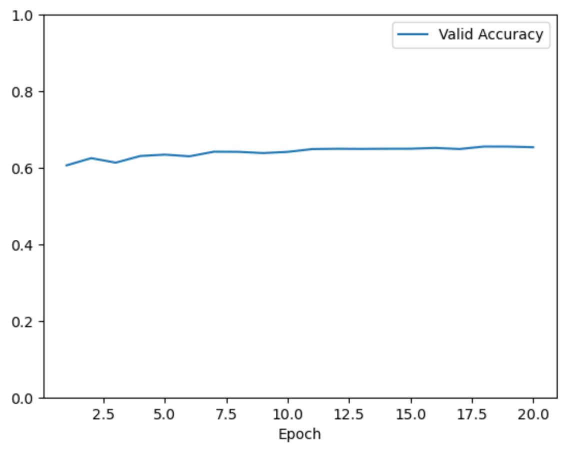

Figure 9: Plot of the validation accuracy using learning rate of $\eta = 0.01$

- η **= 0.001 Training accuracy:** 1.0145 ; **Validation accuracy:** 0.6163

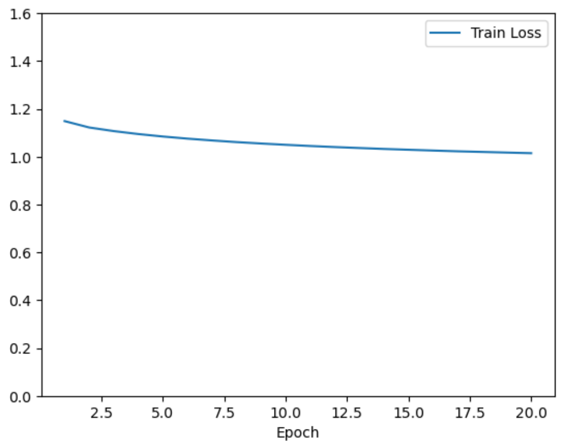

Figure 10: Plot of the training loss using learning rate of $\eta = 0.001$

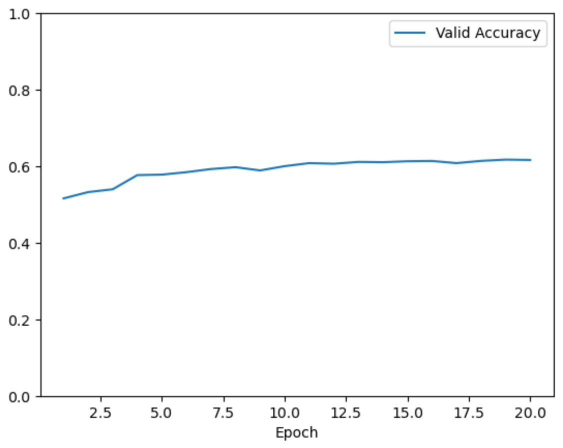

Figure 11: Plot of the validation accuracy using learning rate of $\eta = 0.001$

2.

(a) Compare the performance of your model with batch sizes 16 and 1024 with the remaining hyperparameters at their default value.

Batch size	16
Test accuracy:	0.7372
Time: Real	2m5,151s
Time: User	4m4,846s
Time: Sys	0m1,547s

Batch size	1024
Test accuracy:	0.7372
Time: Real	2m5,151s
Time: User	4m4,846s
Time: Sys	0m1,547s

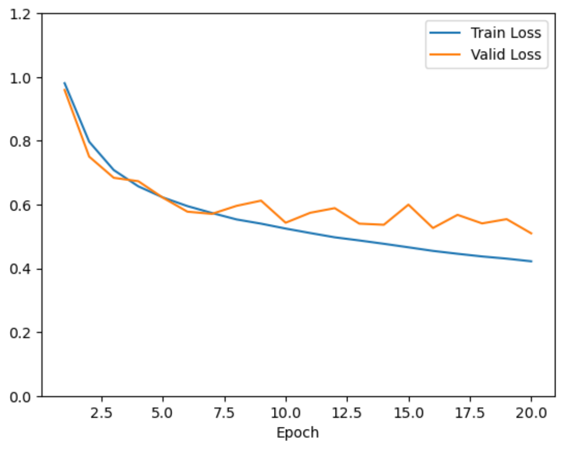

Figure 12: Plot of the train and validation losses with batch sizes 16

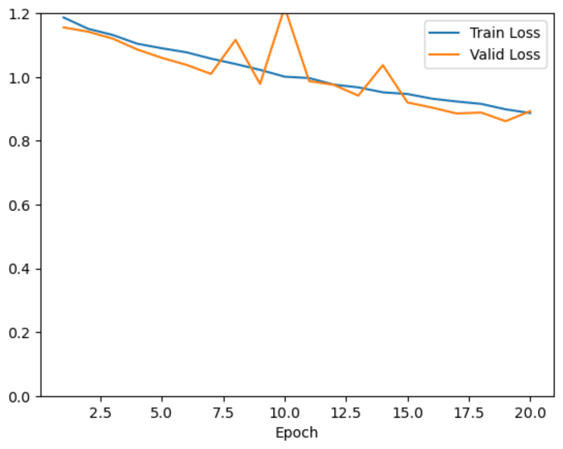

Figure 13: Plot of the train and validation losses with batch sizes 1024

Best test accuracy: 0.7372
The final test accuracy is slightly worse using a batch size of 1024 (0.6919 accuracy) instead of a batch size of 16 (0,7372 accuracy) However, using a larger batch size the computation is 4 times faster. (32s using batch size of 1024 vs 2m5s using batch size of 16)

(b) Train the model with learning rates: 1,0.1,0.01 and 0.001 with the remaining hyper-parameters at their default value.

- $\eta = 1$ **Validation accuracy:** 0.4721; **Test accuracy:** 0.4726

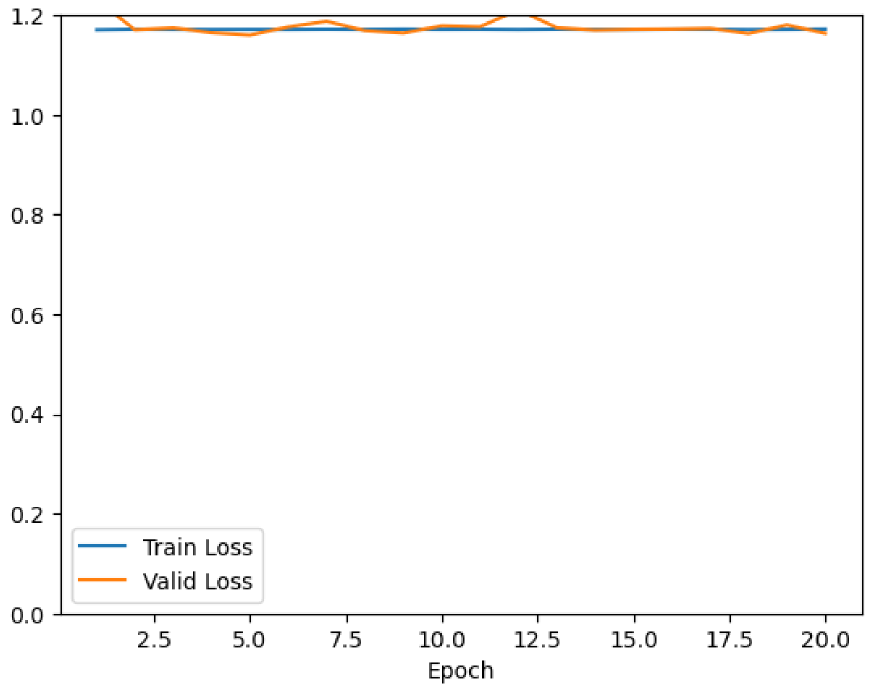

Figure 14: Plot of the train and validation losses using learning rate of $\eta = 1$

- $\eta = 0.01$ **Validation accuracy:** 0.8225; **Test accuracy:** 0.7372

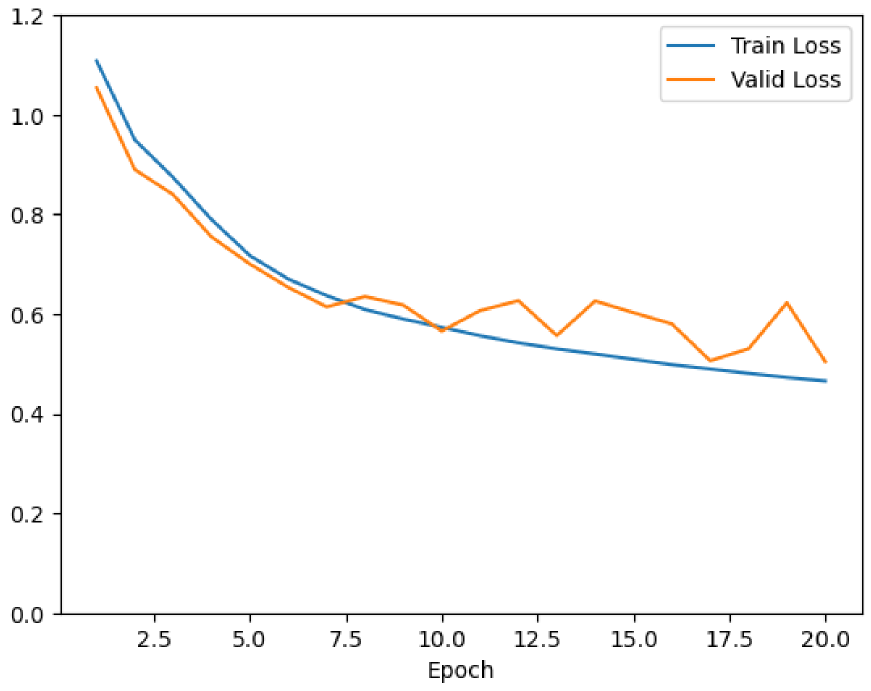

Figure 15: Plot of the train and validation losses using learning rate of $\eta = 0.01$

Worst results:	Learning rate = 1 -> Validation accuracy = 0.4721
Best results:	Learning rate = 0.01 -> Validation accuracy = 0.8225
Best test accuracy:	Test accuracy = 0.7391 for Learning rate = 0.001

The results improved greatly when using a learning rate of 0.1 instead of 1. The difference between the final accuracies of the learning rates 0.1 and 0.01 is not significant, and even if the validation accuracy drops for the learning rate 0.001, the final test accuracy for this learning rate is similar to the other two.

(c) Using a batch size of 256 run the default model for 150 epochs.

- **Batch size 256, 150 epochs l2 regularization parameter: Validation accuracy:** 0.8653; **Test accuracy:** 0.7732

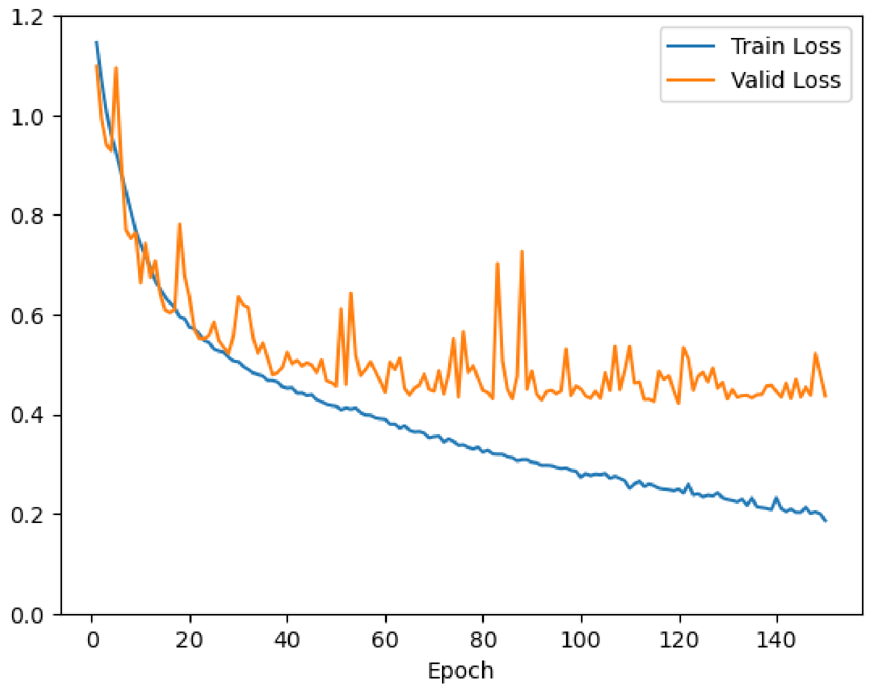

Figure 16: Plot of the train and validation losses l2 regularization parameter

- **Batch size 256, 150 epochs dropout:Validation accuracy:** 0.8614; **Test accuracy:** 0.7921

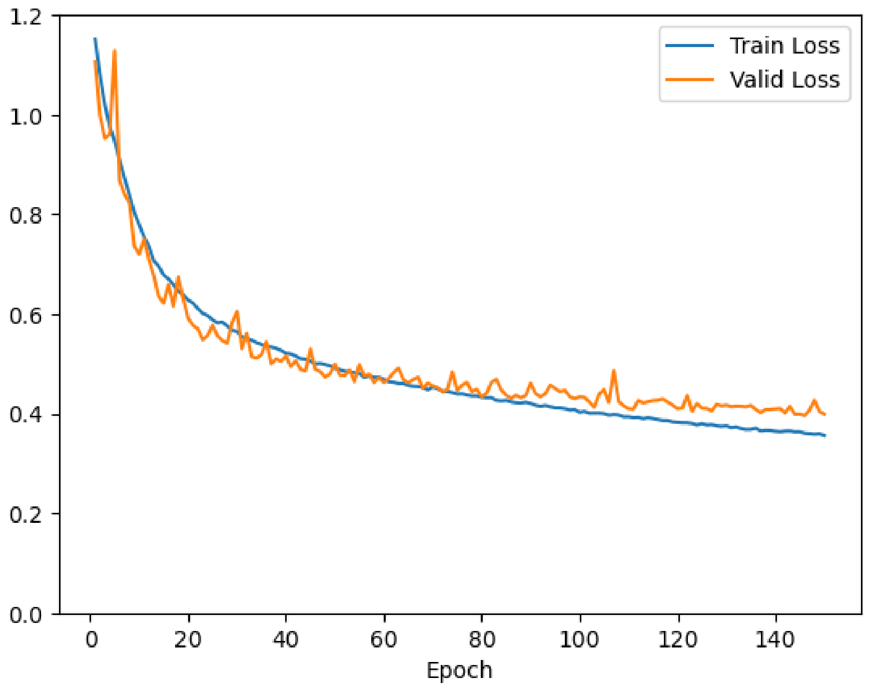

Figure 17: Plot of the train and validation losses dropout

Worst results:	Dropout = 0.2 -> Validation accuracy = 0.8614
Best results:	L2 regularization = 0.0001 -> Validation accuracy= 0.8653
Best test accuracy:	Test accuracy = 0.7921 for dropout = 0.2

Is it Overfitting? Yes, there is overfitting. We can see that the Train set loss keeps improving as it adjusts to the training data but there is a point, around 60 epochs, where the validation set reaches its lowest point, and from that point on, the validation loss gets worse.

Both techniques aim to improve model generalization by preventing overfitting, which we previously saw happening when these techniques were not used. L2 regularization adds a penalty term to the loss function based on the squared magnitudes of model weights; this way it discourages large weights, preventing the model from relying too heavily on any particular feature. Dropout randomly drops a fraction of neurons from the network (in this case 0.2). This introduces noise and helps prevent overfitting by making the network more robust and less reliant on specific neurons.

Question 3

1.

(a) Show that the function above cannot generally be computed with a single perceptron. Hint: think of a simple counter-example.

For a single perceptron to be able to compute a function, that function needs to be linearly separable. If we choose D = 2 for this function, we have 4 possible inputs: {(-1, -1), (-1, +1), (+1, -1), (+1, +1)}.

By choosing -1 as the value for A and +1 as the value for B we can see what the output values for this function look like in the graph below:

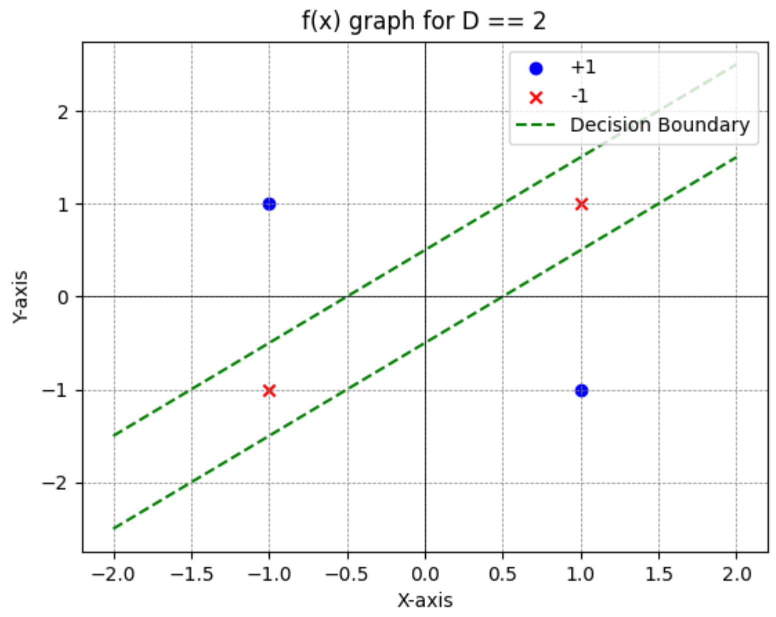

Figure 18: Plot of the outputs for f(x) when D == 2

To explain the results better, when x = -1 and y = +1 or vice-versa, the sum is 0, so -1 ¡= sum ¡= 1, which means f(x) = +1. On the other hand, if x = y = -1 or x = y = +1, the sum of the input is either -2 or +2 respectively, which means it is not contained in the [A, B] interval, so the output of the function is -1.

If we pay attention to the previous graph, we notice it is equal to the graph for the XOR problem. This graph shows that the function is not linearly separable which means it can't be computed by a single perceptron.

(b) Provide all the weights and biases of such network, and ensure that the resulting network is robust to infinitesimal perturbation of the inputs.

As we have seen in the previous exercise, the f(x) function can't be implemented using a single perceptron because the function is not linearly separable. However, using an MLP containing a hidden layer with two hidden units and signal activation functions, it is possible to compute the same function because the function can be divided into linearly separable problems:

The function

$$f(x) = \begin{cases} 1 & \text{if } \sum_{i=0}^{D-1} x_i \in [A, B]. \\ -1 & \text{otherwise} \end{cases}$$

where A and B are integers such that $D \leq A \leq B \leq D$
can also be written as

$$f(x) = \begin{cases} 1 & \text{if } \sum_{i=0}^{D-1} x_i \geq A \text{ and } \sum_{i=0}^{D-1} x_i \leq B. \\ -1 & \text{otherwise} \end{cases}$$

Let's first give a name to each one of these problems:

$$a(x) \rightarrow \sum_{i=0}^{D-1} x_i \geq A$$

$$b(x) \rightarrow \sum_{i=0}^{D-1} x_i \leq B$$

$$c(x) \rightarrow a(x) \wedge b(x)$$

These are all linearly separable problems, which means we can assign a single perceptron to each one of these.
The two perceptrons in the hidden unit will be responsible for the a) and b) problems and the c) problem will be implemented by the perceptron in the output layer.
Let's think first about the c) problem. Let h(x) be the output of the hidden layer. The hidden layer has a signal activation function and therefore h(x) is always a 2 x 1 matrix, with each of the values being either +1 or -1.
Let's assume that the output is +1 if the condition for the perceptron is satisfied. In this case, we want the output layer perceptron to compute the following values for each one of the possible h(x) values:

$h(x)_{[0,0]}$	$h(x)_{[0,1]}$	Predicted Value
+1	+1	+1
+1	-1	-1
-1	+1	-1
-1	-1	-1

Table 1: Predicted value for each h(x) possibility

As we can see, the result for the output layer is +1 only if both of the inputs are also +1. In this case, the sum of the two entries of the h(x) matrix is equal to 2. In all the other cases the same sum is either 0 or -2. So we can solve the problem this way:

$$c(x) = 1 \quad \Leftrightarrow$$
$$\sum_{i=0}^{1} h(x_i) \geq 1$$

Our activation function for this layer is sign(z) so, changing the function to have the same structure as sign(z), we get:

$$\sum_{i=0}^{1} h(x_i) - 1 \geq 0$$

The sum of h(x) can be computed like this:

$$\sum_{i=0}^{1} h(x_i) = [1, 1] \cdot h(x)$$

and replacing it in the previous function:

$$[1, 1] \cdot h(x) - 1 \geq 0$$

Where [1 1] is our Weight matrix and -1 is the Bias. For the hidden layers, we know the dimension of the Weight will be 2 X D, where D is the number of input features and the dimension of the bias will be 2 X 1. We need to set each row to solve one of the problems.

Initially, we will consider a version that is not robust to infinitesimal perturbations of the inputs. In this case, we assume all inputs are either 1 or -1.

The first row needs to solve the A problem. Considering we have a sign activation function we can rewrite the a(x) function this way, so it has the same structure as the activation function:

$$\begin{aligned}
&a(x) = 1 \Leftrightarrow \\
&\Leftrightarrow \sum_{i=0}^{D-1} x_i \geq A \Leftrightarrow \\
&\Leftrightarrow \sum_{i=0}^{D-1} x_i - A \geq 0
\end{aligned}$$

and doing the same for b(x):

$$\begin{aligned}
&b(x) = 1 \Leftrightarrow \\
&\Leftrightarrow \sum_{i=0}^{D-1} x_i \leq B \Leftrightarrow \\
&\Leftrightarrow \sum_{i=0}^{D-1} x_i - B \leq 0 \Leftrightarrow \\
&\Leftrightarrow -\sum_{i=0}^{D-1} x_i + B \geq 0
\end{aligned}$$

Sum(h(x)) can also be written like this:

$$\sum_{i=0}^{D-1} x_i = [1, 1, \ldots, 1, 1] \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{D-2} \\ x_{D-1} \end{bmatrix}$$

Replacing it in the two previous functions we obtain:

$$\begin{aligned}
&a(x) = 1 \Leftrightarrow \\
&\Leftrightarrow -\begin{bmatrix} 1_0 & 1_1 & \ldots & 1_{D-2} & 1_{D-1} \end{bmatrix} \cdot x + \begin{bmatrix} -A \end{bmatrix} \geq 0
\end{aligned}$$

$$\begin{aligned}
&b(x) = 1 \Leftrightarrow \\
&\Leftrightarrow -\begin{bmatrix} 1_0 & 1_1 & \ldots & 1_{D-2} & 1_{D-1} \end{bmatrix} \cdot x + \begin{bmatrix} B \end{bmatrix} \geq 0 \Leftrightarrow \\
&\Leftrightarrow \begin{bmatrix} -1_0 & -1_1 & \ldots & -1_{D-2} & -1_{D-1} \end{bmatrix} \cdot x + \begin{bmatrix} B \end{bmatrix}
\end{aligned}$$

So the Weights and Bias for this layer without robustness to the input noise will be:

$$\mathbf{W} = \begin{bmatrix} 1_0 & 1_1 & \ldots & 1_{D-2} & 1_{D-1} \\ -1_0 & -1_1 & \ldots & -1_{D-2} & -1_{D-1} \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -A \\ B \end{bmatrix}$$

Subsequently, we will explore a variant that incorporates robustness to infinitesimal perturbations of the inputs. Firstly, let us show why the current solution is not robust to noise in the inputs. For example, if our MLP has B = D = 5 and every input is 1.000001, the final sum will be 5.000005 and, as a result, the second perceptron of the hidden layer (which solves the b(x) problem) will wrongly output 0, not satisfying the function. To solve this, we need to make the intervals a bit larger, but not to the point where it will wrongly classify inputs that sum to 6, for example, as being in the interval. To do this we can add 0.5 to the B value, and subtract 0.5 to the A value, making the new interval bigger. The new interval will, therefore, look like this:

$$\begin{bmatrix} A - 0.5 & B + 0.5 \end{bmatrix} \qquad \textit{instead of} \qquad \begin{bmatrix} A & B \end{bmatrix}$$

Replacing the old A and B values with the new values in the Bias matrix, we get:

$$\mathbf{B} = \begin{bmatrix} -(A-0.5) \\ B+0.5 \end{bmatrix} \quad \Leftrightarrow$$
$$\Leftrightarrow \mathbf{B} = \begin{bmatrix} -A+0.5 \\ B+0.5 \end{bmatrix}$$

This solution is now robust, but it doesn't fulfill the requirement that states "all the weights and biases are integers". To fulfill this requirement we just need to multiply the two entries in the Bias by 2 and do the same in the Weight matrix to keep the proportionality. We now have the Weights and Bias for all layers of the MLP:

$$\mathbf{W_1} = \begin{bmatrix} 2_0 & 2_1 & \dots & 2_{D-2} & 2_{D-1} \\ -2_0 & -2_1 & \dots & -2_{D-2} & -2_{D-1} \end{bmatrix} \qquad \mathbf{B_1} = \begin{bmatrix} -2A+1 \\ 2B+1 \end{bmatrix}$$
$$\mathbf{W_2} = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad \mathbf{B_2} = \begin{bmatrix} -1 \end{bmatrix}$$

To prove that this solution is now robust to infinitesimal perturbation of the inputs, we will show how the input where every entry is 1.000001 would be classified, with these new matrixes and the following parameters: D = 5, A = -5, B = 5:

$$\mathbf{W_1} = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ -2 & -2 & -2 & -2 & -2 \end{bmatrix} \qquad \mathbf{B_1} = \begin{bmatrix} 11 \\ 11 \end{bmatrix}$$
$$\mathbf{W_2} = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad \mathbf{B_2} = \begin{bmatrix} -1 \end{bmatrix}$$

With these weights and biases, we can calculate the output.

$$\mathbf{sign}\left(\begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ -2 & -2 & -2 & -2 & -2 \end{bmatrix} \begin{bmatrix} 1.000001 \\ 1.000001 \\ 1.000001 \\ 1.000001 \\ 1.000001 \end{bmatrix} + \begin{bmatrix} 11 \\ 11 \end{bmatrix} \right) =$$
$$= \mathbf{sign}\left(\begin{bmatrix} 21.00010 \\ 0.99999 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Given that both inputs are also +1, according to Table 1, we can conclude that the output is +1.

(c) Repeat the previous exercise if the hidden units use rectified linear unit activations instead.

In this exercise, the hidden layer has the relu activation function, instead of the signal activation function. The relu activation function is the following:

$$relu(z) = max(0, x)$$

If we use the same weights for W1 and B1 as we did in the previous exercise, the h(x) matrix values outputted by the hidden layer will be 0 (instead of -1) if the function is not satisfied and any positive value (instead of +1) if the function is satisfied. With these values for h(x) it is impossible to find Weights and Bias for the last layer that can solve the c(x) problem. This means we need to change the bias and the weights in the hidden layer so that the output of the relu is 0 if the function is satisfied and any positive value otherwise. To find out how, we must change the weights and biases of the hidden layer. Let's first start by rewriting the relu activation function shown above.

$$relu(z) = max(0, z) \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{otherwise} \end{cases}$$

This means that we want the input of the activation function to be z ¡= 0 if the function of the perceptron is satisfied, in contrast to the previous exercise where we wanted it to be z ¿= 0 if the same function is satisfied. We know that

$$z \leq 0 \iff -z \geq 0$$

and as a result, if we replace every entry of the weight and bias matrixes for the hidden layer from the previous exercise with their symmetric value, we will achieve our desired weight and bias matrixes for this layer.

$$\mathbf{W_1} = \begin{bmatrix} -2_0 & -2_1 & \dots & -2_{D-2} & -2_{D-1} \\ 2_0 & 2_1 & \dots & 2_{D-2} & 2_{D-1} \end{bmatrix} \qquad \mathbf{B_1} = \begin{bmatrix} 2A-1 \\ -2B-1 \end{bmatrix}$$

For the output layer, the reasoning is the same as in the previous exercise. We want the output layer perceptron to compute the following values for each one of the possible h(x) values:

$h(x)_{[0,0]}$	$h(x)_{[1,0]}$	y_{pred}
0	0	+1
0	pos	-1
pos	0	-1
pos	pos	-1

Table 2: Predicted value for each h(x) possibility

As we can see, the result for the output layer is +1 if sum(h(x)) = 0 and -1 if sum(h(x)) ¿ 0. Knowing this, we can rewrite the problem in the following way:

$$c(x) = 1 \iff \sum_{i=0}^{1} h(x_i) \leq 0$$

Rewriting it to have the same structure as the sign(z) function:

$$-\sum_{i=0}^{1} h(x_i) \geq 0$$

The sum of h(x) can be computed like this:

$$\sum_{\mathbf{i=0}}^{\mathbf{1}} \mathbf{h(x_i)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \cdot h(x)$$

And replacing it in the function:

$$-\begin{bmatrix} -1 & -1 \end{bmatrix} \cdot h(x) \geq 0 \iff \begin{bmatrix} -1 & -1 \end{bmatrix} \cdot h(x) + \begin{bmatrix} 0 \end{bmatrix} \geq 0$$

This means the weight and bias for the output layer are the following:

$$\mathbf{W_1} = \begin{bmatrix} -1 & -1 \end{bmatrix} \qquad \mathbf{B_1} = \begin{bmatrix} 0 \end{bmatrix}$$

We now have the Weights and Bias for all layers of the MLP:

$$\mathbf{W_1} = \begin{bmatrix} -2_0 & -2_1 & \dots & -2_{D-2} & -2_{D-1} \\ 2_0 & 2_1 & \dots & 2_{D-2} & 2_{D-1} \end{bmatrix} \qquad \mathbf{B_1} = \begin{bmatrix} 2A-1 \\ -2B-1 \end{bmatrix}$$

$$\mathbf{W_2} = \begin{bmatrix} -1 & -1 \end{bmatrix} \qquad \mathbf{B_2} = \begin{bmatrix} 0 \end{bmatrix}$$

The logic implemented for robustness to infinitesimal perturbation in exercise 3(b) transferred to exercise 3(c) because the Weights and Bias of the hidden layer are the same, but with the symmetric values, therefore the robustness to noise in the inputs is also maintained in the solution for this exercise.