IDL Assignment 1

Egecan Karakas Ansuman Sahu

October 2021

1 Introduction

In this assignment we worked on a simplified version of MNIST data set: a collection of 2707 digits represented by vectors of 256 numbers that represent 16x16 images. For task 1 and task 2, first 1707 images used in training, where as last 1000 images used for testing purposes. The output was class prediction for hand-written digits of 0-9. For the third task, we worked on XOR problem which is not solvable by Single Layer Perceptrons. Development is done with Python, where we used Google Colab and local Anaconda environment. Mainly, scikit-learn and numpy libraries used for decompositions, manifolds, KNN classifier and efficiency purposes. Matplotlib is also used for visualization.

2 Task 1 - Data Dimensionality, Distance-Based Classifiers

For the first task we introduced a basic approach based on cloud centers in training set. Then Dimensionality Reduction of the data is made to see whether the problem is linearly separable or not and get a better visualization about the separation of problem classes. Afterwards we used already existing KNN (K-Nearest-Neigbor) classifier in Scikit-Learn library.

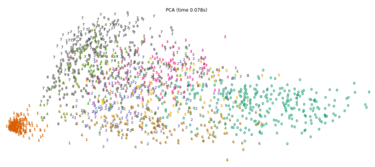
2.1 Question 1

In this part, we calculated data cloud centers of small MNIST data set with 256 dimensions. We also found out that pairs of {7,9}, {4,9}, {3, 5}, {8,9}, {5, 6} are hardest to separate as their cloud centers is closest to each other. Also, {0, 1} pair was the easiest to diversify as their cloud centers are furthest between all pairs. From manifolds in question 2, it is also easy to see that clusters of 0 and 1 locate on the far edges of diagrams in Figure 1.

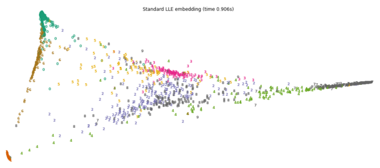
Regarding the accuracy of this simple model, training error rate was %13.65. Although it's not a promising model, it helps to create a acceptance lower-bound for future models

2.2 Question 2

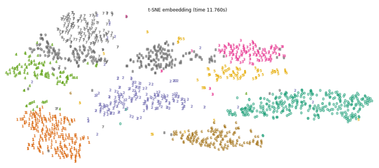
In this part we generated visualizations for dimensionality reduction algorithms of PCA, LLE, and t-SNE. PCA is a deterministic dimensionality reduction algorithm which is also the fastest of all 3 with 0.078s. In this assignment we used PC-1 and PC-2 as the axes, which does not correctly represent all of the variation in the data. LLE finished in 0.906s yet it didn't provide more than PCA. On the other hand, despite its long run time of 11.7s, T-SNE performed the best visually understandable clustering of this linearly separable problem.



(a) PCA



(b) LLE



(c) t-SNE

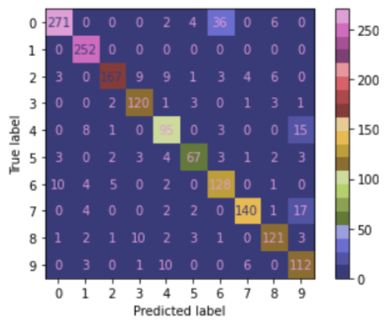
Figure 1: Dimensionality Reduction Algorithms

2.3 Question 3

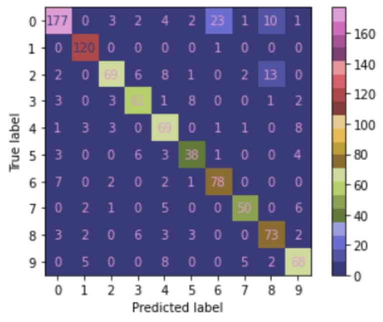
Cloud-centrical model's error rate for training set was 13.65 compared to 19.61 for the test set. As I said before, it can be used as a baseline for future models.

2.4 Question 4

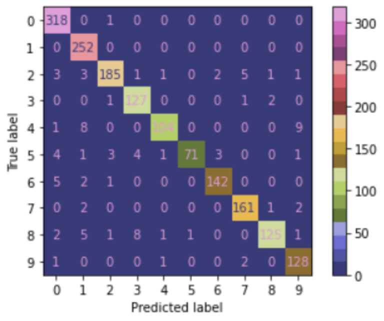
In this part we used the KNN classifier from sklearn library. It brings a less naive approach with 5.45 error rate in train, and 10.71 error rate in test set. Unfortunately, these ratios are still worse compared to best classifiers of KNN with L2 distance at *yann_le_cun*.



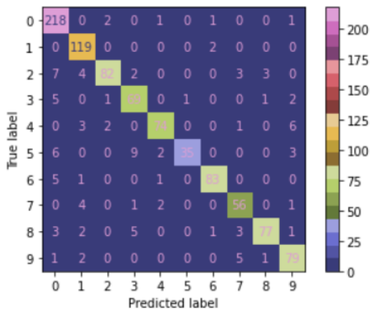
(a) Cloud Train



(b) Cloud Test



(c) KNN Train



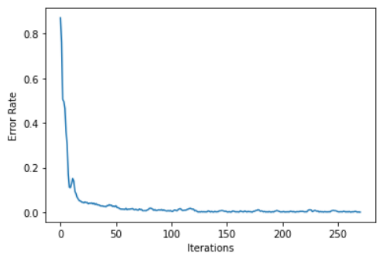
(d) KNN Test

Figure 2: Confusion matrices of the models

Finally, we created confusion matrices for Cloud and KNN models in Figure 2. For cloud model, {0, 6}, {4, 9}, {7, 9} pairs are the most mistaken in train data, and {0, 6}, {0, 8}, {2, 8} in test. Some of these pairs are already present in Question 1 findings. For the missing pairs, their interspersion is noticeable from dimensionality reduction algorithm in Question 2. For instance, even though 0 has a distant cloud center, it has meshed samples with 6 in PCA. Moreover, both Cloud and KNN models seem to have the most difficulty to distinguish 5 from other numbers. This might be due to rarity of 5 in the dataset.

3 Task 2 - Multi-Class Perceptron Algorithm

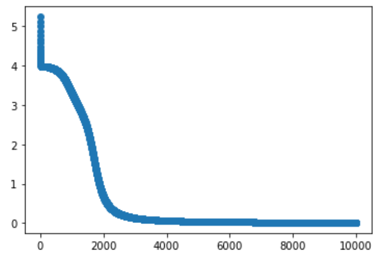
We designed a single layer Multi-Class Perceptron for this part. To do so, we implemented Generalized Perceptron Algorithm (Duda et al.) from Lecture 2. We start with random weights and weight are updated by x values for activated nodes. Correct class's weights increase when it is a false negative, and other classes' weights decrease when they have higher scores (aka. activated). The implementation's error rate is %12.91. It is better than Cloud centrical L2 distance method. However, it is still worse than KNN model. For this problem, there exists too many local minimas. Therefore, it is highly likely for the algorithm to hit and stop on one of these points or clouds. We could increase our success rate through normalizations and better learning algorithms. However, we left those experimentations for the upcoming parts of the lecture. The Figure below shows that error rate quickly decrease on the train data in 271 iterations. At every run, we will have different starting points as initial weights are random and we will hit a different local minimum as there are plenty of those.



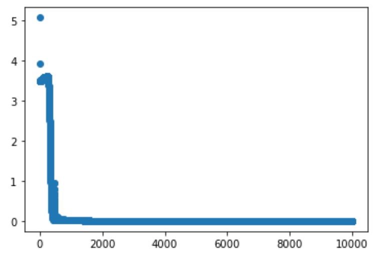
(a) Single Layer Multi Class Perceptron

4 Task 3 - XOR Network and Gradient Descent Algorithm

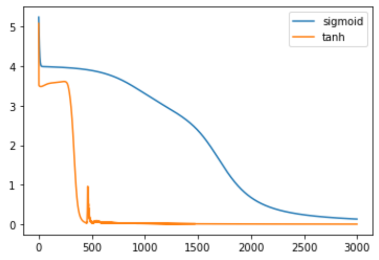
We have implemented the xor problem in neural networks for this task. We start by implementing a xor network with two input nodes, two hidden nodes and one output node along with input vectors {{0,0},{0,1},{1,0},{1,1}} and output targets {0,1,1,0}. After assigning random weights and bias for the hidden and output weights and bias, we use xor network to get the initial output vector. Using a fixed learning rate of 0.1 and sigmoid as the activation function the gradient descent algorithm is implemented, where we keep track of the mean square error which will help us get a better understanding of how quickly the training model converges and number of iterations/epochs required for error to be negligible. We have implemented the same backpropagation model by using hyperbolic tangent as an activation function to compare the mean square error for both training models.The figure below shows the mean square error comparison for xor network using both activation functions. After plotting both the graphs clearly tanh is a better activation function based on the number of epochs (around 1000) required to make the error negligible as compared to that of sigmoid which needed more epochs (around 2000) for the same. The tanh activation works better than sigmoid activation function because the mean of its output is closer to zero rather than sigmoids 0.5, and hence it centers the data better for the next layers.



(b) Sigmoid



(c) Tanh



(d) Mse Comparison for both Activation Functions