# Assignment 0
Introduction to Deep Learning

Coutinho, Filipe s3801454
Moghadasi, Nastaran s3994910
Shen, Yan s3798801

September 2023

## 1 Task 1

### 1.1 Our Work

The central idea of Task 1 revolves around distance-based classifiers. In step 1, our team aimed to find the cloud center for each digit. We used **np.where** to find the indices in the train-in data where train-out data matches a digit (0-9), resulting in matrices for each digit. The centroids for each digit were calculated using **np.mean**. During this process, the **np.linalg.norm** method was used to compute the distances. Based on the distances between the centroids, this led us to believe that it is possible, to some extent, to distinguish between digits 0-9. However, when the distances between centroids of two classes are too close, it can lead to decreased accuracy. For instance, the distance between the centroids of digits 7 and 9 is only 5.426474, the smallest among all pairwise centroid comparisons in the dataset.

In step 2 of Task 1, we explored dimensionality reduction algorithms, including PCA, U-MAP, and T-SNE, to visualize the distances between digits in 2D. From Figure 1, we observed that PCA partially aligned with our expectations, with some overlap between digits 9 and 7. Additionally, digits 4 and 9 appeared close, consistent with the earlier distance calculation where their distance was 6.010408. Digit 1, on the other hand, was quite distant from all other digits. However, it was evident that many digits still overlapped, such as 8 and 4, although not as closely as 9 and 7.

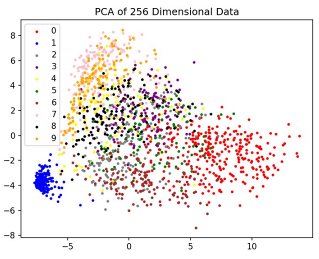

Figure 1: PCA

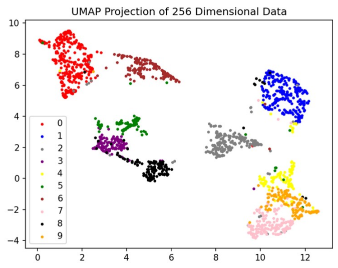

Figure 2: UMAP

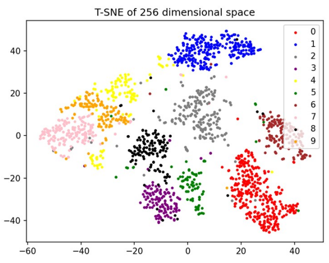

Figure 3: T-SNE

The Figure 2 vividly visualized the relationships between digits 4, 9, and 7. However, it indicated that the distance between 0 and 3 was too close. The Figure 3 appeared slightly more clustered compared to U-MAP but effectively captured relationships between digits 4, 9, and 7. It also suggested that 8 was closer to 5 (distance 6.967386) than to 6 (distance 8.587222). Thus, T-SNE had a closer resemblance to the matrix distance.

To complete steps 3 and 4 of Task 1, we calculated the distances between each input and each centroid. We identified the nearest centroid for each input using **np.argmin**, resulting in predictions for every input. Comparing these predictions with true values (from the train-out or test-out sets) allowed us to calculate accuracy. KNN is also a distance-based classification algorithm, but it calculates distances to custom-defined neighboring nodes to determine the closest type. The key parameter, K, in KNN, was optimized by testing various values. Smaller K values were preferred due to their higher accuracy, but K=1 could introduce noise due to a single reference node. Through experiments, we found that KNN achieved higher accuracy on both training and test sets. Smaller K values led to higher accuracy, resulting in a greater difference in accuracy compared to NM. Analyzing the results, NM struggled more with distinguishing between 4 and 9, 0 and 6, as well as 7 and 9. On the other hand, KNN faced difficulty in distinguishing between 4 and 9 (15% misclassification) and between 4 and 1 (12% misclassification).

### 1.2 Conclusions

In conclusion, Task 1 involved the creation of a model to classify hand-written digits, focusing on distance-based classifiers and dimensionality reduction. Visualizations and comparisons of these algorithms provided valuable insights into the dataset's characteristics and the algorithms' performance.

## 2 Task 2

### 2.1 Objective

The primary goal of Task 2 is to implement a multi-class perceptron training algorithm. We evaluated this network on both the training and testing datasets.

### 2.2 Methodology and Data

The network features a single-layer perceptron with 10 neurons (representing digits 0-9), an input layer with 257 neurons (256 for data, 1 for bias), and a 10-neuron output layer. As a note the input matrix (X train plus bias) is the shape of (1702, 257), and the output matrix is (1702, 10), indicating instances and nodes respectively.

### 2.3 Algorithm Design and Implementation

- **Learning algorithm**: The perceptron learning algorithm is employed for this network.
- **Activation function**: Sigmoid activation function is used to squash the output between 0 and 1 for the perceptron layer. We also needed the derivative of the sigmoid function in the backpropagation process.
- **Weight Initialization**: The perceptron's weights are initialized with small random values close to zero to achieve a more effective learning process by using a standard Gaussian distribution (mean=0, $\sigma = 1$), resulting in a weight matrix of size (257, 10).
- **Training perceptron**: During training, a feed-forward pass determines the initial outputs, and improvements were made using the backpropagation technique.
- **Accuracy calculation**: After every ten epochs, the accuracy of the model is computed for both the training and test sets.

### 2.4 Results

Executing the algorithm at a learning rate of 0.1 for 100 epochs, we recorded accuracy for both training and test sets; for instance, at Epoch 0, the training accuracy was approximately 85.41% and the test accuracy was 78.8%; by Epoch 90, these values rose to 98.59% and 89.2%, respectively.

### 2.5 Comparative with distance based algorithms

The comparison made with the Nearest Mean and KNN classifier. Nearest Mean classifier yielded 86.35% (training) and 80.4% (test) accuracies, while KNN produced 88.99% (training) and 84.4% (test). The perceptron approaches 99% training accuracy by Epoch 90 and 89.2% on the test, but the gap hints at mild overfitting, suggesting techniques like early stopping or regularization.

### 2.6 Conclusions

The single-layer perceptron, given enough training, showcases superior performance in comparison to the distance-based methods used in task 1. Its iterative learning mechanism allows it to capture complex patterns, making it a valuable tool for classification.

## 3 Task 3

### 3.1 Objective

Task 3 aims to build a custom multi-layer perceptron neural network and evaluate its performance across various configurations, such as learning rate, activation function, and weight initialization methods. Successful convergence is defined as achieving a loss value of 0, within a maximum of 10000 training epochs for the neural network.

### 3.2 Sigmoid Activation

Through various experiments, we found that using the `sigmoid` activation function with random weights initialized between $[0, 1]$, a learning rate of 0.2 provides the fastest and most consistent convergence, plateauing around 6000 epochs. A learning rate of 0.1 yields similar results but with slightly slower convergence compared to 0.2. However, 0.2 achieves lower loss and converges twice as fast with the `sigmoid` function. With a learning rate of 0.01, there's an initial rapid loss decrease, but subsequent convergence slows significantly, reaching only around 0.10 loss after 10,000 epochs.

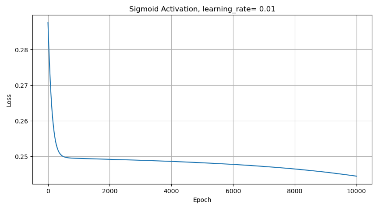

Figure 4: Learning Rate: 0.01

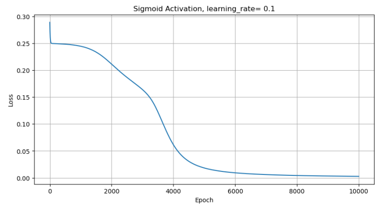

Figure 5: Learning Rate: 0.1

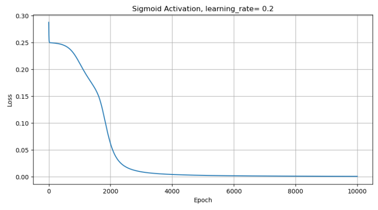

Figure 6: Learning Rate: 0.2

### 3.3 TanH Activation

When we examine the loss of the `tanh` function in comparison to the `sigmoid` function, it becomes apparent that it generally exhibits significantly quicker convergence. Once more, the fastest convergence was observed in the model employing a learning rate of 0.2, followed by 0.1, with 0.01 yielding the slowest convergence.

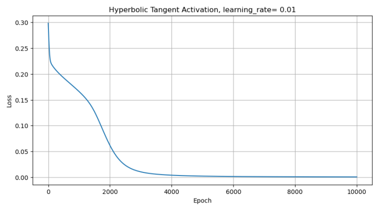

Figure 7: Learning Rate: 0.01

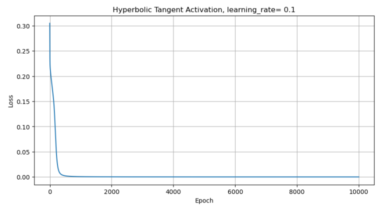

Figure 8: Learning Rate: 0.1

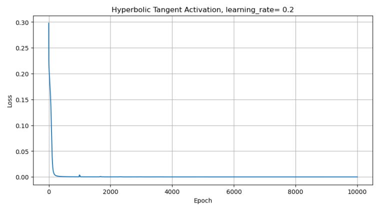

Figure 9: Learning Rate: 0.2

### 3.4 ReLU Activation

Upon analyzing the loss associated with the `relu` function, it becomes evident that this function exhibits the swiftest initial loss reduction. Nevertheless, it consistently maintains a loss level above 0.25. Furthermore, it appears that the learning rate influences the pace at which the loss decreases, but it doesn't appear to have a significant impact on the extent of the loss reduction.

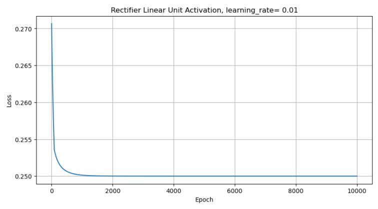

Figure 10: Learning Rate: 0.01

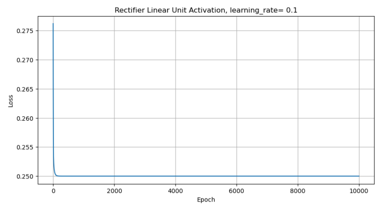

Figure 11: Learning Rate: 0.1

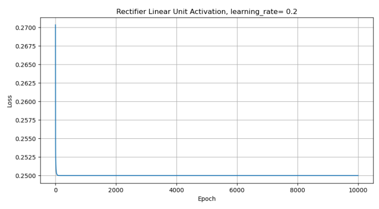

Figure 12: Learning Rate: 0.2

## 3.5 Weight Initialization

We observed that initializing weights using `np.random.randn(9)` outperforms the prior approach using `np.random.uniform(0, 1, (9))`. This enhancement is clearly reflected in the loss plots across all activation functions, namely **`sigmoid`**, **`tanh`**, and **`relu`**. Additionally, this adjustment resolved the previous issue with the latter.

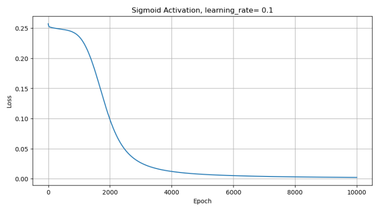

Figure 13: Sigmoid

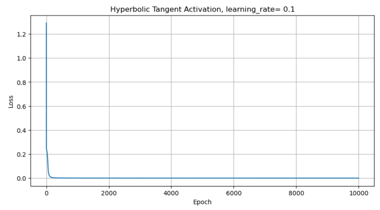

Figure 14: TanH

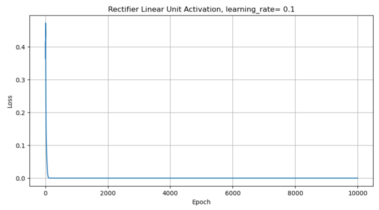

Figure 15: ReLU

## 3.6 Conclusions

In summary, when initializing weights within the range of 0 to 1, a learning rate of 0.2 consistently demonstrates the fastest convergence, irrespective of the selected activation function. As for the choice of activation function, our findings lead us to the conclusion that the tanh function delivers superior results in terms of both minimizing loss and accelerating convergence.