Intro to Deep Learning — Assignment 0

Distance-based Classifiers and a Multi-class Perceptron (from Scratch)

Luc van Driel (s3774740)
Rwik Kamilya (s4822285)
Joris Lans (s2978466)

October 1, 2025

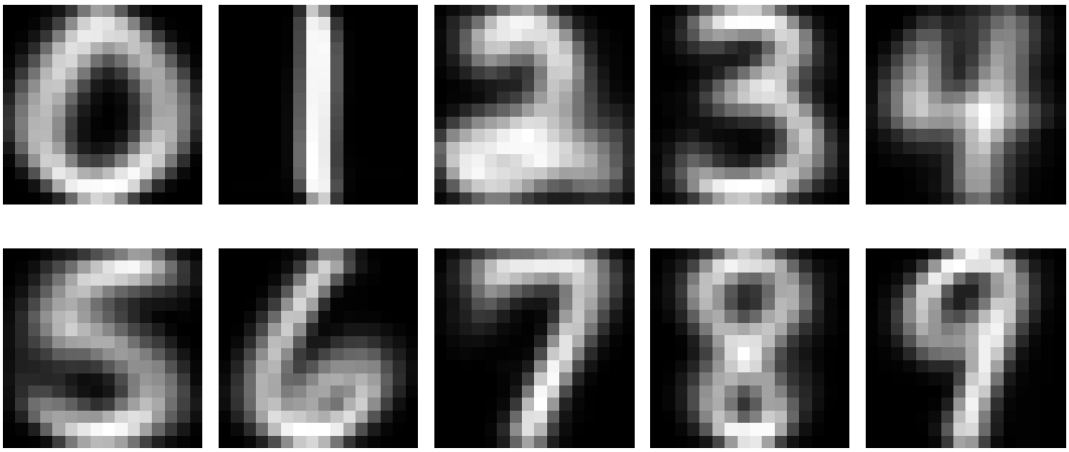


Figure 1: Average image per digit of MNIST data.

1 Introduction

This report captures our experiments with some of the building blocks of machine learning and deep learning. Our aim was to dive into image classification on a simplified MNIST dataset and to design neural networks from scratch. Along the way, we explored dimensionality reduction, distance-based classifiers, and a multi-class perceptron. These tasks gave us an insight into how high-dimensional data behaves, how models learn, and how optimization shapes performance.

2 Method

Data - We used the provided simplified MNIST dataset of 2707 grayscale 16×16 digit images, split into 1707 training and 1000 test samples.

2.1 Task 1: Dimensionality & Distance-based Classifiers

Class centers and pairwise distances We considered for each digit $d \in \{0, \ldots, 9\}$, a cloud of points in 256-dimensional space, C_d, which consists of all training images (vectors) that represent d. For each cloud C_d we calculated a 256-dimensional vector of means over all coordinates of vectors that belong to C_d. We also calculated the distances between the centers of the 10 clouds,

$$D_{ij} \;=\; \text{dist}(\mathbf{c}_i, \mathbf{c}_j) \text{ , for } i, j \in \{0, \ldots, 9\}$$

2-D visualizations We experimented with dimensionality reduction techniques. Specifically, Principal Component Analysis (PCA), Uniform Manifold Approximation and Projection (UMAP), and t-distributed Stochastic Neighbor Embedding (t-SNE).

Nearest Mean and KNN Classifiers We built a nearest-mean classifier from scratch, based on the digit mean values discussed in section 2.1. We used scikit-learn to implement the same nearest-mean classifier and KNN classifier. Then we trained and tested the classifiers on the train and test data. We generated a confusion matrix for the test results of both classifiers.

2.2 Task 2: Multi-class Perceptron (from scratch)

We implemented a multi-class perceptron training algorithm, from scratch. We looped over the training data each epoch and updated the parameters of the perceptron. The parameters were a (256, 10) matrix of weights and a (10,) matrix of biases. We used the weight update rules, found in Chapter 6.1, equations 6.3 and 6.7 of Understanding Deep Learning textbook.

$$\nabla_{\boldsymbol{\phi}} \ell_i = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2\big(\phi_0 + \phi_1 x_i - y_i\big) \\ 2x_i\big(\phi_0 + \phi_1 x_i - y_i\big) \end{bmatrix}$$

We then tried to limit the number of loops by employing matrix multiplication to handle all inputs in parallel. We added a bias vector of 1's to the input matrix making it a $(257 \times \text{number of samples})$ matrix. We experimented with different weight initialization strategies, epochs and learning rates to evaluate their effects and compare both approaches. We sweep learning rates (e.g., $\eta \in \{0.01, 0.1, 0.9\}$), epochs, and weight initializations (zero, normal, uniform) and repeat runs to gauge reliability.

3 Results and Observations

3.1 Task 1: Dimensionality & Distance-based Classifiers

3.1.1 Mean digits and distance analysis.

As expected, the mean cloud (figure 1) shows that the mean of the pixel values for each digit produces an image that is also the visual 'average' 16×16 representation of that handwritten digit. Based on these images, we can only guess which digits would be easier or harder to separate for any ML algorithm. To get a better idea, we can take a look at table 1, where we inspect a distance matrix between each digit pair.

	0	1	2	3	4	5	6	7	8	9
0	**0.00**	14.45	9.33	9.14	10.77	7.52	8.15	11.86	9.91	11.49
1	14.45	**0.00**	10.13	11.73	10.17	11.12	10.61	10.74	10.09	9.93
2	9.33	10.13	**0.00**	8.18	7.93	7.91	7.33	8.87	7.08	8.89
3	9.14	11.73	8.18	**0.00**	9.09	6.12	9.30	8.92	7.02	8.35
4	10.77	10.17	7.93	9.09	**0.00**	8.00	8.78	7.58	7.38	6.01
5	7.52	11.12	7.91	6.12	8.00	**0.00**	6.70	9.21	6.97	8.26
6	8.15	10.61	7.33	9.30	8.78	6.70	**0.00**	10.89	8.59	10.44
7	11.86	10.74	8.87	8.92	7.58	9.21	10.89	**0.00**	8.47	5.43
8	9.91	10.09	7.08	7.02	7.38	6.97	8.59	8.47	**0.00**	6.40
9	11.49	9.93	8.89	8.35	6.01	8.26	10.44	5.43	6.40	**0.00**

Table 1: Heatmap table containing the distances of every average distance between MNIST digits (yellow = small distance, red = large distance).

Table 1 shows a heatmap of the distance between average digit pairs. It suggests that the digits 0 and 1 should be the easiest to separate by any algorithm, based on the fact that they are furthest away from each other in the 256-dimensional space. However, this table uses averages and does not account for variance. We cannot be completely certain that these two digits would be easiest to differentiate, because there might be outliers.

Digit Pair	Distance
0 – 5	7.52
1 – 9	9.93
2 – 8	7.08
3 – 5	6.12
4 – 9	6.01
5 – 3	6.12
6 – 5	6.70
7 – 9	5.43
8 – 9	6.40
9 – 7	5.43

Table 2: Minimum distances between each digit and its closest neighbor.

For a better overview, we inspected a minimum distance matrix (table 2) for each digit and the distance from its closest neighbor. Here, we can get an idea which digits would be hardest to separate if we are assuming uniform variance between classes. That means in practice that 7 and 9 might not necessarily be the most difficult to separate if their clusters were relatively small due to little variation between instances. Therefore, we cannot conclude anything about the separability with absolute certainty. However, we can say that digits 7 and 9 are most similar to each other on average when compared to all other digit pairs.

3.1.2 2-D visualizations.

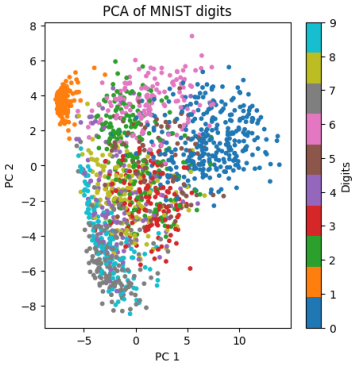


Figure 2: PCA

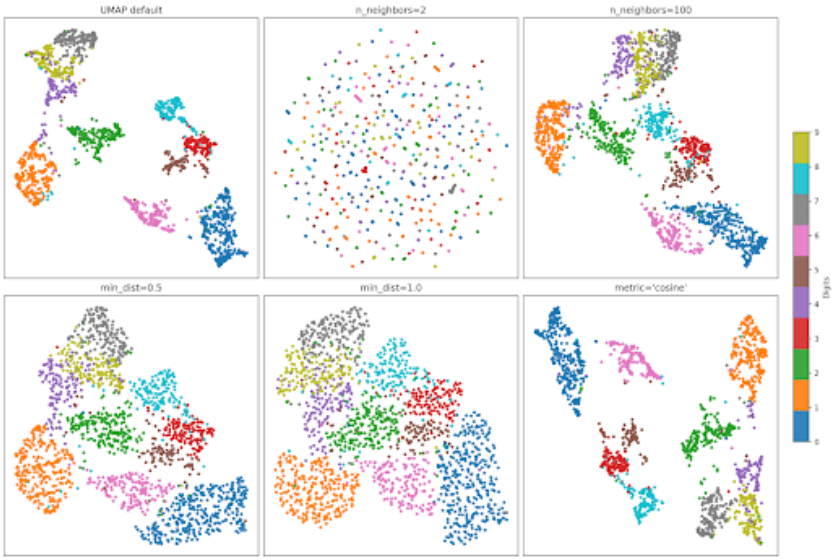


Figure 3: UMAP with varying parameters

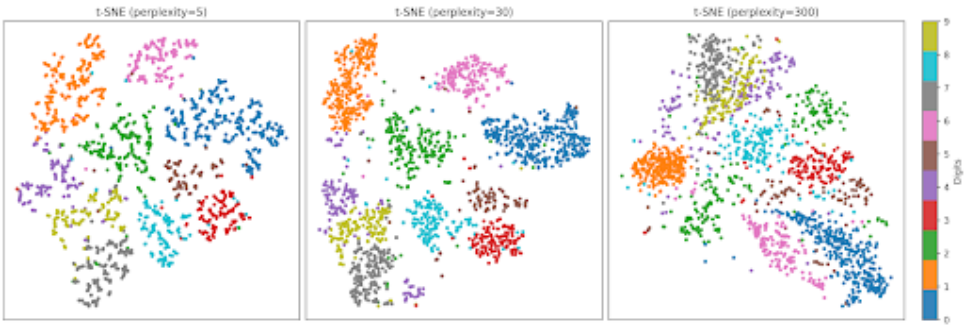


Figure 4: t-SNE with varying perplexity

The PCA plot (figure 2) shows the least separation between classes and looks quite messy. However, digit 1 does show clear separation, which is to be expected following the distance matrix. Likewise, digits with a small distance between them seem to overlap more on the plot as well, 7 and 9 for example.

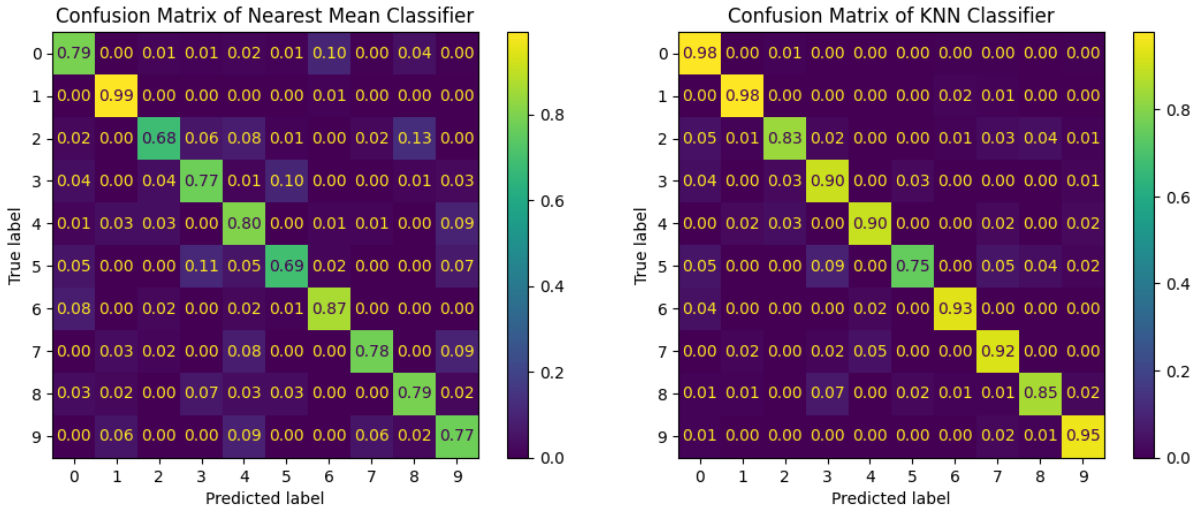
For UMAP and t-SNE we generated a number of plots with varying parameters, which show to make significant difference. In contrast to PCA, the t-SNE projection (figure 4) shows well-formed clusters and a clear separation between digits when using moderate perplexity. The projection resembles the results of the distance matrix; digits with a short distance between them also lay close together on the plot. For example, 9 is surrounded by 4, 7 and 8, which are also the digits with the smallest distance in the matrix. Lower perplexity fragmented the digits into too many small groups and higher perplexity blurred boundaries across classes.

UMAP (figure 3) provided the most flexible control: small `n_neighbors` values emphasized very local structure, larger values emphasized global organization, and `min_dist` directly controlled how tight the clusters are. Changing the metric to cosine slightly reshaped cluster boundaries, highlighting how handwritten stroke directions affect clustering. Compared to the other two projections, the UMAP projection bares less resemblance with the distance matrix. Most digits with a small distance in the matrix are still close together on the plot, but there

are some exceptions. For example, 8 has the smallest distance to 9 in the distance matrix but are quite far apart on the plot. Overall, both UMAP and t-SNE produced much clearer digit separation than PCA, but required careful parameter tuning to balance local detail with global structure.

3.1.3 Nearest Mean and KNN Classifiers

The nearest mean classifier we built from scratch achieved a train and test accuracy of 86.35% and 80.4% respectively. The KNN classifier scores an accuracy of 100% and 91.5% on the train and test set respectively.



(a) Nearest-Mean.

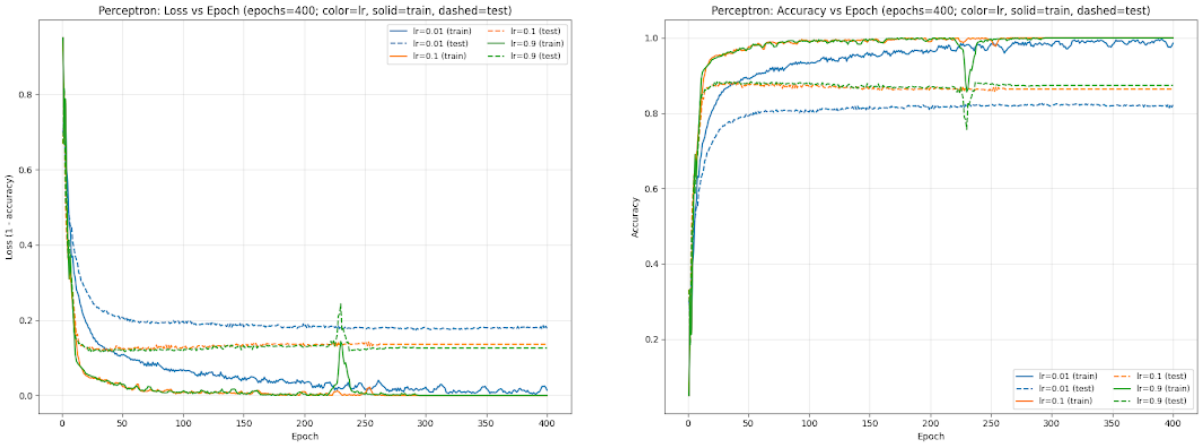
(b) KNN (`n_neighbors = 1`).

Figure 5: Confusion matrices (normalized).

The confusion matrices (figure 5) show that for both classifiers, digit 5 is one of the most difficult digits to classify accurately. This is to be expected, because this digit has a short distance to many of the other digits, as can be seen in table 2. Notably, the KNN classifier performs a lot better at classifying 0's and 9's compared to nearest mean, with an increase in accuracy of 19 and 18 percentage points respectively. Overall, the confusion matrices resemble the distance matrix of table 1 quite accurately, where digit pairs with small distance also get misclassified more frequently. This means that in this case the class distance matrix can be used to predict classification accuracy of distance-based classifiers.

3.2 Task 2: Multi-class Perceptron (from scratch)

We trained a single-layer multi-class perceptron from scratch on the simplified MNIST dataset. The training procedure updated the weights using either per-example loops or fully vectorized matrix multiplication. In both cases, we tracked accuracy and loss on the training and test sets across epochs, as shown in figure 6 below.



(a) Perceptron Loss vs Epoch

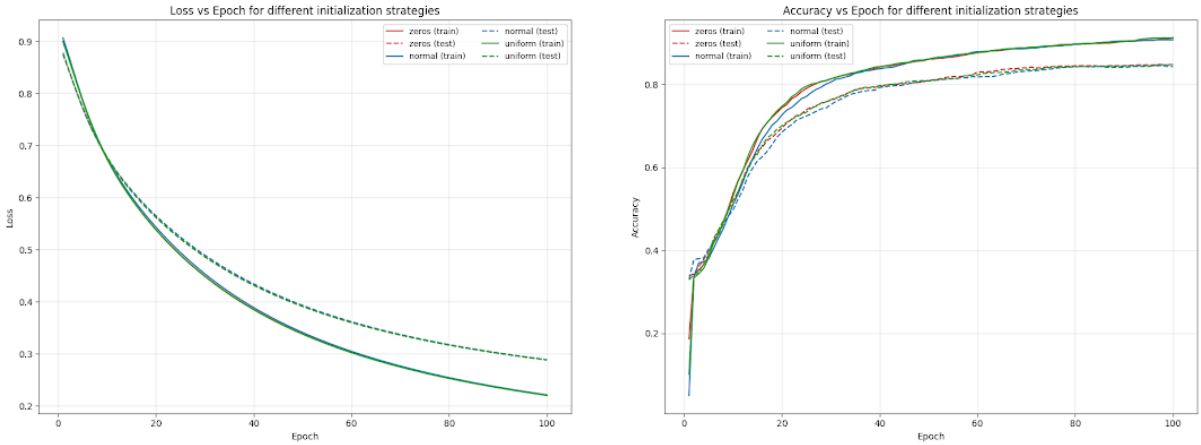
(b) Perceptron Accuracy vs Epoch

Figure 6: Perceptron loss and accuracy during training at different learning rates.

Performance Across experiments, the perceptron converged reliably on the training set, with a training accuracy of around 95–100% depending on the learning rate. However, test accuracy never improved from the 85–90% range (achieving a highest accuracy of 90.3% at $lr = 0.1$), showing that while the perceptron is powerful enough to separate the linearly separable training data, it misclassifies unseen digits.

Learning rate As can be seen in figure 6, learning rate had a major influence on training dynamics. It is basically a trade-off between speed and stability. Very small learning rates waste epochs, while very large ones can destabilize optimization.

- **Low learning rate** ($lr = 0.01$) - slow convergence, smoother curves, low accuracy in limited epochs.
- **Moderate learning rate** ($lr = 0.1$) - fast and stable convergence, high train and test accuracy.
- **High learning rate** ($lr = 0.9$) - very rapid early convergence but with oscillations in both accuracy and loss (figure 6).



(a) Loss with varying initializations

(b) Accuracy with varying initializations

Figure 7: Perceptron loss and accuracy during training for different initialization strategies.

Initialization In figure 7 we compare zero, normal, and uniform initializations. In this single-layer perceptron, all three produced nearly identical learning curves, train/test accuracy and loss overlapped for the full 100 epochs, with final test accuracy differences within $\sim$ 1–2%. As expected in this setup, zero initialization does not affect a linear model, because class labels update parameters to distinct gradients at the first step, avoiding symmetry. If the setup had a hidden layer, the neurons in this hidden layer could have imitated each other, as it would have similar forward results and backward updates.

Computational efficiency We compared the loop-based update approach with the vectorized matrix multiplication implementation. They produced identical learning curves, but the vectorized version was significantly faster. This utilized the efficiency advantage of using numpy matrix operations in parallel over loops in Python.

3.2.1 Perceptron vs Distance Based Classifiers

Method	Test Accuracy
Nearest-Mean	80.4%
k-NN (k=5)	91.5%
Multi-class Perceptron	90.3%

Table 3: Best test accuracies on the simplified MNIST test set.

While the perceptron benefits from its ability to learn weights rather than relying solely on distance to class centers, its best accuracy of 90.3% is very close to, but still slightly below, the 91.5% achieved by k-NN (table 3). This exposes the limitations of linear classifiers, which cannot fully separate overlapping digit classes. Overall, the perceptron generalizes better than the nearest-mean classifier and approaches k-NN performance, while being more efficient once trained.

4 Conclusion

In these experiments, we compared distance-based classification methods with a perceptron trained from scratch on a simplified MNIST dataset. The nearest-mean classifier achieved 80.40% test accuracy, while k-NN achieved 91.5%. Visualizations with PCA, t-SNE, and UMAP highlighted why certain digits overlap, with pairs like 7 and 9 proving particularly difficult to separate. These results showed that while distance-based classifiers are simple and interpretable, their accuracy is constrained by overlapping class distributions. The multi-class perceptron achieved a best test accuracy of 90.3% (with learning rate 0.1), improving upon nearest-mean and coming close to k-NN. Learning rate had the largest impact on training dynamics, while weight initialization mattered little in this shallow setting. Vectorized training produced identical results to loops but ran far more efficiently. From distance-based classifiers to perceptrons, our experiments show that learning weights improve accuracy, but true breakthroughs in performance require deeper, nonlinear models.

Contributions

- **Luc van Driel** - Code: Task 1 Part 1, Part 3; Report: Task 1 Part 1, Part 3.
- **Rwik Kamilya** - Code: Task 1 Part 2, Task 2 Part 1,2; Report: Task 1 Part 2, Task 2 Part 1,2,3: Introduction, Conclusion.
- **Joris Lans** - Code: Task 1 Part 4; Report: Formatting in Latex, Distance-based classifiers.