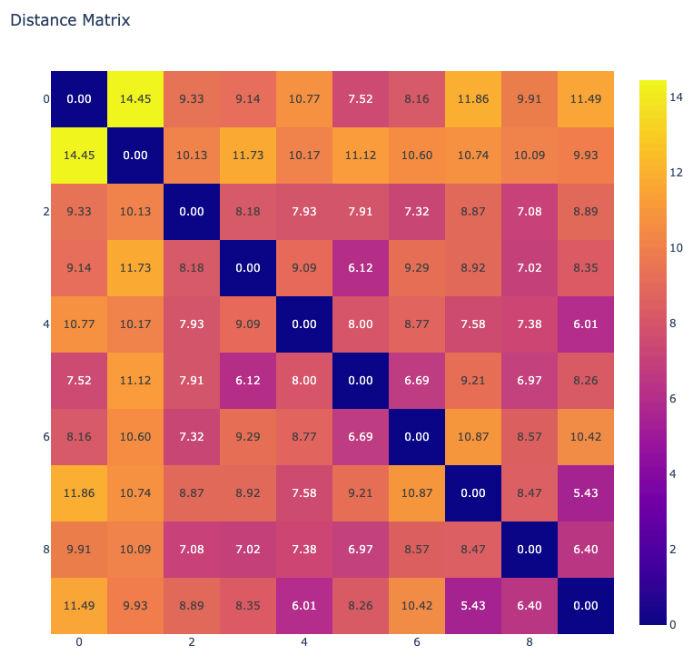
Assignment 0 - Introduction to Deep Learning
Group 53

Task 1: Data dimensionality, distance-based classifiers

Task 1.1

The first point of this task asks us to calculate the center for each digit (from 0 to 9) from the MINST dataset in order to obtain a vector of dimension 256 ($16 * 16$ pixels) and a matrix of shape (10, 256). We have imported the dataset and merged the file `train_in` with `train_out` to have a single labeled dataframe. Then we have calculated the center of each digit using the `groupby` function from pandas and the `mean` function.
Once obtained the centers, we calculated the distance matrix by iterating over each pair of digit centers using two nested loops and computing the L2 norm (Euclidean distance) between them.



0.00	14.45	9.33	9.14	10.77	7.52	8.16	11.86	9.91	11.49
14.45	0.00	10.13	11.73	10.17	11.12	10.60	10.74	10.09	9.93
9.33	10.13	0.00	8.18	7.93	7.91	7.32	8.87	7.08	8.89
9.14	11.73	8.18	0.00	9.09	6.12	9.29	8.92	7.02	8.35
10.77	10.17	7.93	9.09	0.00	8.00	8.77	7.58	7.38	6.01
7.52	11.12	7.91	6.12	8.00	0.00	6.69	9.21	6.97	8.26
8.16	10.60	7.32	9.29	8.77	6.69	0.00	10.87	8.57	10.42
11.86	10.74	8.87	8.92	7.58	9.21	10.87	0.00	8.47	5.43
9.91	10.09	7.08	7.02	7.38	6.97	8.57	8.47	0.00	6.40
11.49	9.93	8.89	8.35	6.01	8.26	10.42	5.43	6.40	0.00

Figure 1: Distance Matrix for the centers of each digit

As it's possible to see from the Figure 1, the digits that are more problematic to classify, using only the centers, are the 9 and 7, where the most distanced classes are 0 and 1.

Task 1.2

In the second part of task 1, we reduced the dimensionality of the data using three different techniques: PCA, U-MAP and T-SNE.

PCA

We have used the PCA implementation from sklearn, setting the number of components to 2, in order to obtain a two dimension space, and setting the random state to 42 to obtain reproducible results.

The Figure 2 shows the results where the different colors represent the different digits. Zero is the most separated class, where the other are in clusters mixed toghether.

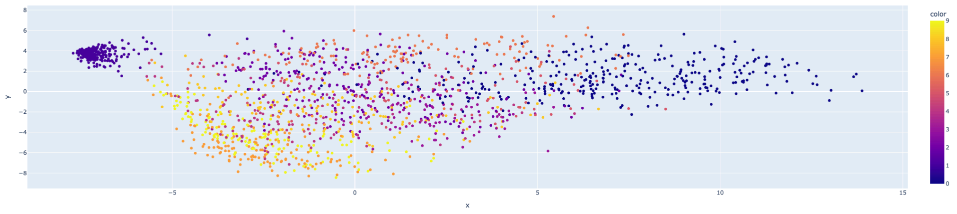


Figure 2: PCA results

U-MAP

The same setting of PCA has been used for U-MAP in terms of dimensionality and reproducibility. The Figure 3 shows how U-MAP is able to describe and separate better the data using only two dimensions. The digits 7, 9 and 4 are the most closest ones but still separated.

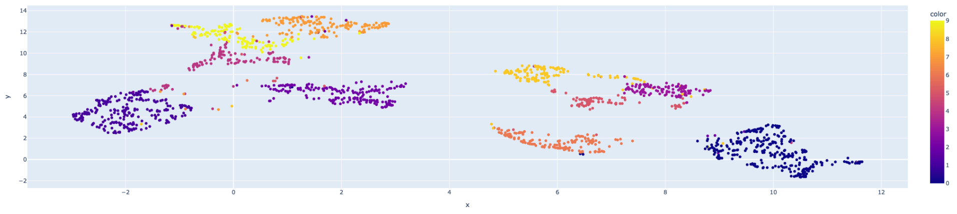


Figure 3: U-MAP results

T-SNE

With T-SNE we have achived a result not good as U-MAP but still better than PCA. The Figure 4 shows cluster almost well separated but distributed in a larger space and less compact.

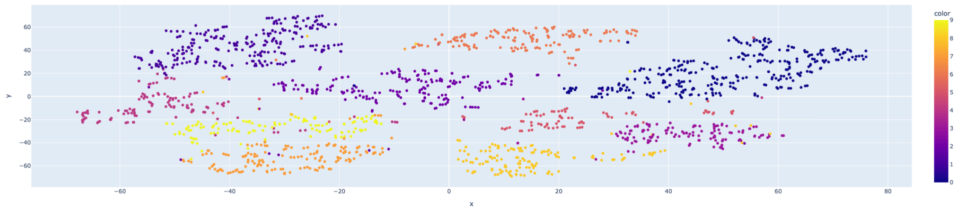
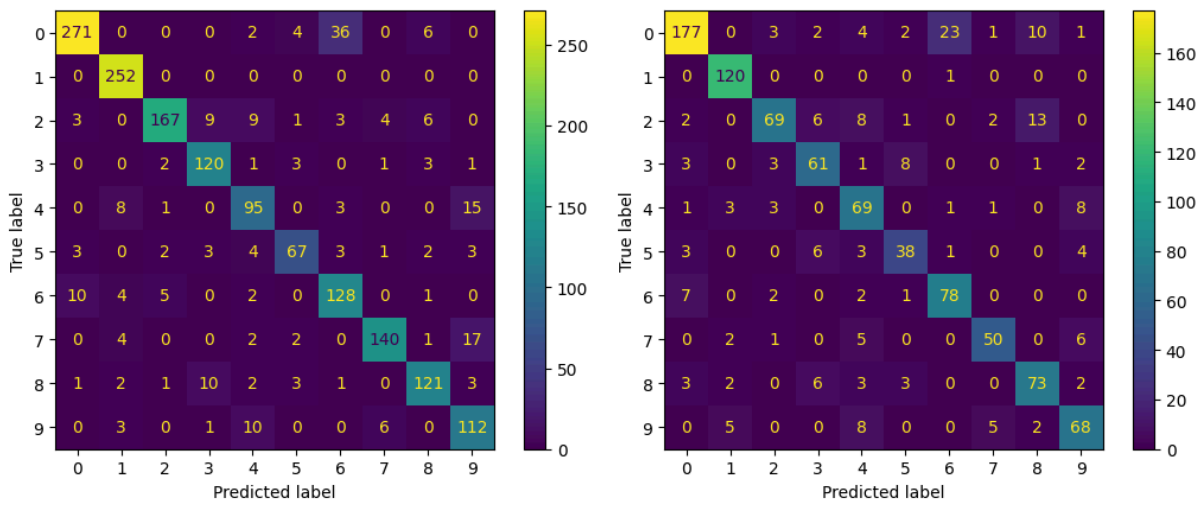


Figure 4: T-SNE results

Task 1.3

We created a `nearest mean classifier` form scratch both on the train and test set: this method is based on evaluation of the L2 norm (Euclidean distance) between the different samples and the center dataset (ten vectors of 256 dimension made in the Task 1.1). Looking at the confusion matrix of the train set in Figure 5, we can see that the most problematic digits to classify are 0 and 6 in both test and train set. There are also some misclassification between the classes 7 and 9 but not as many as we could expect, given the low distance between their centers in the distance matrix.
The results are very similar between the two sets, with an accuracy of 0.86% for the train set and 0.80% for the test set: this is due to the different sample size of the train and the test set (1706 and 999 respectively).



a) train set b) test set

Figure 5: Confusion Matrices

Task 1.4

As last subtask we implemented a `KNN (K-Nearest-Neighbor)` classifier. We iterated over different values of `k` (from 1 to 9) to find how the accuracy changes. We obtained the best results with `k=1`: this happened because low values of `k` capture more details in the data, but it can also lead to overfitting.
We obtained the same accuracy, for both train and test set, as the `nearest mean classifier`, probably because `KNN` without tuning the hyperparameters `metric` and `p` use the L2 norm as well. This explains why the results are the same, as is possible to see in the confusion matrices in Figure 5.

Task 2: implement a multi-class perceptron algorithm

Task 2 required us to implement a multi-class perceptron from scratch in order to classify the same MINST dataset.
We have created the class `Perceptron` with attributes such as learning rate, number of epochs, weight and number of final classes, so it possible to use this class for other datasets with different number of labels.
As methods we have implemented the `fit` function used to train the model, the `predict` method used to predict the labels using the trained weights and `accuracy` to calculate, as the name suggests, the accuracy of the predicted classes.

Fit method

The `fit` function takes as input the training data and the corresponding labels. First we initialize the weights, with values between 0 and 0.01, using the same shape of the input `X` data.
Then we iterate with two loops, first over each epoch, and then for every samples in the training set. We implemented the forward pass using `z = x @ self.W + self.b` to perform the dot product between the input and the weights and adding the bias, and then we used the `argmax` to obtain the most probable class. In this iteration we also calculate the if the prediction is wrong, useful to update the weights and bias, and also the loss (MSE).
For the weights and bias update we used another loop to iterate over each class. We adjust the bias as `bias = bias + learning_rate * error` and the weights as `weights = weights + learning_rate * error * x`.

Results

We trained the model using different learning rates, specifically 0.001, 0.0001 and 0.00001, with 70 epochs and for each of them we plotted the loss curve and calculated the accuracy. In particular, we obtained the best result with a learning rate of 0.0001, reaching, as is possible to see in the Figure 6 and 7, an accuracy of almost 0.95% in the `train_set` where in the `test_set` the accuracy is 0.87%.

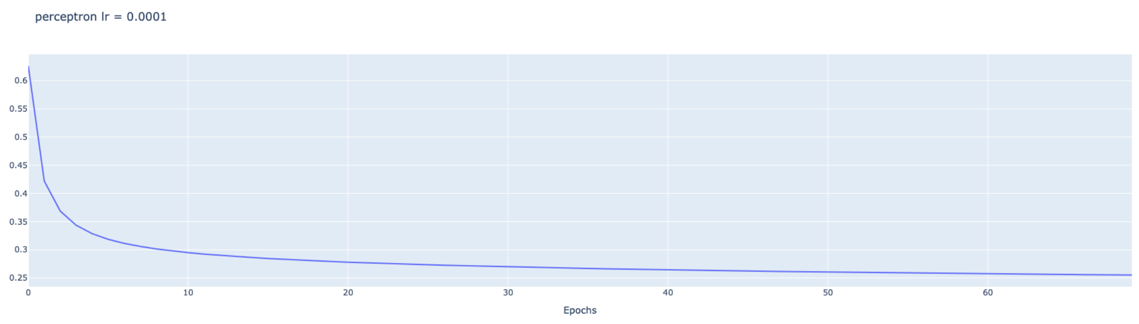


Figure 6: Training loss during the epochs

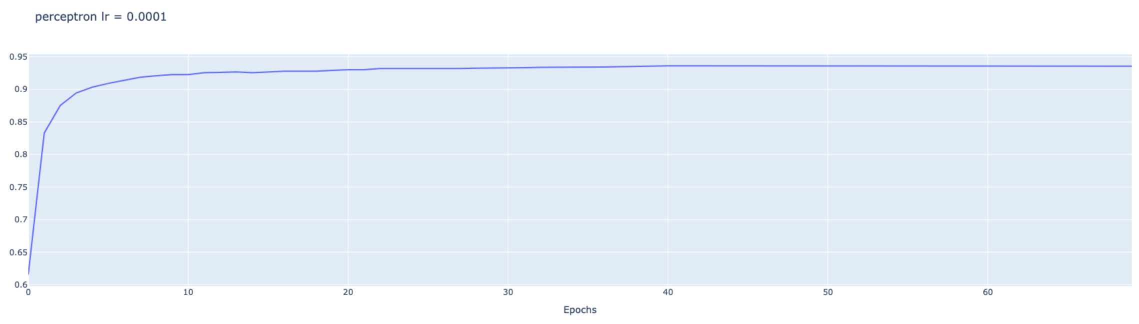


Figure 7: Training accuracy during the epochs

To achieve better results it is necessary to create a more complex Neural Network, with more layers in order to capture more details in the data. A single perceptron is not enough, but an accuracy of 0.87% in the `test_set`, is better than the results obtained in Task 1.3 and 1.4 with the `nearest mean classifier` and `KNN`.

Contributions

Student	**Task 1**	**Task 2**	**Task 3**
Filippo Bomben	Code and Report	Code and Report	/
Sara Casagrande	Code and Report	Report	/
Gian Marco Tomietto	Report	Code and Report	/