Deep Learning: Homework 1

Hugo Mantinhas 95592, João Silveira 95597

December 15, 2023

Work Division

Each member of the group worked together on all the questions in this assignment. For this reason, the workload was divided equally between the two members of the group.

Question 1

- **1. a)**

	Train	**Validation**	**Test**
Accuracy	0.4654	0.4610	0.3422

Table 1: Train, validation and test accuracies for the perceptron model

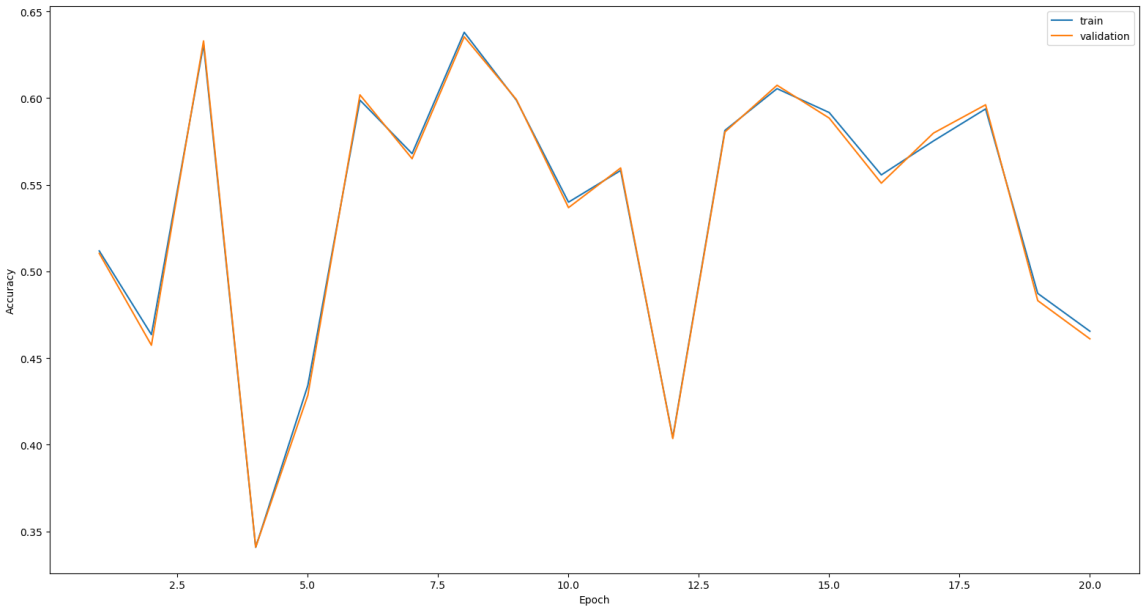

Figure 1: Train and validation accuracies as a function of the number of epochs for the perceptron model

- **1. b)** Analyzing the plot for the learning rate of 0.01, we find that the weight update many times results in accuracy changes in the order of 5%, in both directions. This suggests that the weight adjustments might be overshooting the optimal value. On the other hand, the learning rate of 0.001 results in a more stable learning process, with the accuracy changes being in the order of 1% or less. Unlike the model trained with 0.01 learning rate, this one seems to be tending to grow over time which suggests that the weight adjustments are closer to the optimal value. Although the final test accuracies for both models don't look much different: 0.5784 for learning rate 0.01 and 0.5936 for learning rate 0.001; for the latter the model shows a lot more stability in its accuracy across epochs.

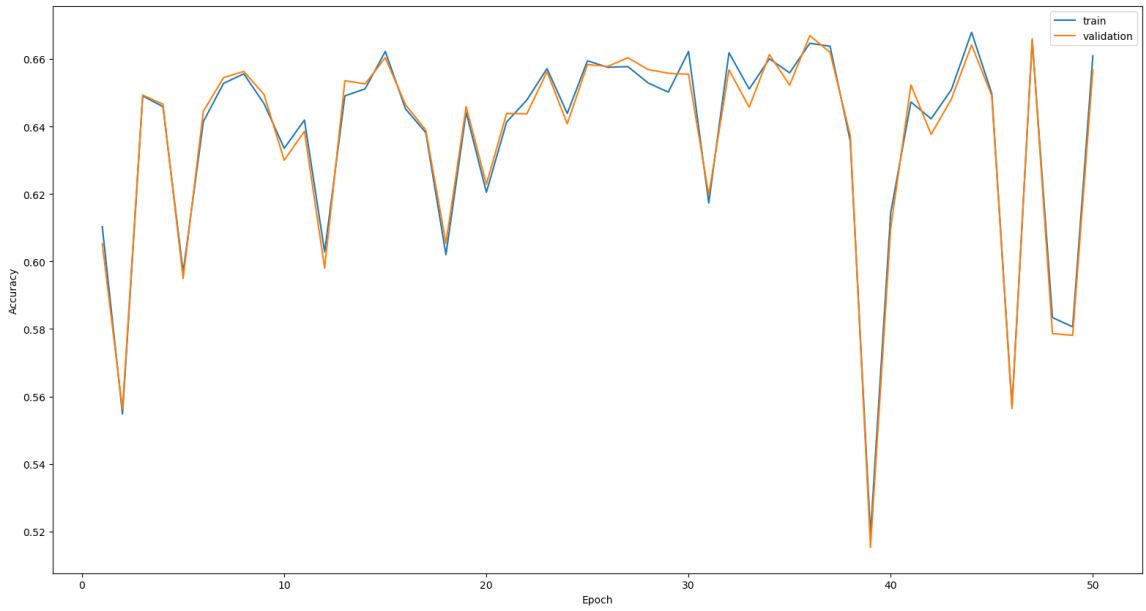

Figure 2: Train and validation accuracies as a function of the number of epochs for learning rate of 0.01

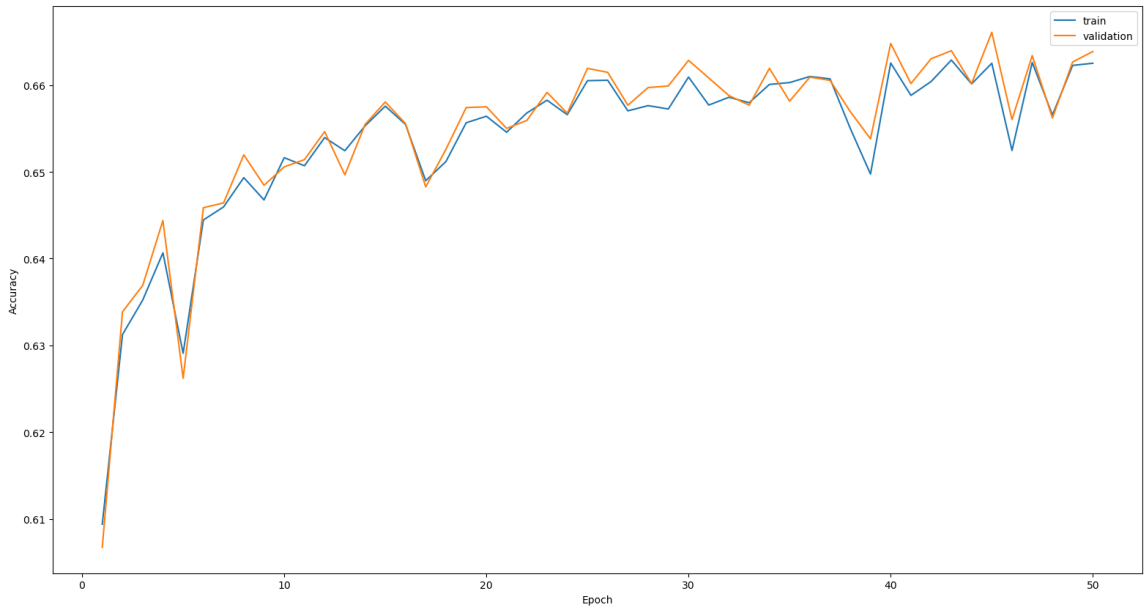

Figure 3: Train and validation accuracies as a function of the number of epochs for learning rate of 0.001

	Train	**Validation**	**Test**
Learning rate 0.01	0.6609	0.6568	0.5784
Learning rate 0.001	0.6625	0.6639	0.5936

Table 2: Train, validation and test accuracies for learning rates of 0.01 and 0.001

- **2. a)** This statement is true. A logistic regression model is a linear model, which means that it can only learn linearly separable data, i.e., a dataset whose classes can be separated by a hyperplane. On the other hand, a multi-layer perceptron using `relu` activations can learn to separate non-linearly separable data because of the non-linearity introduced by the `relu` activation function in between layers. However, it can be shown, by computing the Hessian matrix of the loss function, that the loss function of a logistic regression model using cross-entropy loss is always convex. However, the same cannot be done for the loss function of a multi-layer perceptron, in general. This means that the logistic regression model can be trained to a global optimum, while, when using a multi-layer perceptron, we can never be sure that we have reached a global optimum.

- **2. b)** The final test accuracy is 0.7580.

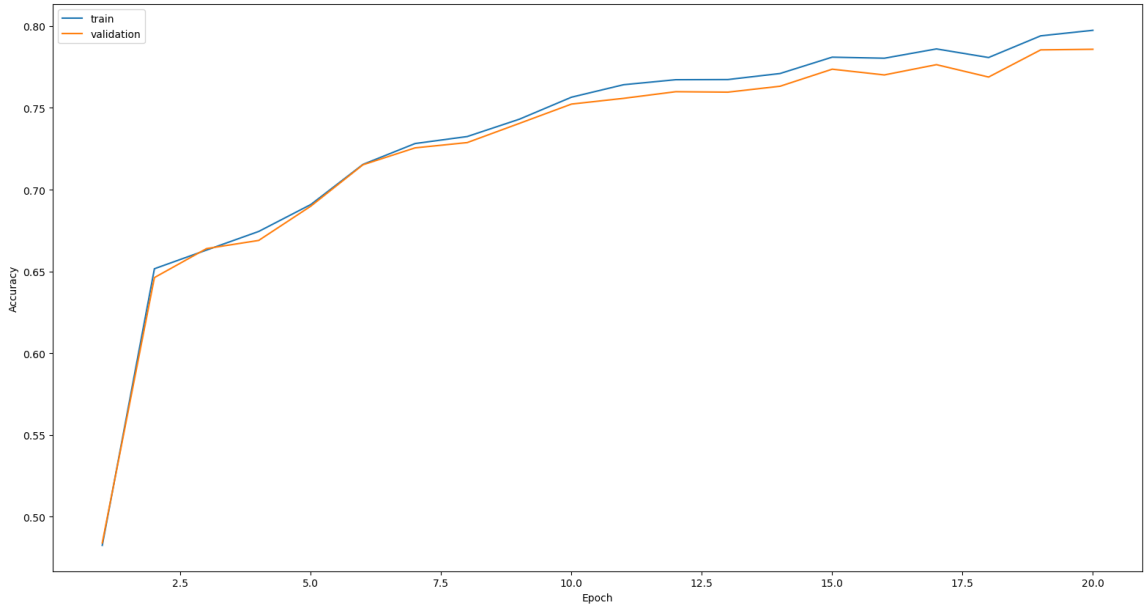

Figure 4: Train and validation accuracies as a function of the number of epochs for the no-toolkit MLP

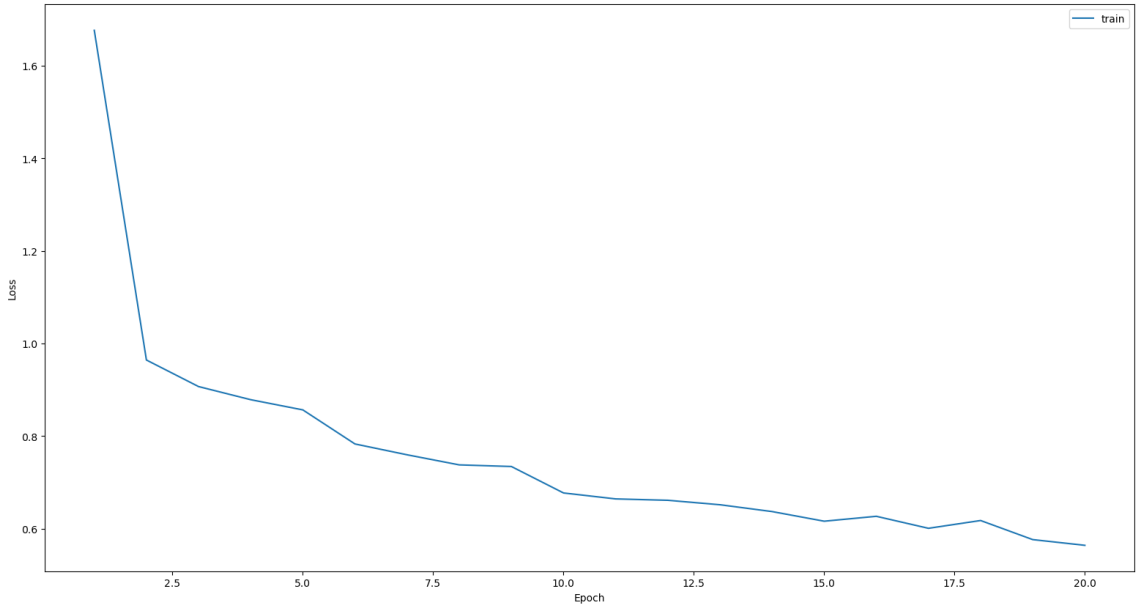

Figure 5: Train loss as a function of the number of epochs for the no-toolkit MLP

Question 2

- **1.** The best configuration in terms of validation accuracy was the one with learning rate 0.01, with a validation accuracy of 0.6535. Its final test accuracy is 0.6200.

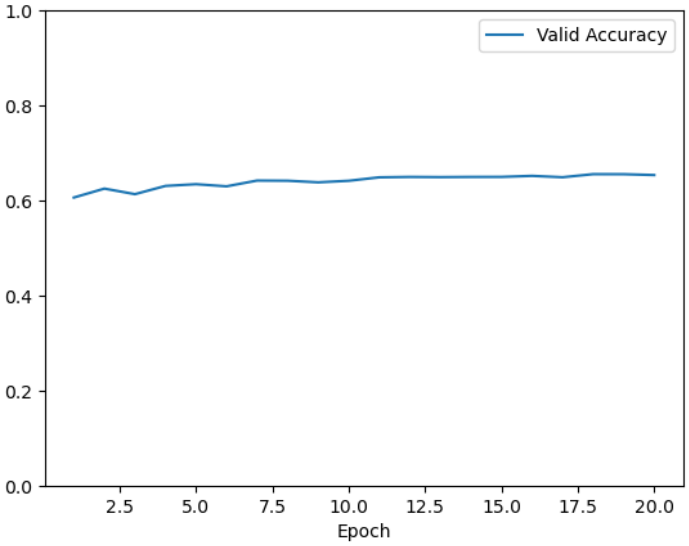

Figure 6: Validation accuracy as a function of the number of epochs for the logistic regression model with learning rate 0.01

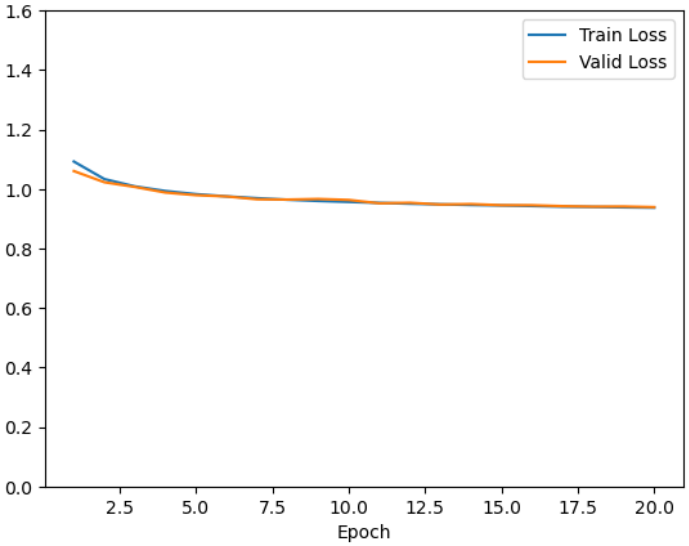

Figure 7: Train and validation loss as function of the number of epochs for the logistic regression model with learning rate 0.01

- **2. a)** The choice of the batch size hyperparameter imposes dealing with a trade-off between test accuracy and training time. Smaller batches produce more frequent gradient updates, but this comes at the cost of making more non-parallelizable computations. Moreover, as we decrease the batch size, the more noisy the gradient signal becomes, which, in the extreme case, causes unstable training. However, when chosen properly, a smaller batch size offers a regularizing effect, since it introduces noise in the gradient signal, which can help the model generalize better. Larger batch sizes, on the other hand, produce less frequent gradient updates, meaning that, in each epoch, we need to do less non-parallelizable computations and, therefore, each epoch takes less time to complete. However, in the extreme case, large batch sizes can lead to overfitting, since the gradient signal is less noisy.

  In this case, we can see that the model with batch size 16 has a better test accuracy than the

model with batch size 1024. However, the model with batch size 1024 has a much faster training time, in this case, 3.5 times faster. This is because, in each epoch, the model with batch size 16 needs to perform 64 times more non-parallelizable computations than the model with batch size 1024.

Batch Size	Test Accuracy	Execution time
16	0.7788	1min17.420sec
1024	0.7208	0min22.466sec

Table 3: Test accuracy and execution time for the feed-forward network with batch sizes 16 and 1024

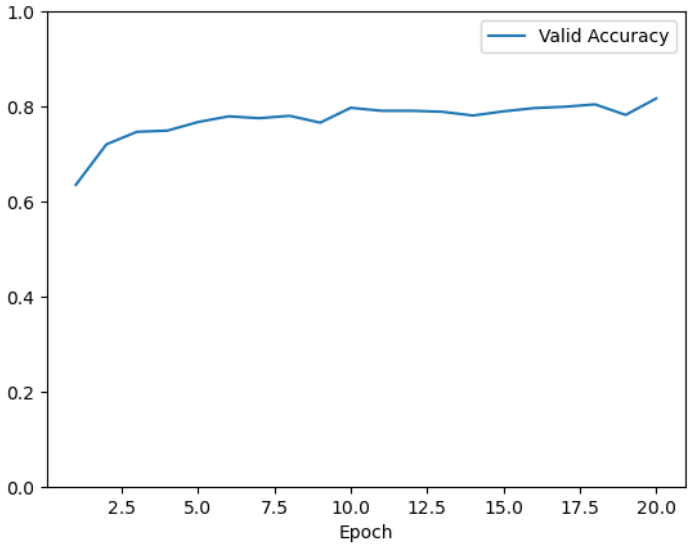

Figure 8: Train and validation accuracies as a function of the number of epochs for the feed-forward network using a batch size of 16

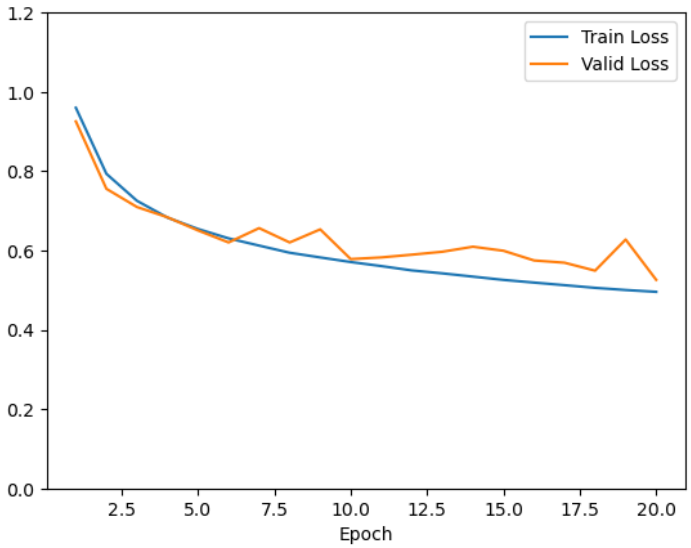

Figure 9: Train and validation loss as a function of the number of epochs for the feed-forward network using a batch size of 16

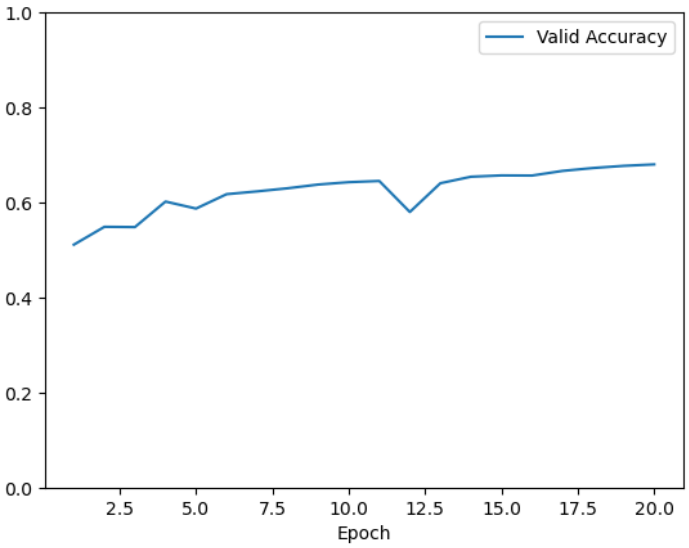

Figure 10: Validation accuracy as a function of the number of epochs for the feed-forward using a batch size of 1024

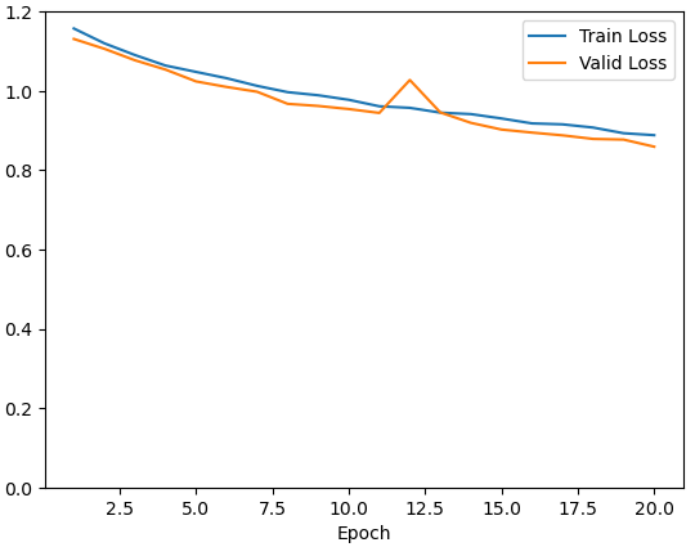

Figure 11: Train and validation loss as a function of the number of epochs for the feed-forward using a batch size of 1024

- **2. b)** The best and worst configurations in terms of validation accuracy were the ones with learning rates 0.1 and 1, respectively. The best configuration has a validation accuracy of 0.8168 and a test accuracy of 0.7788 which is also the best test accuracy achieved across all configurations. The worst configuration, on the other hand, has a validation accuracy of 0.4721 and a test accuracy of 0.4726. The configuration using a learning rate of 1 is the worst because the model overshoots the optimal gradient descent step, which, in turn, inhibits the model from converging to a good solution. On the other hand, the configuration using a learning rate of 0.1 is the best because it is small enough to allow the model to converge to a good solution, but large enough to allow the model to converge in a reasonable amount of time.

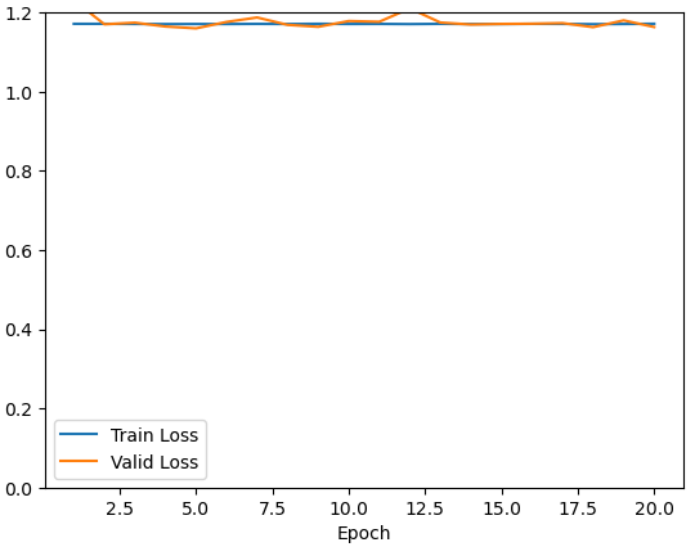

Figure 12: Train and validation loss as a function of the number of epochs for the feed-forward network using a learning rate of 1

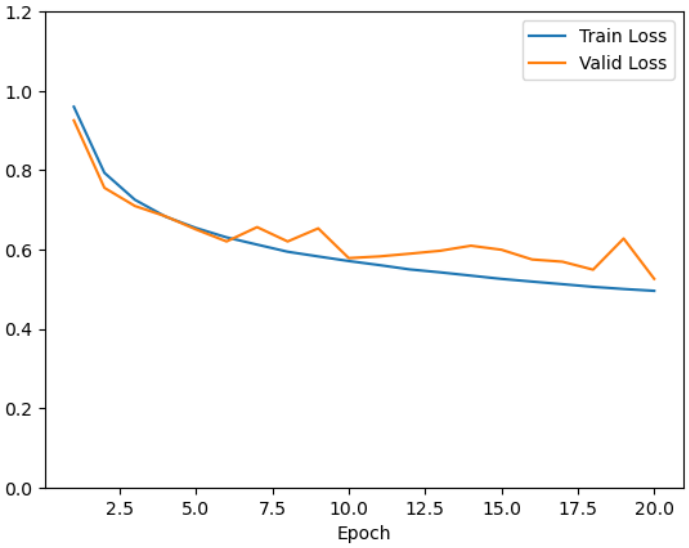

Figure 13: Train and validation loss as a function of the number of epochs for the feed-forward network using a learning rate of 0.1

- **2. c)** The best and worst configurations in terms of validation accuracy were the one using dropout rates of 0.2 and the one using L2 regularization parameter set to 0.0001, respectively. The best test accuracy was achieved by the configuration using dropout rates of 0.2, with a test accuracy of 0.8034. The default model showed signs of overfitting, since the validation loss stopped improving after around 50 epochs, while the training loss kept improving. This means that the model started to learn the training data too well, which caused it to generalize poorly to the validation data. Using a L2 a regularization parameter of 0.0001 did not help much the model generalize better, since the validation loss stopped improving after around 60 epochs. However, the model using dropout rates of 0.2 was able to generalize better, since the validation loss kept improving until 120 epochs. This improved generalization capability is reflected in the test accuracy achieved by each model, with the model using dropout rates of 0.2 achieving a test accuracy of 0.8034, while the

default model and the model using L2 regularization parameter of 0.0001 achieved test accuracies of 0.7750 and 0.7864, respectively.

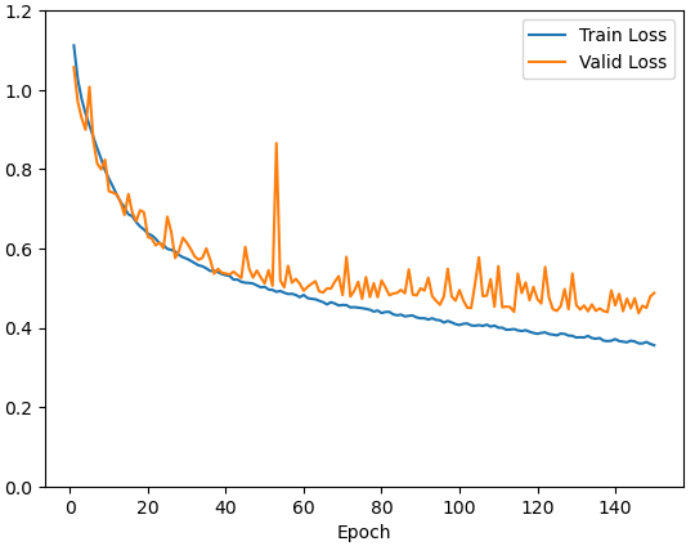

Figure 14: Train and validation loss as a function of the number of epochs for the default feed-forward network

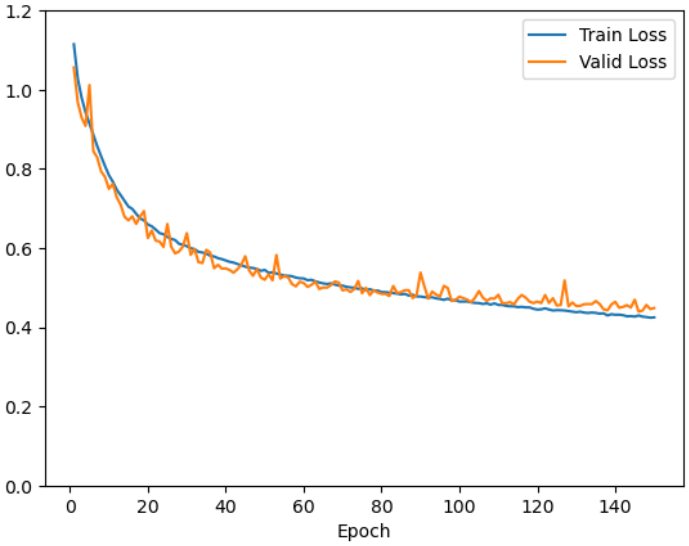

Figure 15: Train and validation loss as a function of the number of epochs for the feed-forward network using dropout rates of 0.2

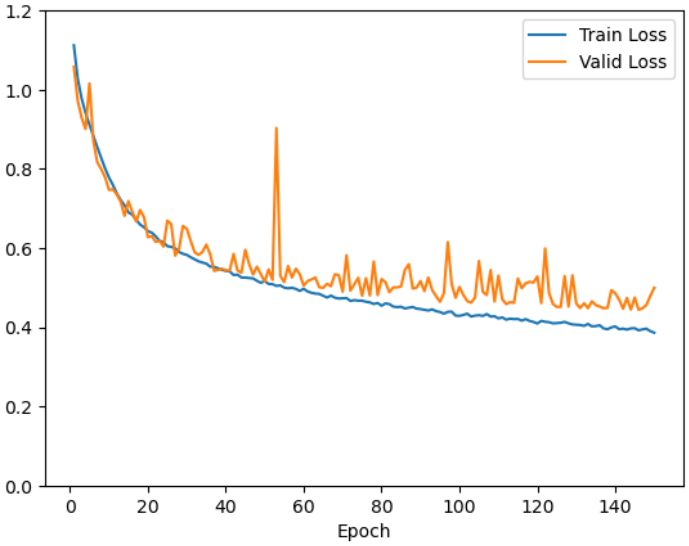

Figure 16: Train and validation loss as a function of the number of epochs for the feed-forward network using L2 regularization parameter set to 0.0001

Question 3

- **1. a)** For the case with $D = 2, A = -1, B = 1$, there are 4 data points in our domain, calculated in Figure 17, and represented in Figure 18.

$$f(-1,1): \qquad \sum_{i=1}^{2} x_i = -1 + 1 = 0 \qquad 0 \in [-1,1] \Rightarrow f(-1,1) = 1$$

$$f(1,1): \qquad \sum_{i=1}^{2} x_i = 1 + 1 = 2 \qquad 2 \notin [-1,1] \Rightarrow f(1,1) = -1$$

$$f(-1,-1): \qquad \sum_{i=1}^{2} x_i = -1 + -1 = -2 \qquad -2 \notin [-1,1] \Rightarrow f(-1,-1) = -1$$

$$f(1,-1): \qquad \sum_{i=1}^{2} x_i = 1 + (-1) = 0 \qquad 0 \in [-1,1] \Rightarrow f(1,-1) = 1$$

Figure 17: Calculation of $f(x)$ for $D = 2, A = -1, B = 1$

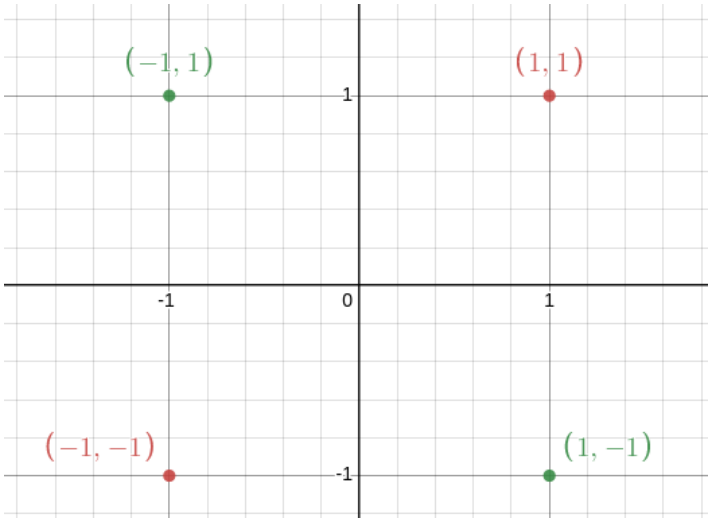

Figure 18: Space of $f(x)$ for $D = 2, A = -1, B = 1$. Green represents class 1 and red represents class -1.

For this example, we can see that the data is not linearly separable, since it is equivalent to the XOR problem which is also not linearly separable. For this reason, we can conclude we cannot use a single perceptron to classify this data.

- **1. b)**

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

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

Let $S = \sum_{i=1}^{D} x_i$, then:

$$f(x) = \begin{cases} 1 & \text{if } A \leq S \text{ and } S \leq B \\ -1 & \text{otherwise} \end{cases}$$

Since $A, B, S \in \mathbb{Z}$, for this domain, the equation above is equivalent to:

$$f(x) = \begin{cases} 1 & \text{if } A - 0.5 \leq S \text{ and } S \leq B + 0.5 \\ -1 & \text{otherwise} \end{cases}$$

$$f(x) = \begin{cases} 1 & \text{if } 0 \leq S - A + 0.5 \text{ and } 0 \leq -S + B + 0.5 \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

For the following:

$$b^{[0]} = \begin{bmatrix} -A + 0.5 \\ B + 0.5 \end{bmatrix} \qquad W^{[0]} = \begin{bmatrix} 1 & \dots & 1 \\ -1 & \dots & -1 \end{bmatrix}$$

We get that:

$$b^{[0]} + W^{[0]} \cdot \mathbf{x} = \begin{bmatrix} -A + 0.5 \\ B + 0.5 \end{bmatrix} + \begin{bmatrix} 1 & \dots & 1 \\ -1 & \dots & -1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{D} x_i - A + 0.5 \\ -\sum_{i=1}^{D} x_i + B + 0.5 \end{bmatrix} = \begin{bmatrix} S - A + 0.5 \\ -S + B + 0.5 \end{bmatrix}$$

This result is quite similar to the conditions where the class is $+1$, in equation 1.

The next step in the MLP is applying the sign function, which applied to the result above, gives us a similar boolean result to the boolean requirements in equation 1, where $true = 1$, and $false = -1$.

$$\text{sign}(b^{[0]} + W^{[0]} \cdot \mathbf{x}) = \begin{bmatrix} S - A + 0.5 \geq 0 \\ -S + B + 0.5 \geq 0 \end{bmatrix}$$

From analysing equation 1, it should get class $+1$ when both conditions are true, that is, when the result is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

The other possible cases are, by permutation, $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} -1 \\ -1 \end{bmatrix}$. All of these should equate to class -1, since they mean at least one of the original conditions is false. However, case $\begin{bmatrix} -1 \\ -1 \end{bmatrix}$ is impossible:

$$\begin{bmatrix} -1 \\ -1 \end{bmatrix} \Leftrightarrow \begin{cases} S - A + 0.5 < 0 \\ -S + B + 0.5 < 0 \end{cases} \Leftrightarrow \begin{cases} S + 0.5 < A \\ B < S - 0.5 \end{cases} \Rightarrow B < A$$

Impossible. $A \leq B$

The last weight and bias matrices will be as following:

$$b^{[1]} = \begin{bmatrix} -1 \end{bmatrix} \qquad W^{[1]} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

For the three scenarios, this will output what we want:

$$b^{[1]} + W^{[1]} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 - 1 = 1$$

$$b^{[1]} + W^{[1]} \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} = 0 - 1 = -1$$

$$b^{[1]} + W^{[1]} \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 0 - 1 = -1$$

Given the codomain of $\{-1, 1\}$, the final activation function, which is $sign(z)$, will not change the result, and will output directly the correct class.

The limits can be demonstrably proven to be robust. For this problem, adding an infinitesimal vector to the input vector, will only effect the $\sum_{i=1}^{D} x_i$. These infinitesimal vectors will result in a S', which is either infinitesimally bigger or smaller or equal than S, which are S^+, S^-, respectively.

Let $S-A$ and $-S+B$ be T and U, respectively, which as a subtraction between integers, are also integers.

$$\lim_{S'\to S} h_0(z_0) = \begin{bmatrix} S'-A+0.5\geq 0 \\ -S'+B+0.5\geq 0 \end{bmatrix} = \lim_{T'\to T \text{ and } U'\to U} h_0(z_0) \begin{bmatrix} T'\geq -0.5 \\ U'\geq -0.5 \end{bmatrix}$$

Since T, U are both integers, this inequality cannot flip with infinitesimal changes. As such, infinitesimal changes to S, in both directions, are equivalent to S.

- **1. c)** For ReLU, it is similar, and we can share work up from the equation 1:

$$f(x) = \begin{cases} 1 & \text{if } 0\leq S-A+0.5 \text{ and } 0\leq -S+B+0.5 \\ -1 & \text{otherwise} \end{cases}$$

$$f(x) = \begin{cases} 1 & \text{if } 0\geq -(S-A+0.5) \text{ and } 0\geq -(-S+B+0.5) \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

Considering the following weight and bias matrices:

$$b^{[0]} = \begin{bmatrix} A-0.5 \\ -B-0.5 \end{bmatrix} \qquad W^{[0]} = \begin{bmatrix} -1 & \dots & -1 \\ 1 & \dots & 1 \end{bmatrix}$$

We get that:

$$b^{[0]}+W^{[0]}\cdot\mathbf{x} = \begin{bmatrix} A-0.5 \\ -B-0.5 \end{bmatrix} + \begin{bmatrix} -1 & \dots & -1 \\ 1 & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix} = \begin{bmatrix} -\sum_{i=1}^{D} x_i + A - 0.5 \\ \sum_{i=1}^{D} x_i - B - 0.5 \end{bmatrix} = \begin{bmatrix} -S+A-0.5 \\ S-B-0.5 \end{bmatrix} =$$

$$= \begin{bmatrix} -(S-A+0.5) \\ -(-S+B+0.5) \end{bmatrix} \tag{3}$$

The result are the same conditions present in equation 2, but in this case, class +1 requires these values to be smaller than or equal to 0.

The next step in the MLP is applying the activation function, which will now be ReLU. Applied to the result above:

$$ReLU(\begin{bmatrix} -(S-A+0.5) \\ -(-S+B+0.5) \end{bmatrix}) = \begin{bmatrix} x\in\mathbb{R}_{\geq 0} \\ x\in\mathbb{R}_{\geq 0} \end{bmatrix}$$

From analysing equation 2, it should get class +1 when both conditions are true, that is, when the values in equation 3 are smaller or equal to zero, and false when bigger than zero. Applying $ReLU$ will convert negative numbers, and zero, to 0, and keep positive values as they are. This means that $x=0 \Leftrightarrow true$ and $x\in\mathbb{R}_+ \Leftrightarrow false$. As such, class +1 corresponds to both conditions being, true, that is, when this result matrix equals $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

The other possible cases are, by permutation, $\begin{bmatrix} 0 \\ x\in\mathbb{R}_+ \end{bmatrix}$, $\begin{bmatrix} x\in\mathbb{R}_+ \\ 0 \end{bmatrix}$ and $\begin{bmatrix} x\in\mathbb{R}_+ \\ x\in\mathbb{R}_+ \end{bmatrix}$. All of these should equate to class -1, since they mean at least one of the original conditions is false. However, like in the previous exercise, case $\begin{bmatrix} x\in\mathbb{R}_+ \\ x\in\mathbb{R}_+ \end{bmatrix}$ is impossible:

$$\begin{bmatrix} x \in \mathbb{R}_+ \\ x \in \mathbb{R}_+ \end{bmatrix} \Leftrightarrow \begin{cases} -(S - A + 0.5) > 0 \\ -(-S + B + 0.5) > 0 \end{cases} \Leftrightarrow \begin{cases} S + 0.5 < A \\ B < S - 0.5 \end{cases} \Rightarrow B < A$$

Impossible. $A \leq B$

The last weight and bias matrices will be as following:

$$b^{[1]} = \begin{bmatrix} 0 \end{bmatrix} \qquad W^{[1]} = \begin{bmatrix} -1 & -1 \end{bmatrix}$$

For the three scenarios, this will output what we want:

$$b^{[1]} + W^{[1]} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} = -0 - 0 + 0 = 0$$

$$b^{[1]} + W^{[1]} \cdot \begin{bmatrix} x \in \mathbb{R}_+ \\ 0 \end{bmatrix} = -x - 0 + 0 = -x$$

$$b^{[1]} + W^{[1]} \cdot \begin{bmatrix} 0 \\ x \in \mathbb{R}_+ \end{bmatrix} = -0 - x + 0 = -x$$

Finally, by applying the sign function, that is, our output activation, we will get:

$$sign(0) = 1$$

$$sign(-x) = -1$$

$$sign(-x) = -1$$

Which are the correct results for the three possible scenarios.

Like above, the limits can be demonstrably proven to be robust. For this problem, adding an infinitesimal vector to the input vector, will only effect the $\sum_{i=1}^{D} x_i$. These infinitesimal vectors will result in a $S^{'}$, which is either infinitesimally bigger or smaller or equal than S, which are S^{+}, S^{-}, respectively. Let $-S + A$ and $S - B$ be T and U, respectively, which as a subtraction between integers, are also integers.

$$\lim_{S' \to S} h_0(z_0) = ReLU(\begin{bmatrix} -(S' - A + 0.5) \\ -(-S' + B + 0.5) \end{bmatrix}) = \lim_{T' \to T \text{ and } U' \to U} h_0(z_0) = ReLU(\begin{bmatrix} T' - 0.5 \\ U' - 0.5 \end{bmatrix})$$

Since T, U are both integers, infinitesimal changes will not change the behaviour of the ReLU, which is converting negative values to 0, and keeping positive values unchanged. Infinitesimal changes will have no effect in changing the activation of the ReLU expression as a whole, since its decimal part is .5 . As such, the operation does not change meaning when approaching from different directions of S, and we can conclude that infinitesimal changes to S are equivalent to S.