Deep Learning 2023
Homework 1 – Group 48

Miguel Vale (99113) Pedro Lobo (99115)

Contribution

We started by dividing the coding part of this homework by the two of us. Questions 1.1. a), 1.1. b) and 2.2 were done by Miguel while questions 1.2. b) and 2.1 were done by Pedro. Although we distributed this questions between the both of us, we double-checked each other's implementation.

Questions 1.2 a) was done by Pedro and later reviewed by Miguel while question 3.1 a) was done by Miguel and later reviewed by Pedro. The remaining questions, namely questions 3.1 b) and 3.1 c), were done by the two of us.

The report was written by the two of us, with each of us writing the answers to the questions that we implemented and then reviewed by the other.

Question 1

Medical image classification with linear classifiers and neural networks

1. a) *Implement the* `update_weights` *method of the* `Perceptron` *class in* `hw1-q1.py`*. Then train 20 epochs of the perceptron on the training set and report its performance on the training, validation and test sets. Plot the train and validation accuracies as a function of the epoch number.*

   The resulting test accuracy for the perceptron is 0.3422. The train and validation accuracies in each epoch are depicted in Table 1. The train and validation accuracies are depicted in Figure 1.

Epoch	Train Acc	Validation Acc
1	0.5118	0.5103
2	0.4635	0.4573
3	0.6308	0.6329
4	0.3407	0.3408
5	0.4342	0.4285
6	0.5988	0.6018
7	0.5679	0.5650
8	0.6379	0.6353
9	0.5988	0.5992
10	0.5399	0.5367
11	0.5582	0.5596
12	0.4041	0.4035
13	0.5813	0.5804
14	0.6054	0.6074
15	0.5916	0.5884
16	0.5556	0.5509
17	0.5752	0.5798
18	0.5937	0.5961
19	0.4873	0.4831
20	0.4654	0.4610

Table 1: Perceptron train and validation accuracies in each epoch.

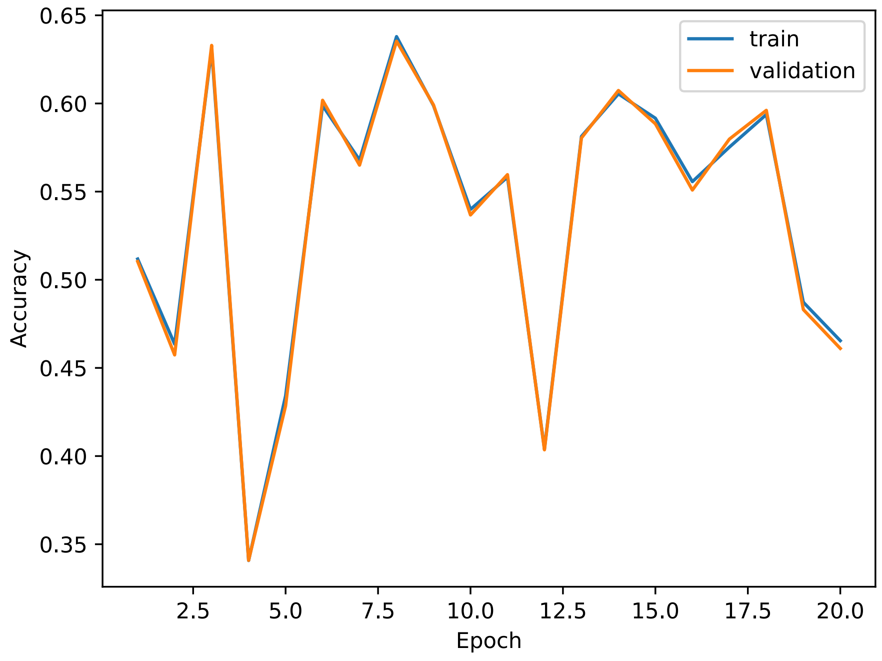

Figure 1: Perceptron train and validation accuracies as a function of the epoch number.

b) *Repeat the same exercise using logistic regression instead (without regularization), using stochastic gradient descent as your training algorithm. Report the final test accuracies and compare, based on the plots of the train and validation accuracies, the models obtained using two different learning rates of $\eta = 0.01$ and $\eta = 0.001$.*

For a learning rate of $\eta = 0.01$, the resulting test accuracy is 0.5784, while for a learning rate of $\eta = 0.001$, the obtained test accuracy is 0.5936. The train and validation accuracies for $\eta = 0.01$ and $\eta = 0.001$ are depicted in Figures 2 and 3, respectively.

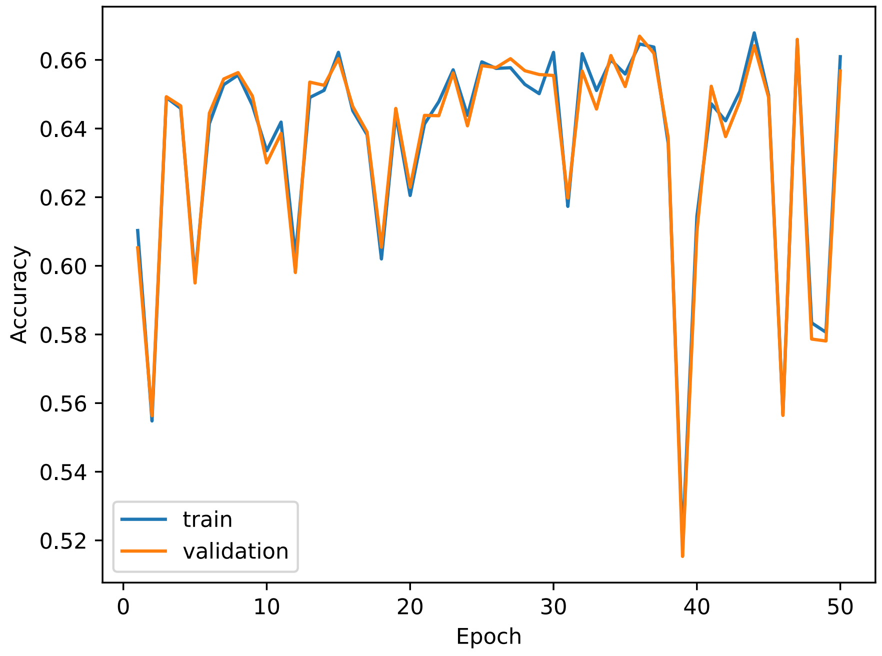

Figure 2: Logistic regression train and validation accuracies as a function of the epoch number for $\eta = 0.01$.

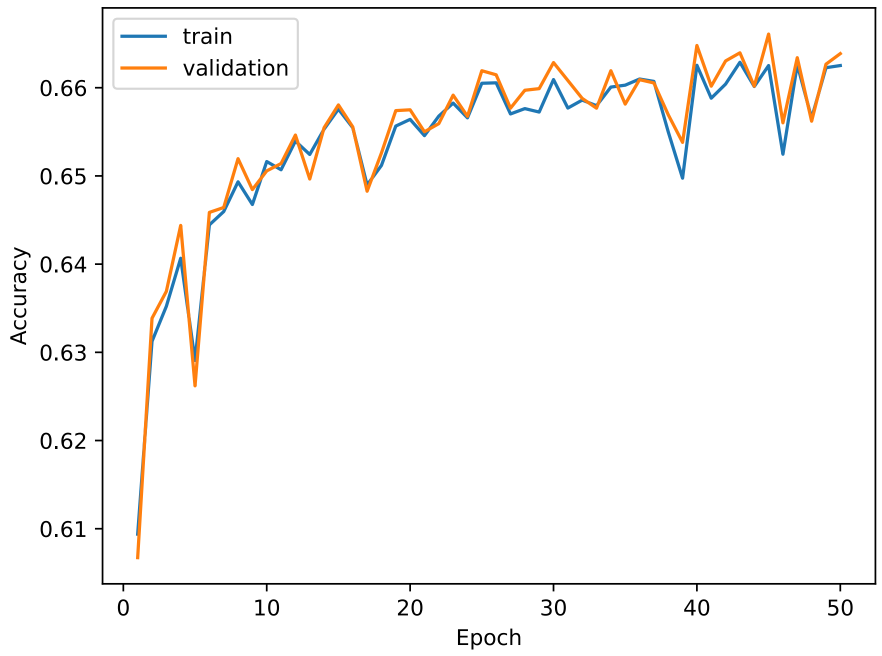

Figure 3: Logistic regression training and validation accuracies as a function of the epoch number for $\eta = 0.001$.

2. a) *Comment the following claim: "A logistic regression model using pixel values as features is not as expressive as a multi-layer perceptron using `ReLU` activations. However, training a logistic regression model is easier because it is a convex optimization problem." Is this true of false? Justify.*

The claim is correct. The following two paragraphs address the expressiveness of the models and their optimization problems, respectively.

A multi-layer perceptron using ReLU activations is more expressive than a logistic regression model as its hidden layers have a non-linear activation which allows the model to learn more complex representations of the input. This more complex representations can turn the input into a linearly separable dataset, that can be learned by the output layer. A logistic regression model can only learn linearly separable datasets, as its output layer is a linear function of the input.

Despite being more expressive, a multi-layer perceptron using ReLU activations is harder to train than a logistic regression model. This is due to the fact that the optimization problem of the former is non-convex, while the optimization problem of the latter is convex. This means that the optimization problem of a logistic regression model has a single global minimum, while the optimization problem of a multi-layer perceptron using ReLU activations, in general, can have multiple local minima. While training the multi-layer perceptron with a gradient descent algorithm, the model can get *stuck* on a local minimum. This makes it harder to find the optimal solution for the latter model.

b) *Without using any neural network toolkit, implement a multi-layer perceptron with a single hidden layer to solve this problem, including the gradient backpropagation algorithm which is needed to train the model. Use 200 hidden units, a ReLU activation function for the hidden layers, and a multinomial logistic loss (also called cross-entropy) in the output layer (even though we are dealing with a binary classification this will allow you to use the same code for a multi-class problem). Don't forget to include bias terms in your hidden units. Train the model for 20 epochs with stochastic gradient descent with a learning rate of 0.001. Initialize biases with zero vectors and values in weight matrices with* $w_{ij} \sim \mathcal{N}(\mu, \sigma^2)$ *with* $\mu = 0.1$ *and* $\sigma^2 = 0.1^2$. *Report final test accuracy and include the plots of the train loss and train and validation accuracies as a function of the epoch number.*

For a learning rate of $\eta = 0.001$, the resulting test accuracy for the MLP is 0.6106. The correspondant train and validation accuracies are depicted in Figure 4, while the train loss is depicted in Figure 5.

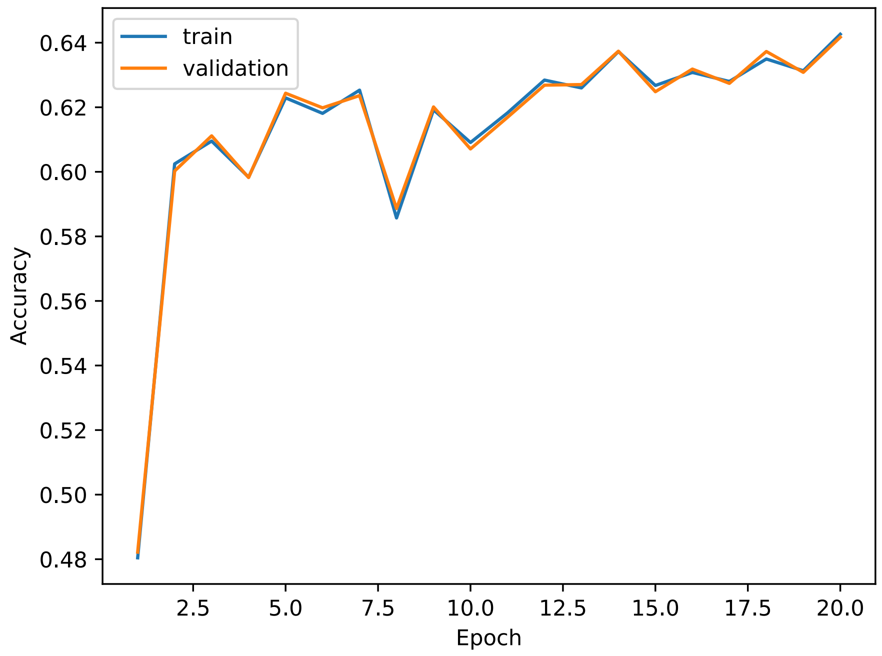

Figure 4: MLP train and validation accuracies as a function of the epoch number for $\eta = 0.001$.

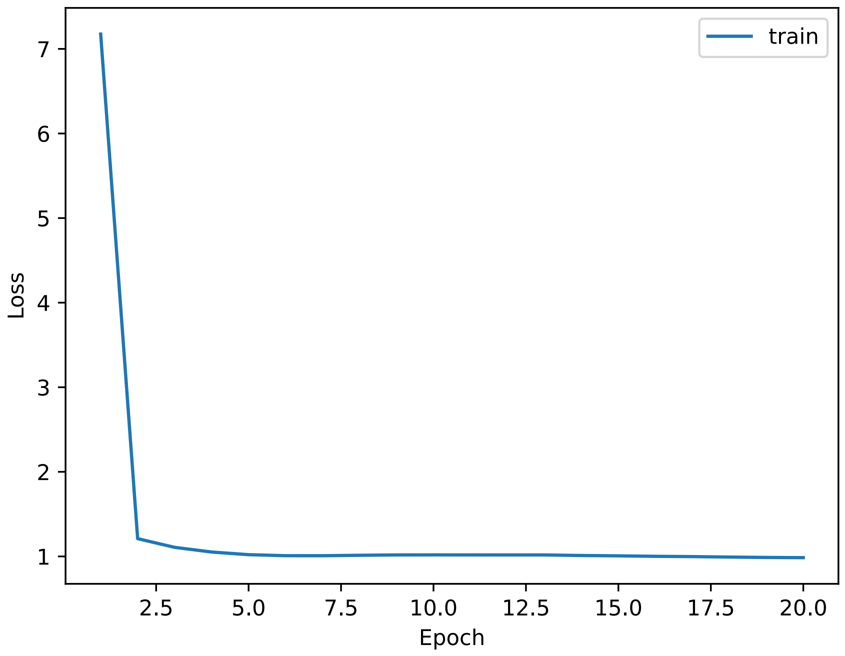

Figure 5: MLP train loss as a function of the epoch number for $\eta = 0.001$.

Question 2

Medical image classification with an autodiff toolkit

1. *Implement a linear model with logistic regression, using stochastic gradient descent as your training algorithm (use a batch size of 16). Train your model for 20 epochs and tune the learning rate on your validation data, using the following values:* $\{0.001, 0.01, 0.1\}$. *Report the best configuration (in terms of final validation accuracy) and plot two things: the training loss and the validation accuracy, both as a function of the epoch number. Report the final accuracy on the test set.*

   The best configuration, in term of final validation accuracy, is $\eta = 0.01$, as shown in Table 2.

Learning Rate	**Validation Accuracy**
0.001	0.6163
0.01	0.6535
0.1	0.6224

Table 2: Final logistic regression validation accuracy for each learning rate.

For the configuration with a learning rate of $\eta = 0.01$, the final accuracy on the test set is 0.6200.

The traning loss and validation accuracy plots for a learning rate of $\eta = 0.01$ are depicted in Figures 6 and 7, respectively.

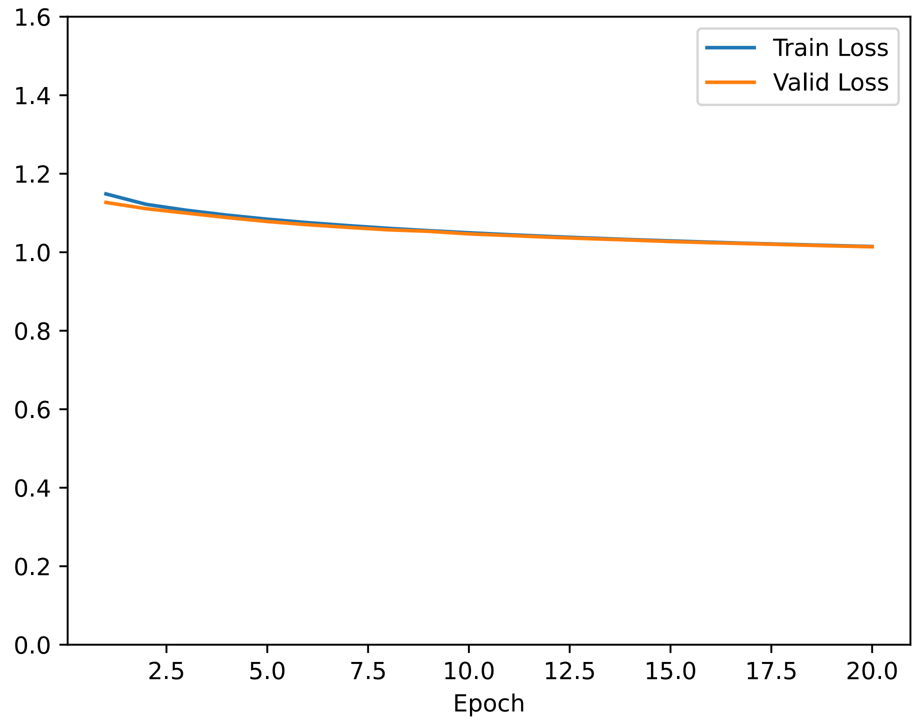

Figure 6: Logistic regression training loss for a learning rate of $\eta = 0.01$.

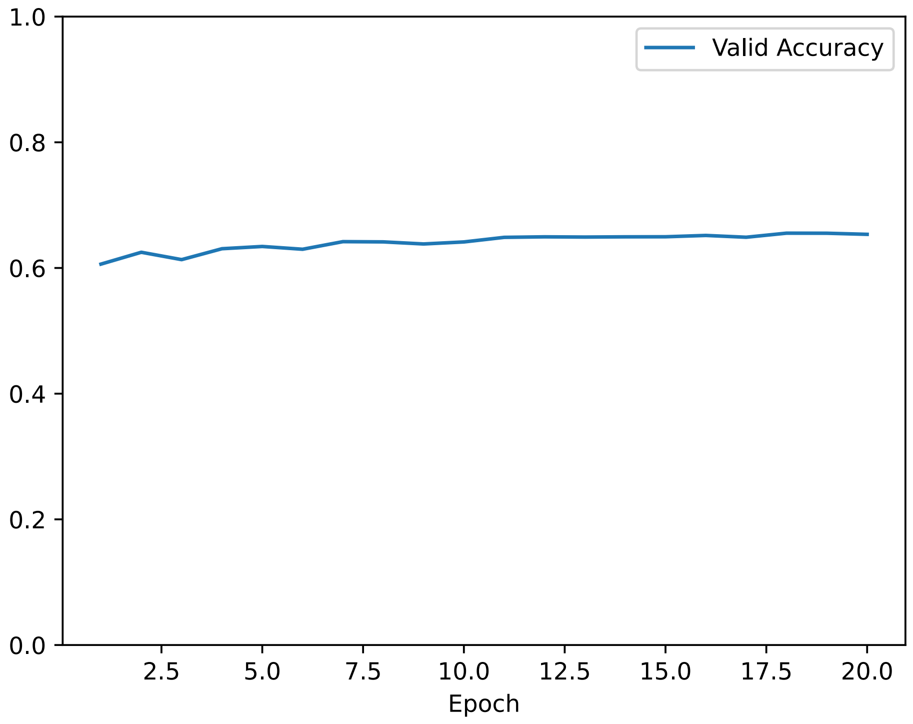

Figure 7: Logistic regression validation accuracy for a learning rate of $\eta = 0.01$.

2. *Implement a feed-forward neural network using dropout regularization. Make sure to include all the hyperparameters and training/model design choices shown in Table 3. Use the values presented in the table as default.*

Hyperparameter	**Value**
Number of Epochs	20
Learning Rate	0.1
Hidden Size	200
Number of Layers	2
Dropout	0.0
Batch Size	16
Activation	ReLU
L2 Regularization	0.0
Optimizer	SGD

Table 3: Default hyperparameters.

a) *Compare the performance of your model with batch sizes 16 and 1024 with the remaining hyperparameters at their default value. Plot the train and validation losses for both, report the best test accuracy and comment on the differences in both performance and time of execution.*

The configuration with a batch size of 16 has the best final test accuracy, as shown in Table 4.

Batch size	Test accuracy
16	0.7713
1024	0.7202

Table 4: Final MLP test accuracy for each batch size.

The configuration with a batch size of 1024 takes less time to train, as shown in Table 5.

Batch size	Time (seconds)
16	242
1024	67

Table 5: Training time of the MLP for each batch size.

The train and validation losses for a batch size of 16 and a batch size of 1024 are depicted in Figures 8 and 9, respectively.

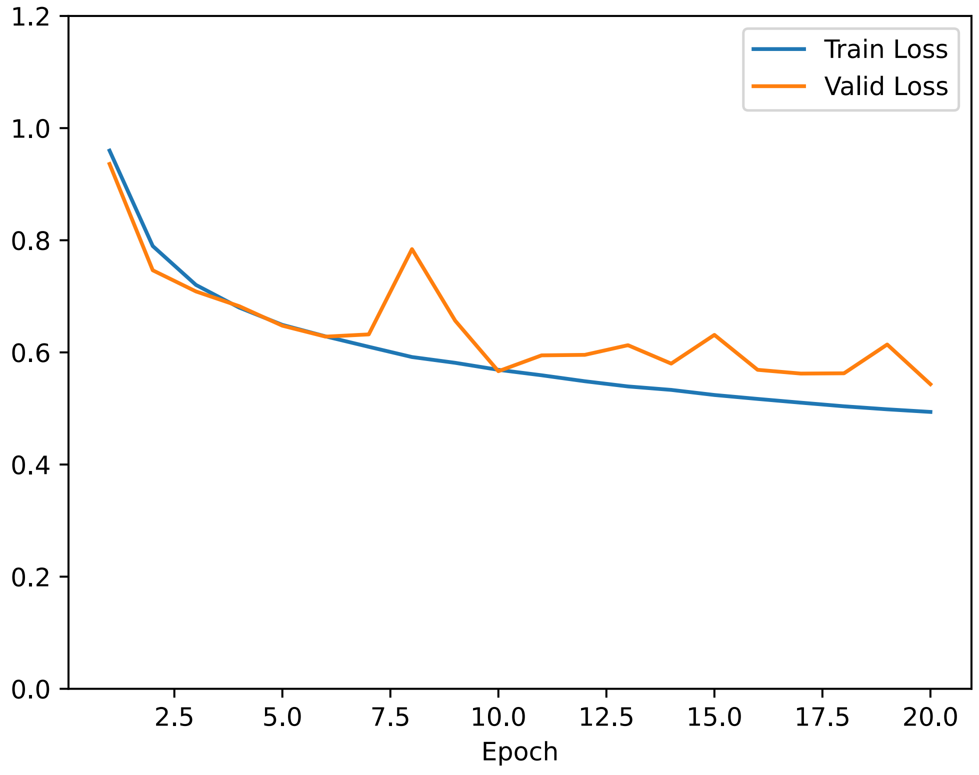

Figure 8: MLP train and validation losses for a batch size of 16.

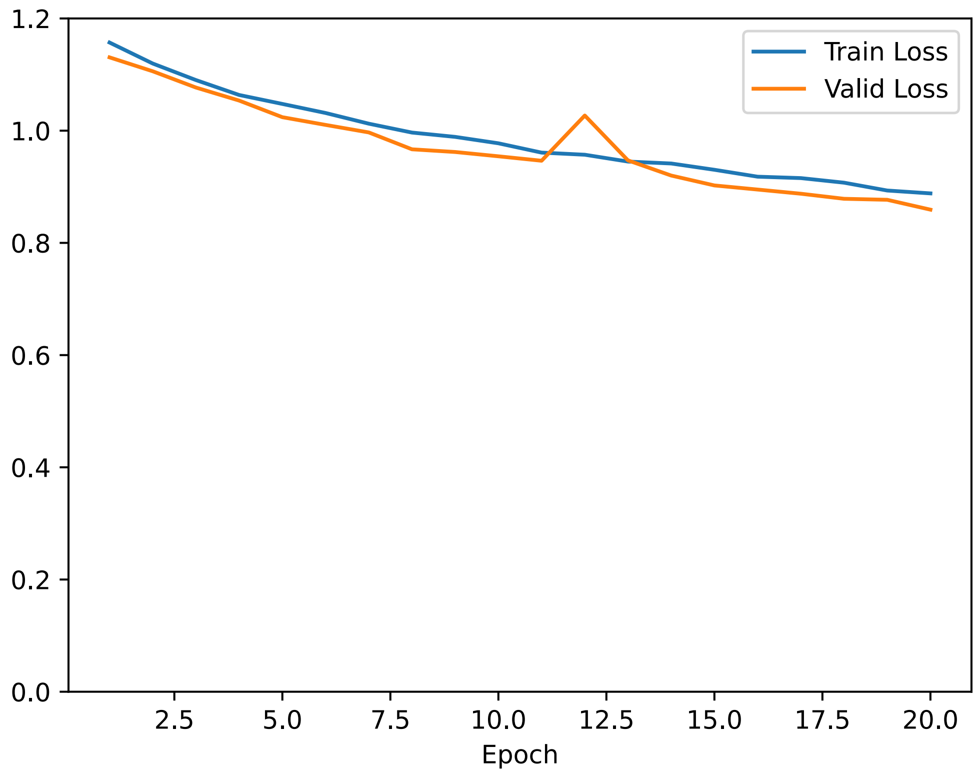

Figure 9: MLP train and validation losses for a batch size of 1024.

The model with a batch size of 16 takes longer to train. This is due to the fact that the gradient computed for each batch is more *noisy* (it is less aligned with the true gradient) than the gradient computed for a batch size of 1024. This means that the stochastic gradient descent algorithm takes longer to converge as it takes more steps to reach the optimal solution. A larger batch size leads to a more accurate estimate of the gradient, which leads to a faster convergence of the stochastic gradient descent algorithm and a faster training time.
The plot of the train and validation losses for a batch size of 1024 shows a smoother training process. This is also indicative of a more accurate gradient computation for each batch.

The model with a batch size of 16 has a higher test accuracy, relative to the model with a batch size of 1024. This is due to the fact that a small batch size can restrict the training of the model, as it introduces more variety into the training process. This can lead to a better generalization of the model, as it is more robust to unseen data, avoiding overfitting. Models less prone to overfitting show better performance on the test set.

b) *Train the model with learning rates: 1, 0.1, 0.01 and 0.001 with the remaining* *hyperparameters at their default value. Plot the train and validation losses for the best and worst configurations in terms of validation accuracy, report the best test accuracy and comment on the differences in performance.*

The best and worst configuration, with regard to the validation accuracy, are $\eta = 1$ and $\eta = 0.1$, respectively, as shown in Table 6.

Learning Rate	**Validation Accuracy**
1	0.4721
0.1	0.8094
0.01	0.8004
0.001	0.6685

Table 6: Validation accuracy of the MLP for each learning rate.

The best configuration, regarding test accuracy, is $\eta = 0.1$, as shown in Table 7.

Learning Rate	**Test Accuracy**
1	0.4726
0.1	0.7713
0.01	0.7543
0.001	0.6900

Table 7: Test accuracy of the MLP for each learning rate.

The train and validation losses for the configurations with $\eta = 1$ and $\eta = 0.1$ are depicted in Figures 10 and 11, respectively.

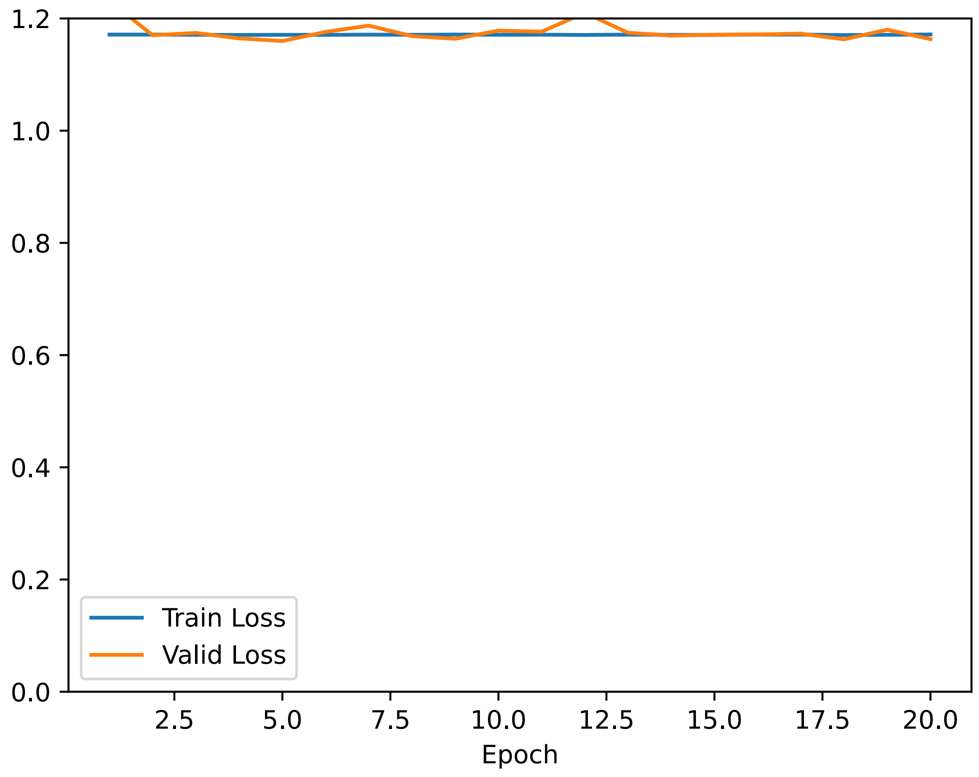

Figure 10: MLP train and validation losses for a learning rate of $\eta = 1.0$.

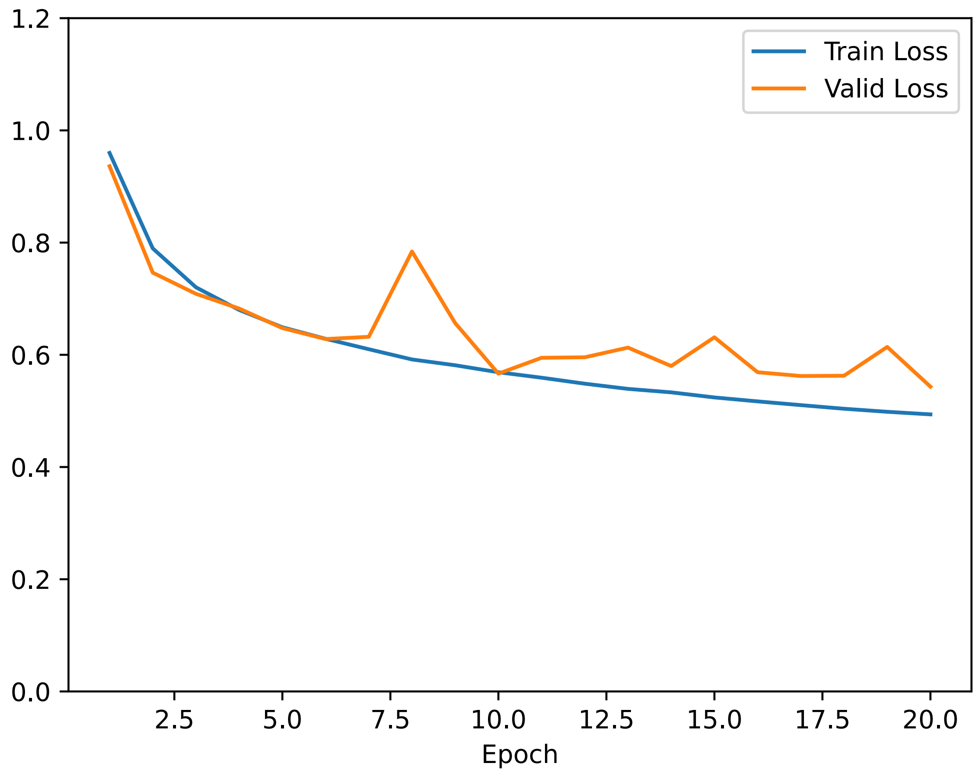

Figure 11: MLP train and validation losses for a learning rate of $\eta = 0.1$.

The models with a very large or very small learning rate exhibit worst test accuracies.

A small learning rate can introduce problems into the training process. Some problems include a slow convergence of the stochastic gradient descent algorithm and a higher chance of getting stuck in a local minimum. As the learning rate is small, the model takes *little steps* taking a lot of training epochs to reach the optimal solution.

On the other hand, a large learning rate can lead to an overshooting of the optimal solution, as the model takes *big steps* in the direction of the gradient. This can lead to the model *bouncing* around the optimal solution, but never reaching it.

Models that have a learning rate that is nor too small nor too large mitigate some of this problems, displaying a better test accuracy.

c) *Using a batch size of 256 run the default model for 150 epochs. Is there overfitting? Train two similar models with the following changes: one with the L2 regularization parameter set to 0.0001 and the other with a dropout probability of 0.2. Plot the train and validation losses for the best and worst configuration in terms of validation accuracy, report the best test accuracy and comment on the differences of both techniques.*

The worst and best configuration with regard to the validation accuracy are the default model (with a batch size of 256) and the model with a dropout probability = 0.001, respectively, as shown in Table 8.

Model	**Validation Accuracy**
Default with batch size = 256	0.8340
L2 regularization parameter = 0.0001	0.8269
Dropout probability = 0.2	0.8445

Table 8: Final MLP validation accuracy for one of the models.

The best configuration, regarding test accuracy, is the configuration with dropout, as shown in Table 9.

Model	**Test Accuracy**
Default with batch size = 256	0.7713
L2 regularization parameter = 0.0001	0.7788
Dropout probability = 0.2	0.7996

Table 9: Final MLP test accuracy for one of the models.

The train and validation losses for the configuration with a batch size of 256 and the configuration with a dropout probability of 0.2 are depicted in Figures 12 and 13, respectively.

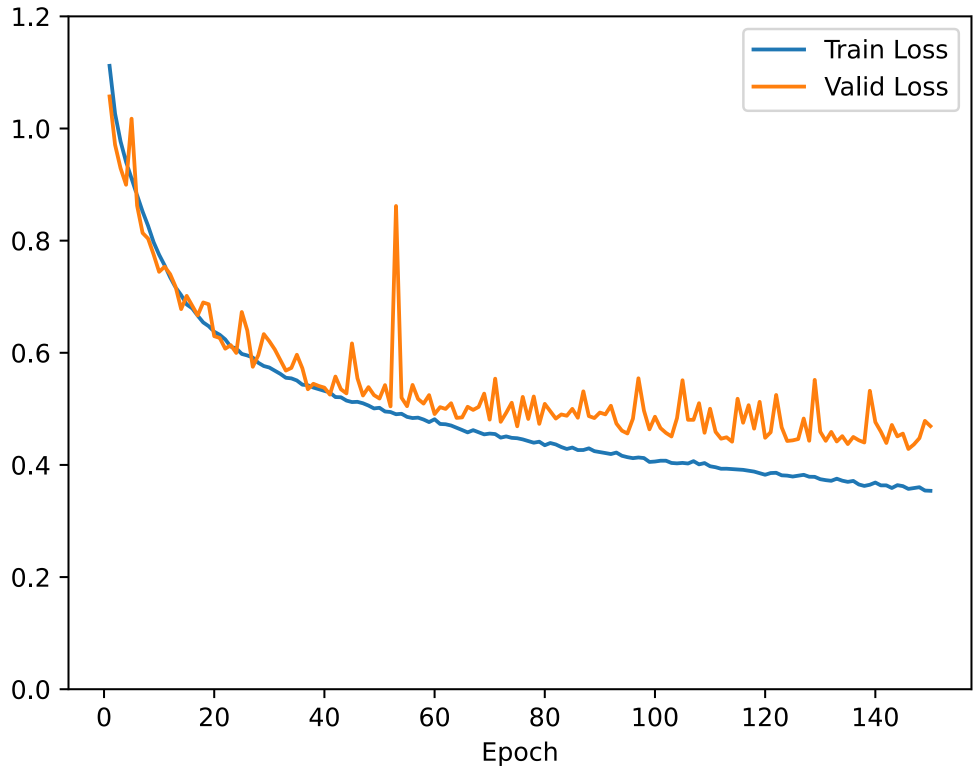

Figure 12: MLP train and validation losses for the default model, with a batch size of 256.

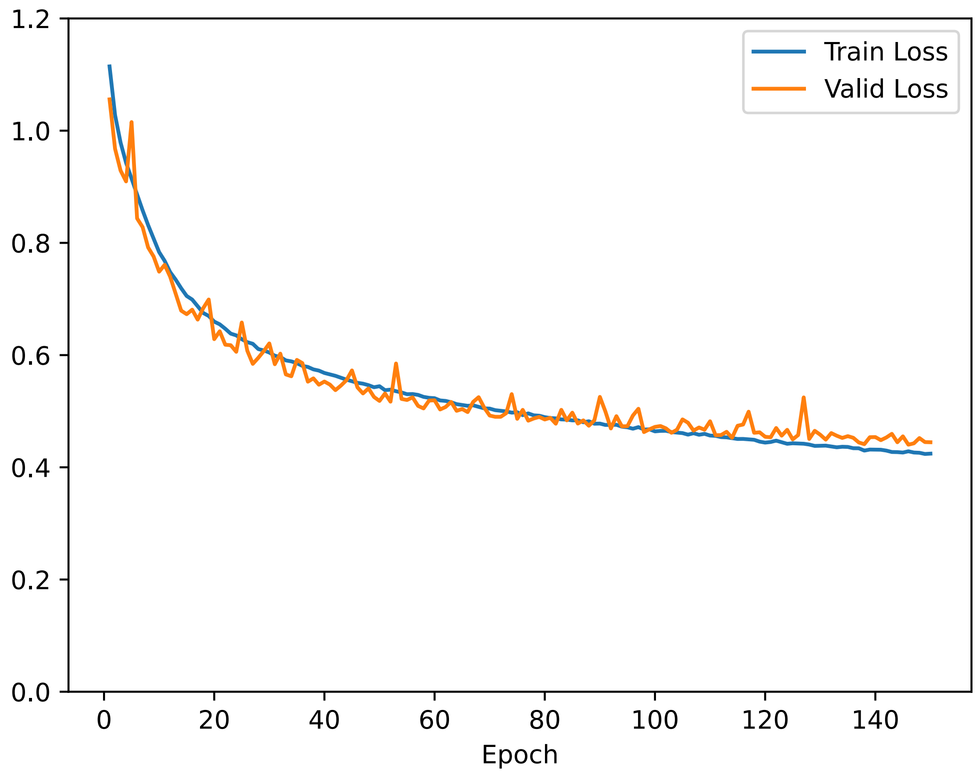

Figure 13: MLP train and validation losses for a dropout probability of 0.2.

The configurations differ as the default model doesn't have any regularization technique, while the model with a dropout probability of 0.2 uses dropout regularization. The random drop of certain connections forces the network to learn a more robust representation of the data, as it cannot rely on the presence of a particular unit.

The presence of a regularization technique leads to lower variations of the validation accuracy and validation loss between epochs. It also translates into a higher test loss at the cost of a lower validation loss. As the model is more restricted in the training process, it is more capable of generalizing for unseen data. The presence of dropout regularization reduces overfitting.

Question 3

1. *In this exercise, you will design a multilayer perceptron to compute a Boolean function of D variables, $f; \{-1,+1\}^D \mapsto \{-1,+1\}$, defined as:*

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

where A and B are integers such that $-D \leq A \leq B \leq D$.

a) *Show that the function above cannot generally be computed with a single perceptron.*

If we consider $D = 2$, $A = -1$ and $B = 1$, the function $f(x)$ is mapped into the XOR function, as suggested by the following truth table:

x_1	x_2	$\sum_{i=1}^{D} x_i$	$f(x)$
-1	-1	-2	-1
-1	1	0	1
1	-1	0	1
1	1	2	-1

A single perceptron can only learn linearly separable functions. As the XOR function is not linearly separable, it cannot be learned by a single perceptron, as there is no hyperplane that can separate the two classes, as suggested by Figure 14.

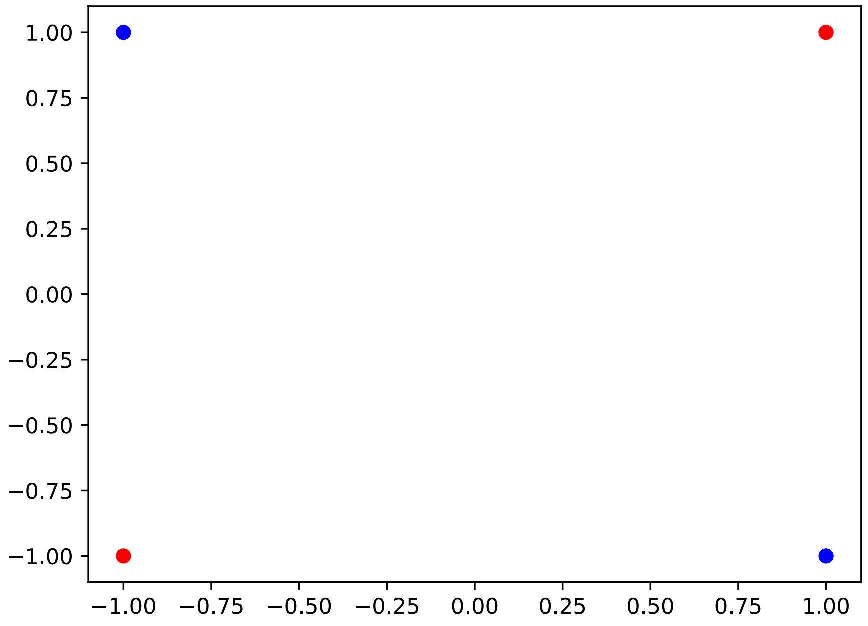

Figure 14: Plot of function f with $D = 2$, $A = -1$ and $B = 1$. Red points represent the class -1, while blue points represent the class 1. The two classes are not linearly separable.

b) *Show that the function above can be computed with a multilayer perceptron with a single hidden layer with two hidden units and hard threshold activations $g : \mathbb{R} \rightarrow$*

$\{-1, +1\}$ *with*

$$g(z) = \operatorname{sign}(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

and where ***all the weights and biases are integers*** *(positive, negative, or zero).Provide all the weights and biases of such network, and ensure that the resulting network is robust to infinitesimal perturbation of the inputs, i.e., the resulting function* $h : \mathbb{R}^D \to \mathbb{R}$ *should be such that* $\lim_{t \to 0} h(x + tv) = h(x) = f(x)$ *for any* $x \in \{-1, +1\}$ *and* $v \in \mathbb{R}^D$.

Considering D = 2, A = -1 and B = 1, the function $f(x)$ is mapped into the XOR function. The two hyperplanes given by the equations $x_1 + x_2 = -1$ and $x_1 + x_2 = 1$ separate the two classes, as suggested by Figure 15.

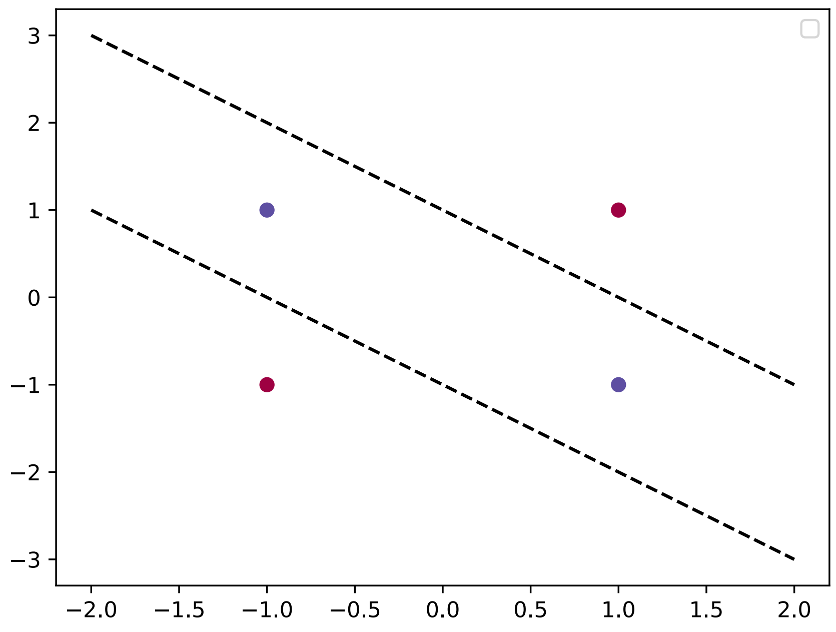

Figure 15: Plot of function f with D = 2, A = -1 and B = 1. Red points represent the class -1, while blue points represent the class 1. The points between the two hyperplanes are classified as 1, while the points outside the two hyperplanes are classified as -1.

In general, for any D, A and B such that $-D \leq A \leq B \leq D$, the two hyperplanes that separate the two classes are given by the equations

$$\sum_{i=1}^{D} x_i = A \tag{1}$$

$$\sum_{i=1}^{D} x_i = B \tag{2}$$

The following weights and biases for the hidden layer and output layer allow the network to learn the function $f(x)$:

$$\begin{aligned}
W^{(1)} &= \begin{bmatrix} 1 & \dots & 1 \\ 1 & \dots & 1 \end{bmatrix} \text{ with dimensions } (2 \times D) \\
b^{(1)} &= \begin{bmatrix} B \\ A \end{bmatrix} \\
W^{(2)} &= \begin{bmatrix} 1 & -1 \end{bmatrix} \\
b^{(2)} &= \begin{bmatrix} -1 \end{bmatrix}
\end{aligned} \tag{3}$$

c) *Repeat the previous exercise if the hidden units use rectified linear unit activations instead.*

Considering $D = 2$ and $A = B = 0$, the following weights and biases for the hidden layer and output layer allow the network to learn the function $f(x)$:

$$\begin{aligned}
W^{(1)} &= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \\
b^{(1)} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\
W^{(2)} &= \begin{bmatrix} -2 & 1 \end{bmatrix} \\
b^{(2)} &= \begin{bmatrix} -1 \end{bmatrix}
\end{aligned} \tag{4}$$