Introduction to Deep Learning
Assignment 1

Irina-Mona Epure, Ramya T Rameshchandra

1 Task 1: Data dimensionality, distance-based classifiers

In this first task, we developed a very basic distance-based classifier for the MNIST dataset. In order to implement it, we considered a 256-dimensional space in which the training images exist as 256-coordinate objects.

1.1 Finding the center of each class

The classes representing each digit are described as the sets of all training samples belonging to those classes. Thus, all samples labeled with class d (a digit from 0 to 9) are part of a cloud of points C_d. Each cloud has a centre c_d, which is computed as a vector of means of all points belonging to the cloud over all 256 dimensions. The distance-based classifier will classify a new point by measuring the Euclidean distance from it to all the cloud centers and choosing the label of the closest one.

We can thus derive some information about how difficult it will be for the distance-based classifier to distinguish between certain classes by looking at the distance between the centres of these classes, which are shown in 1. Some of the difficult to distinguish digit pairs are the following: 7 and 9, 4 and 9, 3 and 5, 5 and 6, 5 and 8, and 8 and 9. We can expect a higher accuracy when distinguishing between classes whose centers are further apart, for example: 0 and 1, 0 and 7, 0 and 9, 1 and 3, and 1 and 5.

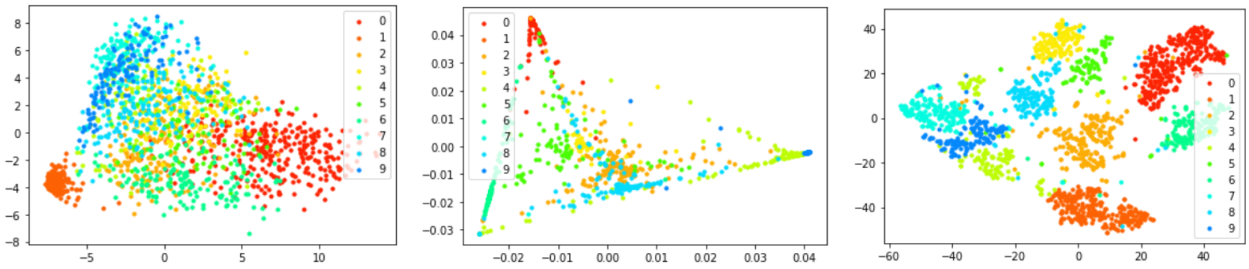
	0	1	2	3	4	5	6	7	8	9
0	0.000000	14.449608	9.334556	9.143734	10.769844	7.519296	8.163019	11.864555	9.907902	11.488875
1	14.449608	0.000000	10.125323	11.733233	10.173786	11.118800	10.597933	10.743154	10.086777	9.932094
2	9.334556	10.125323	0.000000	8.178285	7.932541	7.906796	7.317952	8.872531	7.077516	8.887748
3	9.143734	11.733233	8.178285	0.000000	9.087608	6.118750	9.292426	8.922401	7.020425	8.354350
4	10.769844	10.173786	7.932541	9.087608	0.000000	8.001517	8.765997	7.583012	7.380909	6.010408
5	7.519296	11.118800	7.906796	6.118750	8.001517	0.000000	6.688629	9.211954	6.967386	8.258538
6	8.163019	10.597933	7.317952	9.292426	8.765997	6.688629	0.000000	10.868157	8.570208	10.419744
7	11.864555	10.743154	8.872531	8.922401	7.583012	9.211954	10.868157	0.000000	8.467785	5.426474
8	9.907902	10.086777	7.077516	7.020425	7.380909	6.967386	8.570208	8.467785	0.000000	6.401166
9	11.488875	9.932094	8.887748	8.354350	6.010408	8.258538	10.419744	5.426474	6.401166	0.000000

Figure 1: The Euclidean distances between each pair of centres.

1.2 2-Dimension Visualizations of the Data

We experimented with three dimensionality reduction algorithms to help us visualize the data in a 2-dimensional space. The three algorithms are:

- **Principal Component Analysis (PCA)**, which is used to decompose a multidimensional dataset into the n orthogonal components which explain the most variance in the data [1]. By visualizing the result of this decomposition of the train data in Figure 2a, we can infer the following information: the digit 1 is the most easily distinguishable as it has minimal overlapping with other classes, 0 is easy to differentiate from 7, 8 and 9, and there is a lot of superposition of data labeled with 2, 3, 4, 5, 6, and 7.
- **Locally Linear Embedding (LLE)**, which projects the data on a 2-dimensional space, but preserves distances between local neighbourhoods. Is is implemented as a series of linear PCAs that are compared to find the best non-linear embedding [2]. In Figure 2b, it can be seen that digit 9 is most similar to digit 4, which we also observed by looking at the matrix of distances between the class centres..
- **T-distributed Stochastic Neighbor Embedding (t-SNE)**, which derives joint probabilities from analyzing the similarities between samples. It attempts to reduce the Kullback-Leibler divergence between the original 256-dimensional data and the joint probabilities of the 2-dimensional embedding of the data [3]. In this visaulization in Figure 2c, the classes are separated most clearly from one another and it is easier to observe the relationships between them: class 9 is similar to both class 7 and class 4, 3 is similar to 5 and 8, and the rest of the classes are easier to distinguish, although there are quite a few samples which appear to be in the wrong cluster.



(a) PCA of MNIST dataset. **(b)** LLE of MNIST dataset **(c)** t-SNE of MNIST dataset

Figure 2: 2-dimension visualizations of the classes in the MNIST dataset.

1.3 Implementation and Accuracy of the Distance-Based Classifier

Based on the centres we calculated, we developed a distance-based classifier for new samples. When given a new datapoint, the classifier measures the Euclidean distance from it to all

the centers and returns the label of the closest center. We first tested the performance of the classifier by calculating the percentage of correctly classified samples from the train set (86.34%), and then from the test set (80.38%). Overall, the performance seems quite high, but it is very low compared to the state of the art (99.6% on the test set).

1.4 Comparison to K-Nearest Neighbour

K-Nearest Neighbour (KNN) is a widely used classification algorithm which assigns to an unlabeled datapoint the majority label of the N closest datapoints to it in the train set. By using this method, we obtained an accuracy of 96.6% on the train set and 90.79% on the test set. The confusion matrices of both this method and the distance-based classifier can be seen in Figure 3. By looking at the accuracy scores and confusion matrices of the two classifiers, we can safely say that KNN performs better on both the training and test sets. When looking in closer detail at the confusion matrices for our distance-based classifier, we can see there are some pairs of digits that often raise some difficulties for the classifier, for example: 0 and 6 get confused with one another, and 4 and 7 frequently get misclassified as 9. With KNN, misclassification is less frequent, but it does still happen sometimes in cases such as 2 and 7, 4 and 9, and 8 and 3.

```
[[271   0   0   0   2   4  36   0   6   0]
 [  0 252   0   0   0   0   0   0   0   0]
 [  3   0 167   9   9   1   3   4   6   0]
 [  0   0   2 120   1   3   0   1   3   1]
 [  0   8   1   0  95   0   3   0   0  15]
 [  3   0   2   3   4  67   3   1   2   3]
 [ 10   4   5   0   2   0 128   0   1   0]
 [  0   4   0   0   2   2   0 140   1  17]
 [  1   2   1  10   2   3   1   0 121   3]
 [  0   3   0   1  10   0   0   6   0 112]]
```

(a) Distance-based classifier on train set

```
[[177   0   3   2   4   2  23   1  10   1]
 [  0 120   0   0   0   0   1   0   0   0]
 [  2   0  69   6   8   1   0   2  13   0]
 [  3   0   3  61   1   8   0   0   1   2]
 [  1   3   3   0  69   0   1   1   0   8]
 [  3   0   0   6   3  38   1   0   0   4]
 [  7   0   2   0   2   1  78   0   0   0]
 [  0   2   1   0   5   0   0  50   0   6]
 [  3   2   0   6   3   3   0   0  73   2]
 [  0   5   0   0   8   0   0   5   2  68]]
```

(b) Distance-based classifier on test set

```
[[318   0   1   0   0   0   0   0   0   0]
 [  0 252   0   0   0   0   0   0   0   0]
 [  2   4 189   1   1   0   0   4   0   1]
 [  2   0   0 128   0   0   0   0   1   0]
 [  0   4   0   0 114   0   0   0   0   4]
 [  3   0   1   1   1  80   1   0   0   1]
 [  1   1   1   0   0   0 147   0   0   0]
 [  0   3   0   0   0   0   0 162   0   1]
 [  2   3   0   5   2   0   0   1 129   2]
 [  1   0   0   0   0   0   0   2   0 129]]
```

(c) KNN on train set

```
[[218   0   2   0   1   0   1   0   0   1]
 [  0 119   0   0   0   0   2   0   0   0]
 [  7   2  84   1   0   0   1   3   3   0]
 [  3   0   2  70   0   2   0   0   0   2]
 [  0   2   2   0  79   0   0   1   0   2]
 [  7   1   0   8   1  35   0   0   0   3]
 [  3   1   0   0   1   0  85   0   0   0]
 [  0   4   0   1   2   0   0  56   0   1]
 [  2   2   0   5   0   0   1   1  79   2]
 [  1   0   0   0   0   0   0   4   1  82]]
```

(d) KNN on test set

Figure 3: Confusion matrices of the two classifiers on the train and test sets.

2 Task 2: Implement a multi-class perceptron algorithm

In the second task, we implement a multi-class perceptron with 10 nodes (one per digit), each node having 256+1 inputs (inputs and bias) and one output.

Implementation: The input matrix is a (m,257) matrix where m is the number of training samples. Iterating over each input sample, the weights of the network are updated for each missclassified input. The weights for nodes with activation functions greater than the correct node are reduced while that of the correct node is increased.

Conclusions: The experiment was repeated by iterating over the sample set once and then, 10 times. In the first case, the perceptron gives an accuracy of approximately 89% on training data and 80% on test data. In the case of repeating the experiment for 10 epochs, the perceptron gives an accuracy of approximately 97% on training data and 86% on test data.

The perceptron performs better than the Distance-Based classifier for multiple epochs. The performance of the perceptron and KNN are similar.

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

In this task, we implemented a small neural network from scratch, which takes two bits as input and returns their XOR value. The network consists of 2 hidden nodes and one output node. Each node has three weights, one of which is the weight of the bias. We ran three versions of the network which all use a different activation function of the following: sigmoid, tanh, and ReLU.

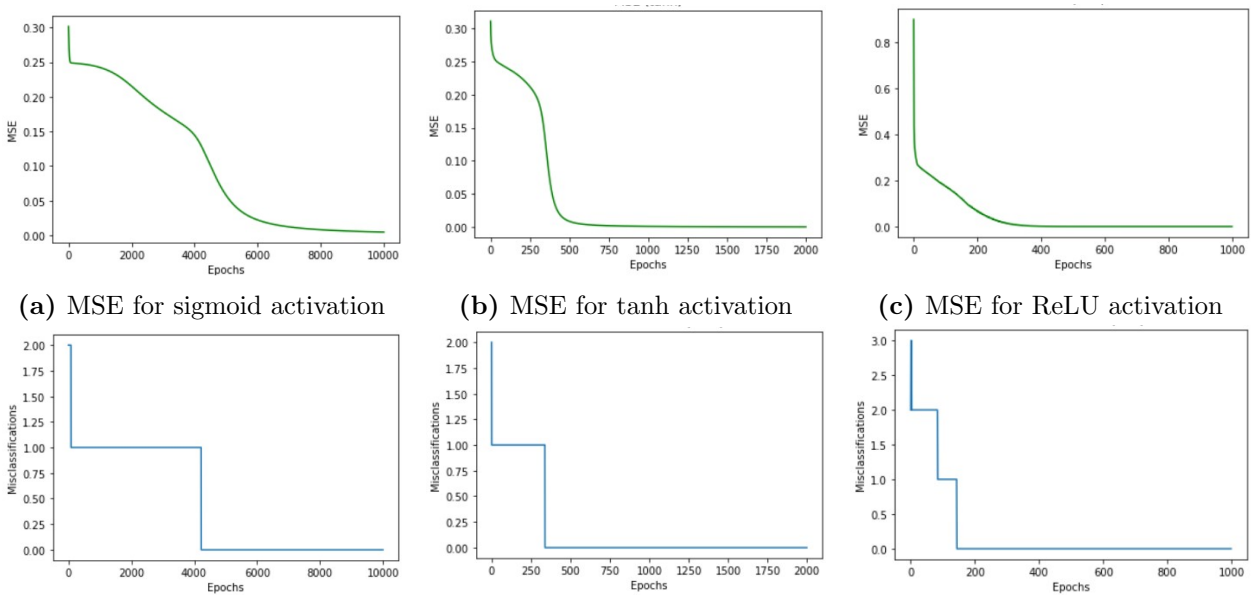
The code we wrote for this task consists of the following functions:

- *xor_net*: computes the output given by the network based on 2 specified inputs, 9 given weights, and the desired activation function (sigmoid, tanh, or ReLU).
- *mse*: loss function that helps us evaluate the accuracy of the network.
- *grdmse*: computes the gradients of the weights based on the MSE loss function. We used *a lot* of Calculus to individually calculate each of the gradients for the case of each activation function.

After implementing these functions, we then trained the networks over multiple epochs until they converged.

After multiple runs, we noticed that the best way to initialize the weights is from a uniform distribution in the range [-1, 1]. The learning rate values that gave the best results were 0.15 for the sigmoid and tanh activation functions and 0.1 for ReLU. We selected the best run with each activation function to be represented in figure 4. It can be seen that with sigmoid activation function, the network takes around 10000 epochs to converge. We saw a very high improvement with tanh (convergence around 600 epochs), and an even better one with ReLU (300 epochs).

Our implementation was the most stable and always converged with sigmoid and tanh. With ReLU, it depends a lot on the initialization values of the weights. In very unlucky cases, because of the initial weights, the network did not learn and never converged.



(a) MSE for sigmoid activation

(b) MSE for tanh activation

(c) MSE for ReLU activation

(d) Number of misclassifications for sigmoid activation

(e) Number of misclassifications for tanh activation

(f) Number of misclassifications for ReLU activation

Figure 4: Best case performance of XOR Neural Network with the three different activation functions.

References

(1) Tipping, M. E.; Bishop, C. M. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **1999**, *61*, 611–622.

(2) Roweis, S. T.; Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *science* **2000**, *290*, 2323–2326.

(3) Van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research* **2008**, *9*.