Deep Learning

Electrical and Computer Engineering

Homework 1 - Report

Students:

Alexandre Reis (100123) alexandre.leite.reis@tecnico.ulisboa.pt
Duarte Morais (100163) duarte.morais@tecnico.ulisboa.pt

Group 49

2023/2024 – First Semester, Q2

Contribution of each member of the group in this project: Question 1 was solved entirely (code, experimental results and report) by Duarte Morais (student 100163); Question 2 was solved entirely (code, experimental results and report) by Alexandre Reis (student 100123); Question 3 (a) and part of question 3 (b) was solved by Duarte Morais (100163); Part of question 3 (b) and the entire question 3 (c) were solved by Alexandre Reis (100123).

1 Question 1

1.1 Question 1.1

1.1.1 Question 1.1 (a)

In Figure 1, 20 training epochs were completed by the perceptron model. To achieve this, the updateWeight() [2] method was implemented to adapt model weights based on the accuracy of predictions. In cases where predictions were incorrect, the weight of the correct class was increased, and the weight of the incorrect class was decreased. This can be expressed as follows:

$$\mathbf{w}_{y_n}^{(k+1)} = \mathbf{w}_{y_n}^{(k)} + \varphi(\mathbf{x}_n)$$
$$\mathbf{w}_{\hat{y}_n}^{(k+1)} = \mathbf{w}_{\hat{y}_n}^{(k)} - \varphi(\mathbf{x}_n)$$

In this context, $w_{y_n}^{(k)}$ represents the weight vector for the correct class, $w^{(k)}{}_{\hat{y}_n}$ denotes the weight vector for the incorrect class, and $\varphi(x_n)$ is a function involving the input vector x_n.

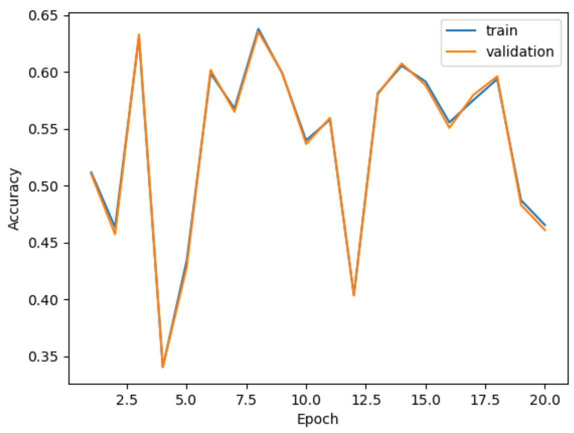


Figure 1: Perceptron Model with 20 Epochs

Metric	Value
Final Train Accuracy	0.4654
Final Validation Accuracy	0.4610
Final Test Accuracy	0.3422

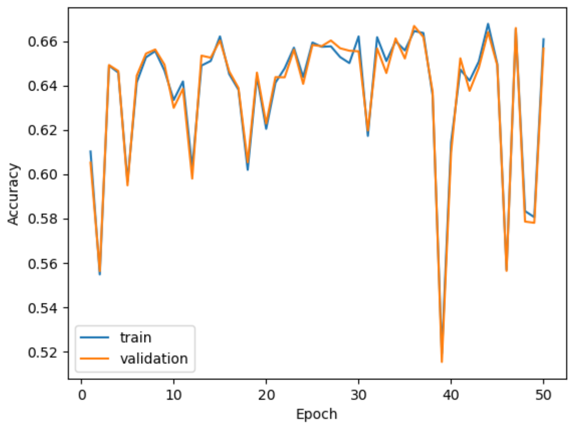
Table 1: Accuracy Metrics for the Perceptron

1.1.2 Question 1.1 (b)

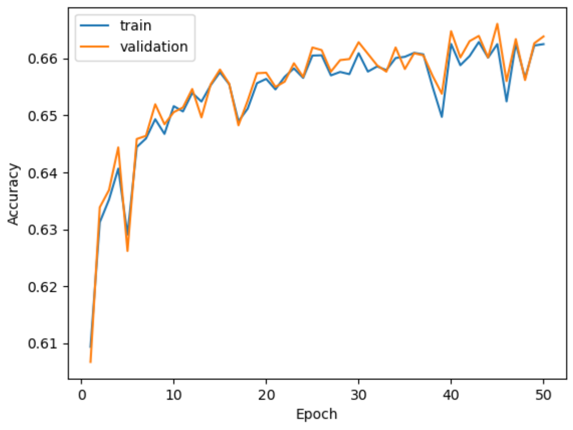
After implementing and running logistic regression with stochastic gradient descent, it was observed that a learning rate of $\eta = 0.001$ result in better convergence compared to $\eta = 0.01$. While a learning rate of $\eta = 0.01$ implies a faster convergence, sometimes leads to the minimum overshooting. This overshooting, could cause oscillations or divergence in the algorithm, as demonstrated in Figure 2a.

The plot with the learning rate equal to 0.001 showed oscillations but, overall, the model accuracy increased over epochs. In contrast, the plot with learning rate equal 0.01 didn't show clear improvement, suggesting instability in the ability to learn the model.

This indicates that a smaller learning rate (0.001) led to a more stable and steadily improving model accuracy, while a larger learning rate (0.01) led to instability and overshooting, preventing the learning process.



(a) Learning rate: 0.01



(b) Learning rate: 0.001

Figure 2

Learning Rate	Train Accuracy	Validation Accuracy	Final Test Accuracy
0.01	0.6205	0.6229	0.4556
0.001	0.6564	0.6575	0.6068

Table 2: Final Accuracy Metrics for Logistic Regression

1.2 Question 1.2

1.2.1 Question 1.2 (a)

The statement suggests that a logistic regression model using pixel values isn't as good at capturing complexity as a multi-layer perceptron with ReLU activations. This statement is true because of how these models work. In essence, a logistic regression used for binary classification is very similar to a perceptron with a sigmoid activation function. This means that, as is the case with a single perceptron, the logistic regression can only represent linearly seperable data.

Unlike logistic regression models, which stick to linear separability, the multi-layer perceptron can handle more complicated relationships between the data, making it adaptable to non-linearly separable data. This, in turn, makes it more expressive and enables the model to pick-up on more information and understand intricate data patterns. This model's expressiveness comes directly from it's complexity, given it's hidden layers and additional parameters.

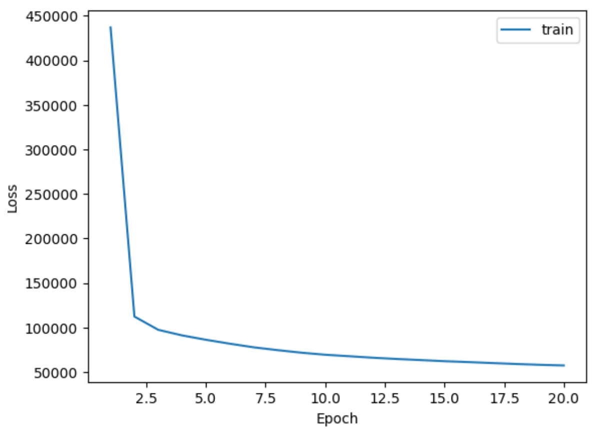
The training process for a logistic regression is simple and follows a convex optimization path, essentially behaving like a basic perceptron. So, in logistic regression, as the functions are strictly convex, they ensure the presence of only a single global minimum.

On the other hand, a multi-layer perceptron with ReLU activations introduces complexity while training, given it's non-convex optimization approach, leading to the possibility of the algorithm converging to different solutions due to multiple local minima.

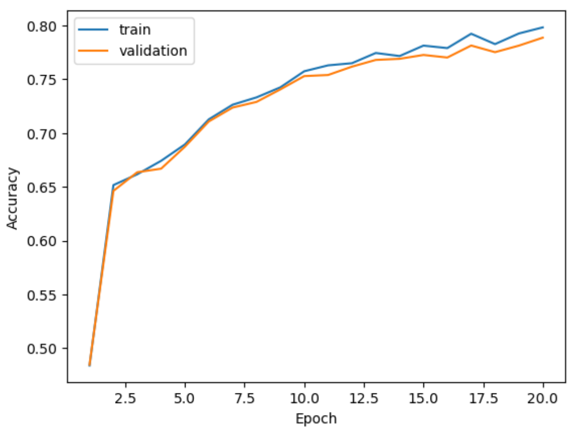
In essence, while logistic regression is easier to train and follows a convex path, a multi-layer perceptron with ReLU activations does better in terms of being more expressive and handling more complex data patterns.

1.2.2 Question 1.2 (b)

The figures bellow, Figure 3a and Figure 3b are the plots for the multi-layer perceptron, illustrating the training and validation accuracies, as well as the training loss:



(a) Train loss



(b) Train and validation accuracies

Figure 3

The final test accuracy achieved by the model is 0.7656 as indicated in the Table 3.

Train Accuracy	Validation Accuracy	Final Test Accuracy
0.7984	0.7888	0.7656

Table 3: Final Accuracy Metrics for the Multi-layer Perceptron

2 Question 2

2.1 Question 2.1

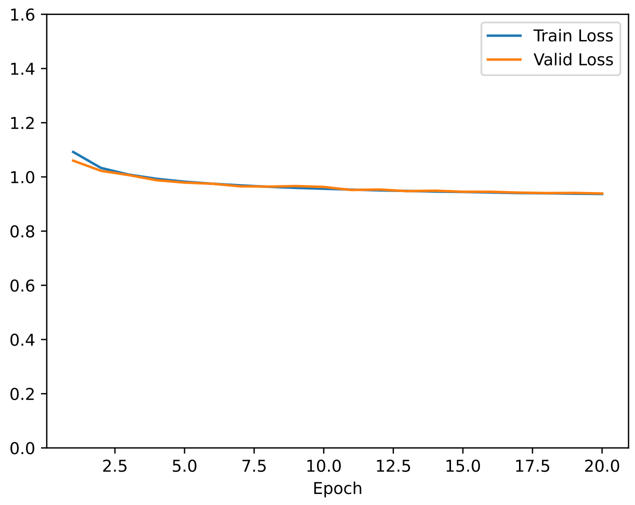
In this question, a linear model with logistic regression was implemented [3] [4]. Separating the data into training dataset, validation dataset and test dataset, several conclusions can be made regarding the effects of the learning rate (parameter study for this question) on the model's results.

The model was trained using the following learning rates: 0.001; 0.01; 0.1 (keeping every other hyperparameter constant).

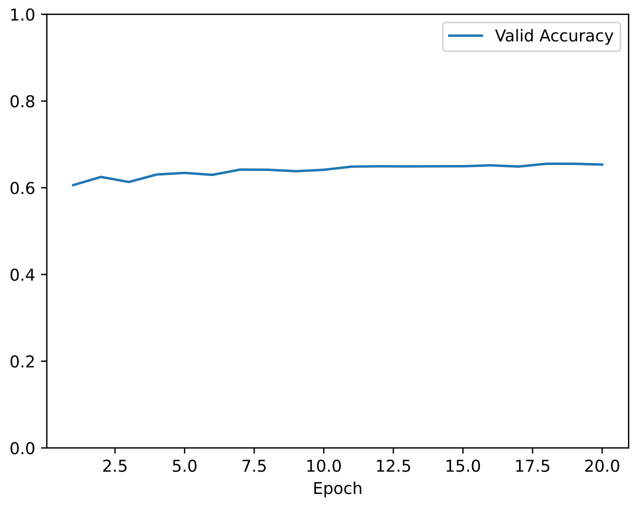
Learning rate	Final test accurary	Final validation accuracy
0.001	0.6503	0.6163
0.01	0.6200	0.6535
0.1	0.5577	0.6224

Table 4: Comparison of test and final validation accuracy values using different learning rates

Analysing the results from table 4 it is possible to conclude that the best validation accuracy value is achieved for the intermediate learning rate (0.01) model. This is the best configuration and it has a final test accuracy of 0.62.



(a) Training and validation datasets loss values



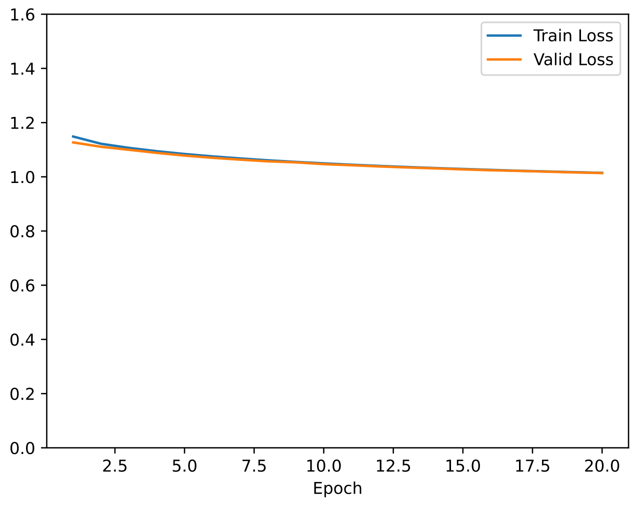
(b) Validation accuracy

Figure 4: Plots for the best configuration (Learning rate = 0.01)

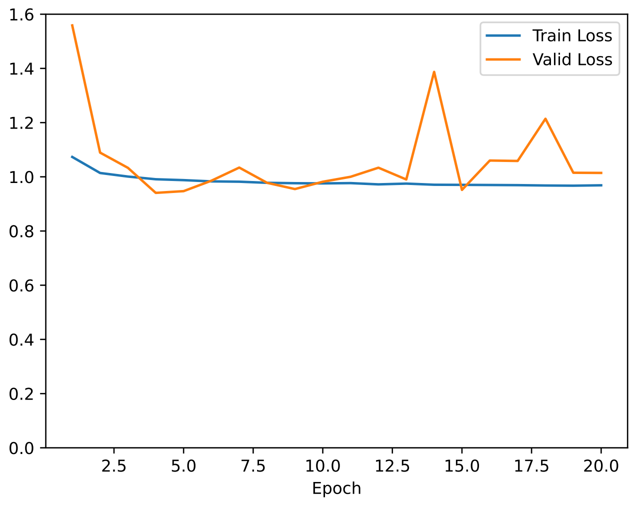
The expected outcome for a model trained with a big learning rate would be to make a lot of progress in the first few epochs of the training, but become more unstable near the end of the training, recurrently overshooting the global minimum. This means that a model trained with a big learning rate should not give us very good result, when trained with a lot of epochs, in relation to smaller learning rates. But, when trained with a very small amount of epochs, this learning rate can be a good choice.

In the other hand, the expected outcome of a model trained with a smaller learning rate would be to make slower progress in the first few epochs. But for a model trained with a lot more epochs, the smaller learning rate model, should end up with better results, as the SGD algorithm can converge to a local minima with more precision.

These expected behaviours were confirmed with the graphs of training and validation loss values for the 3 models. For the bigger learning rate (figure 5b), the loss values are very unstable. For the smaller learning rate (figure 5a), the loss values are very stable but almost constant (not making a lot of progress).



(a) Model trained with small (0.001) learning rate



(b) Model trained with big (0.1) learning rate

Figure 5

20 epochs is not big enough to give a big advantage to the smaller learning rate of 0.001. At the same time, 20 epochs is big enough to make the unstable nature of a big learning rate be noticeable. This justifies that an intermediate learning rate value produces the model with better results in terms of validation accuracy.

2.2 Question 2.2

A feed-foward neural network was implemented in the given skeleton code for this question [2] [3]. For questions (a), (b) and (c), when a hyperparameter is not mentioned, it´s default value was used.

2.2.1 Question 2.2 (a)

In this question, a comparison between training the neural network with a small batch size (16) and bigger batch size (1024) is made.

Analysing the results, for the same number of epochs, the model trained with the bigger batch size takes a lot less time to train than the one with the smaller batch size. This means that each epoch takes less time with a bigger batch size.

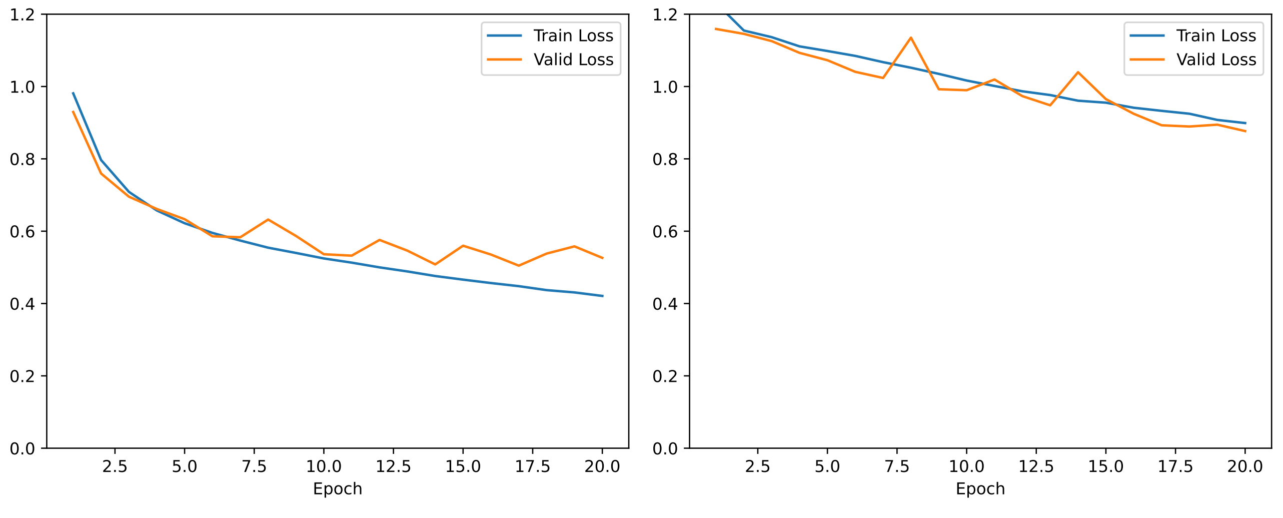
This result was to be expected. For both models, using the Stochastic Gradient Descent algorithm, the model updates the weights one time for each batch, and goes through the entire

Number of Epochs	20
Learning Rate	0.1
Hidden Size	200
Number of (hidden) Layers	2
Dropout (probability)	0.0
Batch Size	16
Activation	ReLU
L2 Regularization	0.0
Optimizer	SGD

Table 5: Default values for feed-foward neural network

Batch Size	Final test accurary	Training time (s)
16	0.7429	151.21
1024	0.7202	37.41

Table 6: Comparison between neural-network trained with different batch sizes



(a) Batch size: 16 **(b)** Batch size: 1024

Figure 6

train dataset in each epoch. This means that if a smaller batch size is being used, the weights need to be updated more times for each epoch and, in consequence, this takes more time.

The effect of doing forward propagation and back-propagation on a bigger batch should not increase the time to run too much, as these processes can be parallelized trough matrix multiplications.

In terms of the final test accuracy, and the train and validation loss throughout the training, the smaller batch size achieves the best results. This was also to be expected, as training with smaller batches induces some noise on the weight updates, which can serve as regularization. This can improve the model's generalization ability, hence improving the test and validation accuracy values.

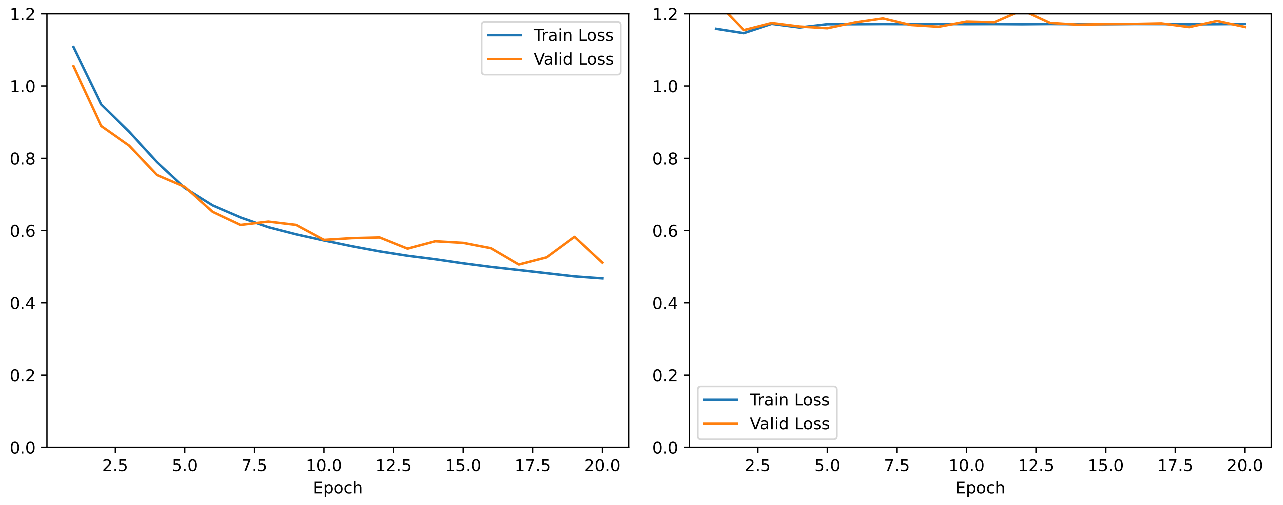
2.2.2 Question 2.2 (b)

For this question the feed-forward neural network was trained with various learning rates. These values should affect how big the change in the weights and biases is, each time they are updated.

Learning rate	Final test accurary	Final validation accuracy
1.0	0.4726	0.4721
0.1	0.7429	0.8195
0.01	0.7486	0.8200
0.001	0.7391	0.7230

Table 7: Test and validation accuracy values - Comparison for different learning rates

Analysing table 7 it is possible to conclude that the model with better performance in terms of final validation accuracy is the one with 0.01 learning rate.



(a) Learning rate: 0.01 - Best performing model **(b)** Learning rate: 1.0 - Worst performing model

Figure 7

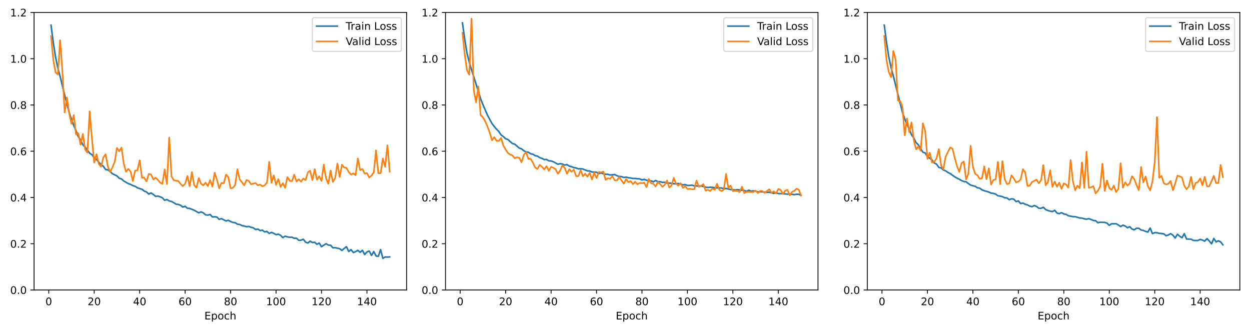
As explained in Question 2.1, a very small learning rate will only be a good approach when using a lot of epochs (mainly in the end of the training process). Small values for this hyperparameter slow down the progress from one epoch to another and end up not having very good results for only 20 epochs (this case).

Big learning rate values tend to amplify the noise from each batch in SGD. This means that throughout one epoch, each batch can overcompensate the weights values in different directions. In figure 7b it is possible to see that in the end of each epoch, not much progress is made, due to these sequential over-compensations canceling each other.

This means that an intermediate learning rate should work best in this case, which ended up being confirmed with the results with the 0.01 learning rate (figure 7a). This hyperparameter value was not only the best one in terms of final validation accuracy, but also final test accuracy, proving the good generalization ability of a model trained with a well chosen learning rate.

2.2.3 Question 2.2 (c)

When using many epochs to train a model, it can become too specific for predictions on the training data set. The consequence of this is that the model will not be able to provide good predictions for unseen data. This is called overfitting.



(a) No regularization used **(b)** Dropout regularization - 0.2 **(c)** L2 Regularization - 0.0001

Figure 8: Train and validation loss values across 150 epochs

One clear indicator that overfitting is happening consists on a divergence between the train loss and validation loss graph throughout several epochs. This means that the train loss is becoming very low but the validation loss is constant or increasing (model became very good at predicting the test set, but not good in unseen data - model is to specific and cannot generalize well).

The graph of image 8a, represents the loss values for a model trained with no regularization method and with a lot of epochs. It is clear that the two graphs diverge from approximately epoch number 40. This means that overfitting occurred.

Regularization methods help to make the model less specific to the train data set and better at generalizing and predicting results on unseen data [1].

Two regularization methods were tried: Dropout regularization with a probability of dropout of 0.2 - this was implemented in every layer except the output one; L2 regularization with value 0.0001.

L2 regularization is implemented on the loss function. This type of regularization increases the loss value when the L2 norm of the weights vector is larger. The consequence is that the model tends to states where the weight vectors have lower norms, instead of a state where the model is too good at predicting the train data set. This usually improves generalization.

Dropout regularization consists on setting to zero some amount of perceptron outputs in each layer, in each iteration of model training. The probability of each perceptron's output being set to zero is a parameter of the dropout regularization (in this case, 0.2). This method makes the perceptrons that were not a target of the dropout better at predicting the output on their own (without being dependent on other perceptrons of that layer). This makes the model more robust and better at generalizing.

Analysing the results from figure 8, it is possible to conclude that the dropout regularization prevented overfitting, as there is no divergence between the 2 loss graphs.

In the other hand, the L2 regularization was not as effective. It is possible to see that the L2 regularization had some effect, as the validation loss graph does not tend upwards, which is not the case when no regularization is used (figure 8a). But this effect was not sufficient as overfitting still happened.

The hyperparameter chosen for the L2 regularization (0.0001) was very small, which can make the effect of this regularization method almost unnoticeable. This means that if this hyperparameter was increased, most likely it would have a better effect at preventing overfitting and improving generalization.

Used Regularization	**Final test accurary**	**Final validation accuracy**
None	0.7561	0.8651
Dropout - 0.2	0.7845	0.8596
L2 - 0.0001	0.7656	0.8536

Table 8: Test and validation accuracy values - Comparison for different regularization methods

As already predicted just by analysing the loss graphs, the **best model at generalizing** and predicting unseen data is the one with dropout regularization. This can be confirmed with the final test accuracy values in table 8.

The model that had the **best final validation accuracy value** is the one trained without any type of regularization (figure 8a) and the **worst one in terms of validation accuracy** is the one with L2 regularization (figure 8c). This was not the expected result. The expected result would have been very similar to the test accuracy results (best model: Dropout regularization; worst model: No regularization). These results can be explained to some extent by small variances that happen in each epoch to the validation accuracy values, alongside the fact that the 3 values are vary close. This can make it so the last epoch validation accuracy values are not the best ones for each model.

3 Question 3

3.1 Question 3.1

3.1.1 Question 3.1 (a)

Formulating the problem assuming that D=2, A=1 and B=1. This implies that the function $f(x)$ can be defined as follows:

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

Evaluating this function for a given set of points:

$$X = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_{11} & x_{21} & x_{31} & x_{41} \\ x_{12} & x_{22} & x_{32} & x_{42} \end{bmatrix}$$

Results in the following calculations:

$$\sum_{i=1}^{2} x_{1i} = 0 + 0 = 0 \notin [A, B] \rightarrow f(x_1) = -1$$
$$\sum_{i=1}^{2} x_{2i} = 0 + 1 = 1 \in [A, B] \rightarrow f(x_2) = 1$$
$$\sum_{i=1}^{2} x_{3i} = 1 + 0 = 1 \in [A, B] \rightarrow f(x_3) = 1$$
$$\sum_{i=1}^{2} x_{4i} = 1 + 1 = 2 \notin [A, B] \rightarrow f(x_4) = -1$$

The analysis indicates that the data is not linearly separable, as illustrated in the Figure 9. This implies that a single perceptron cannot effectively solve the problem, as it relies on linear decision boundaries. The non-separability of the data suggests the need for a more complex model or a different approach to accurately classify the given points.

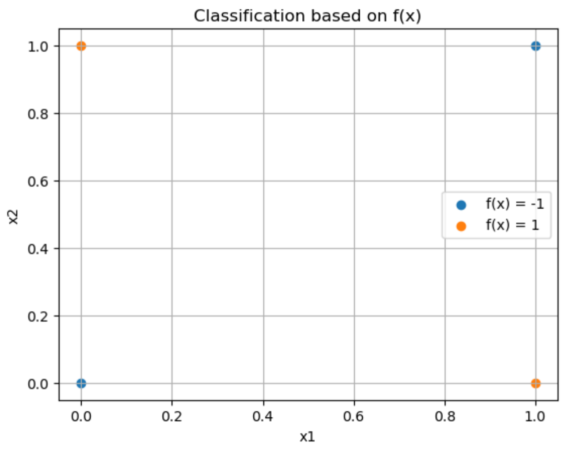


Figure 9: Data pattern for D=2, A=1 and B=1

3.1.2 Question 3.1 (b)

A multilayer perceptron can solve non linearly separable problems if given enough hidden units. This means that a priori, a multilayer perceptron should be able to solve this problem.

First of all, we need to deconstruct the problem: In order to know if $\sum_{i=1}^{D} x_i \in [A, B]$, we need to check if $\sum_{i=1}^{D} x_i \geq A$ **and** $\sum_{i=1}^{D} x_i \leq B$.

If we can get as output from one of the units from the hidden layer the following condition:

$$\begin{cases} 1 \quad if \sum_{i=1}^{D} x_i \geq A \\ -1 \quad if \sum_{i=1}^{D} x_i < A \end{cases} \tag{2}$$

And if the other hidden layer unit can represent the following condition:

$$\begin{cases} 1 \quad if \sum_{i=1}^{D} x_i \leq B \\ -1 \quad if \sum_{i=1}^{D} x_i > B \end{cases} \tag{3}$$

Then, the output layer is just an **AND condition**, which is known to be a linearly separable problem and can be solved by just one unit (the output unit).

We can now define the output layer's unit:

$$h_3 = z_1 * 1 + z_2 * 1 - 1 \quad ; \quad output = z_3 = g(h_3) \tag{4}$$

With z_1, z_2 being the output of the units in the hidden layer, and:

$$g(x) = sign(x) \tag{5}$$

This means that the **weights and biases of the output unit** are: $w_{23} = w_{13} = 1$; $b_3 = -1$.

With the output unit defined, we just need to define the 2 units in the hidden layer, with robustness to infinitesimal perturbation of the inputs.

Hidden layer unit 1: check condition of equation 3:

$$\sum_{i=1}^{D} x_i \leq B \Leftrightarrow -\sum_{i=1}^{D} x_i + B \geq 0 \tag{6}$$

If $-\sum_{i=1}^{D} x_i + B \geq 0$, then $g(-\sum_{i=1}^{D} x_i + B) = +1$, and if $-\sum_{i=1}^{D} x_i + B < 0$, then $g(-\sum_{i=1}^{D} x_i + B) = -1$. This means that the unit could be defined with:

$$h_1 = \sum_{i=1}^{D} -1 * x_i + B \quad ; \quad z_1 = g(h_1) \tag{7}$$

The problem is that this is not robust to infinitesimal perturbation of the inputs. To demonstrate it the following example is shown, where the sum of the inputs is approximately equal to B, which means it (approximately) falls into the [A,B] interval, and the output of the unit should be +1:

$$B = 3; \quad \sum_{i=1}^{D} x_i = 3.001 \longrightarrow z_1 = g(-0.001) = -1$$

As the output is -1, this is not robust.

In order to make it robust, some changes can be made to the weights and biases of the unit, in order to make the output equal to +1, when the sum of the input is infinitesimally bigger than B:

$$h_1 = \sum_{i=1}^{D} -2 * x_i + 2B + 1 \quad ; \quad z_1 = g(h_1) \tag{8}$$

Proof that it is robust:

B value	$\sum_{i=1}^{D} x_i$	z_1
3	3	$z_1 = g(1) = +1$
3	3.001	$z_1 = g(0.999) = +1$
3	4	$z_1 = g(-1) = -1$
3	2	$z_1 = g(3) = +1$

This means that the **weights and bias for this hidden unit** should be $w_{01} = -2$ and $b_1 = 2 * B + 1$.

For the other hidden layer unit, a similar approach can be made:

$$\sum_{i=1}^{D} x_i \geq A \Leftrightarrow \sum_{i=1}^{D} x_i - A \geq 0 \longrightarrow g(\sum_{i=1}^{D} x_i - A) = +1$$

$$h_2 = \sum_{i=1}^{D} 1 * x_i - A \quad ; \quad z_2 = g(h_2)$$

This is once again not robust to infinitesimal perturbation of the inputs, so the weights and biases need to be changed:

$$h_2 = \sum_{i=1}^{D} 2 * x_i - 2A + 1 \quad ; \quad z_2 = g(h_2) \tag{9}$$

Proof that this is a robust unit, with an example:

A value	$\sum_{i=1}^{D} x_i$	z_2
2	2	$z_2 = g(1) = +1$
2	1.999	$z_2 = g(0.998) = +1$
2	3	$z_2 = g(3) = +1$
2	1	$z_2 = g(-1) = -1$

This means that the **weights and bias for this hidden unit** should be $w_{02} = 2$ and $b_2 = -2A + 1$.

All the weights and biases of the multilayer perceptron are now defined and it is proven that it is robust to infinitesimal perturbation of the inputs.

3.1.3 Question 3.1 (c)

A similar approach to the previous question is going to be used. First of all let's deconstruct the problem into one that is useful in this situation (hidden units have ReLU activation function): in this case, if the hidden units are constructed in a way in which their output is null if $\sum_{i=1}^{D} x_i \geq A$ **and** $\sum_{i=1}^{D} x_i \leq B$, respectively; and their output is positive otherwise. Then the output function should simply implement an AND logic.

Let's, once again, start by defining the output unit (knowing that the output z_1 and z_2 of the hidden units is equal to zero if they meet their condition successfully; and bigger than zero, otherwise).

One possibility for the output unit can be:

$$h_3 = z_1 * -1 + z_2 * -1 \quad ; \quad output = z_3 = g(h_3) \tag{10}$$

With z_1, z_2 being the output of the units in the hidden layer, and:

$$g(x) = sign(x) \tag{11}$$

With this implementation of the output unit, if $\sum_{i=1}^{D} x_i \geq A$ and $\sum_{i=1}^{D} x_i \leq B$, then:

$$z_1 = z_2 = 0 \longrightarrow h_3 = 0 \longrightarrow g(z_3) = +1$$

If any of the 2 condition are not meet, then $z_1 > 0$ and/or $z_2 > 0$, which will make:

$$h_3 < 0 \longrightarrow g(z_3) = -1$$

This means that the chosen **weights and bias for the output unit** are: $w_{13} = w_{23} = -1$; $b_3 = 0$.

For the hidden layer units, a lot can be kept the same from Question 3.1.2. We just need to use the symmetrical of h_1 and h_2, because we want their value to be negative when their respective conditions are meet (instead of positive, which was the goal in the previous question). This is because, when their respective conditions are meet, we need $z_1 = z_2 = 0$, using the ReLU activation function.

$$ReLU(s) = max(0, s)$$

This means that hidden unit 1 should represent the following condition:

$$\begin{cases} 0 & if \sum_{i=1}^{D} x_i \leq B \\ k & if \sum_{i=1}^{D} x_i > B \quad , with\ k > 0 \end{cases} \tag{12}$$

And should have the following expressions:

$$h_1 = \sum_{i=1}^{D} 2 * x_i - 2B - 1 \quad ; \quad z_1 = ReLU(h_1) \tag{13}$$

Hidden unit 2 should represent the following condition:

$$\begin{cases} 0 & if \sum_{i=1}^{D} x_i \geq A \\ k & if \sum_{i=1}^{D} x_i < A \quad , with\ k > 0 \end{cases} \tag{14}$$

And has the following expressions:

$$h_2 = \sum_{i=1}^{D} -2 * x_i + 2A - 1 \quad ; \quad z_2 = ReLU(h_2) \tag{15}$$

This means that the chosen **weights and bias for the hidden units** are: $w_{01} = 2$; $b_1 = -2B - 1$; and $w_{02} = -2$; $b_2 = 2A - 1$.

These 2 hidden units are equally as robust to infinitesimal perturbation of the inputs as they were in Question 3.1.2. In turn, the multilayer perceptron is also robust to these perturbations.

References

[1] Andrew Fogarty. Regularization. https://seekinginference.com/deep$_learning/regularization.html$.

[2] Mário Figueiredo André Martins, Francisco Melo. *Deep Learning Course (Lecture Slides and Pratical Lessons)*. 2023-2024.

[3] Patrick Loeber. Pytorch - feed forward neural network. https://www.python-engineer.com/courses/pytorchbeginner/13-feedforward-neural-network/ Accessed on December, 15, 2023.

[4] Pytorch. Pytorch - linear. https://pytorch.org/docs/stable/generated/torch.nn.Linear.html Accessed on December, 15, 2023.