Question 1

Medical image classification with linear classifiers and neural networks. In this exercise, you will implement a linear classifier for a simple medical image classification problem, using the OCTMNIST dataset. The dataset contains 109,309 valid optical coherence tomography (OCT) images for retinal diseases comprised of 4 diagnosis categories - multi-class classification task. The original training set was split with a ratio of 9:1 into training and validation set, and use its source validation set as the test set. The source images are gray-scale with sizes varying between (384–1,536)×(277–512). Images were center-cropped with a window size of length of the short edge and resize them into 1×28×28. **Please do not use any machine learning library such as scikit-learn or similar for this exercise; just plain linear algebra (the numpy library is fine).** Python skeleton code is provided (`hw1-q1.py`).

In order to complete this exercise, you will need to download the OCTMNIST dataset. You can do this by running the following command in the homework directory:

```
python download_octmnist.py
```

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. You can do this with the command

```
python hw1-q1.py perceptron
```

We started by acquiring the OCTMNIST dataset as instructed, followed by visualizing a selection of images from the dataset, illustrated in Figure 1.

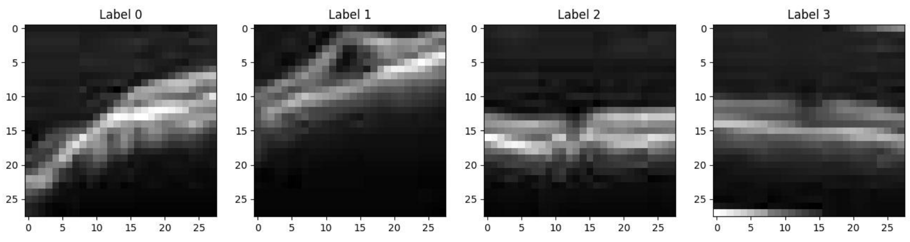


Figure 1: Examples of images from the OCTMNIST dataset.

Subsequently, we implemented the remaining Perceptron code, according to what we learned in the practical classes. This algorithm starts by adding an extra dimension to the input vector and initializing the weight matrix to zero, where the first dimension corresponds to the bias. For each epoch, the model is applied to the inputs in order to get the corresponding prediction. If a prediction is correct, no adjustments are made. If it is wrong, the weights are updated by adding or subtracting the corresponding input feature.

The perceptron algorithm was trained for 20 epochs and the achieved accuracies are as follows: 0.4654 for the training set, 0.4610 for the validation set, and a final test accuracy of 0.3422. The accuracies for the training and validation sets as a function of the epoch number are represented in Figure 2.

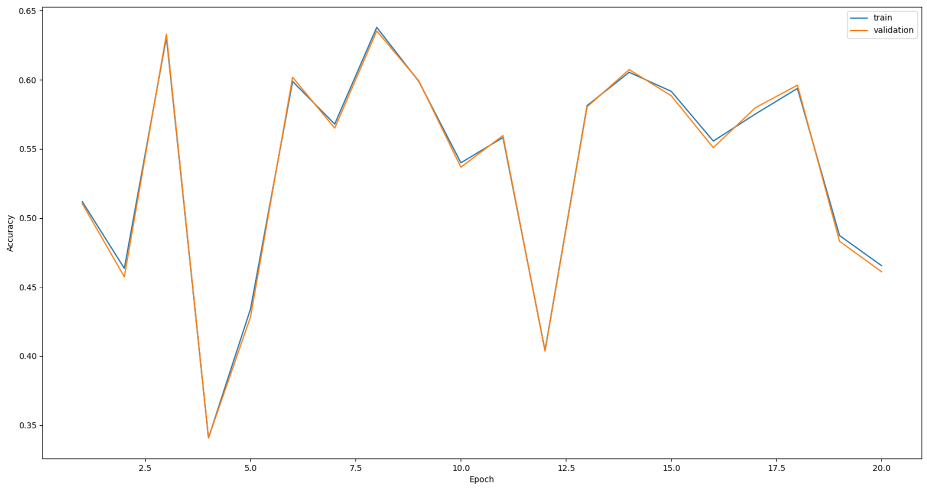


Figure 2: Training and validation accuracies as a function of the epoch number for the perceptron model on the OCTMNIST dataset.

As illustrated in Figure 2, the model's accuracy on both the training and validation sets is relatively low. This suggests that the perceptron model might be too simple to capture the underlying patterns in the data. Furthermore, the even lower accuracy on the test set indicates that the model does not generalize well to new unseen data.

1. (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 $\eta = 0.001$. This can be solved by implementing the `update_weights` method in the `LogisticRegression` class. You can do this with the command

```
python hw1-q1.py logistic_regression -epochs 50 -learning_rate 0.01
```

In the logistic regression model, the weights are updated according to Eq. 1,

$$\mathbf{W} = \mathbf{W} + \eta\left((\mathbf{e_{y_i}} - \mathbf{p_i}) \otimes \mathbf{x}_i\right), \text{ with } p_i = \frac{e^{\mathbf{W}_i \cdot \mathbf{x}_i}}{\sum_j e^{\mathbf{W}_j \cdot \mathbf{x}_i}}, \tag{1}$$

where η is the learning rate and e_{y_i} is the one-hot vector in the direction of the gold label.

This model was trained using 2 different learning rates and the accuracy plots for the train and validation sets for $\eta = 0.01$ and $\eta = 0.001$ are shown in Figures 3 and 4, respectively. With a learning rate of 0.01, the model achieves a training accuracy of 0.6609, a validation accuracy of 0.6568, and a final test accuracy of 0.5784 after 50 epochs. In contrast, employing a smaller learning rate of 0.001 results in a slightly improved performance, yielding a training accuracy of 0.6625, a validation accuracy of 0.6639, and a final test accuracy of 0.5936.

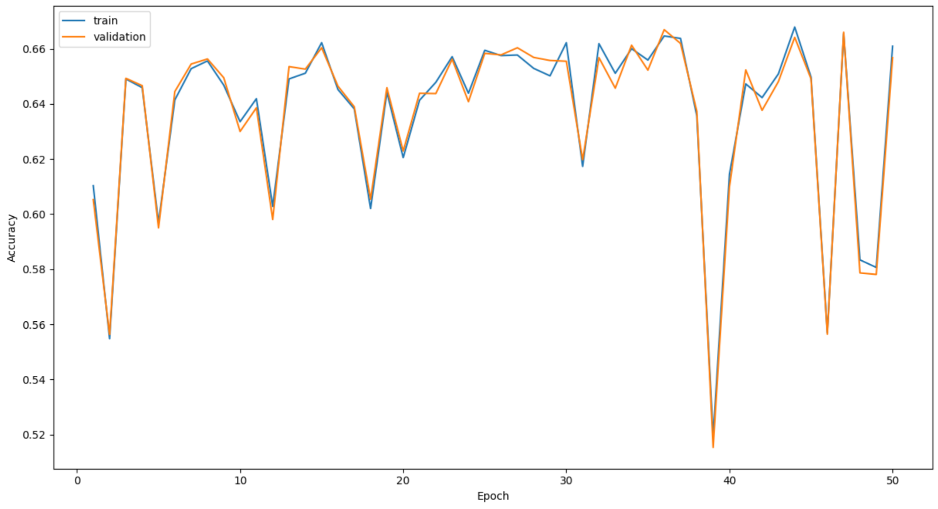


Figure 3: Training and validation accuracies as a function of the epoch number for the logistic regression model on the OCTMNIST dataset with learning rate $\eta = 0.01$.

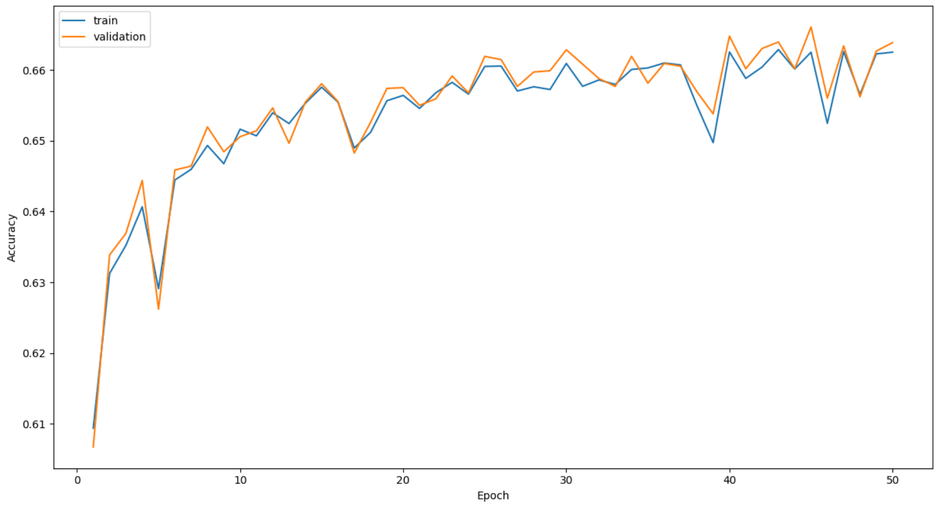


Figure 4: Training and validation accuracies as a function of the epoch number for the logistic regression model on the OCTMNIST dataset with learning rate $\eta = 0.001$..

The accuracy plot for the lower learning rate exhibits a smoother trend, indicating more stable convergence during training. This suggests that a smaller learning rate prevents overshooting, leading to better generalization, and highlights the importance of hyperparameter tuning.

In comparison to the previous perceptron model, the logistic regression not only exhibits substantially higher accuracy values but also demonstrates improved generalization. This may be related to the probabilistic nature of the logistic regression which allows for smoother predictions and better adaptation to complexities in the data. However, it is important to note that neither model reached optimal accuracy.

2. Now, you will implement a multi-layer perceptron (a feed-forward neural network) again using as input the original feature representation (i.e. simple independent pixel values).
(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.

In this context, the term "expressiveness" refers to a model's capacity to capture complex patterns and relationships within the data. The logistic regression is a linear model that takes a feature vector—pixel values in this scenario—as input and outputs a probability of the image belonging to a class. It does so by applying a sigmoid transformation to a linear combination/weighted sum of the features. However, the logistic regression assumes the linearity and independence of the features, making it struggle to capture more complex relationships, such as feature interactions and non-linear relationships.

On the other hand, a multi-layer perceptron (MLP) is a neural network architecture with multiple layers of interconnected nodes. Each node in a layer connects to every node in the preceding layer, and the output of each node is a non-linear transformation using an activation function, in our case using the non-linear function ReLU, of the weighted sum of its inputs. This allows the MLP to capture non-linear relationships and interactions between features effectively. Furthermore, each subsequent hidden layer learns increasingly abstract features resulting from the previous layer, allowing the model to learn even more complex relationships from the data.

As previously, mentioned both models use weighted sums, therefore for both models it is required to find the optimal weights/coefficients that minimize the loss. As implied, the process of training the models is an optimization problem. The logistic regression is straightforward to train due to being a convex optimization problem, meaning that there exists a global minimum. Using optimization algorithms such as the gradient descent with an appropriate step size ensures a unique and optimal, or very close to optimal depending on the step-size, solution.

The MLP due to having multiple hidden layers and non-linear and, in the case of ReLU, non-differentiable activation functions makes the optimization problem non-convex, meaning that there are multiple local minima, making it much more complex to find the optimal solution. To find the optimal solutions we rely on forward and backward propagation for several epochs. Furthermore, appropriate weight initialization and learning rate are also necessary to ensure good results.

In conclusion, the affirmation is true. Training a logistic regression is much simpler than training the MLP, however, the MLP can better capture intricate complex relationships in the data.

2. (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 $\sigma^2 = 0.1^2$ (hint: use `numpy.random.normal`). Run your code with the base command, adding the necessary arguments

```
python hw1-q1.py mlp
```

Report final test accuracy and include the plots of the train loss and train and validation accuracies as a function of the epoch number.

The MLP was trained using the default settings: 20 epochs, 200 hidden nodes in the single hidden layer and a learning rate of 0.001. It achieved a final loss of 57887.3192, a training accuracy of 0.7970, a validation accuracy of 0.7873 and a final test accuracy of 0.7467. The accuracies for the training and validation sets and the train loss are illustrated in Figures 5 and 6, respectively.

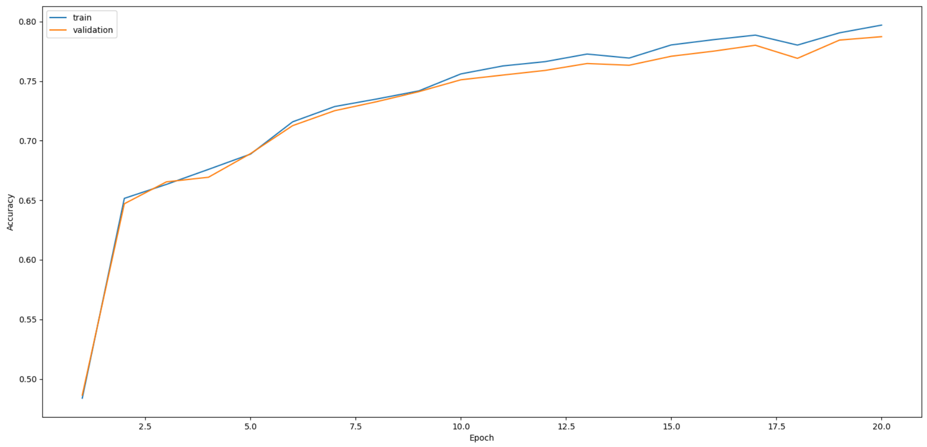


Figure 5: Training and validation accuracies as a function of the epoch number for the MLP model on the OCTMNIST dataset

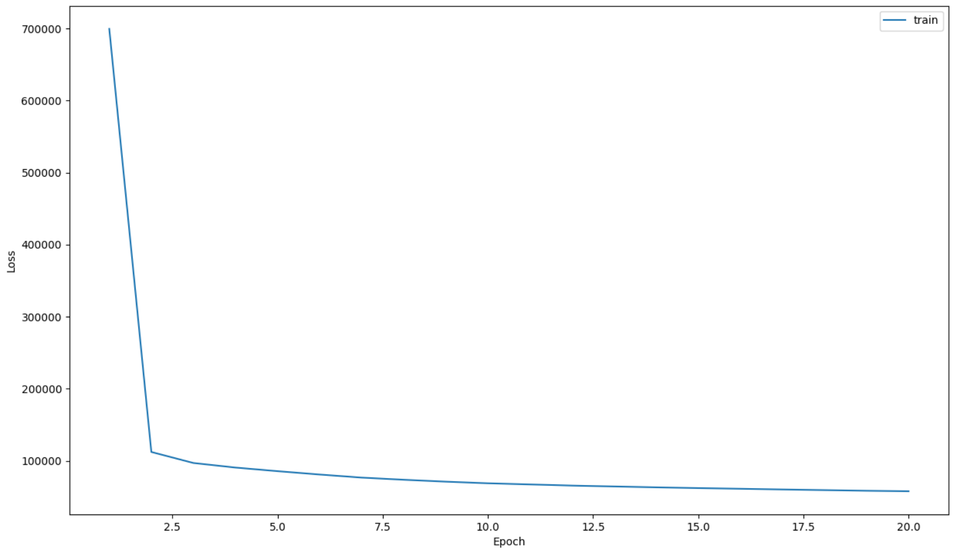


Figure 6: Training loss as a function of the epoch number for the MLP model on the OCTMNIST dataset

When comparing with the results obtained with the previous implementations of the logistic regression, we can see that the accuracies across all three sets are significantly higher. The MLP, in particular, outperformed the other models, especially when predicting the test set. This emphasizes the effectiveness of non-linear models in capturing complex patterns in data, particularly when dealing with scenarios where the data isn't easily linearly separable, which might be the case in this exercise.

Question 2

Medical image classification with an autodiff toolkit. In the previous question, you had to write gradient backpropagation by hand. This time, you will implement the same system using a deep learning framework with automatic differentiation. Pytorch skeleton code is provided (`hw1-q2.py`) but if you feel more comfortable with a different framework, you are free to use it instead.

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.

In the skeleton code, you will need to implement the method `train_batch()` and the class `LogisticRegression`'s `__init__()` and `forward()` methods.

In this exercise, the model featured a linear layer without any activation function, along with Cross Entropy Loss. The inclusion of a sigmoid layer was unnecessary in this case, as the Cross Entropy Loss function automatically applies Log SoftMax loss, better suited for multi-class problems.

Learning rate	Train loss	Val. accuracy	Test accuracy
0.1	0.9687	0.6224	0.5577
0.01	0.9370	0.6535	0.6200
0.001	1.0145	0.6163	0.6503

Table 1: Loss and accuracies for different learning rates for the linear model with logistic regression.

From Table 1, we can see that, even though for a learning rate of 0.01 the logistic regression has the highest validation accuracy and lowest training loss for a batch size of 16, a learning rate of 0.001 provides the best results in the final test accuracy. This configuration has a final training loss of 1.0145, a validation accuracy of 0.6163 and final test accuracy of 0.6503. The corresponding training loss and validation accuracy are plotted as a function of the epoch number in Figure 7.

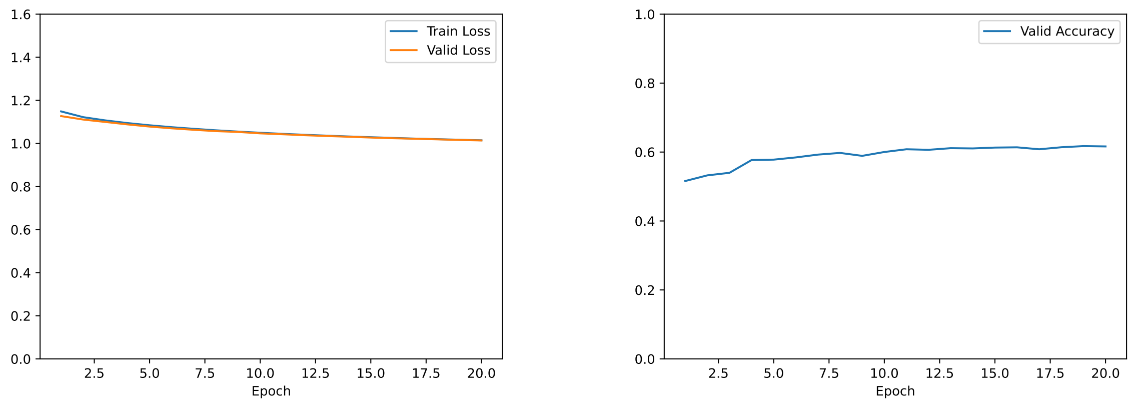


Figure 7: Training loss and validation accuracy as a function of the epoch number for the Logistic Regression with a learning rate of 0.001 on the OCTMNIST dataset.

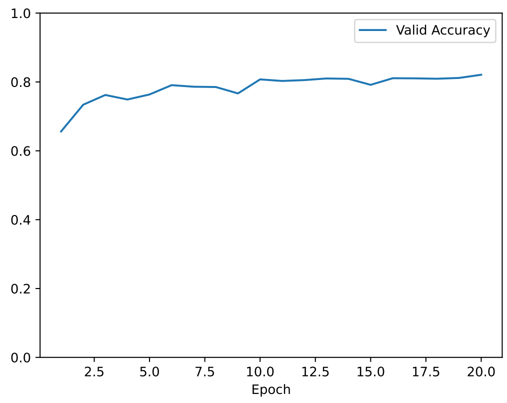
2. Implement a feed-forward neural network using dropout regularization. Make sure to include all the hyperparameters and training/model design choices shown in Table 1. Use the values presented in the table as default. In the skeleton code, you will need to implement the class `FeedforwardNetwork`'s `__init__()` and `forward()` methods.

(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.

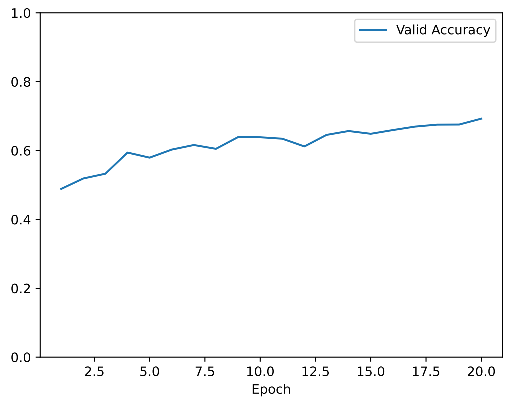
The performance of the neural network with batch sizes 16 and 1024 with the remaining hyperparameters at their default value is shown in Table 2. The corresponding validation accuracies and training and validation losses as a function of the epoch number are shown in Figure 8 and 9, respectively.

Batch size	Train loss	Val. accuracy	Test accuracy	Time (s)
16	0.4377	0.8209	0.7391	231.5
1024	0.8760	0.6927	0.7353	52.27

Table 2: Loss, accuracies and time for different batch sizes for the feed-forward neural network model.

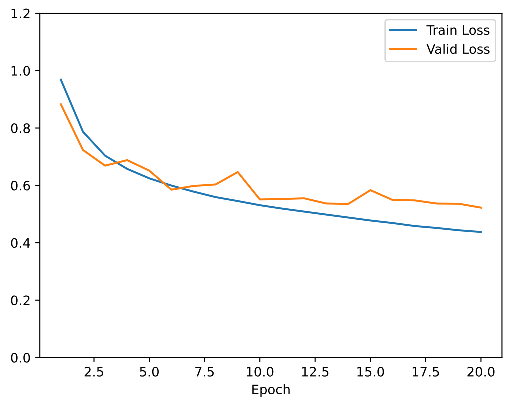


(a) Validation accuracies for the feed-forward neural network model model with batch size of 16.

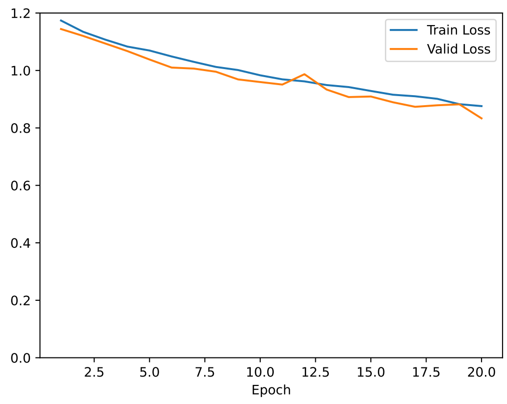


(b) Validation accuracies for the feed-forward neural network model model with batch size of 1024.

Figure 8: Comparison of validation accuracies as a function of the epoch number.



(a) Training and validation loss for the feed-forward neural network model with batch size of 16.



(b) Training and validation loss for the feed-forward neural network model with batch size of 1024.

Figure 9: Comparison of training and validation loss as a function of the epoch number.

For the batch size of 16, the model achieved a lower training loss, higher validation accuracy, and a slightly better final test accuracy compared to the batch size of 1024. This indicates that the model with a batch size of 16 was able to capture more intricate patterns in the training data, however it is more prone to overfitting. This superior performance also came at the cost of a significantly longer execution time, taking 231.5 seconds to complete, which is expected due to the more frequent weight updates. The choice of batch size involves a trade-off between model performance and computational efficiency. On the one hand, smaller batch sizes often lead to better generalization but at the expense of increased computational costs and longer training times. On the other hand, larger batch sizes are computationally more efficient but might compromise the model's sensitivity to subtle data patterns. Therefore, the choice, depends on the problem at hand and the computational resources available. In this particular case, despite the better validation accuracy, the final test accuracy for the batch size of 16 is only slightly better than the batch size of 1024, which raises questions about whether the improvement in accuracy justifies the longer training time.

(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 performance of the neural network for different learning rates is represented in Table 3.

Learning rate	Train loss	Val. accuracy	Test accuracy
1	1.1714	0.4721	0.4726
0.1	0.4370	0.8205	0.7599
0.01	0.4868	0.8074	0.7561
0.001	0.8426	0.6926	0.7164

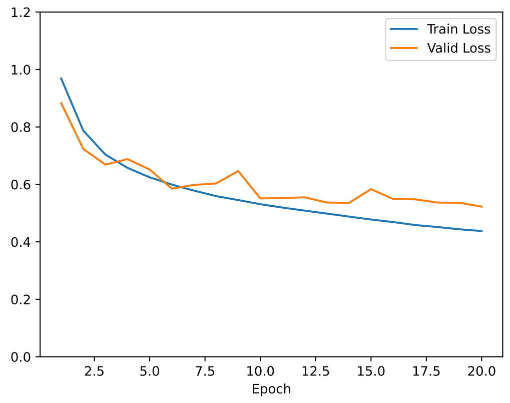
Table 3: Loss and accuracies for different learning rates for the feed-forward neural network model.

The model with learning rate 1, performed poorly since it achieved high training losses and low and test validation accuracies. This is expected since high learning rates often cause models to overshoot during training, leading to oscillations, getting stuck in local minima and affecting the model's ability to generalize well. In contrast, the models with the learning rate values, 0.1 and 0.01, achieved considerably lower training losses and much higher accuracies. This is due to the fact that a lower learning rate enables more precise weight updtates, contributing to a more stable convergence. However, the model with learning rate of 0.001 achieved a lower validation and test accuracy compared to the previous two. This means that the learning rate might be too low, slowing down the convergence process. The training loss is also higher, indicating that the model might need more iterations (more than 20 epochs) to reach optimal weights. It is important to note that these results highlight the critical role of fine-tuning the learning rate in model training in order to achieve optimal model performance.

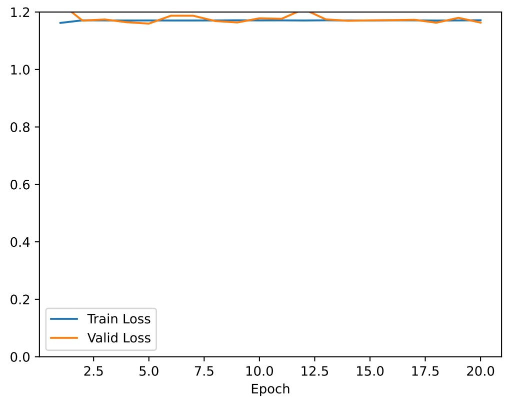
According to Table 3, the worst and best configuration were the ones with learning rates 1 and 0.1, respectively. The corresponding train and validation losses are illustrated in Figure 10.

(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 training and validation losses as well as the validation accuracy for the model with a batch size of 256 and 150 epochs are represented in Figure 11.

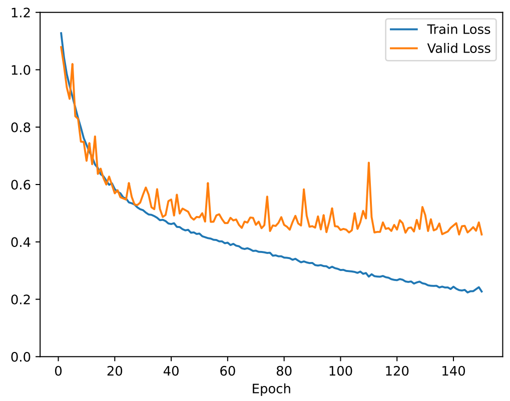


(a) Training and validation loss for the feed-forward neural network model with learning rate of 0.1 and remaining default parameters.



(b) Training and validation loss for the feed-forward neural network model with learning rate of 1 and remaining default parameters.

Figure 10: Comparison of training and validation loss for the best and worst learning rate configuration as a function of the epoch number.



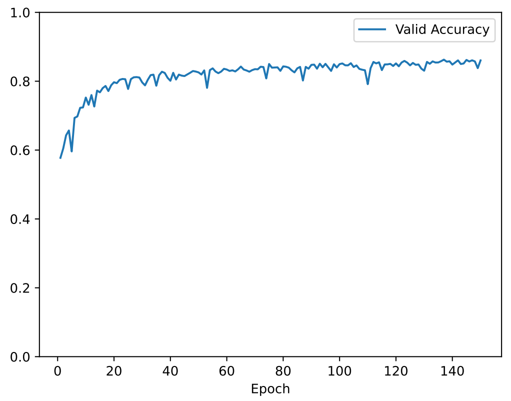


Figure 11: Training and validion losses (left) and validation accuracy (right) as a function of the epoch number for the feed-foward neural network with a batch size of 256 and 150 epochs as a function of the epoch number.

The model achieved a high validation accuracy of 0.8605, indicating good performance on the training set. However, as we can see from the plot, the plateau of the validation accuracy combined with the significantly lower test accuracy suggests overfitting (0.7675) suggest overfitting. This means that the model is too tailored to the training data and does not generalize well to new, unseen data. A possible solution to the overfitting problem is to introduce regularisation techniques such as L2 regularisation or dropout. The introduction of L2 regularisation with the l2 parameter = 0.0001 slightly mitigated overfitting compared to the default model. This regularisation term penalizes large weights preventing the model from fitting noise in the training data. However, even tough the test accuracy improved (0.7921), there is still a noticeable gap between training and test accuracy (0.8582) Dropout regularisation demonstrated superior performance. By randomly dropping out a fraction of neurons during training, dropout prevents dependencies between hidden units and enhances the model's ability to generalize. This is reflected in the higher test accuracy of 0.8110 which is much closer to the validation accuracy of 0.8582. This results demonstrate that regularization techniques are powerful techniques for preventing overfitting and must be taking into account when training a model. To conclude, the worst and best configurations corresponded to the default model and the model with dropout regularisation, respectively. The corresponding train and validation losses and validation accuracies are illustrated in Figure 11 and 12.

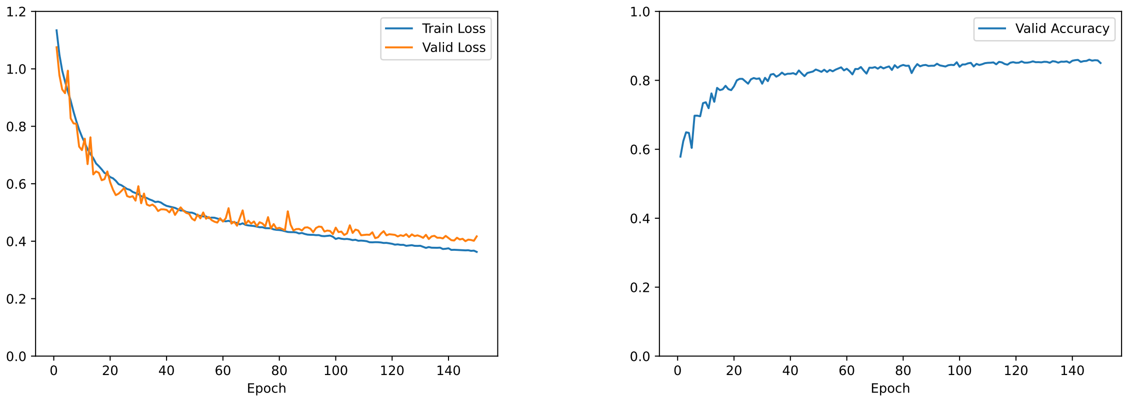


Figure 12: Training and validion losses (left) and validation accuracy (right) as a function of the epoch number for the feed-foward neural network with a batch size of 256, 150 epochs and dropout probability of 0.2 as a function of the epoch number.

Question 3

a) Assuming that D = 2, A= -1 and B = 1, we have: f(x) = $\begin{cases} 1 & \text{if } x_1 + x_2 \in [-1, 1] \\ -1 & \text{otherwise} \end{cases}$

the points be:

$$\begin{aligned}
\mathbf{x} &= \begin{bmatrix}-1 & 1\end{bmatrix}^\top, \quad -1+1=0 && \Rightarrow \quad y = 1\\
\mathbf{x} &= \begin{bmatrix}1 & -1\end{bmatrix}^\top, \quad 1+(-1)=0 && \Rightarrow \quad y = 1\\
\mathbf{x} &= \begin{bmatrix}1 & 1\end{bmatrix}^\top, \quad 1+1=2 && \Rightarrow \quad y = -1\\
\mathbf{x} &= \begin{bmatrix}-1 & -1\end{bmatrix}^\top, \quad -1+(-1)=-2 && \Rightarrow \quad y = -1
\end{aligned}$$

To show that the perceptron cannot learn the function, we just have to show a single example, for which there isn't a hyperplane that is able to separate all the positive outputs (+1) from the negative outputs (-1). Since for this example we are using two dimensions, the hyperplane will be a line. As we can see in the plot bellow there isn't any line that is able to separate all the positive outputs from the negative outputs.

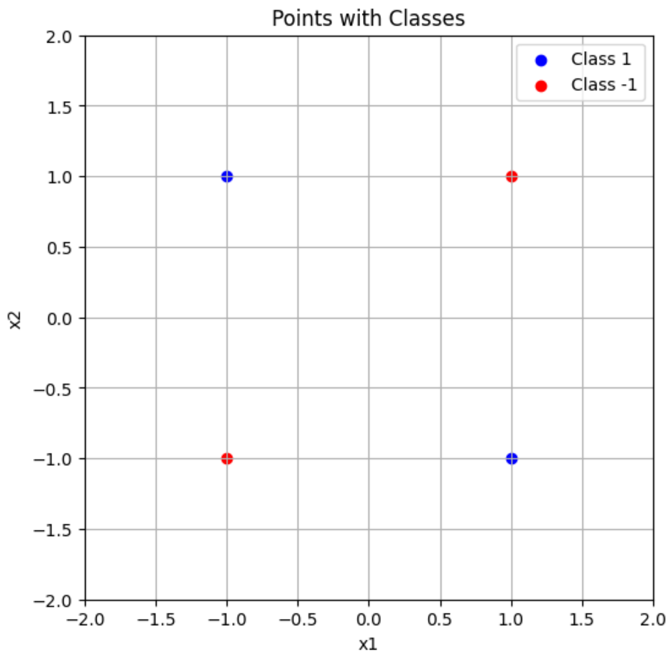


Figure 13: Counter example

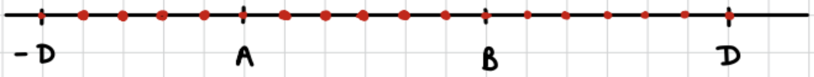
(b) Boolean function $f:\{-1,+1\}^D \rightarrow \{-1,+1\}$

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

$\hookrightarrow \sum_{i=1}^{D} x_i \in \{\dots, A-2, A-1\} \cup \{B+1, B+2, \dots\}$

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

We want to ensure that the resulting network is robust to infinitesimal perturbations of the inputs. This means that if inputs suffer a small perturbation (eg. some inputs are -0,999 instead of -1) the output of the network should not change.
Therefore, $\sum x_i$ can fall in any or infinitesimally close to any of the red points (representing integer values).



If $f(x) = 1$:

$$\begin{cases} x_1 + \dots + x_D - A \geq 0 \\ x_1 + \dots + x_D - B \leq 0 \end{cases}$$

If $f(x) = -1$:

$$\begin{cases} x_1 + \dots + x_D - A < 0 \\ x_1 + \dots + x_D - B < 0 \end{cases} \quad \text{OR} \quad \begin{cases} x_1 + \dots + x_D - A > 0 \\ x_1 + \dots + x_D - B > 0 \end{cases}$$

We have a problem, $h(x)$ may to be 1 to solve this. We know that $\sum_{i=1}^{D} x_i$ is an integer, so:

$$\sum x_i \geq A \Rightarrow \sum x_i > A - 1$$

Let's check if $h(x)$ is robust. Assuming $D = 2$, $A = 1$, $B = 2$

$a = \Sigma x_i = 0{,}9999$

$(0{,}9999 - 1) = -0{,}0001$
$a - A \geq 0 \rightarrow y = -1$ ✗ wrong output, but $(0{,}9999 - 1 + 1 = 0{,}9999)$ $a - A + 1 > 0 \rightarrow y = 1$ ✓ correct output

However, this does not work if $b = 0{,}0001$

$(0{,}0001 - 1) = -0{,}9999$
$b - A \geq 0 \rightarrow y = -1$ ✓ correct output, but $(0{,}0001 - 1 + 1) = 0{,}0001$ $b - A + 1 > 0 \rightarrow y = 1$ ✗ wrong output

Therefore, we propose the following solution:

$(2 \cdot 0{,}999 - 2 \cdot 1 + 1) = 0{,}9998$
$2 \cdot a - 2 \cdot A + 1 > 0 \rightarrow y = +1$ ✓ correct output

$2 \cdot b - 2 \cdot A + 1 < 0 \rightarrow y = -1$ ✓ correct output
$(2 \cdot 0{,}0001 - 2 \cdot 1 + 1) = -0{,}9998$

It's important to note that 2 is not the only possible choice. Any positive integer not infinitely large would work:

$-\beta \cdot (A - 0{,}0001) + \beta \cdot A - 1 < 0$
$\Leftrightarrow \beta \times 0{,}0001 - 1 < 0$

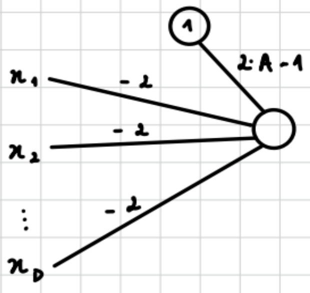
So, we convert the condition $x_1 + \cdots + x_D - A \geq 0$ to

$$2 \cdot x_1 + \cdots + 2x_D - 2 \cdot A + 1 > 0$$

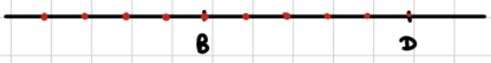
And since the sign function doesn't have the > condition, the solution is to multiply by -1:

$$\boxed{-2 \cdot x_1 + \cdots - 2x_D + 2 \cdot A - 1 < 0}$$

Then, the first part of the neural network becomes:



For the second part of the NN, we can follow the same procedure:



If $\sum x_i \leq B \Rightarrow \sum x_i < B+1$

Assuming $D = 2$, $A = 0$, $B = 1$

$a = \sum x_i = 1{,}001$

$1{,}001 - 1 = 0{,}001$

$a - B \leq 0 \rightarrow y = -1$ × wrong output

$1{,}001 - 1 - 1 = -0{,}999$

$a - B - 1 < 0 \rightarrow y = +1$ ✓ correct output

$b = \sum x_i = 1{,}999$

$b - B \leq 0 \rightarrow y = -1$ ✓ correct output

$1{,}999 - 1 - 1 = -0{,}001$

$b - B - 1 < 0 \rightarrow y = +1$ ✓ wrong output

Applying the same logic as before:

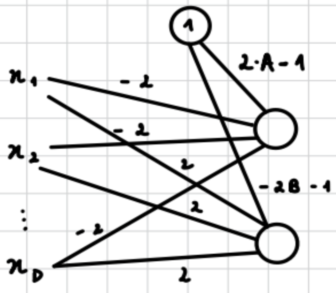
$2 \cdot a - 2 \cdot B - 1 < 0 \rightarrow y = +1$ ✓ correct output

$2 \cdot 1{,}001 - 2 \cdot 1 - 1 = -0{,}998$

$2b - 2 \cdot B - 1 < 0 \rightarrow y = -1$ ✓ correct output

$2 \cdot 1{,}999 - 2 \cdot 1 - 1 = 0{,}998$

Thus, the first and second part of the NN correspond to:



Now, we need to construct the final layer:

If $y = +1$:

$$\begin{cases} \text{sign}(-2\sum x_i + 2\cdot A - 1) = -1 \\ \text{sign}(2\sum x_i - 2\cdot B - 1) = -1 \end{cases}$$

If $y = -1$, we have to cases:

1. $\sum x_i$ is below $[A, B]$

$$\begin{cases} \text{sign}(-2\sum x_i + 2A - 1) = +1 \\ \text{sign}(2\sum x_i - 2\cdot B - 1) = -1 \end{cases}$$

2. $\sum x_i$ is above $[A, B]$

$$\begin{cases} \triangle\ \text{sign}(-2\cdot\sum x_i + 2\cdot A - 1) = -1 \\ \square\ \text{sign}(2\sum x_i - 2\cdot B - 1) = +1 \end{cases}$$

Note: The case where $\triangle = +1$ and $\square = +1$ cannot happen, since that would mean $\begin{cases} \sum x_i < A \\ \sum x_i > B \end{cases}$, which is impossible as $B \geq A$

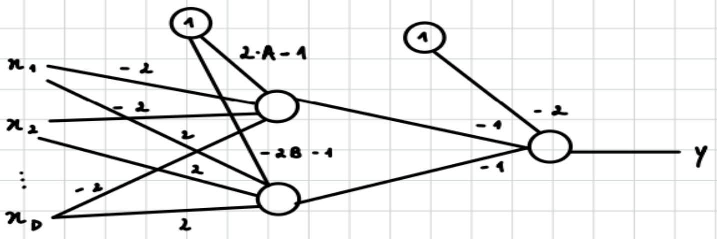
From this we can extract the following equations, where X, Y and Z are the weights:

this has to be a valid solution

$$\begin{cases} X\cdot(-1) + Y\cdot(-1) + Z\cdot 1 \geq 0 \\ X\cdot(+1) + Y(-1) + Z < 0 \\ X\cdot(-1) + Y(+1) + Z < 0 \end{cases} \Rightarrow \begin{cases} -X - Y + Z = 0 \\ X - Y + Z < 0 \\ -X + Y + Z < 0 \end{cases} \Leftrightarrow \begin{cases} Z = X + Y \\ X - Y + X + Y < 0 \\ -X + Y + X + Y < 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} Z = X + Y \\ X < 0 \\ Y < 0 \end{cases} \longrightarrow \text{Choosing the smaller integers: } \boxed{\begin{cases} X = -1 \\ Y = -1 \\ Z = -2 \end{cases}}$$

The final Neural Network is :



with the corresponding weights and bias matrices :

$$W^{(1)} = \begin{bmatrix} -2 & \cdots & -2 \\ 2 & \cdots & 2 \end{bmatrix}_{(2 \times D)} \qquad b^{(1)} = \begin{bmatrix} 2 \cdot A - 1 \\ -2 \cdot B - 1 \end{bmatrix}$$

$$W^{(2)} = [\,-1 \quad -1\,] \qquad b^{(2)} = [\,-2\,]$$

$$\underset{(D+1 \times 1)}{x} = [\,1 \;\; x_1 \;\; x_2 \; \cdots \; x_D\,]^T$$

3 c)

$$\text{ReLU}(z) = \max\{0, z\}$$

Let's try same $W^{[1]}$ and $b^{[1]}$ from the previous exercise

$y = +1$:

$\text{ReLU}(-2 \cdot \Sigma x_i + 2 \cdot A - 1) = 0$

$\text{ReLU}(2 \cdot \Sigma x_i - 2 \cdot B - 1) = 0$

$$\begin{cases} X \cdot 0 + Y \cdot 0 + Z \cdot 1 \geq 0 \\ X(-2 \cdot \Sigma x_i + 2 \cdot A - 1) + Y \cdot 0 + Z \cdot 1 < 0 \\ X \cdot 0 + Y(2 \cdot \Sigma x_i - 2 \cdot B - 1) + Z \cdot 1 < 0 \end{cases}$$

$y = -1$:

1st − case Σx_i is bellow $[A, B]$

$\text{ReLU}(-2 \cdot \Sigma x_i + 2 \cdot A - 1) = -2 \cdot \Sigma x_i + 2 \cdot A - 1$

$\text{ReLU}(2 \cdot \Sigma x_i - 2 \cdot B - 1) = 0$

2nd − case Σx_i is above $[A, B]$

$\text{ReLU}(-2 \cdot \Sigma x_i + 2 \cdot A - 1) = 0$

$\text{ReLU}(2 \Sigma x_i - 2 \cdot B - 1) = 2 \cdot \Sigma x_i - 2 \cdot B - 1$

$$\overset{\text{valid solution}}{\Rightarrow} \begin{cases} Z = 0 \\ X(\overbrace{-2 \cdot \Sigma x_i + 2 \cdot A - 1}^{>0}) < 0 \\ Y(2 \cdot \Sigma x_i - 2 \cdot B - 1) < 0 \end{cases} \Leftrightarrow \begin{cases} Z = 0 \\ X < 0 \\ Y < 0 \end{cases}$$

So we propose

$X = -1$
$Y = -1$

$$W^{[1]} = \begin{bmatrix} -2 & -2 & \cdots & -2 \\ 2 & 2 & \cdots & 2 \end{bmatrix}_{2 \times D}$$

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

$$W^{[2]} = [\,-1 \quad -1\,] \quad b^{[2]} = [\,0\,]$$

1 What each member did in this homework

The project tasks were spread almost evenly between both colleagues. Question 1 was done by both members side by side, Exercise 2 was solved by Marta and 3 by Vasco. Regarding the report, each colleague wrote the answers to the questions they solved, except for Question 1, which was equally divided between them.