MALIS Project 2 Report: The Perceptron

EL BACHA Tonia SAADE Tarek

1 Introduction

The primary objective of this project is to focus on implementing the perceptron algorithm and applying it to a real-world classification problem. Specifically, we work with the MNIST dataset, a collection of 8x8 pixel images of handwritten digits from 0 to 9. The goal of this project is two fold: first, to implement and understand the mechanics of the perceptron algorithm, and second, to experiment with its performance in classifying digit images.

2 Implementing a Perceptron Algorithm

The `train` method implements the training of the perceptron. This is achieved by iterating on the training data set for a number of **epochs**, updating the weights and bias for misclassified samples using the stochastic gradient descent (SGD) algorithm, which try to minimize the perceptron criterion by adjusting the parameters.

The `predict` method is responsible for making predictions on the data after the model has been trained. It uses the learned weights and bias to classify each sample.

Two hyper-parameters can be adjusted for optimal performance: the number of **epochs** and the **learning rate** (alpha). Usually, smaller learning rates are associated with larger number of epochs as the algorithm takes more steps converge towards an optimum.

2.1 Testing the Efficiency of Our Custom Perceptron Algorithm

After implementing the perceptron, we evaluated our algorithm on a linearly separable dataset. We achieved an impressive accuracy of 97.50%, which indicates its ability to correctly classify data points between the two classes in the data set.

The visualization below was expected, as it demonstrates the model's ability to correctly separate data.

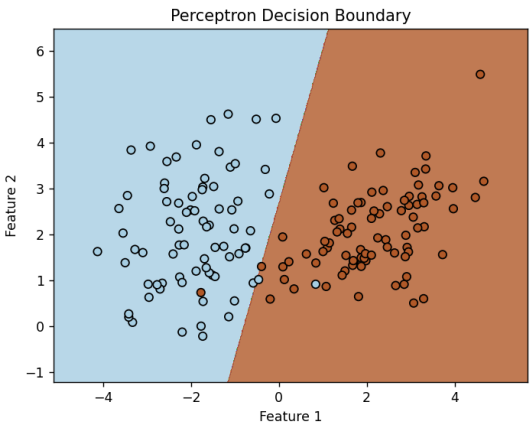

Figure 1: Traning set

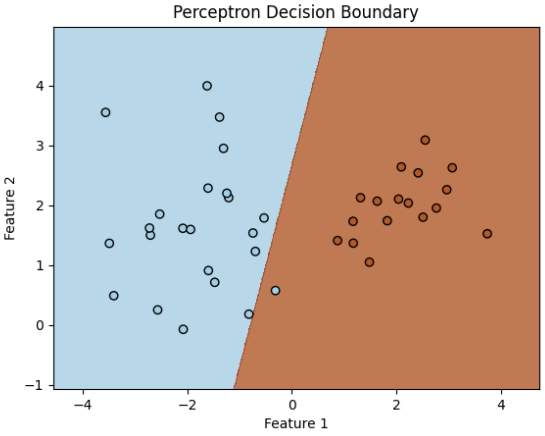

Figure 2: Testing set

2.2 Our Custom Perceptron Algorithm vs Scikit-learn Perceptron

After achieving good performance with our custom perceptron, we decided to compare this result with the scikit-learn perceptron to study the differences.

The custom perceptron, while effective, is a basic implementation that may lack some of the advanced optimizations found in the scikit-learn version. This means that the scikit-learn version is faster and achieves slightly better accuracy than the custom perceptron.

However, the custom perceptron provides a deeper understanding of the algorithm and can be improved through optimization techniques, such as using adaptive learning rate or mini-batches for gradient descent.

3 Using the Perceptron

3.1 One vs All (OvA)

To extend the perceptron algorithm to handle multi-class classification, we implemented the one-vs-all (OvA) technique. This approach involved training ten separate perceptron classifiers, each dedicated to distinguishing a specific digit (0 to 9) from all other digits. For each digit, the corresponding classifier was trained with binary labels.

The OvA approach yielded a very low accuracy of 9.17%, which is close to random guessing for a data set with ten classes. This poor performance can be attributed to the inability of individual classifiers to generalize effectively in a multi-class setting.

3.2 One vs One (OvO)

After observing poor performance with the one-vs-all approach, we implemented the one-vs-one technique to address the multi-class classification problem. In the OvO approach, we trained 45 separate perceptron classifiers for every unique pair of digits. Each classifier was trained to distinguish between two specific digits, using binary labels. During the prediction phase, each test sample was evaluated across all classifiers, and a voting mechanism was used to determine the predicted class. Each classifier contributed one vote for its respective class based on its prediction.

This approach proved highly effective, achieving an accuracy of 100% on the training set and 97% on the testing set. The significant improvement in performance compared to the OvA method highlights the strength of OvO in handling multi-class problems. The outcome can be improved by implementing some techniques to avoid over-fitting.

4 Conclusion

This project showed the limitations of using **a single perceptron** for classifying multi-class data sets as the decision boundaries starts to overlap. However, we start to touch how using **several perceptrons**, where each one is responsible for a specific task, can indeed push the limitations further. This gives the motivation to design a more complex system of perceptrons, interconnected with adjustable weights, where the activation of certain number of them can lead to the activation of others and so on. Other tools can be used to optimize the algorithm further, such as reducing the dimension of the data or using kernels to extend the efficiency of perceptrons in classifying non-linearly separable datasets.

Contributions: Both of us conducted research on perceptrons and their application to classification problems.
Tonia focused on implementing the perceptron class and developing the corresponding tests. Tarek worked on implementing the classification algorithm.
Chatgpt was used to formulate some ideas in the report and helped us to implement some functions, specially in the test files.