P1: Basic MLP Implementation, PyTorch

Using Optimizers and AutoGrad

Group Members

Alejandro González Álvarez, 252658

Luca Franceschi, 253885

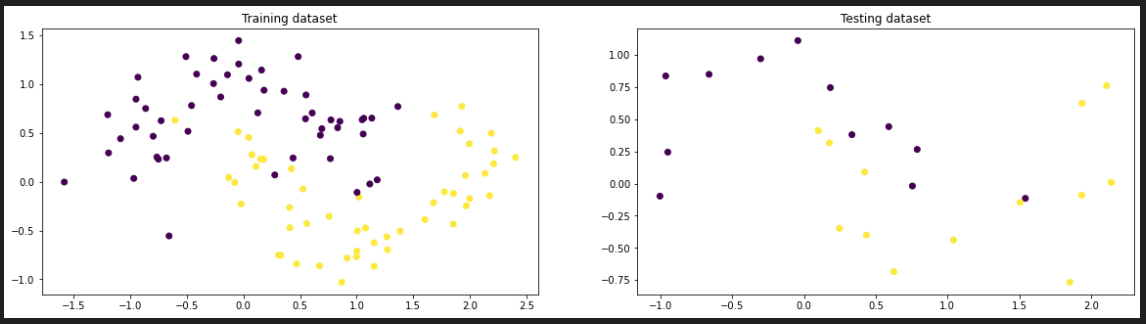
Júlia Othats-Dalès, 254435

In this report we aim to provide an overview of the results obtained from our exploration of different concepts, ranging from multi-layer perceptrons, gradient descent optimization and automatic differentiation. For each exercise we will discuss the results and provide an explanation of our approach and solution, along with any experiments made.

Exercise 1

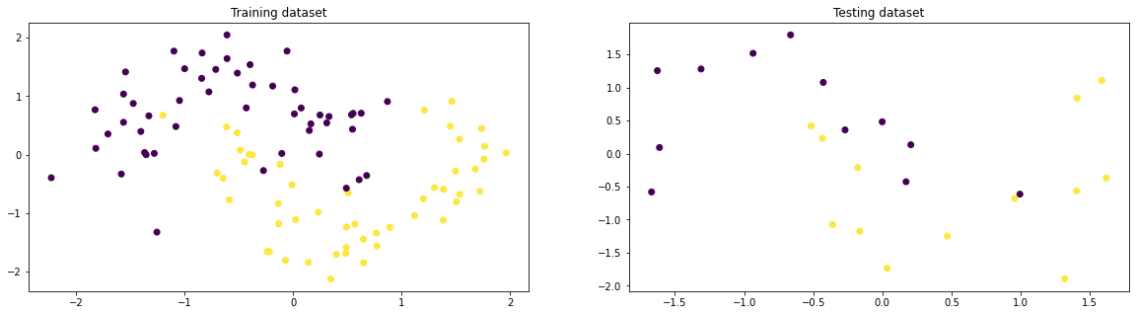
The goal of this exercise is to train a Multi-Layer Perceptron (MLP) using the Numpy library to then solve a classification problem.

We began by loading the provided datasets **train.csv** and **test.csv**, using the Pandas library.

Then, in order to visualize and normalize the datasets we generated some scatter plots of the original training and testing datasets. Each plot represents the dataset with points distributed across the X and Y dimensions, color-coded based on their labels. The visualization helps to understand the distribution of the points and their separability. We obtain the following plots:

Next, we wanted to normalize the datasets’ dimensions, as it helps in improving the convergence of optimization algorithms. We first extracted the X and Y features of both the training and testing datasets and stacked the arrays horizontally to create a combined dataset. After that we applied Z-score normalization to the new datasets so that the mean of all the values is 0 and the standard deviation is 1.

We used the following formula: .

We can then re-plot the normalize datasets to see how they changed. As we can see on the resulting plots, the data is now centered.

In the third step we implement the MLP using Numpy, for which we used the MLP class provided in the **P1-Examples** file, tweaking it a little to fit our needs. In this case, we wanted to use the binary cross-entropy loss function, instead of the L2 loss. Therefore, we modified the loss function of our class.

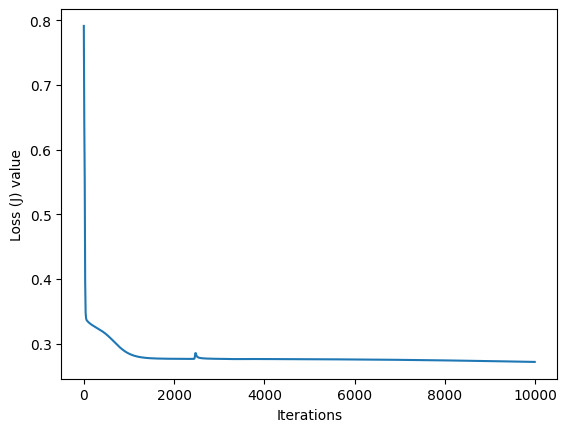
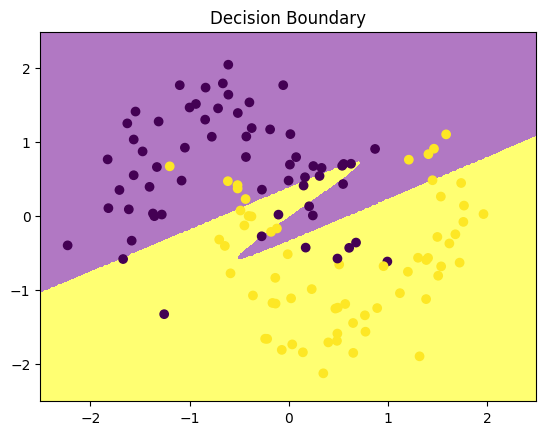
To define this function we used the np.mean function, as it solves the same thing as the summation over all i divided by number of points.

The only other modification we made was in the backward propagation function, were we simplified the multiplication of , also known as delta3 in our code. We follow these calculations to simplify.

Then we implement the training function, to which we made no significant changes. Finally, in order to predict the classification of our testing data, along with its accuracy, we built a predict() function which essentially applies the forward() function to our neural network. To compute the accuracy, we counted for every pair of y, y\_hat, the number of differences and returned the percentage of accurate answers over the entire dataset.

After those changes, all necessary functions were implemented to be able to train the MLP and visualize the decision boundary in 2D of the classification.

For the training, when analyzing the loss function graph over iterations, we can see a quick decrease in the loss, which indicates that it quickly converged and that the model was learning effectively.

To visualize the decision boundary we decided to generate a mesh grid of points with values between in the same ranges were our data was. This grid covers the entire 2D space defined by the X and Y dimensions. Then we were able to —as we did with our original datasets— stack the features horizontally to then predict their classification, using the trained perceptron. If we then plotted this mesh (plt.pcolormesh) we could clearly see the decision boundary. We then plotted both the normalized training and testing datasets on top, to see if we had a large amount of error.

Finally, we were able to predict the classification of our testing dataset and computing its accuracy, using the functions we had previously defined. We found that the accuracy obtained by our trained perceptron was of 72%. We would say that this level of accuracy indicates a reasonable level of performance. However, there is always room for further optimization and improvement, if we for example modified the hyper-parameters.