Assignment Zero

Group 45

Introduction

Neural networks are ubiquitous in our modern world. One of the common applications of neural networks is classification tasks, this assignment explores the development and evaluation of several algorithms related to classification. The assignment is split into two related tasks followed by the comparison, analysis and conclusions:

1. Classification using distance-based classifiers and the effects of dimensionality reduction
2. Classification via a multi-class perceptron
   One of the most common applications of neural networks is in classification tasks, where the goal is to assign inputs into predefined categories. This assignment explores the development and evaluation of several algorithms that are tasked with classifying images of handwritten digits.

We will be using a subset of the MNIST with unique 16x16 pixels of handwritten digits, split into training and test data. The input data is represented as a vector of size 256.

Task 1

All the necessary libraries are imported, and the data is read and loaded.

Part 1: High Dimensional Cloud of Points

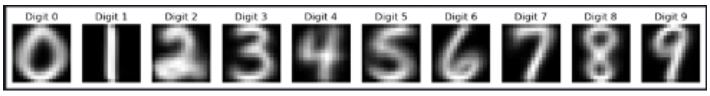
Classification in this case is done by combining all the data points and finding the mean for each class (digit). Here each row's mean is computed and assigned to the corresponding row of the centroid for the class. We do so with all the rows thereby forming the center of the cloud of points for that class.
Mathematically, this is: $c_d = \frac{1}{N_d} \sum_{i=0}^{9} X_i$

Pseudocode:

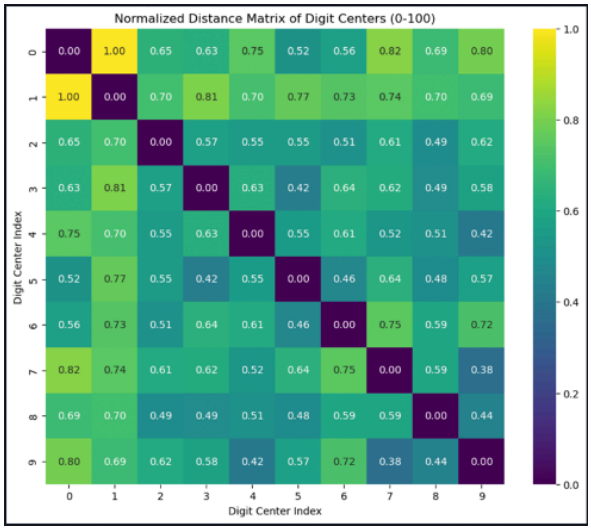
Algorithm 1: Compute Centers for Digits 0-9

```
1: procedure GET_VECTORS(train_in, train_out, digit)
2:     return all rows from train_in where train_out equals digit
3: end procedure

4: procedure COMPUTE_CENTERS(train_in, train_out)
5:     digit_centers ← zeros(10, 256)
6:     for digit ← 0 to 9 do
7:         digit_vectors ← GET_VECTORS(train_in, train_out,digit)
8:         digit_centers[digit] ← MEAN(digit_vectors)
9:     end for
10: end procedure
```

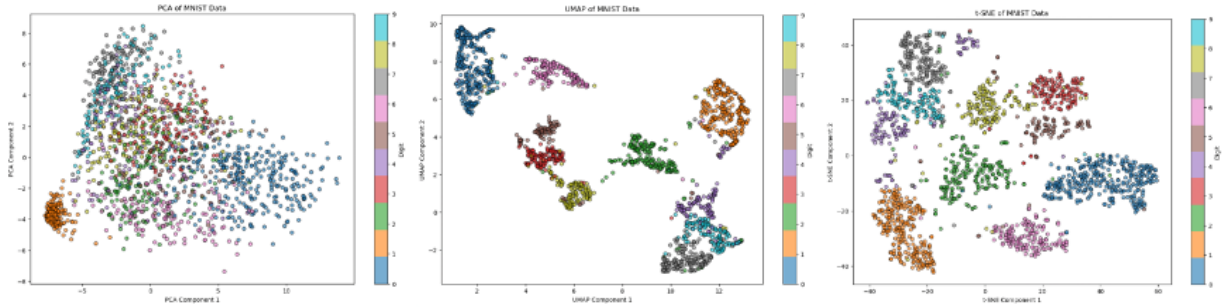


Results: Figure 1: Visualization of the centroid, Figure 2: Pairwise distance between centroids



The expected accuracy of the model can be somewhat inferred by visualizing the distances. Figure 2 shows that the shortest distance is between the centroids for 7 and 9, and hence seem to be the most difficult to separate. Additionally, the distribution of data is analyzed to check for fair and equal representation of all digits in the data. An unequal distribution can lead to biases in not accurately identifying the true center of an underrepresented digit due to its fewer number of datapoints. The distribution of digits in the training data is plotted in the form of a bar chart for visualization. Here, the digit 5 here has the lowest count, while the digit 1 has the highest - almost twice that of 5.

Part 2: Dimensionality Reduction



Various benefits come with dimensionality reduction; by reducing the number of features, the complexity of the data goes down as well. Further, the reduction of non-crucial features reduces complications and confusion – allowing for better accuracy.

To implement this, PCA, UMAP, and T-SNE algorithms are used with the 2D results plotted on a graph as shown below (respectively).

The best performing dimensionality reduction seems to be UMAP, with PCA performing the worst. As predicted in Part 1 – 7 and 9 seem to cluster together, with the addition of 4 as well. The digit ‘1’ clusters the most distinctively in all cases.

Part 3: Nearest Mean Classifier

This part deals with using the centroids computed in Part 1 to classify all the images in train and test datasets by using the Nearest Mean algorithm and calculating the corresponding accuracies. This is done by finding the "closest" center digit to the sample, and the image as assigned as being of the same class. Here, the "closeness" is measured by way of comparison of the input 256 length vector to each of the centroid's vectors using *euclidean distances* and the centroid with the smallest distance is assigned.

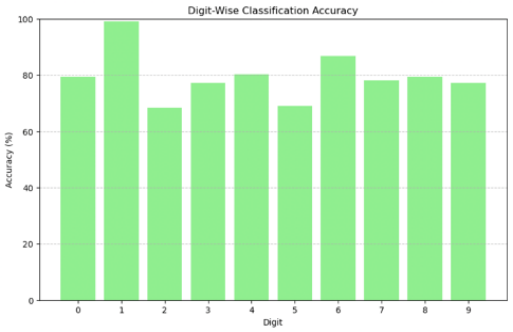
Pseudocode:

Algorithm 2: Classify Image Using Nearest Center

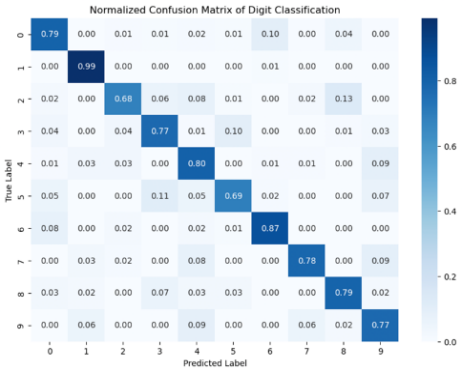
1: **procedure** CLASSIFY_IMAGE(*image,centers*)
2: **distances** ← COMPUTE_DISTANCE(*image,centers*)
3: **closest_label** ← FIND_MIN_INDEX(*distances*)
4: **return** closest_label
5: **end procedure**

6: **procedure** COMPUTE_DISTANCE(*image,centers*)
7: **distances** ← Euclidean_Distance(*image,centers*)
8: **return** distances
9: **end procedure**

10: **procedure** FIND_MIN_INDEX(*distances*)
11: **min_index** ← index of the smallest value in distances
12: **return** min_index
13: **end procedure**



Additionally, the accuracies of the classification on the train and test models are computed by comparing the predicted labels to the actual labels and computing the proportion of likeness. To add on, digit-wise accuracies are also calculated and presented as a confusion matrix to provide further insight for analysis.



The test and train set accuracies were computed to be 80.40% and 86.35% respectively. Additionally, the following graph and figure depicts the digit-wise classification accuracy on the test set and a normalized confusion matrix for the same, respectively.

The overall accuracy of 80.4% on the test set and 86.35% on the train set. shows that the algorithm did not overfit nor underfit the train set. Furthermore, a digit-wise analysis was used to discern what digits are being recognized poorly therefore bring down total accuracy. The lowest ones here were found to be the digits 2 and 5, with the highest being digits 1 with 99% accuracy. As the accuracies are similar on the train and test set - the test set is used here.

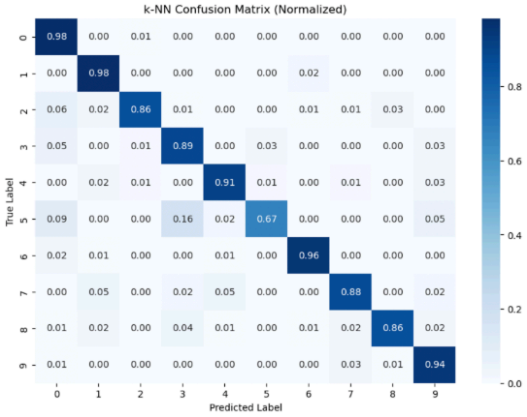
The greatest values in the confusion matrix depict '2' being predicted as '8' and '5' being predicted as '3'.

Part 4: K Nearest Neighbour

In this part, the k-NN algorithm from sklearn is implemented and the data is fit to the model with standard hyperparameter k=3. The confusion matrix is plotted similarly.

The k-NN training and test set accuracies were 97.89% and 91.40% respectively. The following figure illustrates this model's confusion matrix.

Overall accuracies on both train and test do much better here, with the confusion matrix depicting much less confusion overall as well.



Task 2

In this task, we design and implement a simple neural network (a single-layer multi-class perceptron) from scratch to classify images of handwritten digits from the simplified MNIST dataset. We will train the network using gradient descent, explore different hyperparameters (weight initialization strategies and learning rates), and evaluate the model's performance.

Model Architecture

The network consists of one input layer with 256 nodes (corresponding to the 16x16 pixel brightness values) and one output layer with 10 nodes (corresponding to the digits 0–9). These two layers are fully connected in a feed-forward manner, i.e., every input node has forward connections to all output nodes. In addition, each output node is assigned an offset (bias) with an associated dummy input node. Essentially, this multi-class perceptron represents 10 independent univariate regression classifiers with 256 slope parameters and one intercept parameter each, resulting in a weight matrix Ω of size 257x10. The model can be summarized by the following equation:

$$z_{ki} = f\left[x_i, \Phi_k\right] = \phi_{k0} + \sum_{j=1}^{256} \phi_{kj} x_{ij} \quad with \quad \Phi_k \in \Omega,$$

where z_{ki} is the output of the k-th classifier for the i-th example, x_i is the vector of pixel values for the i-th example, Φ_k are the parameters for the k-th classifier, ϕ_{k0} is the bias (intercept) parameter for the k-th classifier, ϕ_{kj} is the slope parameter for the k-th classifier corresponding to pixel j, and x_{ij} is the value of pixel j in the i-th example.

When performing inference, the final prediction of the model corresponds to the index k of the classifier with the largest activation:

$$\hat{y}_i = argmax_k\left[z_i\right],$$

where $\hat{y}_i$ is the predicted class label for the i-th example and z_i is the vector of output activations z_{ki} for the i-th example.

Weight Initialization

We use and compare two weight initialization strategies: random initialization and Glorot initialization.

In random initialization, the model weights are sampled from a uniform distribution between -1 and 1: $\omega_{io} \sim U(-1, 1)$, where ω_{io} is the weight connecting the i-th input node to the o-th output node. In Glorot initialization, the weights are initialized using the Glorot uniform distribution: $\omega_{io} \sim U\left(-\sqrt{\frac{6}{n_i + n_o}}, \sqrt{\frac{6}{n_i + n_o}}\right)$, where n_i is the number of input nodes and n_o is the number of output nodes.

Model Training

We fit the model on a training set of examples $\{x'_i, y_i\}_{i=1}^{I}$, where x'_i is the vector of pixel values for the i-th example with a fixed value of 1 appended (for the bias), y_i is the correct label for the i-th example and $I = 1707$. We define a loss function:

$$L = \frac{1}{I}\sum_{i=1}^{I}\sum_{k=1}^{10}\left(z_{ki} - y_{ki}\right)^2,$$

where L is the average loss over all examples and y_{ki} is the one-hot encoded correct output of the k-th classifier for the i-th example:

$$y_{ki} = \{1,\ if\ k = y_i\ 0,\ if\ k \neq y_i\ .$$

We use the gradient descent algorithm to train the model. To this end, we first determine the gradients:

$$\frac{\partial L}{\partial \omega_{io}} = \frac{2}{I}\sum_{i=1}^{I}\left(z_{ki} - y_{ki}\right)x'_{ij}.$$

Then, we update the weights using the following rule:

$$\omega_{io} \leftarrow \omega_{io} - \alpha\cdot\frac{\partial L}{\partial \omega_{io}},$$

where α is the learning rate. We use and compare α-values of 0.01 and 0.001.

To compare the two weight initialization strategies and the two learning rates, we train a total of four models, each with 1000 training epochs.

Results

The performance (average loss and classification accuracy) on the training and test datasets over the training epochs for random initialization with two different learning rates is shown in Figure 1. Similar plots for Glorot initialization can be found in Figure 2.

The final classification accuracies (for $\alpha = 0.01$ / $\alpha = 0.001$) on the test dataset were 40.3% / 20.6% for random initialization, and 84.4% / 63.4% for Glorot initialization.

Discussion

The choice of weight initialization strategy and learning rate impacts the convergence speed and the final performance of the model. Glorot initialization resulted in much faster learning than random initialization (the latter did not yield convergence within 1000 training epochs). An α of 0.01 likewise resulted in markedly faster (yet initially noisier) learning than an α of 0.001 (again, the latter did not yield convergence within 1000 training epochs).

The non-deterministic nature of the algorithms developed here was evident when performing the same training regimes a few times: performance trajectories differed notably and ultimate classification accuracy changed by up to a few percentage points (more so for the models which had not converged upon 1000 epochs of training).

Finally, after 1000 training epochs, the single-layer multi-class perceptron with Glorot weight initialization and a learning rate of 0.01 (i.e., the only model of the four developed here that had converged by this point) performed better than the simplistic nearest mean classifier, but still significantly worse than the KNN classifier.

Figure 1. Model Performance Throughout Training for Random Initialization

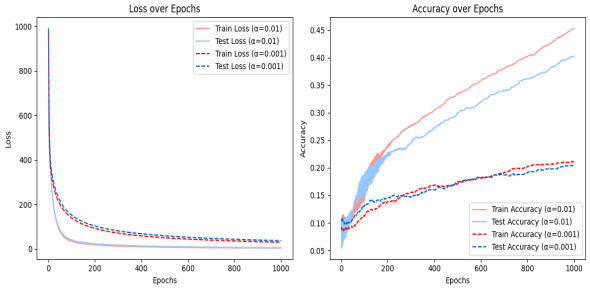


Figure 2. Model Performance Throughout Training for Glorot Initialization