INSTITUTO SUPERIOR TÉCNICO

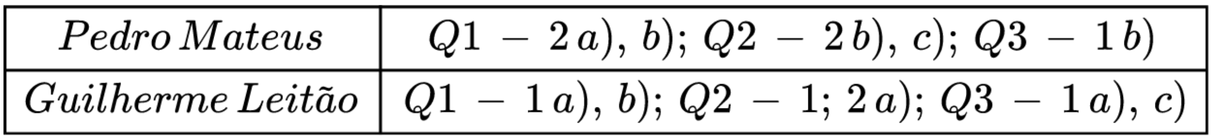
Departamento de Engenharia Informática

Deep Learning (Dei)

MEIC 2023-2024 – 2nd Period

Homework 1

Pedro Mateus 99306; Guilherme Leitão 99951



Pedro Mateus	$Q1 - 2\,a),\, b);\; Q2 - 2\,b),\, c);\; Q3 - 1\,b)$
Guilherme Leitão	$Q1 - 1\,a),\, b);\; Q2 - 1;\; 2\,a);\; Q3 - 1\,a),\, c)$

Question 1

1.

(a)

The perceptron algorithm, upon implementation, displayed a big variance in accuracies, with the training and validation accuracies ranging between ~0.35 and ~0.65, as shown in Figure 1.

Ultimately, the test accuracy proved to be quite low, at only 0.3422, indicating that the model performed poorly.

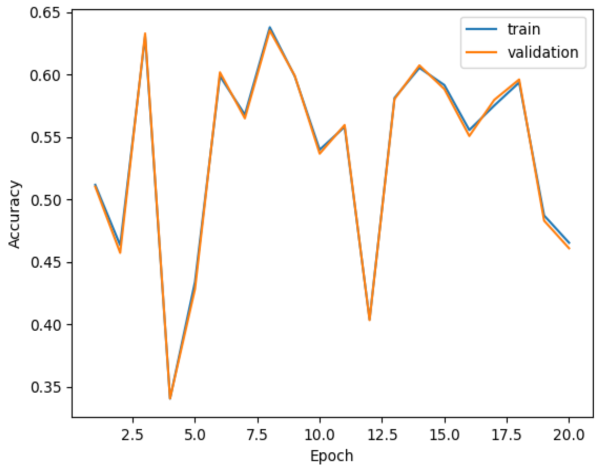
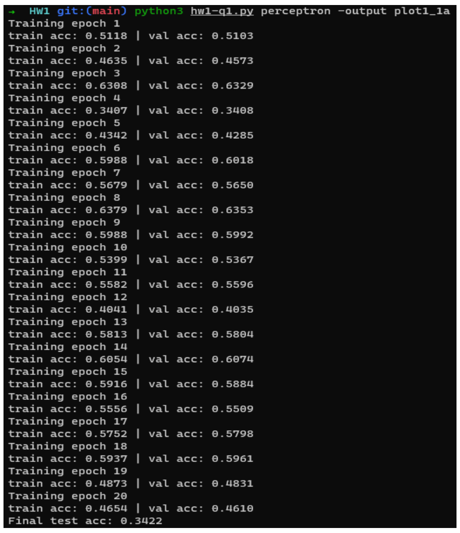


Figure 1: plot1_1a.png



```
→  HW1 git:(main) python3 hw1-q1.py perceptron -output plot1_1a
Training epoch 1
train acc: 0.5118 | val acc: 0.5103
Training epoch 2
train acc: 0.4635 | val acc: 0.4573
Training epoch 3
train acc: 0.6308 | val acc: 0.6329
Training epoch 4
train acc: 0.3407 | val acc: 0.3408
Training epoch 5
train acc: 0.4342 | val acc: 0.4285
Training epoch 6
train acc: 0.5988 | val acc: 0.6018
Training epoch 7
train acc: 0.5679 | val acc: 0.5650
Training epoch 8
train acc: 0.6379 | val acc: 0.6353
Training epoch 9
train acc: 0.5988 | val acc: 0.5992
Training epoch 10
train acc: 0.5399 | val acc: 0.5367
Training epoch 11
train acc: 0.5582 | val acc: 0.5596
Training epoch 12
train acc: 0.4041 | val acc: 0.4035
Training epoch 13
train acc: 0.5813 | val acc: 0.5804
Training epoch 14
train acc: 0.6054 | val acc: 0.6074
Training epoch 15
train acc: 0.5916 | val acc: 0.5884
Training epoch 16
train acc: 0.5556 | val acc: 0.5509
Training epoch 17
train acc: 0.5752 | val acc: 0.5798
Training epoch 18
train acc: 0.5937 | val acc: 0.5961
Training epoch 19
train acc: 0.4873 | val acc: 0.4831
Training epoch 20
train acc: 0.4654 | val acc: 0.4610
Final test acc: 0.3422
```

Figure 2: execution values

(b)

The logistic regression algorithm showed an undoubted improvement in training and validation accuracies, as well as final test accuracies in comparison to the perceptron.

The model with η=0.01 had its training and validation accuracies ranging between ~0.52 and ~0.66 with a final test accuracy of 0.5784.
The model with a lower η=0.001 resulted in its training and validation accuracies ranging between ~0.61 and ~0.66, with an improved final test accuracy of 0.5936.

The plots (*Figure 3* and Figure 4) indicate that the model with the lower learning rate exhibits a more stable convergence and a slightly better generalization on the test set. The higher learning rate shows more fluctuation in accuracy, suggesting that a smaller learning rate might be more suitable for this dataset in achieving consistent performance improvements.

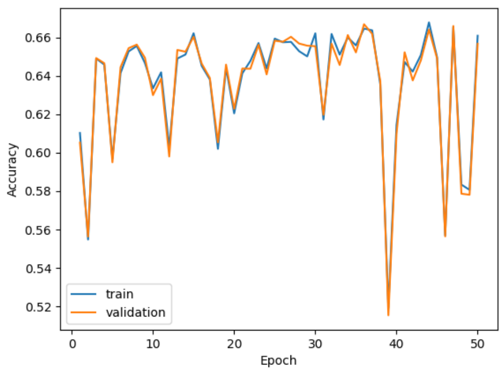


Figure 3: plot1_1b_0.01.png

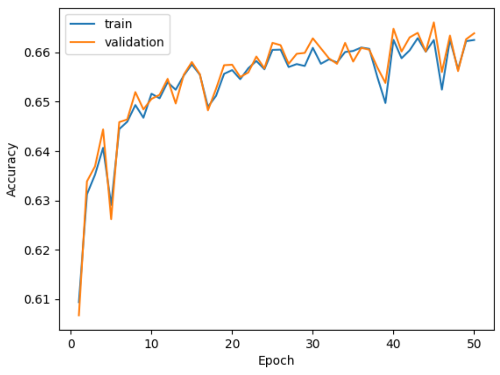


Figure 4: plot1_1b_0.001.png

2.

(a)

Logistic regression is a discriminative probabilistic classifier that maximizes the conditional log-likelihood of the training data, therefore we can conclude that the objective function is strictly convex, which means we can consider any local minimum as a global minimum. On the other hand, the Multi-Layer Perceptron is a fead-forward neural network that is characterized by one input layer, one output layer and one or multiple hidden layers. Since the logistic regression is a linear classifier it cannot solve non-linearly separable problems, making the Multi-Layer Perceptron a more expressive model as it is reported in the statement.

Regarding the ease of training, logistic regression has the upper hand as it is a convex optimization problem making it easier to train reliably, in contrast, since training an MLP involves non-convex optimization we have to deal with local minima, which makes the training process harder.

In conclusion we can confirm that the claim is indeed true.

(b)

Final test acc: 0.7561

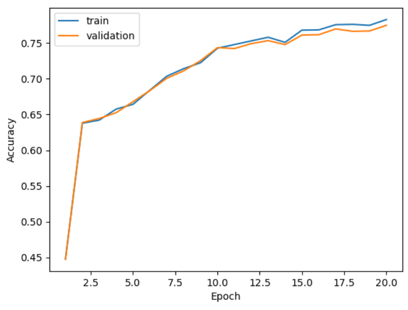


Figure 5: plot1_2b.png

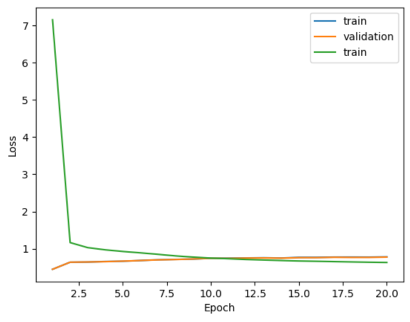


Figure 6: plot1_2b_loss.png

Question 2

1.

The best final validation accuracy was achieved with η=0.01, which was 0.6449.

Correspondingly, the final test accuracy with η=0.01 was also the highest at 0.5803, having the others η=0.1 and η=0.001 a test accuracy of 0.5652 and 0.5728, respectively.
In the case of η=0.01 and η=0.001, both the training and validation loss decrease steadily over time, while the validation accuracy improves, indicating a good fit without apparent overfitting.

However, it is still apparent from the figures below that both the drops in loss and improvement in accuracy are more evident in the η=0.01, proving this to be the best fit for this model.

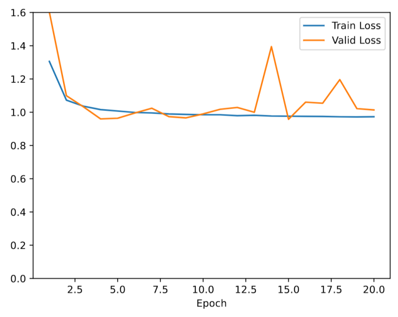


Figure 7:
plot2_1_0.1-training-loss.pdf

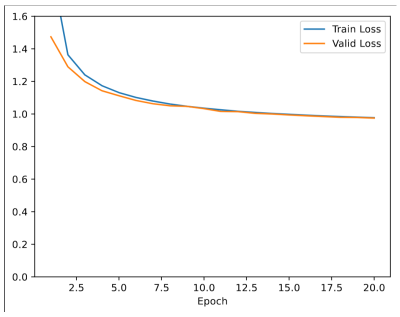


Figure 8:
plot2_1_0.01-training-loss.pdf

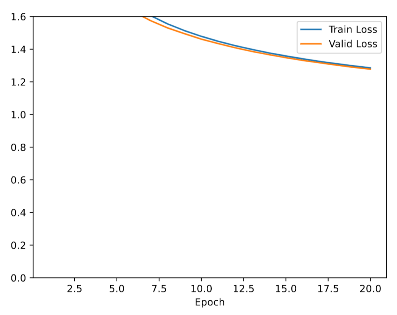


Figure 9:
plot2_1_0.001-training-loss.pdf

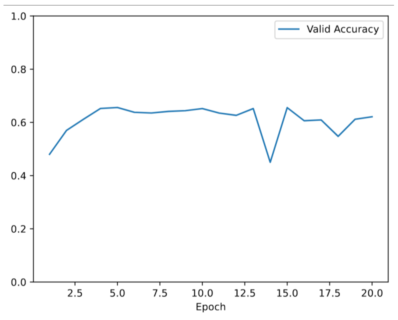


Figure 10:
plot2_1_0.1-validation-accuracy.pdf

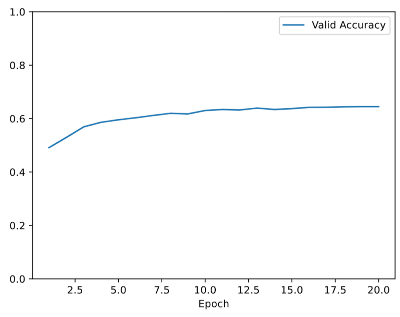


Figure 11:
plot2_1_0.01-validation-accuracy.pdf

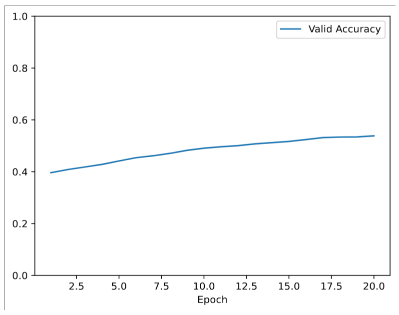


Figure 12:
plot2_1_0.001-validation-accuracy.pdf

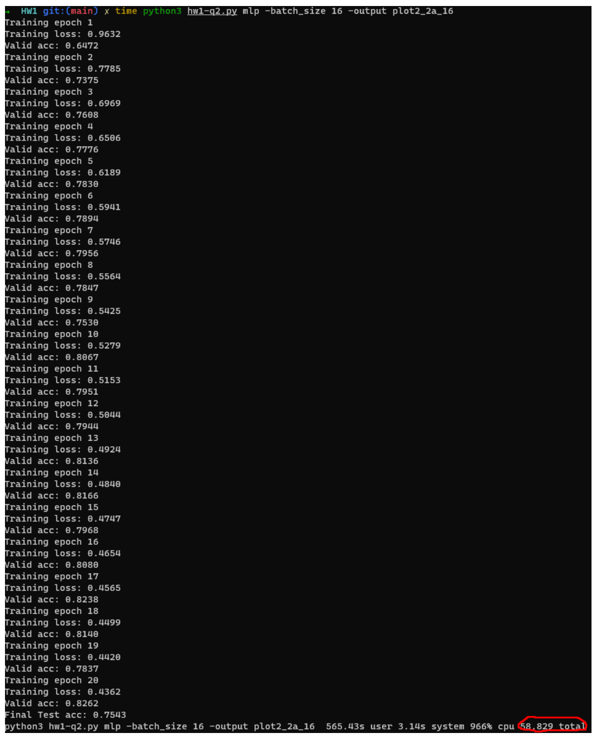
2.

(a)

The smaller batch size of 16 resulted in a slightly higher final test accuracy of 0.7543 compared to 0.7353 for the larger batch size of 1024. However, this came at the cost of increased computational time (58.829 seconds versus 16.161 seconds, as you can see in Figure 13 and Figure 14).

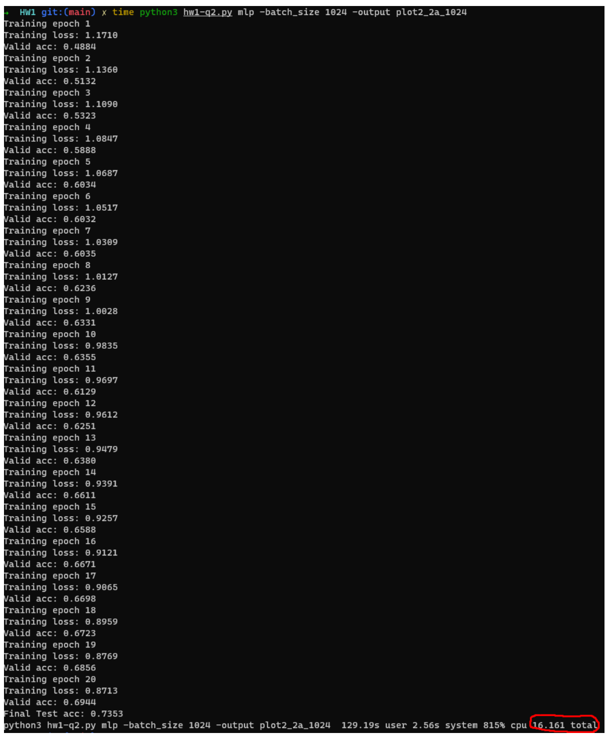
Training loss for the batch size of 16 showed more fluctuations, which might indicate overfitting, as opposed to the smoother decrease in training loss for the larger batch size. Validation accuracy was higher with the smaller batch size, peaking at ~0.82 versus ~0.69 for the larger batch size, suggesting better generalization despite the longer training time and potential overfitting.

Overall, the smaller batch size achieved better performance in terms of accuracy but required more training time and presented a higher variance in validation loss, indicating potential overfitting. The larger batch size, while faster, did not achieve as high accuracy but showed smoother convergence.



```
→  HW1 git:(main) ✗ time python3 hw1-q2.py mlp -batch_size 16 -output plot2_2a_16
Training epoch 1
Training loss: 0.9632
Valid acc: 0.6472
Training epoch 2
Training loss: 0.7785
Valid acc: 0.7375
Training epoch 3
Training loss: 0.6969
Valid acc: 0.7608
Training epoch 4
Training loss: 0.6506
Valid acc: 0.7776
Training epoch 5
Training loss: 0.6189
Valid acc: 0.7830
Training epoch 6
Training loss: 0.5941
Valid acc: 0.7894
Training epoch 7
Training loss: 0.5746
Valid acc: 0.7956
Training epoch 8
Training loss: 0.5564
Valid acc: 0.7847
Training epoch 9
Training loss: 0.5425
Valid acc: 0.7530
Training epoch 10
Training loss: 0.5279
Valid acc: 0.8067
Training epoch 11
Training loss: 0.5153
Valid acc: 0.7951
Training epoch 12
Training loss: 0.5044
Valid acc: 0.7944
Training epoch 13
Training loss: 0.4924
Valid acc: 0.8136
Training epoch 14
Training loss: 0.4840
Valid acc: 0.8166
Training epoch 15
Training loss: 0.4747
Valid acc: 0.7968
Training epoch 16
Training loss: 0.4654
Valid acc: 0.8080
Training epoch 17
Training loss: 0.4565
Valid acc: 0.8238
Training epoch 18
Training loss: 0.4499
Valid acc: 0.8140
Training epoch 19
Training loss: 0.4420
Valid acc: 0.7837
Training epoch 20
Training loss: 0.4362
Valid acc: 0.8262
Final Test acc: 0.7543
python3 hw1-q2.py mlp -batch_size 16 -output plot2_2a_16  565.43s user 3.14s system 966% cpu 58.829 total
```

Figure 13: execution values for batch size 16



```
→  HW1 git:(main) ✗ time python3 hw1-q2.py mlp -batch_size 1024 -output plot2_2a_1024
Training epoch 1
Training loss: 1.1710
Valid acc: 0.4884
Training epoch 2
Training loss: 1.1360
Valid acc: 0.5132
Training epoch 3
Training loss: 1.1090
Valid acc: 0.5323
Training epoch 4
Training loss: 1.0847
Valid acc: 0.5888
Training epoch 5
Training loss: 1.0687
Valid acc: 0.6034
Training epoch 6
Training loss: 1.0517
Valid acc: 0.6032
Training epoch 7
Training loss: 1.0309
Valid acc: 0.6035
Training epoch 8
Training loss: 1.0127
Valid acc: 0.6236
Training epoch 9
Training loss: 1.0028
Valid acc: 0.6331
Training epoch 10
Training loss: 0.9835
Valid acc: 0.6355
Training epoch 11
Training loss: 0.9697
Valid acc: 0.6129
Training epoch 12
Training loss: 0.9612
Valid acc: 0.6251
Training epoch 13
Training loss: 0.9479
Valid acc: 0.6380
Training epoch 14
Training loss: 0.9391
Valid acc: 0.6611
Training epoch 15
Training loss: 0.9257
Valid acc: 0.6588
Training epoch 16
Training loss: 0.9121
Valid acc: 0.6671
Training epoch 17
Training loss: 0.9065
Valid acc: 0.6698
Training epoch 18
Training loss: 0.8959
Valid acc: 0.6723
Training epoch 19
Training loss: 0.8769
Valid acc: 0.6856
Training epoch 20
Training loss: 0.8713
Valid acc: 0.6944
Final Test acc: 0.7353
python3 hw1-q2.py mlp -batch_size 1024 -output plot2_2a_1024  129.19s user 2.56s system 815% cpu 16.161 total
```

Figure 14: execution values for batch size 1024

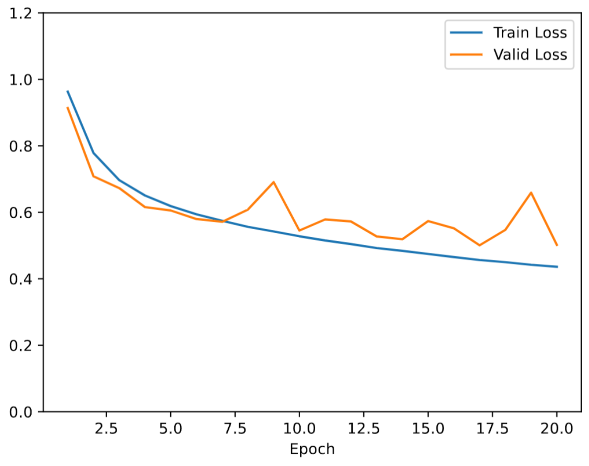


Figure 15: plot2_2a_16-training-loss.pdf

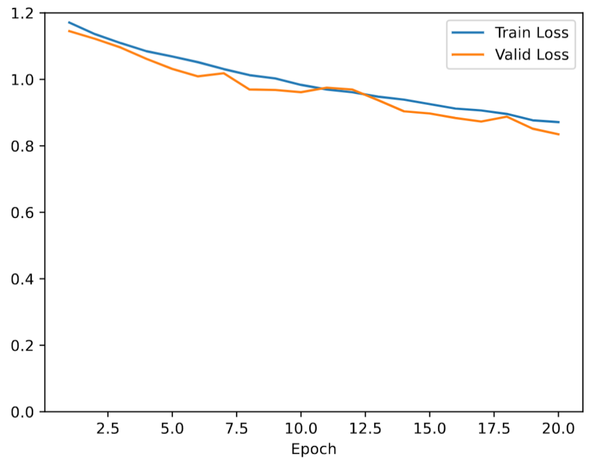


Figure 16: plot2_2a_1024-training-loss.pdf

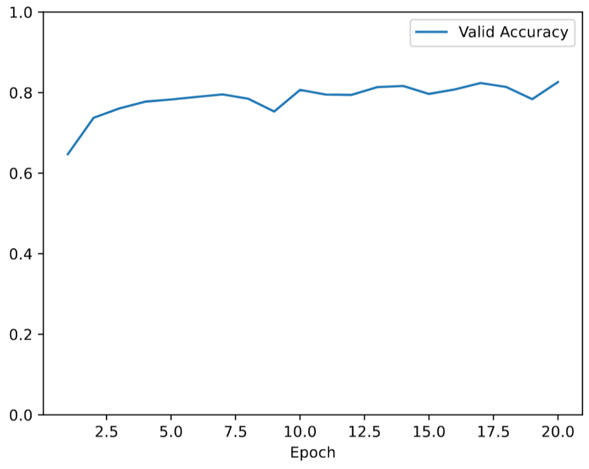


Figure 17: plot2_2a_16-validation-accuracy.pdf

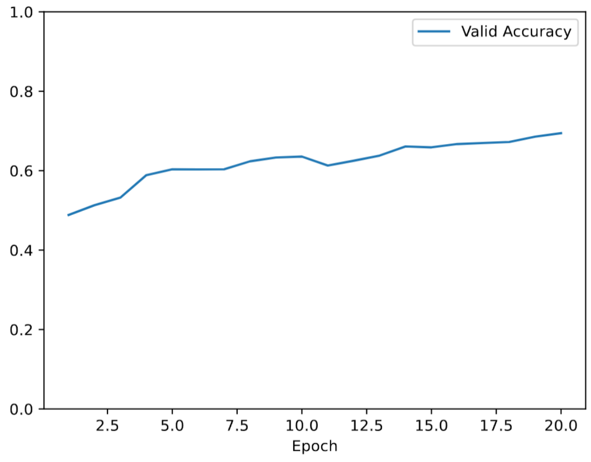


Figure 18: plot2_2a_1024-validation-accuracy.pdf

(b)

For this exercise we measured performance with accuracy.

As we can see in the graphs the model that contains the learning rate hyperparameter set to 0.1 (Figure 19) both the training and validation loss decrease, whereas in the model that contains the learning rate set to 1 (Figure 20) the model's training and validation loss are constant.

Since the learning rate is the hyperparameter that controls how quickly the model is adapted to the system, we can conclude that when the learning rate is set to 0.1 the model starts converging not reaching an optimal solution, on the other hand, when the learning rate is set to 1 the model converges too quickly and gets stuck in a local minimum.

For the other learning rates (0.001 and 0.0001) with only 20 epochs the models reach validation accuracy values between the other 2 models.

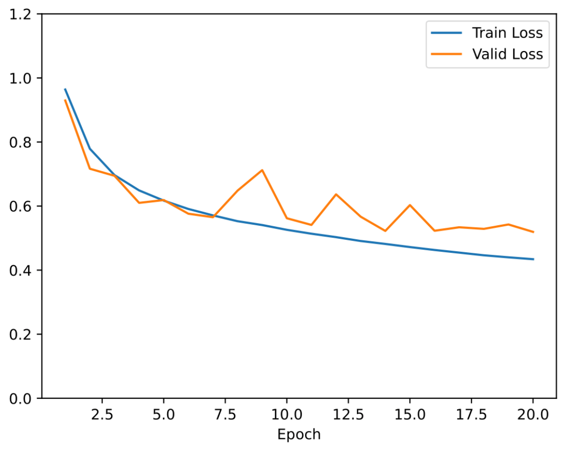


Figure 19: **MLP** with learning rate set to 0.1

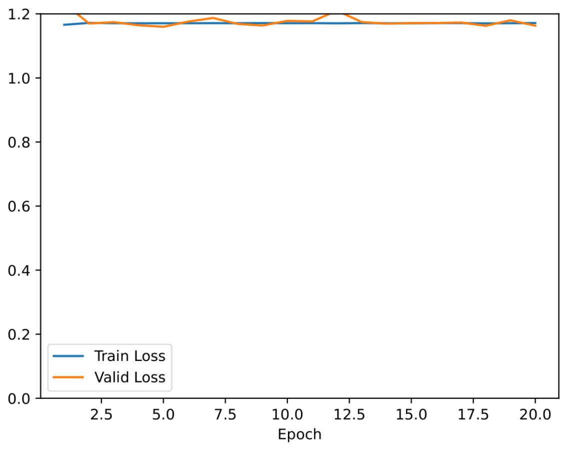


Figure 20: **MLP** with learning rate set to 1

(c)

For this exercise we measured performance with accuracy.

Test Accuracies:

batch size = 256 -> 0.7732
batch size = 256 and l2 decay = 0.0001 -> 0.7543
batch size = 256 and dropout probability = 0.2 -> 0.7940

By looking at the graphs present in both of the figures we can see that in the graph present in Figure 21 the train loss keeps decreasing while validation loss is constant after a certain amount of epochs, on the other hand in the graph present in Figure 22 both the train loss and validation loss keep decreasing.

First let's define the dropout process. The dropout process consists in receiving an input value and setting some of the features to zero with a given probability p, this process introduces variability in the input and therefore prevents overfitting.

Therefore we can conclude that Figure 21 where the dropout probability is set to 0 the model shows clear signs of overfitting, whereas in Figure 22 where the dropout probability is set to 0.2 the model doesn't overfit.

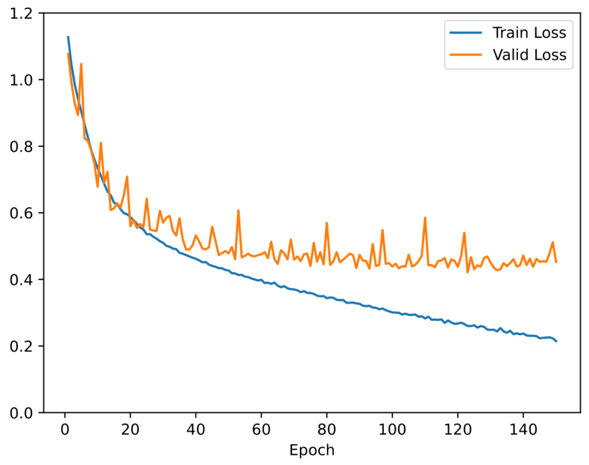


Figure 21: **MLP** with batch size set to 256

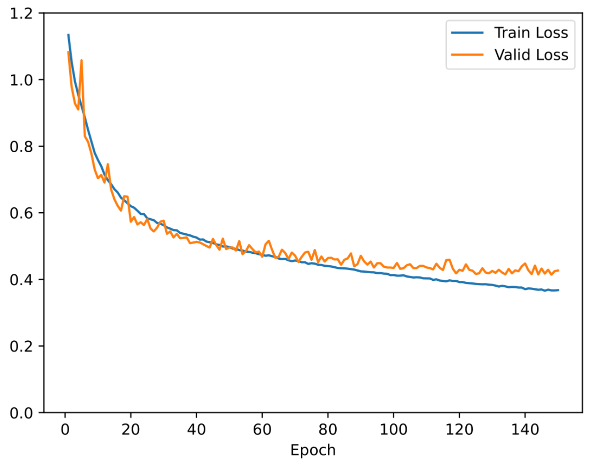


Figure 22: **MLP** with batch size set to 256 and dropout probability set to 0.2

Question 3

1.

(a)

The function given cannot be computed by a single-layer perceptron because it involves classification based on a sum being within an interval, rather than on one side of a threshold.

A single-layer perceptron can only create a linear decision boundary, which is insufficient for distinguishing inputs that sum to values within a range defined by two boundaries.

Example:

A = -1 | B = 1 | D = 2

x: (-1, -1) -> f(x) = -1

x: (-1, 1) -> f(x) = 1

x: (1, -1) -> f(x) = 1

x: (1, 1) -> f(x) = -1

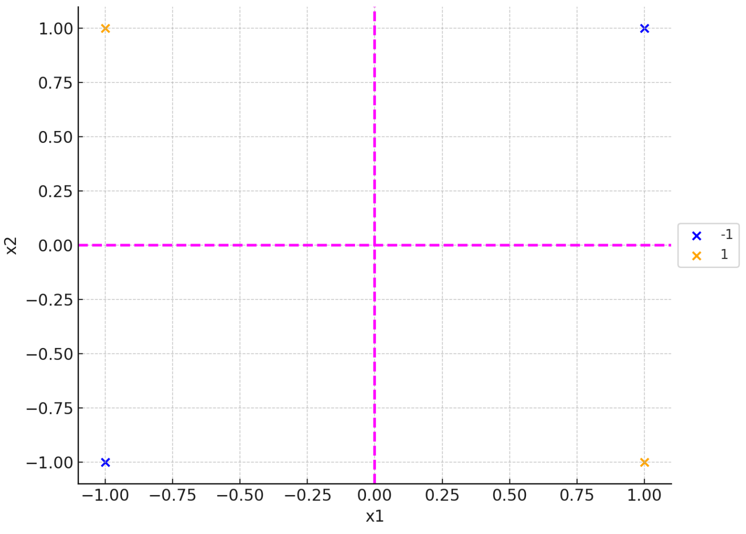


Figure 23: data points for the example provided

As you can see in Figure 23, there is no single line that can separate the two classes, indicating that the function is not linearly separable.

Therefore, a single-layer perceptron cannot compute it.

(b)

Considering the following weight and bias values:

$$w1 = \begin{bmatrix} 2 & 2 & \cdots & 2 \\ 2 & 2 & \cdots & 2 \end{bmatrix}_{2 \times D} ;\ b1 = \begin{bmatrix} -2A + 1 \\ -2B - 1 \end{bmatrix}_{2 \times 1}$$

$$w2 = [1 \quad -1]_{1 \times 2} ;\ b2 = -2$$

What the multilayer perceptron will do is that in the first layer the product of $w1 \cdot x0$ will double of the sum of the values of the input and after that it will subtract (2*A) and add 1 in one of the cells and it will subtract (2 * B + 1) in the other cell after applying the sign function to the matrix we have obtained, we will either 1 or -1 in the returned matrix depending if the sum of the values present in the input is smaller, bigger or equal to A/B.

Since we want the value of the sum to be between A and B, the output of the second layer has to be $[1 \quad -1]^T$, so in the last layer we only need to check that.

The multilayer perceptron will do that by subtracting the value in the second cell of the matrix to the value in the first cell of the matrix. We subtract 2 to the value to take into consideration the fact that in case the sum is bigger than B, the output of the first layer would be $[1 \quad 1]^T$ and therefore the value calculated in the second layer by the product of $w2 \cdot x1$ would be zero, which would be accepted due to the sign function.

When in comes to the robustness of the network to infinitesimal perturbation of the inputs, we can see that in the case the perturbation makes the sum bigger this will only be a problem for the upper bound, so for that reason we subtract 1 extra unit to the sum in the second cell of $b1$, this will not affect the outcome of the multilayer perceptron, because in case the sum is equal to $2B + 1$, the result will be 0 and therefore the value returned by the activation function will be according to what it's supposed, in the case perturbation makes the sum smaller this will only affect the lower bound, so for that reason we add 1 extra unit to the sum in first cell of $b1$, this will not affect the outcome of the multilayer perceptron because if the sum is equal to $2A - 1$, the result will be 0 and therefore the value returned by the activation function will be according to what it's supposed to be.

(c)

ReLU(z) = max(0,z)

h_1: **$w_{1i} = -2, b_1 = 2A - 1$** $\Rightarrow$ $\underline{h_1(x) = \text{ReLU}(2A - 1 - 2\sum x_i)}$ (activates if the sum is lower than A)
h_2: **$w_{2i} = 2, b_2 = -2B - 1$** $\Rightarrow$ $\underline{h_2(x) = \text{ReLU}(2\sum x_i - 2B - 1)}$ (activates if the sum is bigger than B)

output function: w_o = **$[-1, -1]^T, b_o = 0$** $\Rightarrow$ $\underline{f(x) = \text{sign}(-h_1(x) - h_2(x))}$

Example:

$\underline{D = 2;\ A = B = 0}$

$x = (-1, -1)$: $h_1(x) = 3$; $h_2(x) = 0$; $f(x) = \text{sign}(-3 - 0) = -1$ ✓
$x = (-1, 1)$: $h_1(x) = 0$; $h_2(x) = 0$; $f(x) = \text{sign}(-0 - 0) = 1$ ✓
$x = (1, -1)$: $h_1(x) = 0$; $h_2(x) = 0$; $f(x) = \text{sign}(-0 - 0) = 1$ ✓
$x = (1, 1)$: $h_1(x) = 0$; $h_2(x) = 3$; $f(x) = \text{sign}(-0 - 3) = -1$ ✓

Let's say that $\varepsilon \in \mathbb{R}^+$ is a sufficiently small number:

$\underline{\sum x_i = A}$: $f(x) = \text{sign}(-h_1(x) - h_2(x)) = \text{sign}(-\text{ReLU}(2A - 1 - 2A) - \text{ReLU}(2A - 2B - 1))$
$=_{(B\geq A)}$ $\text{sign}(-0 - 0) = \text{sign}(0) = 1$ ✓

$\underline{\sum x_i = A + \varepsilon}$: $f(x) = \text{sign}(-h_1(x) - h_2(x)) = \text{sign}(-\text{ReLU}(2A - 1 - 2(A + \varepsilon)) - \text{ReLU}(2A + 2\varepsilon - 2B - 1))$
$=_{(B>A)}$ $\text{sign}(-0 - 0) = \text{sign}(0) = 1$ ✓
...
$= \text{sign}(-h_1(x) - h_2(x)) = \text{sign}(-\text{ReLU}(2A - 1 - 2(A + \varepsilon)) - \text{ReLU}(2A + 2\varepsilon - 2B - 1))$
$=_{(B=A)}$ $\text{sign}(-0 - 0) = \text{sign}(0) = 1$ ✓

$\underline{\sum x_i = A - \varepsilon}$: $f(x) = \text{sign}(-h_1(x) - h_2(x)) = \text{sign}(-\text{ReLU}(2A - 1 - 2(A - \varepsilon)) - \text{ReLU}(2A - 2\varepsilon - 2B - 1))$
$=_{(B\geq A)}$ $\text{sign}(-0 - 0) = \text{sign}(0) = 1$ ✓

The above is also similarly true for the respective border case of B.
We have thus proven that this network is robust to infinitesimal perturbation of the inputs ($\pm\varepsilon$).