Assignment 1 - Perceptron Learning Rule

Edge and Neuromorphic Computing - CSCE790

Misagh Soltani
University of South Carolina
msoltani@email.sc.edu

Spring 2024

1 Introduction

In this assignment, we will implement the perceptron learning rule for learning a two-input AND gate. The perceptron is a simple model of an artificial neuron that can perform binary classification. The perceptron learning rule is an algorithm that updates the weights and bias of the perceptron based on the errors made on the training data. We will use Python to code the perceptron and plot the decision boundary at the end of each epoch.

2 Perceptron Model

The perceptron model consists of two layers: an input layer and an output layer. The input layer has two nodes, corresponding to the two binary inputs of the AND gate. The output layer has one node, corresponding to the binary output of the AND gate. The output node computes a weighted sum of the inputs and applies a step activation function to produce the output. The step activation function returns 1 if the input is greater than or equal to zero, and 0 otherwise. The perceptron model can be represented by the following equation:

$$y = f(w_1 x_1 + w_2 x_2 + b)$$

where x_1 and x_2 are the inputs, w_1 and w_2 are the weights, b is the bias, f is the step activation function, and y is the output.

3 Python Code

Below is pseudocode outlining the implementation of the perceptron learning rule to train a two-input AND gate. The corresponding Python implementation is accessible in the following locations:

1. Accompanying this document
2. GitHub repository[1]

The Python code leverages NumPy for numerical calculations and Matplotlib for visualization. It outputs the epoch number and the corresponding decision boundary in the form $X_2 = f(X_1)$ after each epoch for tracking progress.

Main function may be used to change the input data, initial weights and bias, and other parameters.

Algorithm 1 Train Perceptron

procedure TRAINPERCEPTRON($train\_data, target\_data, perceptron, lr, epochs$)
  for $epoch \in \{1, 2, \ldots, epochs\}$ **do**
    $errors\_sum \leftarrow 0$
    for $X, Y \in zip(train\_data, target\_data)$ **do**
      $output \leftarrow perceptron.activate(X)$
      $e \leftarrow Y - output$
      $errors\_sum \leftarrow errors\_sum + |e|$
      $W, b \leftarrow perceptron.get\_w\_b()$
      $W \leftarrow W + lr * e * X$
      $b \leftarrow b + lr * e$
      $perceptron.update\_w\_b(W, b)$
    end for
    if $errors\_sum = 0$ **then**
      print("Perceptron fully trained in epoch", $epoch$)
      return
    end if
    plot_decision_boundary($train\_data, target\_data, epoch, W, b$)
  end for
end procedure

[1] `https://github.com/misaghsoltani/perceptron_learning_rule`

Algorithm 2 Main Function

```
procedure MAIN
    X ← {}                                   ▷ Training data for AND gate
    Y ← {}                                   ▷ Target data for AND gate
    W ← {}                                   ▷ Initial weights
    b ← 0                                    ▷ Initial bias
    learning_rate ← 1
    epochs ← 100
    perceptron ← Perceptron(W, b)
    TrainPerceptron(X, Y, perceptron, learning_rate, epochs)
end procedure
```

4 Results

The perceptron learning rule was trained for 5 epochs where the error e for all the iterations in that epoch was 0. The initial weights and bias are $W = [0.5, 1]$ and $b = -1$ respectively. Furthermore, final weights and bias are [2.5, 1] and -3, respectively. The network is fully trained after 5 epochs and can be used as an AND gate.
The decision boundary at the end of each epoch is shown in Figure 1 to 5.

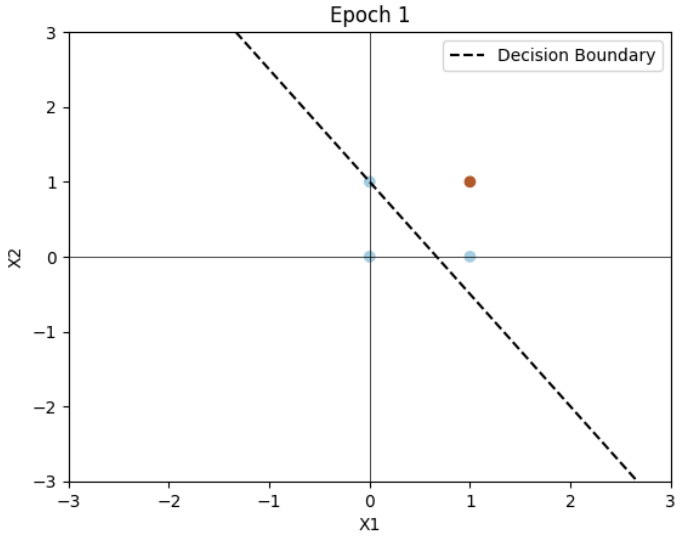

Figure 1: Decision boundary (Epoch 1): $X_2 = -1.5 \cdot X_1 + 1.0$

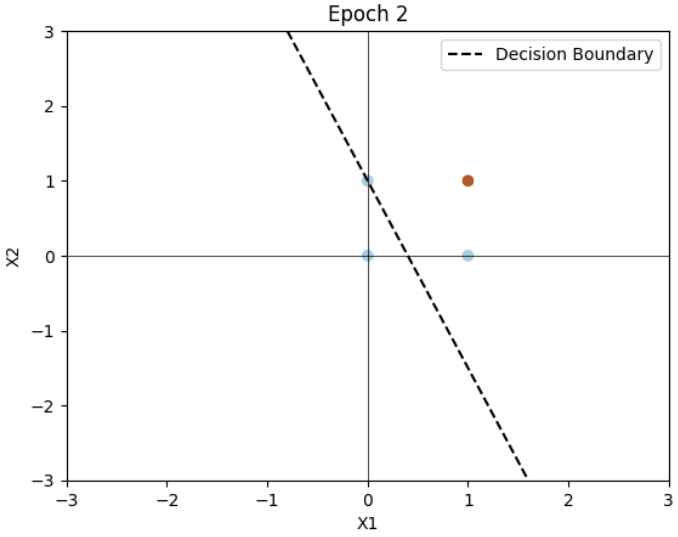

Figure 2: Decision boundary (Epoch 2): $X_2 = -2.5 \cdot X_1 + 1.0$

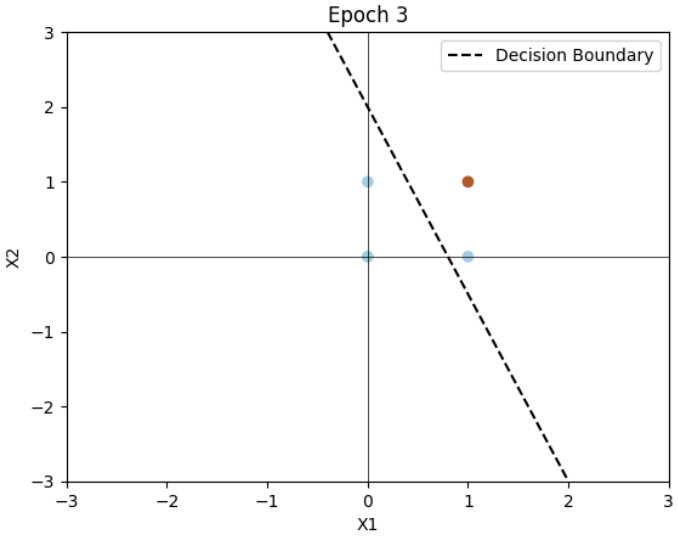

Figure 3: Decision boundary (Epoch 3): $X_2 = -2.5 \cdot X_1 + 2.0$

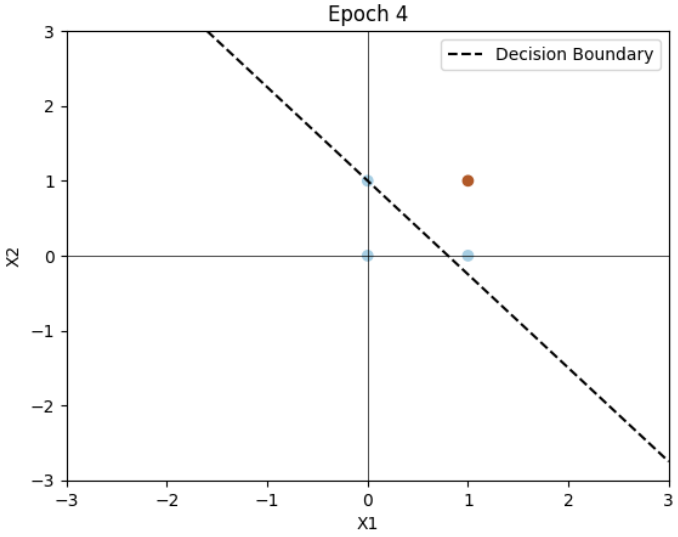

Figure 4: Decision boundary (Epoch 4): $X_2 = -1.25 \cdot X_1 + 1.0$

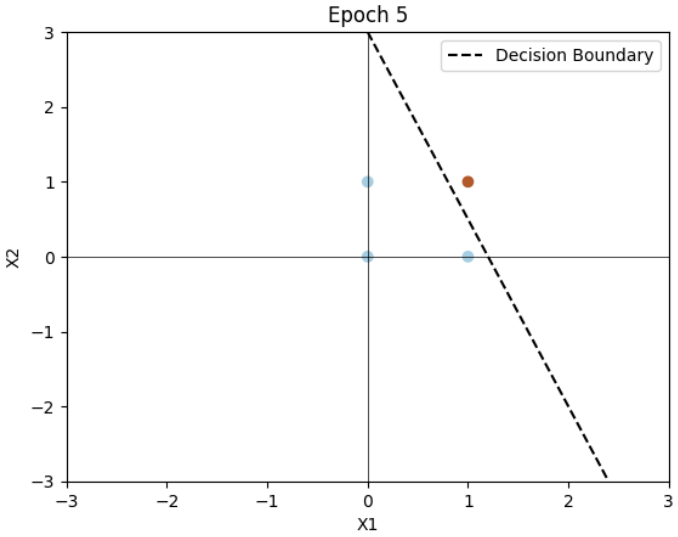

Figure 5: Decision boundary (Epoch 5): $X_2 = -2.5 \cdot X_1 + 3.0$

5 Conclusion

The perceptron learning rule was successfully implemented in Python to train a two-input AND gate. The network converged after five epochs, and decision boundaries were plotted at the end of each epoch to visualize the training process.

The plots reveal a progressive improvement in the perceptron's output with each iteration and epoch. Factors such as initial weights, bias, learning rate, and activation function can influence the number of epochs required to achieve an error of 0.

Further experiments with varying initializations demonstrated that the number of epochs needed for full training differs depending on these factors. As anticipated, fewer epochs were required when the initial weights and bias were closer to the target values. Additionally, adjusting the learning rate also impacted the number of epochs needed for convergence. In this specific example, changing the learning rate to a value less than 1 will require more epochs