Gonçalo Gonçalves, 99226
Matilde Heitor, 99284

Homework 1- Group 33

Question 1

1.

a) The result for final test accuracy is 0.3422, for train accuracy 0.4654 and for validation accuracy 0.4610.

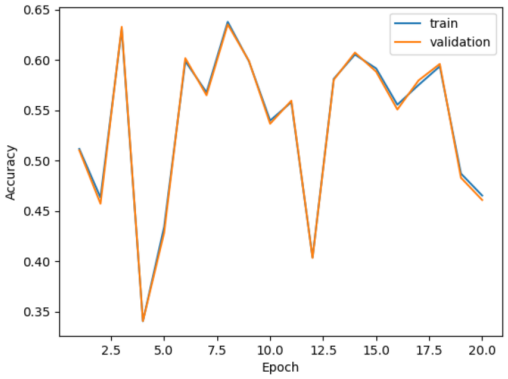


Figure 1

b) The result for the final test accuracy is 0.5936 for the model where the learning_rate is 0.001 and 0.5784 for the one where it is 0.01.

The learning rate is a parameter used to train learning models to help calculate the size of the step that will be used to minimize the loss function.

In the graph showing the learning rate at 0.01, we can see that the model doesn't converge because the learning rate is too large, oscillating between values above and below a local minimum (Figure 3).

In the other graph, where the learning rate is 0.001, the values continue to oscillate, but in this case the assumed values are closer to the local minimum, which indicates that it is converging more quickly (Figure 2).

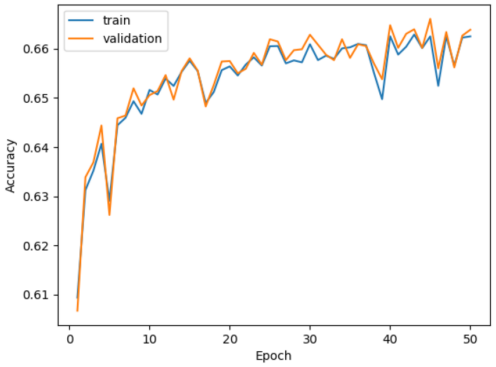


Figure 2

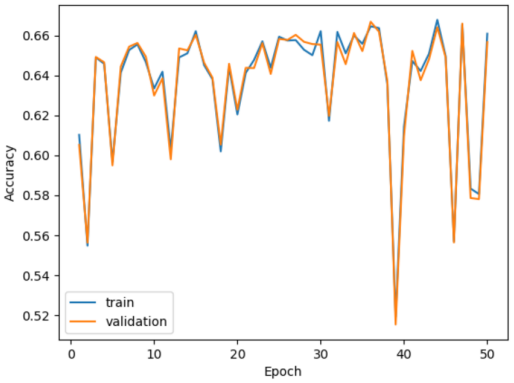


Figure 3

2.

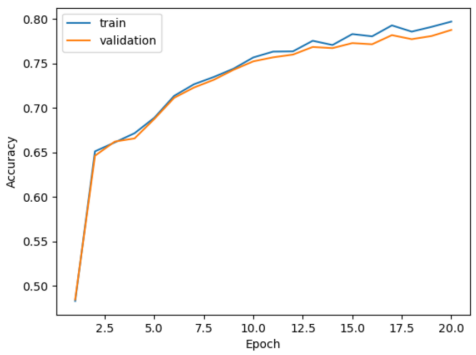
a)

The expressiveness of a machine learning model is its ability to capture complex patterns, relationships, and structures within the data. Logistic regression, being a linear model, assumes a linear relationship between input features and output, making it less expressive for complex, non-linear patterns, especially in image classification. In contrast, a multi-layer perceptron (MLP) with Rectified Linear Unit (ReLU) activations is more expressive due to non-linearities introduced by the activation function, allowing it to capture intricate patterns and representations.

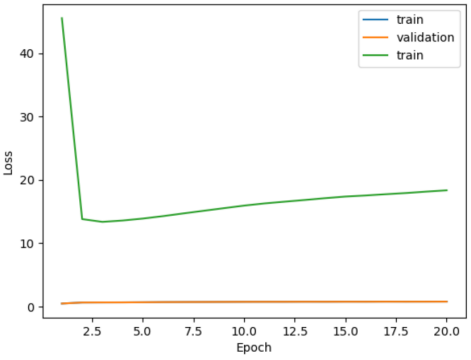
Addressing the second part of the claim, the optimization landscape is a key factor in training efficiency. Logistic regression benefits from convex optimization, ensuring a single global minimum, simplifying training, and guaranteeing convergence with methods like gradient descent. On the other hand, an MLP with non-linear activations lacks convexity, introducing challenges in training. It involves multiple local minima, and convergence to a global minimum is not guaranteed. Training an MLP requires sophisticated techniques, such as initialization strategies and adaptive learning rates.

To answer the question, the claim is partially true. While an MLP with ReLU activations is more expressive, logistic regression is easier to train due to convex optimization. On the other hand, it should also be considered that the training difficulty of an MLP can be addressed with appropriate techniques and so, the choice between the two models depends on factors like problem complexity, available data, and computational resources.

b) The result for the final test accuracy is 0.7656



Train and Validation accuracies - Figure 4



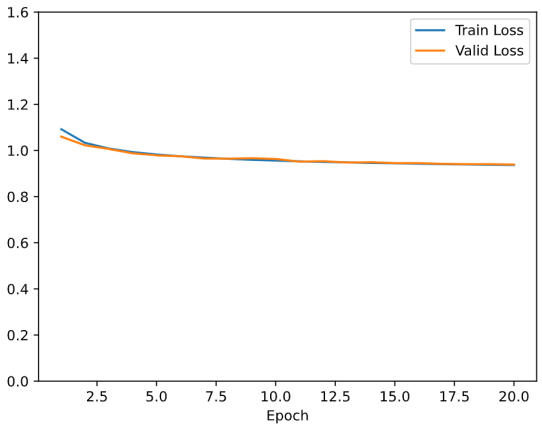
Train Loss - Figure 5

Question 2

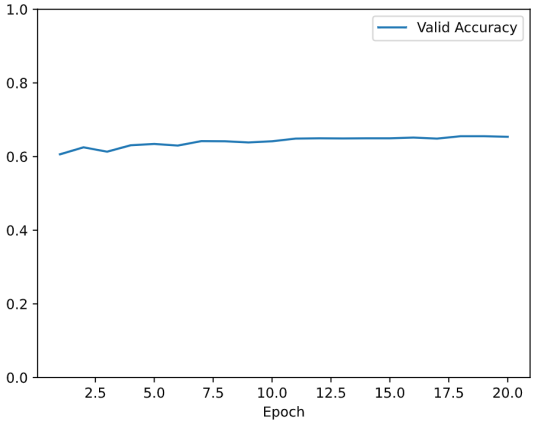
1.

learning_rate = 0.1 => Valid acc = 0.6224 => Final Test acc = 0.5577
learning_rate = 0.01 => Valid acc = 0.6535 => Final Test acc = 0.6200
learning_rate = 0.001 => Valid acc = 0.6163 => Final Test acc = 0.6503

Our best result in terms of validation accuracy was 0.6535 for the model in which we used 0.01 for the learning rate and its final test accuracy was 0.6200.



Training loss - Figure 6



Validation accuracy - Figure 7

2.

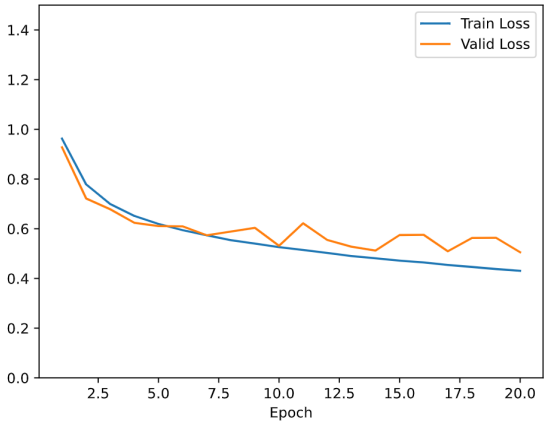
a)

batch_size = 16 => Valid acc = 0.8241 => Final Test acc = 0.7561
batch_size = 1024 => Valid acc = 0.6957 => Final Test acc = 0.7316

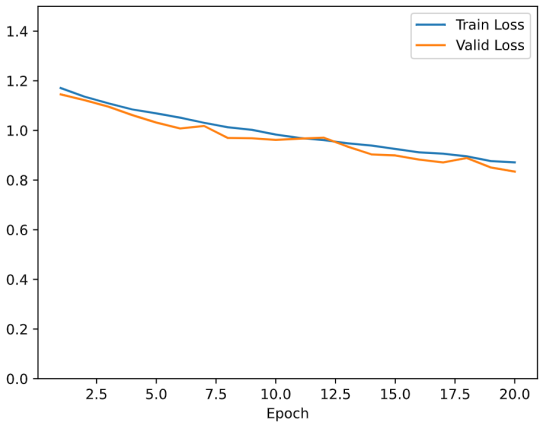
For the model in which we used batch_size 16, our test accuracy was 0.7561 with a run time of 1m32.873s. For the model with a batch_size of 1024, the test accuracy was 0.7316 with a run time of 0m30.169s.

The model with the best test accuracy was the one in which we used batch_size of 16. This may be due to the fact that in this case the number of iterations per epoch is greater, and our model learns better.

As for execution time, we noticed a decrease in the case where the batch_size was 1024, because when we have larger batch sizes, for the same number and size of our epochs, the number of iterations decreases, and consequently so does the execution time.



Batch 16 - Figure 8



Batch 1024 - Figure 9

b)

learning_rate = 0.1 => Valid acc = 0.8241 => Final Test acc = 0.7561
learning_rate = 0.01 => Valid acc = 0.8165 => Final Test acc = 0.7599
learning_rate = 0.001 => Valid acc = 0.6920 => Final Test acc = 0.7127
learning_rate = 1 => Valid acc = 0.4721 => Final Test acc = 0.4726

Our best result in terms of validation accuracy was 0.8241 for the model in which we used 0.1 for the learning rate and its final test accuracy was 0.7561. In contrast, the worst was for the model in which the learning rate was 1 and its validation accuracy was 0.4721 with a final test accuracy of 0.4726.

These results are due to the fact that, assuming a learning rate of 1, our model converges too quickly, at the end of the first epoch, to a value that seemingly minimizes the loss function but overshooting the minimum point and resulting in a suboptimal solution (Figure 11).

On the other hand, we can see that with a learning rate of 0.1 it converges to a local minimum that optimizes our loss function. We can also conclude that this is the optimal value for this learning rate, not too large and not too small as it converges to a better result more quickly than other options (Figure 10).

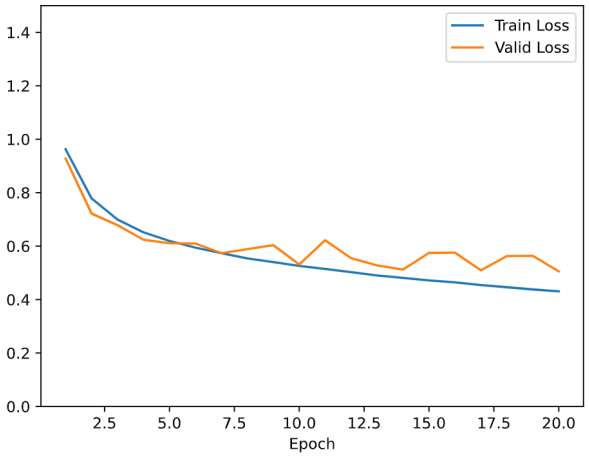


Figure 10

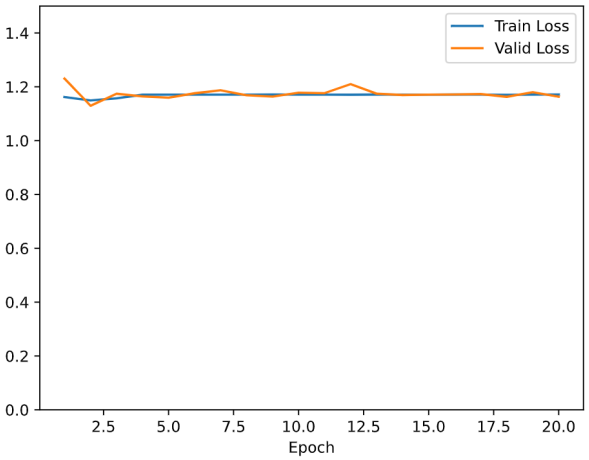


Figure 11

c)

base => Valid acc = 0.8493 => Final Test acc = 0.7656
dropout = 0.2 => Valid acc = 0.8561 => Final Test acc = 0.7864
l2_decay = 0.0001 => Valid acc = 0.8512 => Final Test acc = 0.7694

Looking at the graph, we can see that there is overfitting, one of the indicators being that the train loss is much lower than the validation loss. This means that the network is learning to classify the training data much more accurately than the validation data. We therefore used two different methods to avoid overfitting by changing the dropout and L2_decay parameters.

Our best result in terms of validation accuracy was 0.8561 for the model in which we used 0.2 for the dropout and its final test accuracy was 0.7864 (Figure 12). In contrast, the worst was in the model where we didn't use any regularization, 0 for L2_decay and 0 for the dropout with a validation accuracy of 0.8493 and a final test accuracy of 0.7656 (Figure 13).

In our situation, the dropout parameter is used as a technique which, when applied to the network layers, deactivates 20% of the units during the training process, so that the network doesn't become too dependent on specific units. In this case, there is a very sharp reduction in the difference between the two losses, which indicates that there has been a reduction in overfitting.

The default model without regularization parameters had the worst performance of all methods as it has no method for countering overfitting on the training.

L2 regularization, also known as Ridge regularization, adds a penalty term to the cost function of the model that is proportional to the square of the weights, penalizing large weights in the network. This encourages distribution of weights evenly across all the features trying to refrain the model from becoming too dependent on any one feature and reduce overfitting. In our scenario, although using the L2_decay parameter did improve the model by reducing overfitting and outperforming the default method without any regularization but did not outperform the dropout parameter in terms of accuracy.

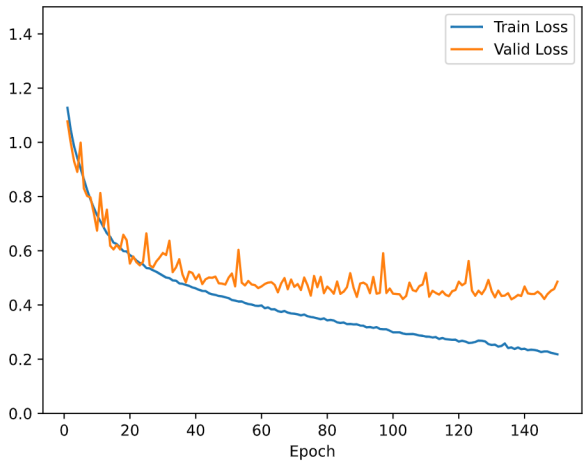


Figure 12

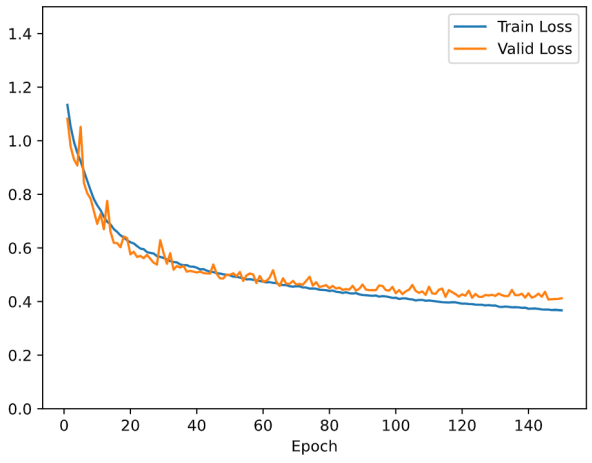


Figure 13

3.1

a) As a single layer perceptron can only solve linearly separable problems, a simple example to prove the proposition that this problem can't be solved by this model is proving the function $f(x)$ can represent non-linearly separable problems.

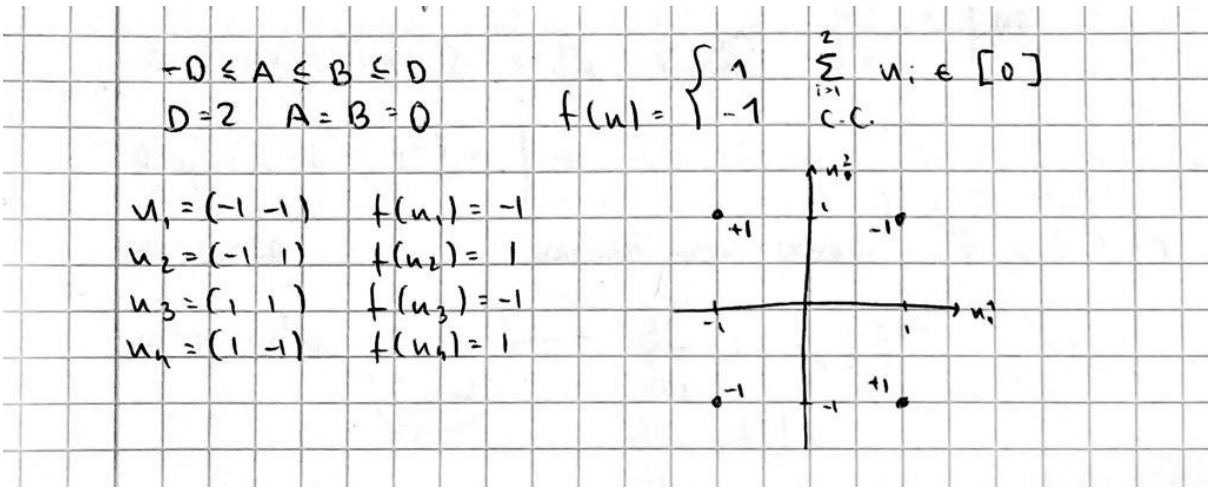


Figure 14

b) $$W^1 = \begin{bmatrix} 2 & 2 \\ 2 & 2 \\ \dots & \dots \\ 2 & 2 \end{bmatrix}, b^1 = \begin{bmatrix} -2A+1 \\ -2B-1 \end{bmatrix}, W^2 = \begin{bmatrix} 2 \\ -2 \end{bmatrix}, b^2 = [-2]$$

Choosing values for W1 and b1 and with a hidden layer with 2 units, the multiplication of $\left[W^1\right]^T x^0 = \sum_{i=1}^{D} 2x_i = 2\sum_{i=1}^{D} x_i$ is equal to the sum of all coordinates (times 2) and b1 is used to verify this sum is is contained in $[A, B]$:

$$A \leq \left[W^1\right]^T x^0 \leq B$$

$$A \leq \left[W^1\right]^T x^0 \wedge \left[W^1\right]^T x^0 \leq B$$

$$\left[W^1\right]^T x^0 - A \geq 0 \wedge \left[W^1\right]^T x^0 - B \leq 0$$

For robustness purposes we need to ensure that the sum is contained in $[A - \alpha, B + \alpha]$, $\alpha < 1$ to account for any computational mistakes. For this example we consider $\alpha = 0.5$:

$$\left[W^1\right]^T x^0 \geq A - 0.5 \wedge \left[W^1\right]^T x^0 \leq B + 0.5$$

$$2\left[W^1\right]^T x^0 - 2A + 1 \geq 0 \wedge 2\left[W^1\right]^T x^0 - 2B - 1 \leq 0$$

Resulting in the final values for b^1. As we account for perturbations, possible small values that otherwise might disrupt the network, these disruption values tend to 0 and so the residual values added for expanding the intervals ensure the model is robust and thus concluding $\lim_{t \to 0} h(x + tv) = h(x)$.

$$z^1 = \left[W^1\right]^T x^0 + b^1$$

$$z_1^1 = \left[W^1\right]^T x^0 + b^1 = 2\sum_{i=1}^{2} x_i^0 - 2A + 1$$

$$z_2^1 = \left[W^1\right]^T x^0 + b^1 = 2\sum_{i=1}^{2} x_i^0 - 2B - 1$$

$$h_1^1 = sign(z_1^1) = \begin{cases} 1 & , if \ \ 2\sum_{i=1}^{2} x_i^0 - 2A + 1 > 0 \\ -1 & , c.c \end{cases}$$

$$h_2^1 = sign(z_2^1) = \begin{cases} 1 & , if \ \ 2\sum_{i=1}^{2} x_i^0 - 2B - 1 > 0 \\ -1 & , c.c \end{cases}$$

From this step we can have 3 different types of values for $\begin{pmatrix} 1 \\ -1 \end{pmatrix} x^1$: $\sum_{i=1}^{2} x \in [A, B], \begin{pmatrix} -1 \\ -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ if outside. As for W^2 and b^2

$$z^2 = \left[W^2\right]^T x^1 + b^2 = \begin{bmatrix} w_1^2 & w_2^2 \end{bmatrix} h^1 + b^2 = w_1^2 h_2^1 + w_2^2 h_2^1 + b^2$$

$$h^2 = sign(z^2) = \begin{cases} w_1^2 - w_2^2 + b^2 > 0 \\ -w_1^2 - w_2^2 + b^2 < 0 \\ w_1^2 + w_2^2 + b^2 < 0 \end{cases}$$

for robustness purposes mentioned before, we apply the same basis for multiplication the values, and as a result we solve :

$$W^2 = \begin{bmatrix} 2 \\ -2 \end{bmatrix}, b^2 = [-2]$$

$$z^2 = \left[W^2\right]^T x^1 + b^2 = [2 \ -2]h^1 - 2 = 2h_1^1 - 2h_2^1 - 2 = \begin{cases} 2, \ 2A - 1 < 2\sum_{i=1}^{2} x_i^0 < 2B + 1 \\ -2 \ , c.c \end{cases}$$

$$f(x) = h^2 = sign(z^2) = \begin{cases} 1 & , if \ \ 2A - 1 < 2\sum_{i=1}^{D} x_i^0 < 2B + 1 \Leftrightarrow \\ -1 & , c.c \end{cases}$$

$$f(x) = h^2 = \begin{cases} 1 & , A < \sum_{i=1}^{D} x_i^0 < B \\ -1 & , c.c \end{cases}$$

Proving $\lim_{t \to 0} h(x + tv) = h(x) = f(x)$.

c) $$W^1 = \begin{bmatrix} 2 & 2 \\ 2 & 2 \\ \dots & \dots \\ 2 & 2 \end{bmatrix}, b^1 = \begin{bmatrix} -2A + 1 \\ -2B - 1 \end{bmatrix}, \ W^2 = \begin{bmatrix} 4 \\ -2 \end{bmatrix}, b^2 = -3$$

The logic for W^1 and b^1 is the same as the previous exercise. For z^1 and h^1 it differs as we use rectified linear unit activations instead:

$$z^1 = \left[W^1\right]^T x^0 + b^1$$

$$z_1^1 = \left[W^1\right]^T x^0 + b^1 = 2\sum_{i=1}^{D} x_i^0 - 2A + 1$$

$$z_2^1 = \left[W^1\right]^T x^0 + b^1 = 2\sum_{i=1}^{D} x_i^0 - 2B - 1$$

$$h_1^1 = relu(z_1^1) = \begin{cases} 1 & ,if \;\; 2\sum_{i=1}^{D} x_i^0 - 2A + 1 > 0 \\ 0 & ,c.c \end{cases}$$

$$h_2^1 = relu(z_2^1) = \begin{cases} 1 & ,if \;\; 2\sum_{i=1}^{D} x_i^0 - 2B - 1 > 0 \\ 0 & ,c.c \end{cases}$$

From this step we can have 3 different types of values for x^1: $\begin{pmatrix}1\\0\end{pmatrix}, \sum_{i=1}^{2} x_i \in [A,B], \begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}$

if outside. As for W^2 and b^2 :

$$z^2 = \left[W^2\right]^T x^1 + b^2 = \left[w_1^2 \;\; w_2^2\right] h^1 + b^2 = w_1^2 h_2^1 + w_2^2 h_2^1 + b^2$$

$$h^2 = sign(z^2) = \begin{cases} w_1^2 + b^2 > 0 \;, & \begin{pmatrix}1\\0\end{pmatrix} \\ b^2 < 0 \;, & \begin{pmatrix}0\\0\end{pmatrix} \\ w_1^2 + w_2^2 + b^2 < 0 \;, & \begin{pmatrix}1\\1\end{pmatrix} \end{cases}$$

For robustness purposes mentioned before, we apply the same basis for multiplication the values, and as a result we solve :

$$W^2 = \begin{bmatrix}4\\-2\end{bmatrix}, b^2 = -3$$

$$z^2 = \left[W^2\right]^T h^1 + b^2 = [4 \;\; -2] h^1 - 2 = 4h_1^1 - 2h_2^1 - 2 = \begin{cases} 2 \;, & \begin{pmatrix}1\\0\end{pmatrix} \\ -2 \;, & \begin{pmatrix}0\\0\end{pmatrix} \\ 0 \;, & \begin{pmatrix}1\\1\end{pmatrix} \end{cases}$$

$$f(x) = h^2 = relu(z^2) = \begin{cases} 1 & ,\sum_{i=1}^{D} x_i \in [A,B] \\ 0 & ,c.c \end{cases}$$

Proving $\lim_{t \to 0} \; h(x + tv) = h(x) = f(x).$

Contributions

In this project, both members collaborated effectively to address the diverse set of tasks across all three questions. Matilde took the lead in implementing and training the Perceptron (1.1a) and contributed to the critical analysis of logistic regression versus multi-layer perceptrons (1.2a). Gonçalo, on the other hand, led the implementation of logistic regression with varying learning rates (1.1b) and played a key role in implementing a multi-layer perceptron with a single hidden layer (1.2b). Additionally, both parties equally contributed for the implementation of the linear model with logistic regression for Question 2.1 and to the implementation of a feed-forward neural network with dropout regularization in Question 2.2. In Question 3, both members jointly tackled the design of a multilayer perceptron for a Boolean function. Throughout the project, both members worked collaboratively and simultaneously, leveraging their individual strengths and expertise to contribute effectively to each task.