Victor MAYAUD
Hichem KHETTAB

Project 2: The Perceptron

Part1 :

The initial task was to implement a perceptron. In perceptron.py, we created two Perceptron classes. One class utilizes the log loss function as the cost function with the sigmoid as the activation function, while the second employs the mean square error function, again with the sigmoid as the activation function. Although the two classes use different cost functions, they share the same structural design, each with its respective gradient formula. The class constructor accepts several inputs, including the dataset, learning rate, momentum, and the selected optimization method for training. The perceptron's parameters are initialized within the constructor, with weights set using the He initialization method to promote better convergence. The train method facilitates the updating of weights and biases using different approaches:

The classic method applies gradient descent, where updates are informed by the gradient computed from either the log loss or the mean square error function. We dynamically adjust the learning rate during training to encourage better convergence.

The momentum method enhances gradient descent by accounting for both the current gradient and a portion of the previous update vector. It calculates gradients for weights and biases, updates their velocities by amalgamating past velocities (scaled by the momentum coefficient) with current gradients (scaled by the learning rate), and then modifies the weights and biases by subtracting these velocities. This approach is designed to accelerate gradient descent, steering it along pertinent directions while diminishing oscillations within the cost function's landscape.

The predict function calculates the output of the sigmoid function for each data point. To ascertain the model's classification, we assess whether the output is below or above 0.5. Outputs under 0.5 are categorized as 0, and those over 0.5 are classified as 1.

Part2 :

After implementing both perceptrons, the subsequent step was to conduct tests. We began by importing only the digits 1 and 0 from the sklearn.datasets library and divided the dataset into two segments: two-thirds for training and one-third for testing. This division was essential for assessing the perceptron's efficacy on data it had not previously encountered.

We then initiated training with our first perceptron (utilizing the log loss function and sigmoid activation) on the training dataset. To pinpoint the optimal hyperparameters—namely, the learning rate and the momentum coefficient—we calculated the average accuracy across 20 iterations. This exercise was replicated with varying values for the coefficients to obtain a more stable result. The best performance was recorded with a learning rate of 1 and a momentum coefficient of 0.7 (refer to Figure **A-1**). At these settings, the perceptron swiftly attained a 100% accuracy rate on the test set. Although the traditional gradient descent method was efficient, the momentum method proved superior, frequently delivering outstanding results on the test set after merely 2 or 3 training epochs. Figure **A-2** demonstrates the momentum method's capacity to reach favorable outcomes more expeditiously than the classical approach.

A similar procedure was employed for the second perceptron (mean square error with sigmoid activation). We conducted a parameter search, determining the most effective values to be 10 for the learning rate and 0.5 for the momentum. Upon training our model with these parameters, although the learning speed was slower compared to the first perceptron **(A-3)**, it still yielded satisfactory results.

Part3 :

The exercises outlined are integral for understanding the mathematical mechanics of machine learning. Calculating derivatives sharpens our insights into how models learn and adjust from data. Grasping the sigmoid function's intricacies, for instance, is key for neural network training. Exercises on Taylor polynomials and Jacobians enhance our grasp of model behavior and sensitivity, which are crucial for model optimization. The use of the chain rule in derivative computation reflects the complexity of training deep

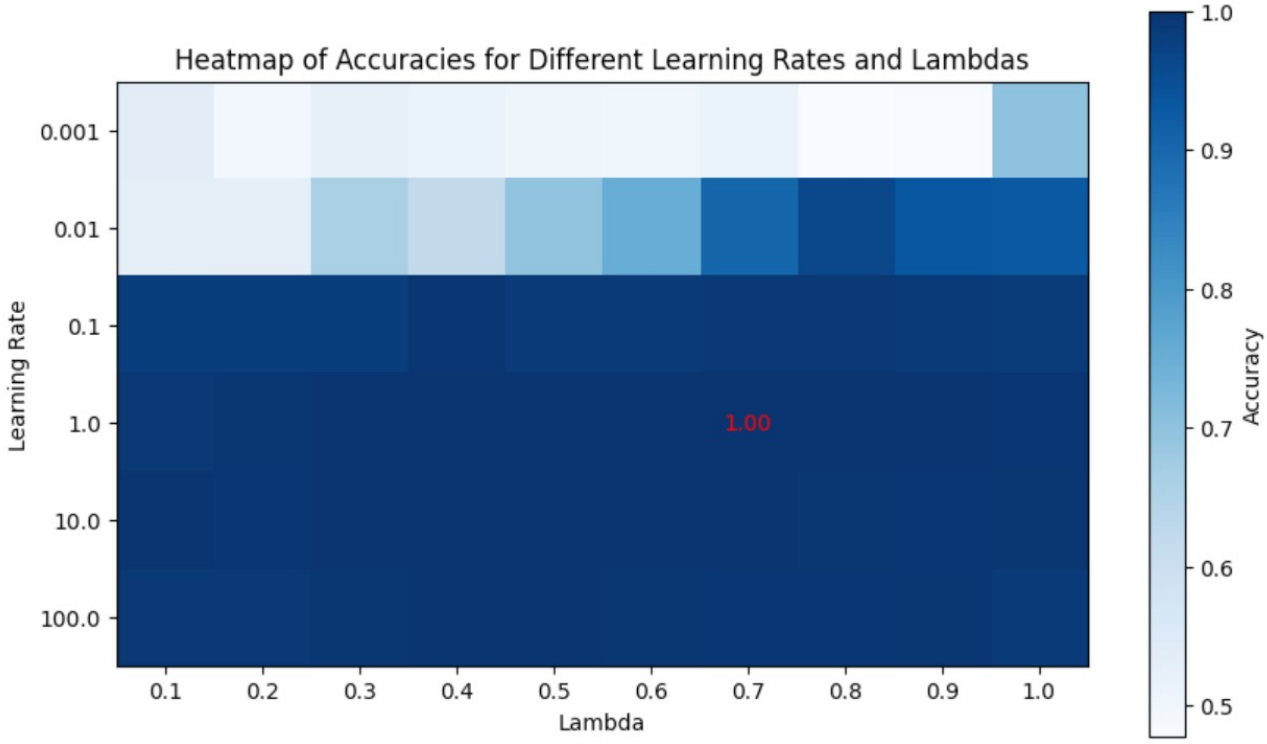
learning models, equipping us with the necessary skills to tackle real-world machine learning challenges efficiently.

Conclusion:

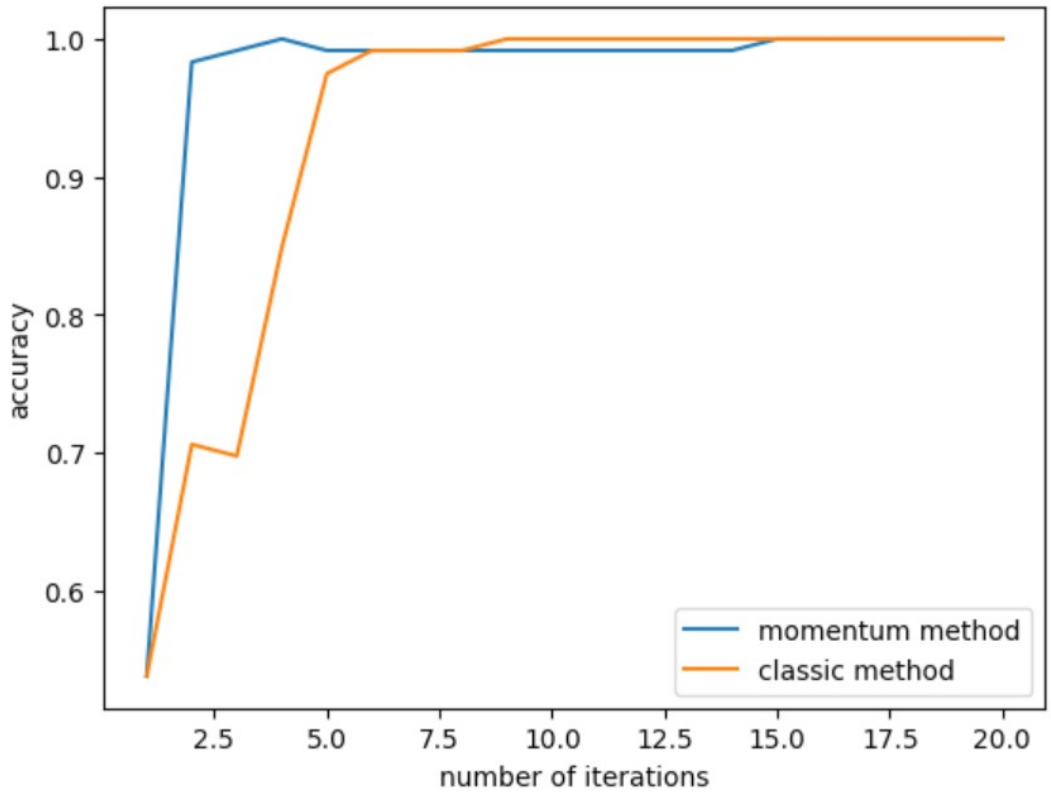
This laboratory exercise enabled us to construct and execute a perceptron on a dataset, significantly enhancing our understanding of perceptrons as well as gradient and vector calculus. The perceptron utilizing a log loss function paired with sigmoid activation demonstrated high efficiency for binary classification tasks, and it appeared to outperform the alternative perceptron that employed mean square error with sigmoid activation. The provided dataset was relatively straightforward for the perceptron to differentiate between two classes, which accounts for the high accuracies achieved by both perceptrons at the conclusion of training.

Annexe :

A-1:



A-2: Comparison between momentum method and classic gradient descent method on perceptron



A-3: Accuracy of the Perceptron Using Momentum Method and MSE Cost Function Across Iterations