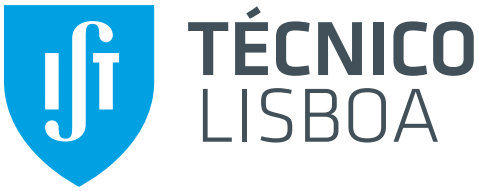


Electrical and Computer Engineering

Deep Learning

Homework 1

Authors:

Rodrigo Coimbra : 100078

Rodrigo Pereira : 100080

December 2023

Index

Contribution of each member to the project

For this project, our group divided the first two questions, assigning one to each member. Therefore, Rodrigo Coimbra (100078) addressed Question 1, and Rodrigo Pereira (100080) addressed Question 2. For the 3^{rd} group, since these are *'Show that'* questions, we decided that both would think about how to deduce what's asked and then join to discuss, so we did the question together.

1 Question 1 - Medical image classification with linear classifiers and neural networks

1.1 Question 1.1a

For this problem, we implemented the method *update_weights* for a perceptron model. Afterwards, we trained it for 20 epochs and obtained the following results.

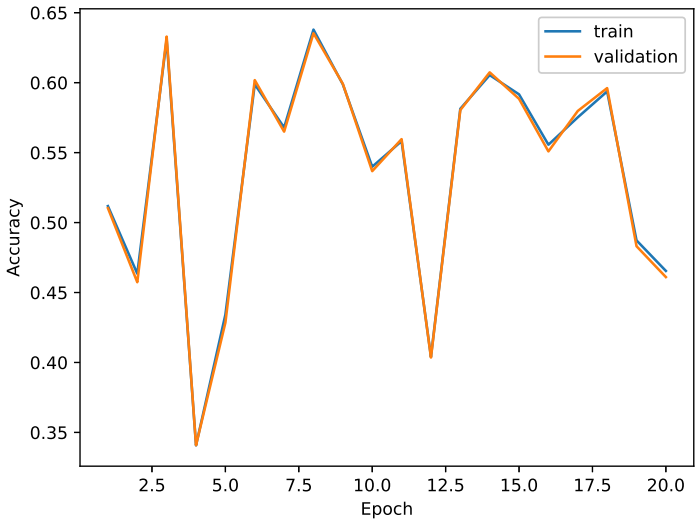


Fig. 1: Train and validation accuracies as a function of the epoch number

As we can see, the training and validation accuracies exhibit nearly identical trends. Additionally, we observe a very unstable trend, which may suggest that the perceptron is encountering difficulty in finding a linear decision boundary. This indicates that there may not exist a linear decision boundary capable of perfectly separating the data, requiring a non-linear decision boundary that this model is unable to achieve.

In the end, we obtained the following accuracies:

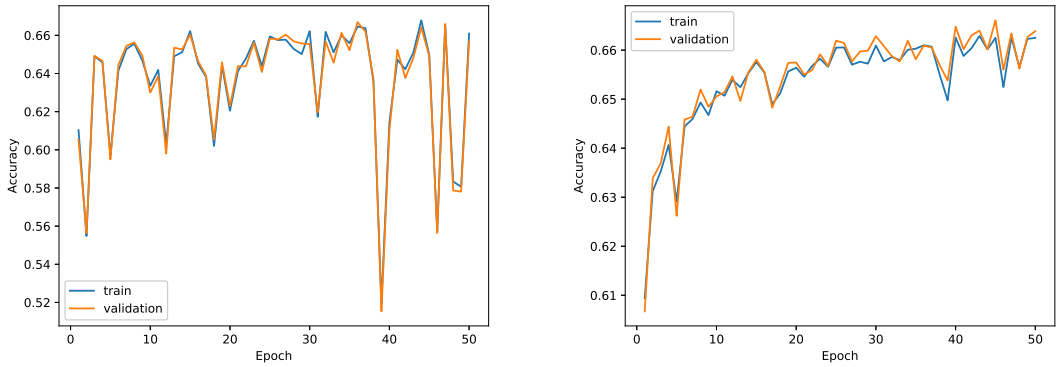
Train Acc	0.4654
Validation Acc	0.4610
Test Acc	0.3422

Tab. 1: Accuracy after 20 epochs

As can be seen, all the values are below the 0.5 accuracy mark, indicating that this perceptron cannot represent the data well. This is better seen when observing the test accuracy that is below 0.35, showing very poor results.

1.2 Question 1.1b

For this second problem, we were asked to implement the method *update_weights* again, but this time for a logistic regression instead. After that, we tested it with two learning rates, $\eta = 0.01$ and $\eta = 0.001$. The results obtained are represented in Fig. 2 and in Tab. 2.



(a) $\eta = 0.01$ (b) $\eta = 0.001$

Fig. 2: Train and validation accuracies as a function of the epoch number

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

Tab. 2: Accuracy after 50 epochs

As we can see, the final values of accuracy were almost the same, with the most noticeable difference being in the test accuracy, and it was a difference of less than 0.02. However, the plots show that the model with a learning rate of 0.001, despite being slower to make changes, had a much more consistent improvement in accuracy, with fewer significant decreases between epochs. Therefore, the lower learning rate was the best choice for this case.

1.3 Question 1.2a

To better comment on this claim, let's start by analyzing the first phrase. The claim asserts that a logistic regression model is not as expressive as a multi-layer perceptron, and indeed, this is true. Logistic regression is a linear model, meaning it has a linear decision boundary and can only solve linearly separable problems. In this case, we are dealing with pixel values of an image, which often exhibit a non-linear relationship between them. On the other hand, a multi-layer perceptron has multiple layers, with each hidden layer computing a representation of the input. This structure enables it to learn complex patterns.

Now, let's analyze the second phrase. Due to the convex nature of logistic regression, it becomes easy to train. This is because it has only one minimum (strictly convex), and therefore, utilizing a gradient method such as gradient descent always leads us to this minimum. On the other hand, for the multi-layer perceptron, it's not as simple. This is due to its multiple hidden layers, which often result in an optimization function with multiple local minima. Consequently, using gradient descent may not guarantee reaching the global minimum. To conclude, by demonstrating the truth of both sentences, we can affirm that the claim as a whole is also true.

1.4 Question 1.2b

For this problem, we implemented a multi-layer perceptron with a single hidden layer of 200 units. Additionally, their activation fuction was ReLU and we used a multinomial logistic loss. The results are as follows:

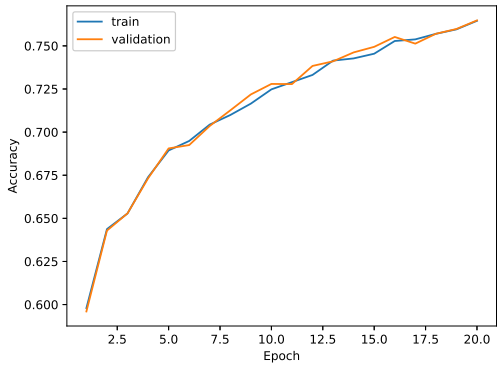


Fig. 3: Train and validation accuracies as a function of the epoch number

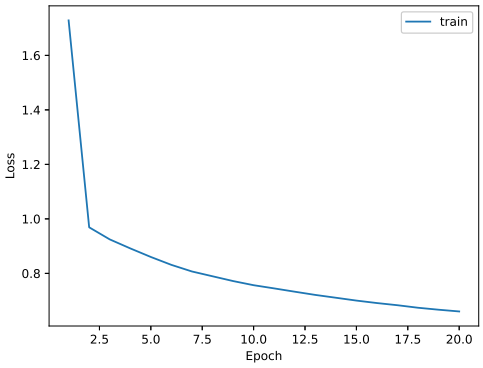


Fig. 4: Train loss as a function of the epoch number

Loss	0.6602
Train Acc	0.7645
Validation Acc	0.7648
Test Acc	0.7202

Tab. 3: Accuracy and loss after 20 epochs

As can be observed, the accuracies of this type of model are much better than before. This shows that with the proper choice of a non-linear activation function, MLP can have much better representations of the input to correctly predict labels, as explained in 1.3. Besides that, we can see that the loss function initially has a big decrease, but after a few epochs its descent slows down. The accuracy follows a similar trend, where, at the end, it keeps increasing much slower than in the beginning.

2 Question 2

For this Question, we were tasked with the development of a Pytorch neural network, using a skeleton code provided by the professors.

2.1 Question 2.1

For this first problem, we implemented a linear model with logistic regression with SGD. Then we trained the model for 20 epochs. The parameters necessary to use were a batch size of 16 and a learning rate that we were changing through different runs, using the following values: 0.001, 0.01, 0.1. Then we plotted the validation and train loss and validation accuracy. The plots obtained were the following:

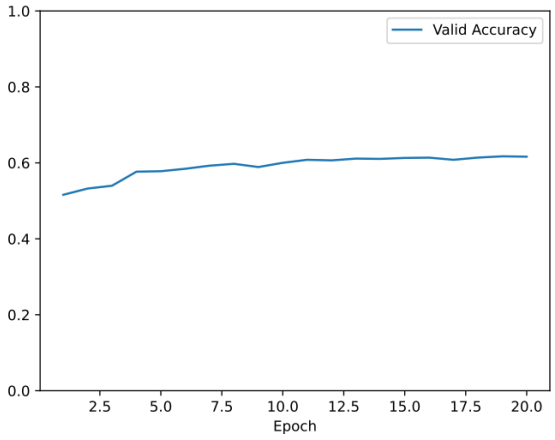


Fig. 5: Validation Accuracy with Lr=0.001

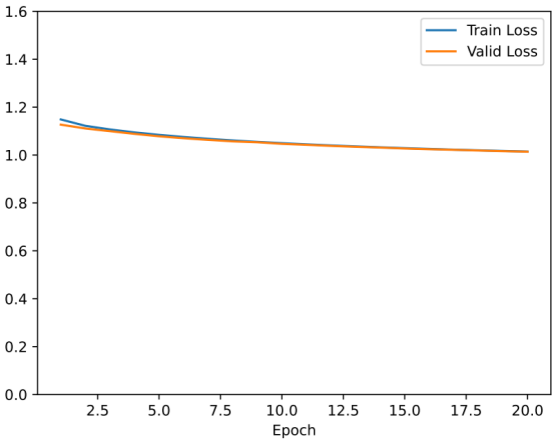


Fig. 6: Validation and Train Loss with Lr=0.001

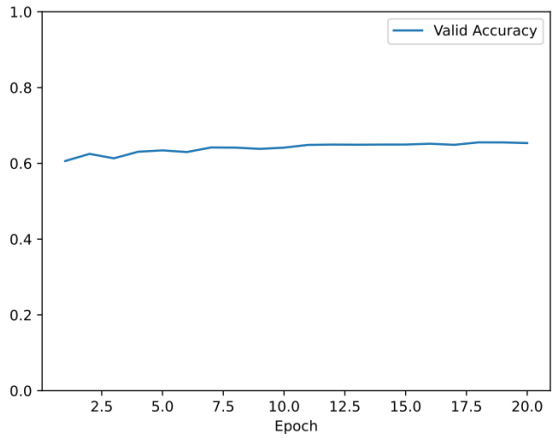


Fig. 7: Validation Accuracy with Lr=0.01

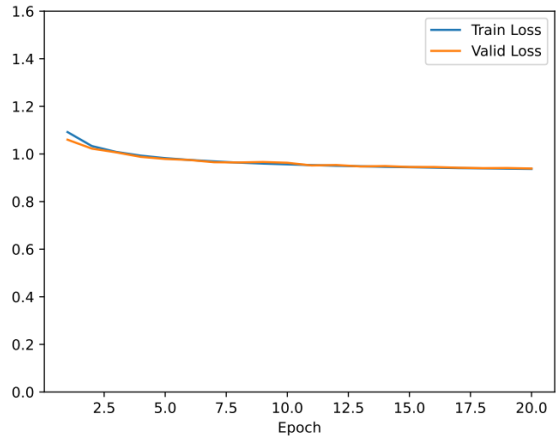


Fig. 8: Validation and Train Loss with Lr=0.01

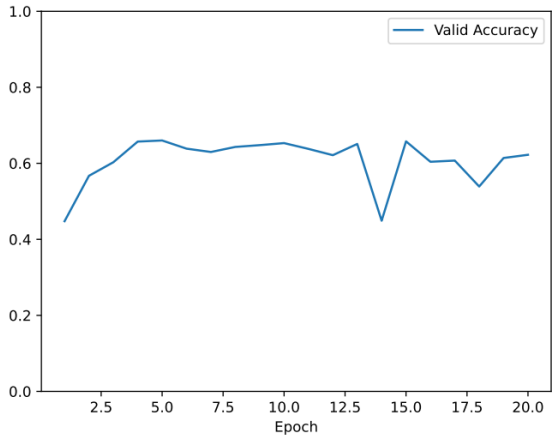


Fig. 9: Validation Accuracy with Lr=0.1

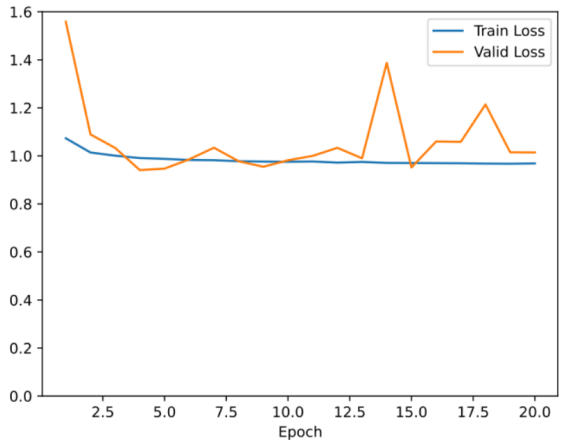


Fig. 10: Validation and Train Loss with Lr=0.1

Learning rate	0.001	0.01	0.1
Train Loss	1.0145	0.9370	0.9687
Validation Acc	0.6163	0.6535	0.6224
Test Acc	0.6503	0.6200	0.5577

Tab. 4

In Tab. 4 we can see the results at the final epoch for each model. The best test accuracy was obtained by the model with the lowest learning rate.

2.2 Question 2.2a)

Now, we were asked to make a feed-forward neural network using dropout regularization, with the hyperparameters and training models given. For this first part, we had to run our code with a batch size of 16 and 1024 and get the time that each one took to run, in addition to the plots of accuracy and loss.

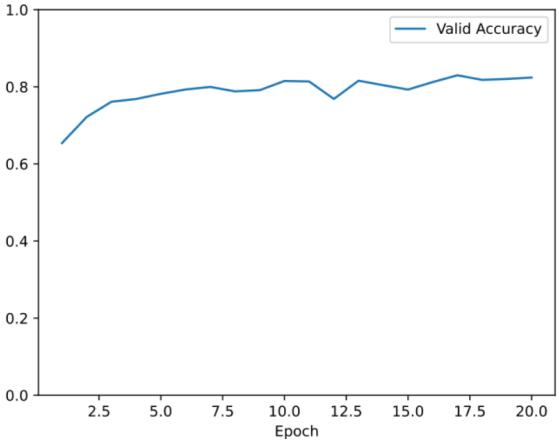


Fig. 11: Validation Accuracy with batch size 16

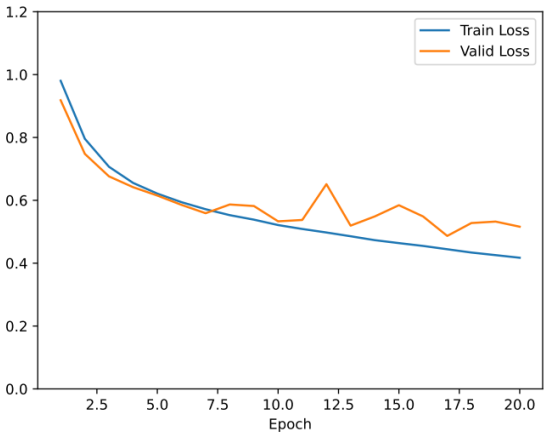


Fig. 12: Validation and Train Loss with batch size 16

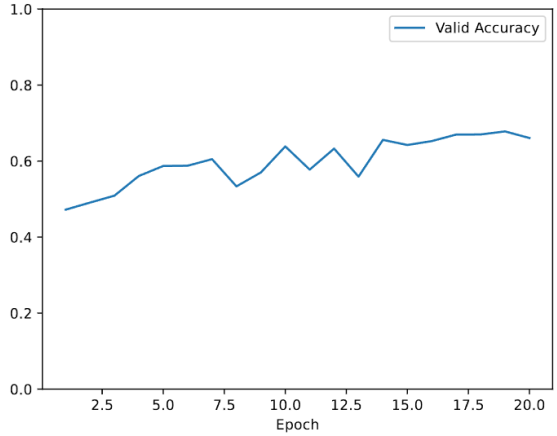


Fig. 13: Validation Accuracy with batch size 1024

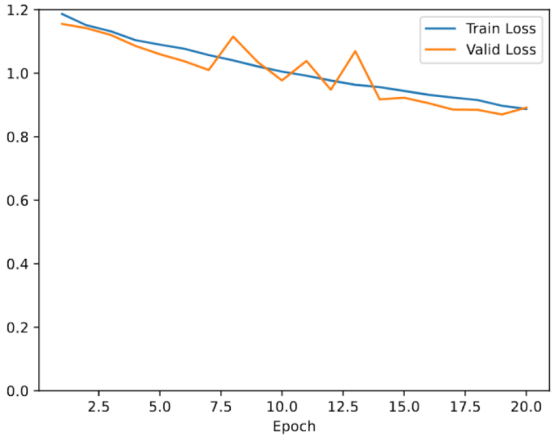


Fig. 14: Validation and Train Loss with batch size 1024

As we can see in Table 5, a batch size of 1024 took 27 seconds and 16 took almost 2 minutes. This can be explained since by taking smaller batches, our code has to process less data once at a time. If we increase it, our MLP is able to process more data at once making it faster. Even though our larger batch size runs way faster, its accuracy is lower, which means that a lower batch size might be the best way to approach this problem, however more tests are needed to assume that.

Batch Size	1024	16
Train Loss	0.8874	0.4207
Validation Acc	0.6647	0.8230
Test Acc	0.6975	0.7486
Time	0m27,178s	1m56,498s

Tab. 5

2.3 Question 2.2b)

Like the question 2.1, we had to execute our MLP with the default parameters, except the learning rate had the values 1, 0.1, 0.01, 0.001. Then we plotted the train/validation loss and validation accuracy.

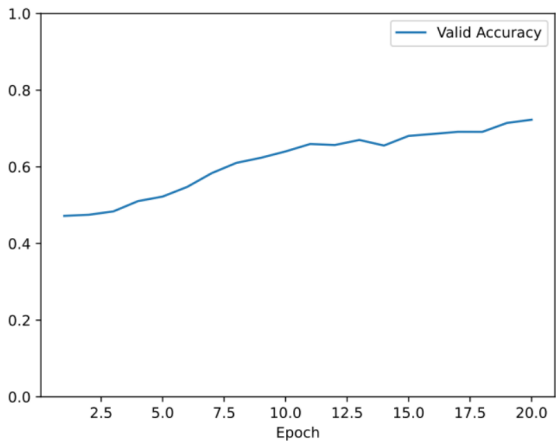


Fig. 15: Validation Accuracy with Lr=0.001

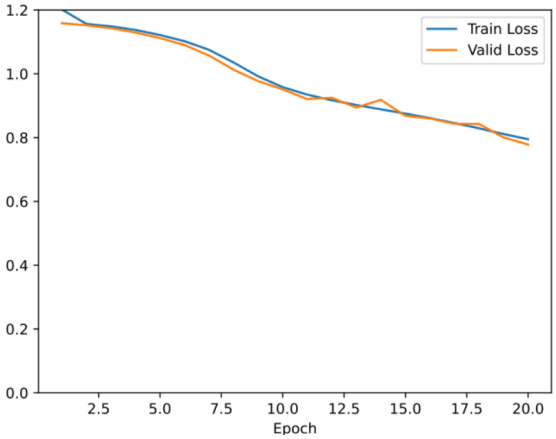


Fig. 16: Validation and Train Loss with Lr=0.001

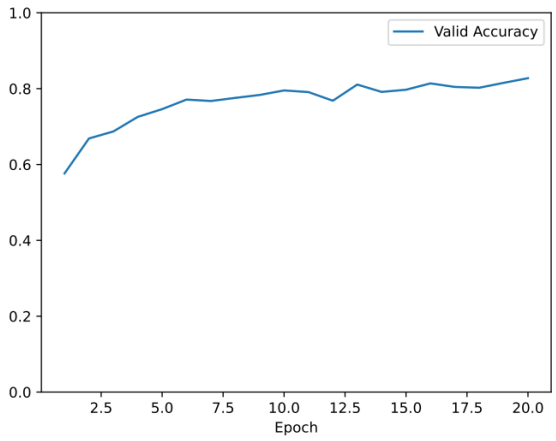


Fig. 17: Validation Accuracy Lr=0.01

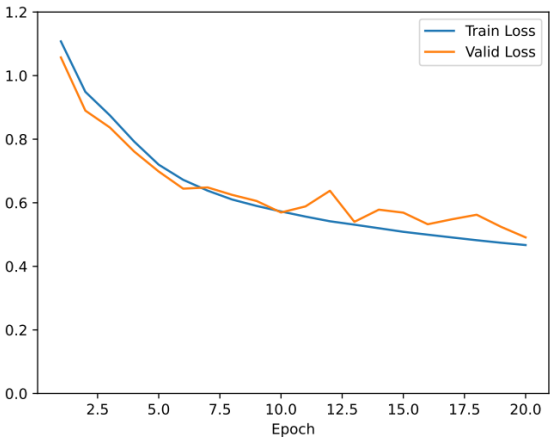


Fig. 18: Validation and Train Loss with Lr=0.01

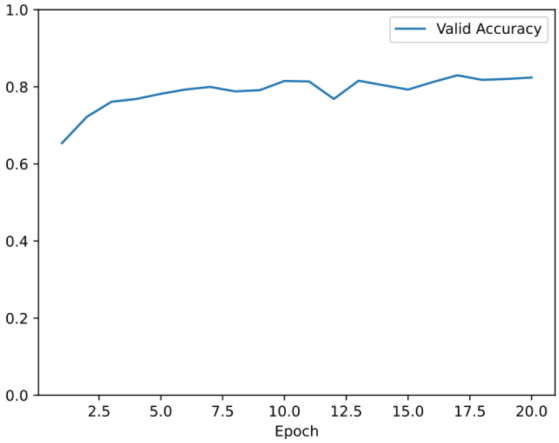


Fig. 19: Validation Accuracy with Lr=0.1

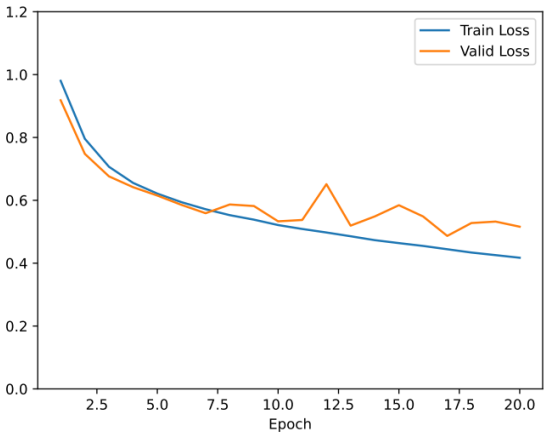


Fig. 20: Validation and Train Loss with Lr=0.1

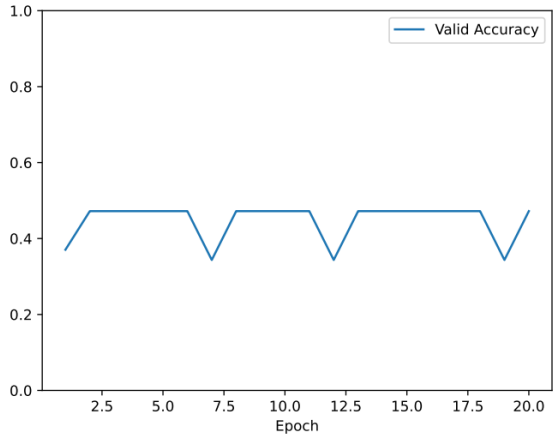


Fig. 21: Validation Accuracy with Lr=1

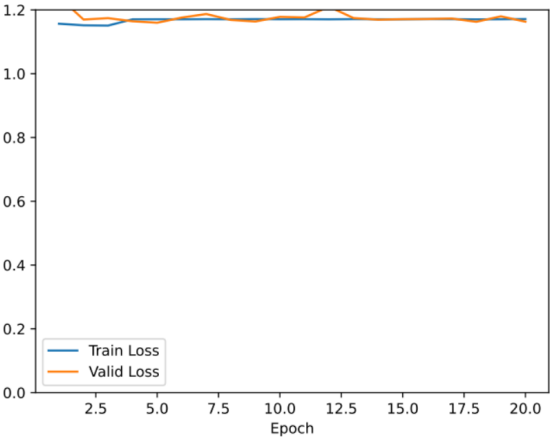


Fig. 22: Validation and Train Loss with Lr=1

Looking at Table 6, we noticed the learning rate that obtained the best results was 0.01. Since the learning rate scales the steps taken to the optimal parameters, a learning rate too high might cause an overshoot of those parameters, making it sub-optimal. Something similar can be said about the lower learning rate where it takes too long to obtain the optimal parameters.

Learning rate	0.001	0.01	0.1	1
Train Loss	0.7949	0.4669	0.4358	1.1714
Validation Acc	0.7230	0.8276	0.8253	0.4721
Test Acc	0.7410	0.7561	0.7505	0.4726

Tab. 6

2.4 Question 2.2c)

For the last problem of Question 2, we changed the number of epochs from 20 to 150 and made the batch size 256. We also did a run with L2 parameter set to 0.0001

and another with dropout at 0.2.

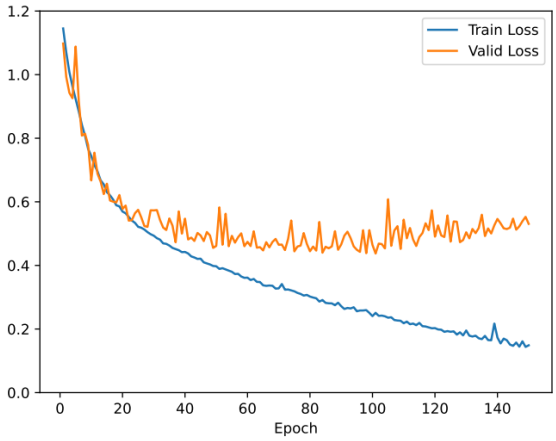


Fig. 23: Validation and Train Loss with batch size 256

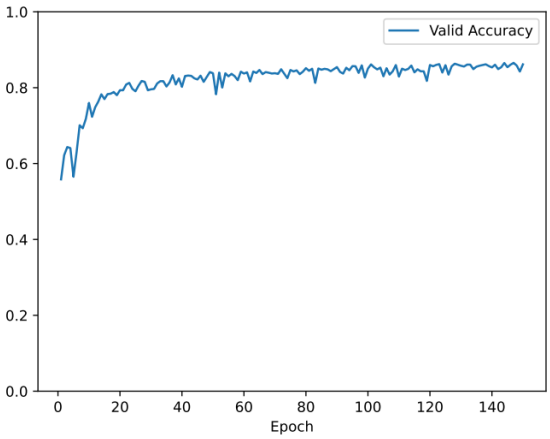


Fig. 24: Validation Accuracy with batch size 256

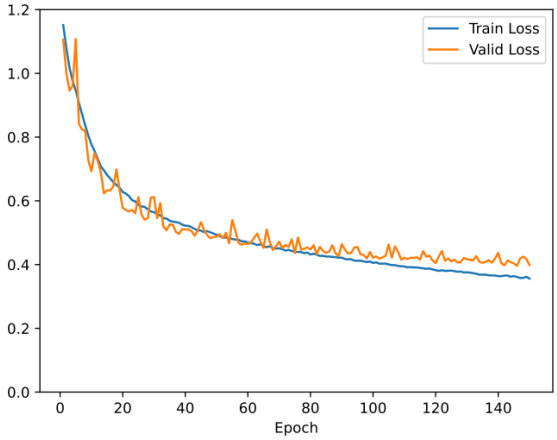


Fig. 25: Validation and Train Loss with dropout 0.2

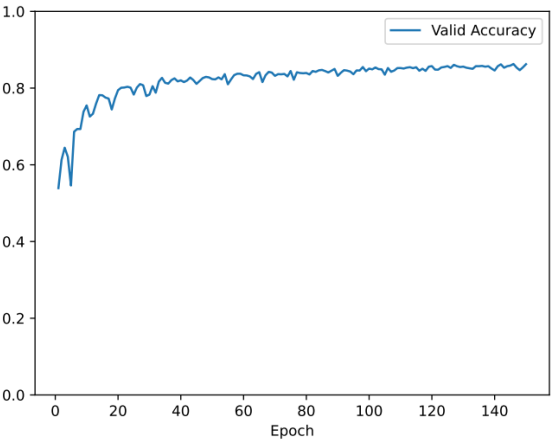


Fig. 26: Validation Accuracy with dropout 0.2

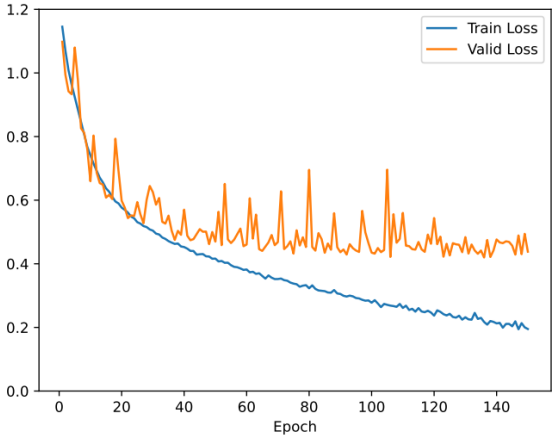


Fig. 27: Validation and Train Loss with L2 0.0001

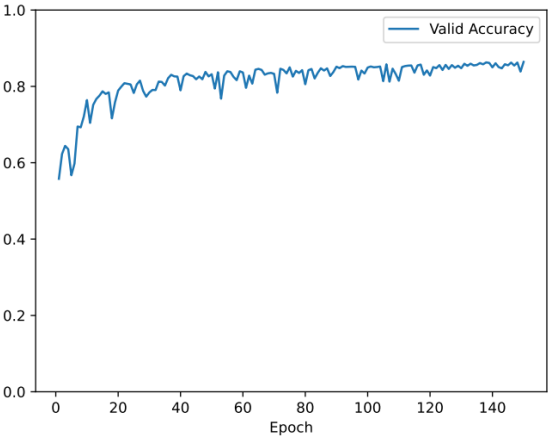


Fig. 28: Validation Accuracy with L2 0.0001

Looking at Figure 23 we notice that there is overfitting since our training loss kept decreasing while our validation loss kept close to the same values. Comparing the three models tested, we see that the one with the best accuracy is the one where we use dropout. As we can see, both models, except the one with dropout, appear to have some form of overfitting, where our loss kept decreasing but our accuracy stagnated. One reason that may justify the fact that the model with dropout regularization worked better is that the dropout is used, usually, to keep the model from overfitting. Even though the L2 regularization also pretends to prevent overfitting, it depends on the type of data and architecture which might not be the most effective method for this specific problem.

	L2	Dropout	Nothing
Train Loss	0.1951	0.3561	0.1488
Validation Acc	0.8642	0.8624	0.8615
Test Acc	0.7807	0.7864	0.7543

Tab. 7

3 Question 3

3.1 Question 3a)

For this problem, we were tasked to prove that it is not possible to compute f(x) through a single perceptron. When we look at this problem with D=2, which means we are looking at the problem in 2 dimensions. The only possible combination of points that we have are (1,1), (-1,1), (1,-1) and (-1,-1). This situation is similar to the XOR problem. When using a single perceptron, we would want to categorize the points (1,1) and (-1,-1) together using a single line, however we also want to make the (-1,1) and (1,-1) be part of the same group. This situation is impossible using a single line but using an MLP it is possible to compute, as shown in figure ??.

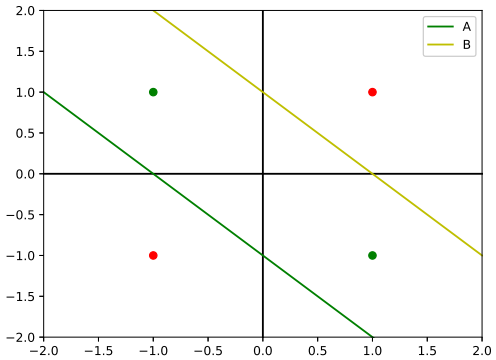


Fig. 29

3.2 3b

For this question, the best approach is start by thinking when D = 2. When this happens we have the following equations:

$$signal_1 = sign(X_1 \times W_{11}^{(1)} + X_2 \times W_{12}^{(1)} + B_1^{(1)}) \tag{1}$$

$$signal_2 = sign(X_1 \times W_{21}^{(1)} + X_2 \times W_{22}^{(1)} + B_2^{(1)}) \tag{2}$$

$$output = sign(signal_1 \times W_{11}^{(2)} + signal_2 \times W_{12}^{(2)} + B^{(2)}) \tag{3}$$

Looking now to the problem, we know that if our $\sum X$ is between [A,B] it will be equal to 1. Therefore, if we want to state the required conditions to be 1 we will end up with the following:

$$X_1 + X_2 \leq B \Leftrightarrow B - X_1 - X_2 \geq 0$$
$$X_1 + X_2 \geq A \Leftrightarrow X_1 + X_2 - A \geq 0$$

If we now look to both *signal* equations (1 and 2), we can easily see that we can transform this two inequations by modifying the weights(1 to the sums and -1 to minus) and the bias(by adding -A or B). Additionally, due to them having the sign as their activation function, we can make them be 1 if True and -1 if False. Therefore, we obtained the following:

$$W^{(1)} = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \qquad B^{(1)} = \begin{bmatrix} B \\ -A \end{bmatrix}$$

Now that we have both conditions for the next layer, we just need perform an *AND* operation, where both conditions are true, giving us our final output of 1 if its sum is between [A, B] and -1 if it is not part of our "desired solution".

To compute the AND function we just make a sum of the weights, since we have the signal function as our activation function for our hidden layer, we want that when our outputs from each node are 1, the final output also be 1 and if not to be -1, making the following condition:

$$W_1 * h_1 + W_2 * h_2 \geq 2 \Rightarrow f(x) = 1$$

$$W_1 * h_1 + W_2 * h_2 < 2 \Rightarrow f(x) = -1$$

Knowing that h_1 and h_2 are either 1 or -1, we want that both our outputs from the hidden layer have the same weight. Additionally, we want it to be 1 when the sum of the two nodes of the hidden layer are greater or equal than $2 \times weight\_value$. In this case we put both weights at 1, then we make our bias equals to $2 \times 1 = 2$ obtaining the following weights and biases for our output layer:

$$W^{(2)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad B^{(2)} = -2$$

We also want to keep our model immune to noise, for that we add a value to the bias. Let's use $x_1 = 1.001$ and $x_2 = 1.001$ as an example. If our $B = 2$, we would want our f(x)=1 however, that doesn't apply because of the noise. Lets look at our equations once again.

$$X_1 + X_2 \leq B \Leftrightarrow B - X_1 - X_2 \geq 0$$

$$X_1 + X_2 \geq A \Leftrightarrow X_1 + X_2 - A \geq 0$$

Since our output layer is working as the logical port AND, we then want to make this condition true $B - X_1 - X_2 \geq 0$. However, looking at the example purposed before, this condition will output as false, since $\sum x_i$ will be grater than B. But if we add a value to this inequation, the condition will be still be true. Making it:

$$B - X_1 - X_2 + b_1 \geq 0$$

This new value should not be greater or equal than 1 nor less than 0, since assuming it's greater or equal than 1 could make -1 values become 1. This is only a true solution, because the values X1, X2, A and B are all integers, so adding a decimal bias won't affect the outcome of other values. Thinking about the previous example, let's make B=1 and keep the x_i values but without any noise. We know that those points are outside of our model, resulting -1 but making the calculations we will have $-1 \geq 0$ and this false. However, adding a constant value greater or equal than 1, we will make it true which is not what is purposed. It is also important to note that the value can be as close to one, as close the noise, so if the noise is 0.2 then our bias to correct that noise should not be greater than 1-0.2. The reason to keep it greater than 0 is so that it keeps being always positive since making it negative could affect the noise cancellation. Applying the same thinking to the A inequation we will obtain the following $B^{(1)}$:

$$B^{(1)} = \begin{bmatrix} B + b_1 \\ -A + b_2 \end{bmatrix}$$

All this can be generalized to multiple dimensions, because by doing this way, the multiple dimensions will only affect the first layer, and instead of the previous inequations we will end up with this:

$$B - \sum_{n=1}^{D} X_i + b1 \geq 0$$

$$\sum_{n=1}^{D} X_i - A + b2 \geq 0$$

After all this, we obtain the following weights and biases:

$$W^{(1)} = \begin{bmatrix} -1 & -1 & ... & -1 \\ 1 & 1 & ... & 1 \end{bmatrix} \qquad B^{(1)} = \begin{bmatrix} B + b_1 \\ -A + b_2 \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad B^{(2)} = -2$$

3.3 Question 3c)

For the final problem we were tasked to try to achieve the same result as in Question 3b) but this time, instead of using the signal function as our activation function, we must use the ReLU in the hidden layer. Once again thinking only in two dimensions, we have the following equations for the output of the hidden layer and the final output:

$$relu_1 = max(0, X_1 \times W_{11}^{(1)} + X_2 \times W_{12}^{(1)} + B_1^{(1)}) \tag{4}$$

$$relu_2 = max(0, X_1 \times W_{21}^{(1)} + X_2 \times W_{22}^{(1)} + B_2^{(1)}) \tag{5}$$

$$output = sign(relu_1 \times W_{11}^{(2)} + relu_2 \times W_{12}^{(2)} + B^{(2)}) \tag{6}$$

In this problem, we can't use the same weights and biases as the problem before since we now run into a problem because of ReLU as an activation function. With ReLU, we can't make our logical our as before, because now one node could be greater or equal than 0 (one of the conditions explained before True) with values different from 1. This makes that we cannot reduce a constant bias to assume the function of AND as before. To do that, we changed the conditions of the nodes and ended up with something similar to what we had before. For that, we created the following conditions:

$$X_1 + X_2 \leq B \Leftrightarrow X_1 + X_2 - B \leq 0 \implies X1 + X2 - B - c < 0 \tag{7}$$

$$X_1 + X_2 \geq A \Leftrightarrow A - X_1 - X_2 \leq 0 \implies A - X1 - X2 - c < 0 \tag{8}$$

*Being c a positive constant with a very small value ($c \to 0^+$), in order to every number that was 0, will now be less than 0.

If we now look to our ReLU activation functions, we know that it will be 0 if the conditions describe above are met and a positive value greater than 0 otherwise. Therefore, we obtain the following:

$$W^{(1)} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \qquad B^{(1)} = \begin{bmatrix} -B - c \\ A - c \end{bmatrix}$$

With this approach we now know that if both outputs from the hidden layer are 0 we have a point that is between [A, B] and, therefore, the output should be 1. Otherwise, we just need that output to be -1.

Looking now to the output activation function, sign function, we can observe that it will only output 1 if the input is ≥ 0. Therefore, we need to negate the

values so that the positive numbers from the output of the hidden layer become negative, while the 0 remains unchanged.

$$-h_1 + -h_2 + bias \geq 0 \Rightarrow f(x) = 1$$
$$-h_1 + -h_2 + bias < 0 \Rightarrow f(x) = -1$$

*Being h_1 and h_2 the outputs from the nodes of the hidden layer As observed, if our goal is to output only 0 as 1, there is no need for any bias. With this approach, if any of the nodes in the hidden layer produces a value other than 0, we simply output -1. So we obtain the following:

$$W^{(2)} = \begin{bmatrix} -1 & -1 \end{bmatrix} \qquad B^{(2)} = 0$$

However, this solution in not robust against perturbations, so we need to adjust the values that goes to the first hidden layer.

Looking again to the equations 7 and 8, we can see that if X1 and X2 have a little perturbation they can go from < 0 to ≥ 0, making the output of that node positive when it shouldn't be. To solve this, we can subtract a bias, impeding that little perturbations on the input to affect the result. This is only possible to be done without affecting other outputs due to the fact that all the values (X1,X2, A,B) being integers, therefore decimal values like 0.1 won't affect it. So we have, for example, that $bias\_extra = -c - new\_bias = -0.1$. So the final values will be:

$$W^{(1)} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \qquad B^{(1)} = \begin{bmatrix} -B - 0.1 \\ A - 0.1 \end{bmatrix}$$

All this can be generalized to multiple dimensions, because by doing this way they will only affect the first layer, and instead of eq. 7 and 8 we will end up with:

$$\sum_{n=1}^{D} X_i - B - 0.1 < 0$$
$$A - \sum_{n=1}^{D} X_i - 0.1 < 0$$

To conclude, we can solve this problem with the following values to the weights and Biases:

$$W^{(1)} = \begin{bmatrix} 1 & 1 & ... & 1 \\ -1 & -1 & ... & -1 \end{bmatrix} \qquad B^{(1)} = \begin{bmatrix} -B - 0.1 \\ A - 0.1 \end{bmatrix}$$
$$W^{(2)} = \begin{bmatrix} -1 & -1 \end{bmatrix} \qquad B^{(2)} = 0$$