Introduction to Deep Learning - Assignment 0

Haoxiang YUAN, Yağmur DOĞAN, Barbaros IŞIK

Task 1: Data dimensionality, distance-based classifiers

To get started in Deep Learning, we did some experiments with the MNIST dataset. The MNIST dataset is a collection of handwritten digits.

Question 1

By calculating the center of each cloud, we obtain a 256-dimensional vector representing the mean coordinates of the images in that cloud. These centers are then used to classify new images based on their distance from the centers and the closest center determines the image's label. When certain pairs of digits have smaller distances, it means that those digits are more difficult to be recognized because their cloud centers are closer.
We can guess that the pairs of digits that are more difficult to be recognized are 0 and 6 or 8, 1 and 7, 8 and 9.

Question 2

To visualize the data, we use the PCA, t-SNE and UMAP methods to reduce the dimensionality of the data to 2 dimensions.

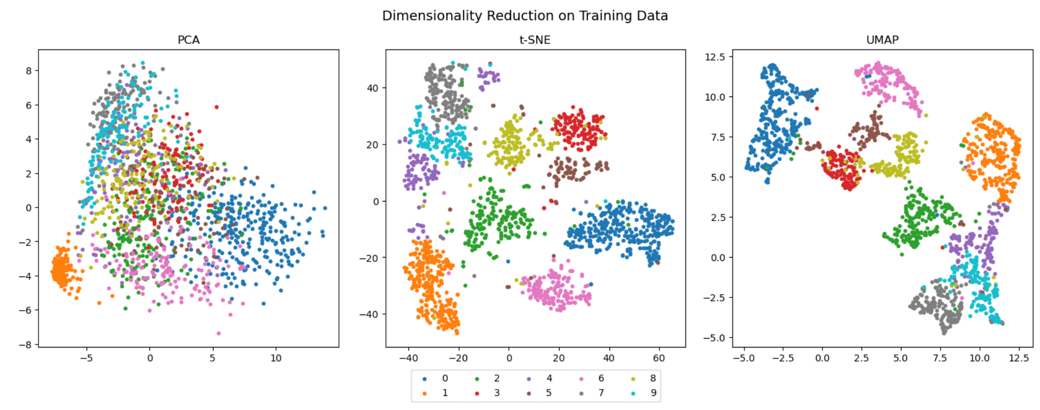


Figure 1: Visualize Reduce the Dimensionality

We can see that PCA is not able to separate the data well. While t-SNE and UMAP perform better than PCA, UMAP is better than t-SNE. Which means that UMAP is better at preserving the global structure of the data.

Question 3 & 4

We implement a simple mean nearest classifier. And applied it to both training set and test set. The accuracy of the training set is 86.35% and the accuracy of the test set is 80.40% KNN is a simple but effective algorithm used for classification tasks. We compared this with the mean nearest classifier. After the evaluation process for both test and train sets, the accuracy output of the training set was 96.60% and the accuracy of the test set was 90.80%.

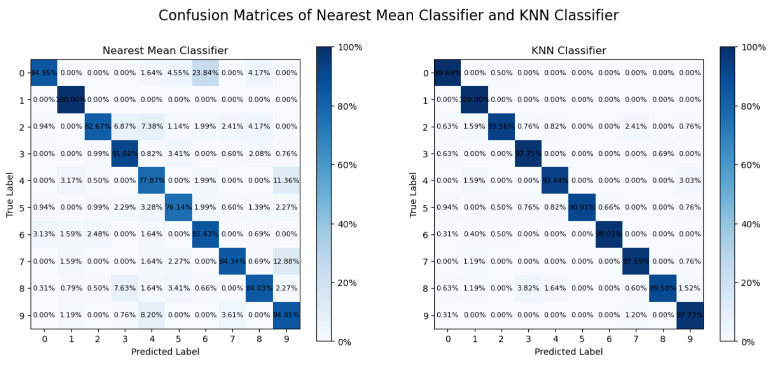
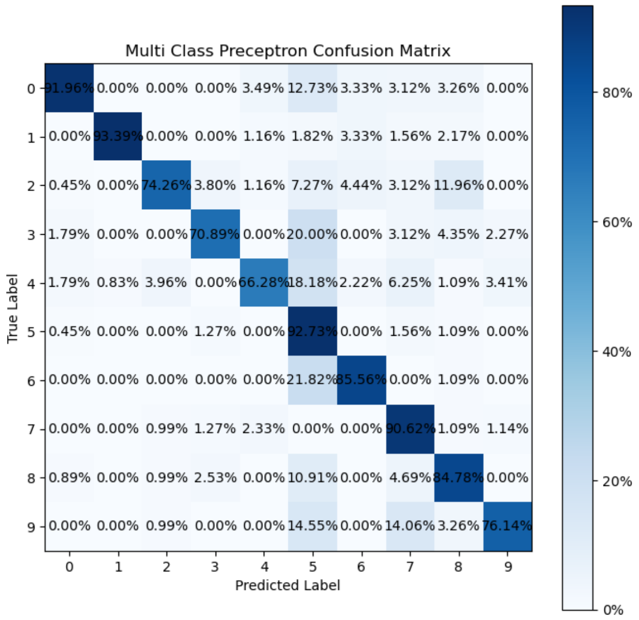


Figure 2: Confusion Matrix of Nearest Mean Classifier & KNN Classifier

Task 2: Implement a multi-class perceptron algorithm

Multi-Class Perceptron is an extension of the perceptron algorithm for binary classification tasks which handles multiple classes. At first we initialized the algorithm with bias and random weight values. Make it predict the class then update weight values, make it predict again and as last step trained & tested the model and get accuracy values of 98.54% for Training set and 85.80% for Test set.
Compare to the KNN algorithm, the accuracy of the training set is higher but the accuracy of the test set is lower. And the confusion matrix of the multi-class perceptron is more balanced than the confusion matrix of the KNN algorithm.



	0	1	2	3	4	5	6	7	8	9
0	91.96%	0.00%	0.00%	0.00%	3.49%	12.73%	3.33%	3.12%	3.26%	0.00%
1	0.00%	93.39%	0.00%	0.00%	1.16%	1.82%	3.33%	1.56%	2.17%	0.00%
2	0.45%	0.00%	74.26%	3.80%	1.16%	7.27%	4.44%	3.12%	11.96%	0.00%
3	1.79%	0.00%	0.00%	70.89%	0.00%	20.00%	0.00%	3.12%	4.35%	2.27%
4	1.79%	0.83%	3.96%	0.00%	66.28%	18.18%	2.22%	6.25%	1.09%	3.41%
5	0.45%	0.00%	0.00%	1.27%	0.00%	92.73%	0.00%	1.56%	1.09%	0.00%
6	0.00%	0.00%	0.00%	0.00%	0.00%	21.82%	85.56%	0.00%	1.09%	0.00%
7	0.00%	0.00%	0.99%	1.27%	2.33%	0.00%	0.00%	90.62%	1.09%	1.14%
8	0.89%	0.00%	0.99%	2.53%	0.00%	10.91%	0.00%	4.69%	84.78%	0.00%
9	0.00%	0.00%	0.99%	0.00%	0.00%	14.55%	0.00%	14.06%	3.26%	76.14%

Figure 3: Confusion Matrix of Multi-Class Perceptron

Task 3: Implement the XOR network and the Gradient Descent Algorithm

We have implemented a neural network from scratch in this task. So for all sub-questions of this task we are using the same neural network architecture. For this we defined a sigmoid, ReLU and tanh (hyperbolic tangent) and their derivative functions.
To update every weights, we implemented the gradient descent algorithm.

$$W = W - \eta \frac{\partial E}{\partial W}$$

Where W is the weight, η is the learning rate and E is the error.

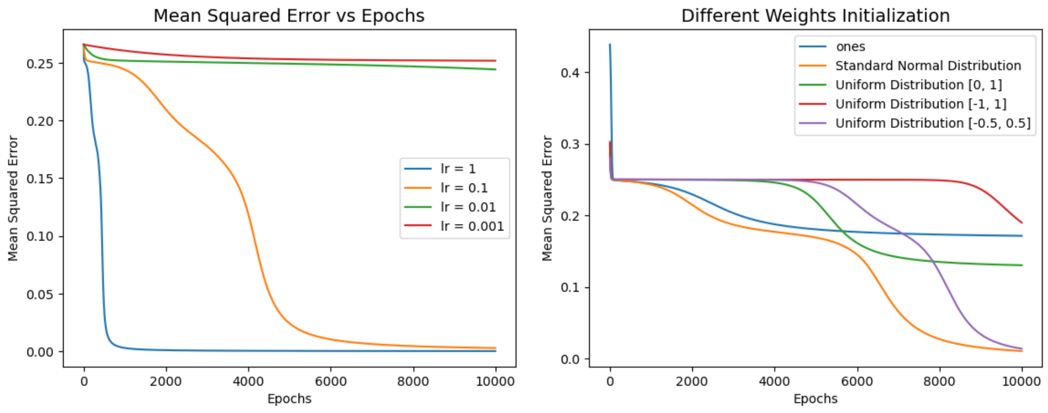


Figure 4: Mean Squared Error & Different Weights Initialization

According to Figure 4, we can see that when the learning rate is 1 or 0.1, the error converges to a value close to 0, while the learning rate is 0.01 or 0.001 the error converges to a value close to 0.25. We can infer that when the learning rate is too small, the error converges to a local minimum.
We tried different ways to initialize the weights. Standard normal distribution has the best performance. It converges to a value near 0 faster, around 10000 epochs. Uniform distribution at [-0.5, 0.5] can also converge to a value near 0, but it takes more epochs than standard normal distribution.
Other methods, such as uniform distribution at [-1, 1], uniform distribution at [0, 1] and constant 1 cannot converge to a value near 0 with 10000 epochs.

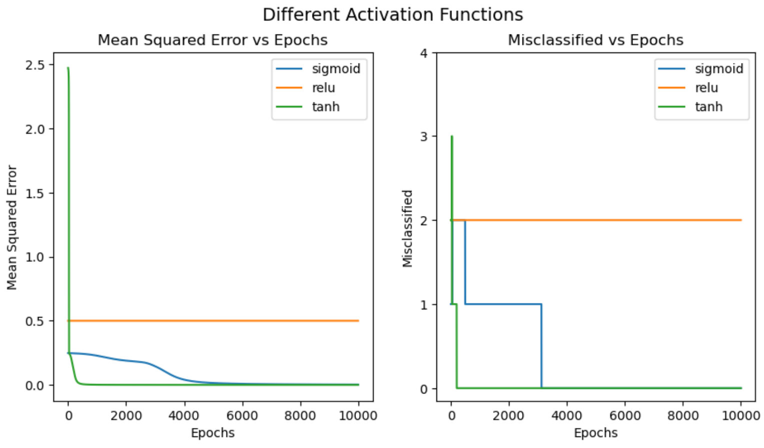


Figure 5: Evaluation of Different Activation Functions

We did some experiments on different activation functions. In Figure 5, we can see that ReLU has the worst performance. It might be because the ReLU function suffers from the problem of vanishing gradient. Tanh has similar performance to Sigmoid. They both converge to a value close to 0. But Tanh converges faster than Sigmoid.

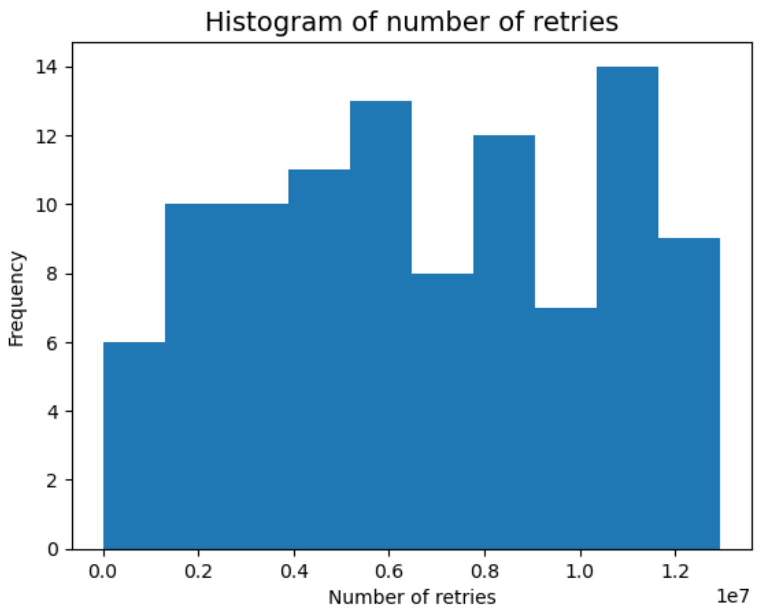


Figure 6: Histogram of Retries

Lastly, we try the "Lazy apporach" by generating a random weight vector and check if the network converges. If not, generate a new random weight vector and try again.
We did 100 experiments and plot the histogram of the number of retries. It takes around 6,500,000 retries to get a network that can make correct predictions.