TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
THAPATHALI CAMPUS

A Lab Report
ON
Perceptron Learning Algorithm (PLA)

Submitted by:

Amrit Kandel	(THA078BEI004)
Prasish Timalsina	(THA078BEI026)

Submitted to:
Department of Electronics and Communication Engineering,
Thapathali Campus,
Kathmandu, Nepal

May 26, 2025

Abstract

This lab report presents the implementation of a single-layer perceptron to simulate basic logic gates using the Perceptron Learning Algorithm. The perceptron was trained to classify inputs for AND and OR gates using a sigmoid activation function and a threshold value of 0.5 for decision-making. The results showed that the model correctly learned the behavior of linearly separable gates such as AND and OR. However, it was unable to model the XOR gate due to its non-linear nature. The experiment highlights the basic working of artificial neural networks and shows the limitations of single-layer perceptrons for solving complex problems.

---

1 Introduction

1.1 Perceptron

Perceptron is the simplest form of artificial neural network. It performs binary classification that maps input features to an output decision, usually classifying data into one of two categories, such as 0 or 1.

There are two types of perceptron. They are:

- **Single-Layer Perceptron**: It is limited to learning linearly separated patterns. It is effective for tasks where the data can be divided into distinct categories through a straight line. While powerful in its simplicity, it struggles with more complex problems where the relationship between inputs and outputs is non-linear.
- **Multi-layer Perceptron**: It possess enhanced processing capabilities as they consist of two or more layers for handling more complex patterns and relationships within the data.

1.2 Binary Classifier

A binary classifier is a type of machine learning model that categorizes input data into one of two distinct classes. It learns from labeled training data, where each input is associated with a binary output, typically represented as 0 or 1. The goal is to find a decision boundary that can accurately separate the two classes.

In the context of logic gates like AND and OR, the binary classifier receives pairs of binary inputs and predicts whether the output should be 0 or 1 based on learned rules. The perceptron is one of the simplest binary classifiers. It uses a linear equation combining the input values and corresponding weights, plus a bias term. The output is then passed through an activation function, like the unit step function or sigmoid, to produce a final binary prediction.

1.3 Activation Function

An activation function is a mathematical function used in artificial neural networks to determine the output of a neuron given a set of inputs. It introduces non-linearity into the model, allowing the network to learn and represent complex patterns beyond simple linear relationships.

In a perceptron, the activation function takes the weighted sum of the inputs (plus a bias) and converts it into an output signal, either a 0 or 1 for binary classification tasks.

Some of the most common types of activation functions are given below.

- Step Function

$$\mathrm{f(x)} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

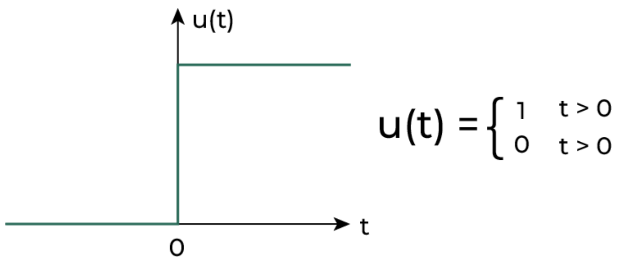


Figure 1-1: Step Function Graph

- Sigmoid Function

$$\mathrm{f(x)} = \frac{1}{1+e^{-x}}$$

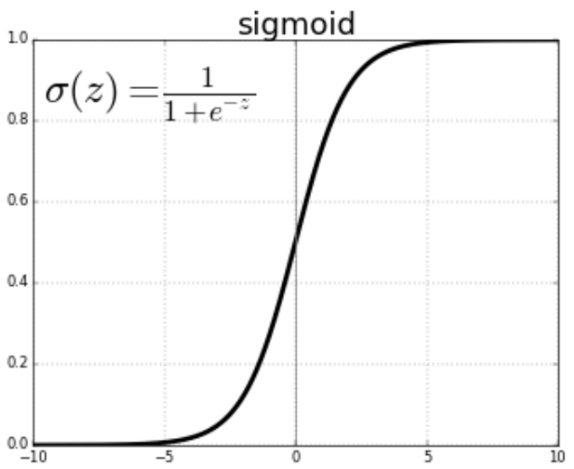


Figure 1-2: Step Function Graph

1.4 Perceptron Learning Algorithm (PLA)

The Perceptron Learning Algorithm (PLA) is a supervised learning rule used to train a perceptron. It updates the weights and bias incrementally based on the error between the predicted and actual output.

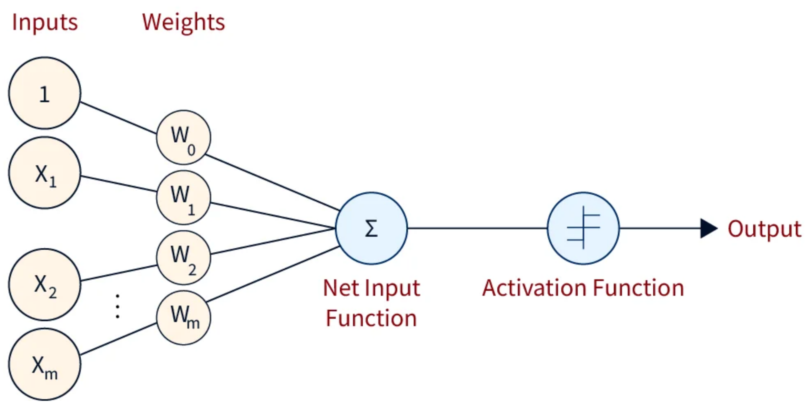


Figure 1-3: PLA

The learning rule is given as below.

$$w_i \leftarrow w_i + \eta \cdot E \cdot x$$

$$b \leftarrow b + \eta \cdot E$$

where, w_i is the weight associated to the input, η is the learning rate, $E = (y - \hat{y})$ is the error, y is the targeted output, $\hat{y}$ is the predicted output and b is the bias.

2 Dataset

The truth tables of the AND gate and the OR gate were used as dataset for training the perceptron.

The truth tables of AND gate and OR gate are given below in Figure 2-1 and Figure 2-2.

Table 2-1: Truth table for AND gate

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Table 2-2: Truth table for OR gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

3 Experimentation

A single-layer perceptron was implemented using Python to classify binary logic gates, specifically, the AND and XOR gates. The perceptron model used a sigmoid activation function to map input features to binary output values.

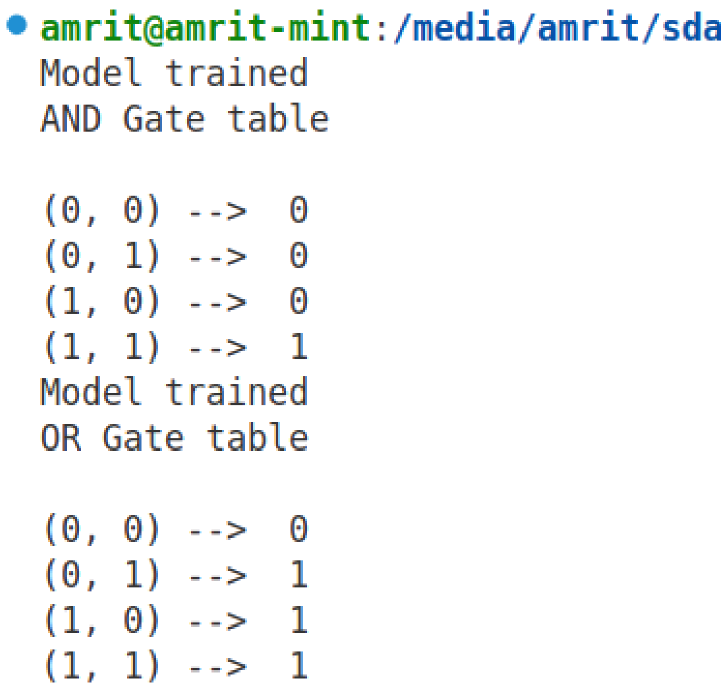
The perceptron was trained using supervised learning where inputs and expected outputs (truth tables) were predefined. Weights and bias were updated using the Perceptron Learning Algorithm (PLA) with a learning rate of 0.001. The experiment consisted of training the perceptron with two inputs and predicting the output using the learned weights.

The structure of the implementation included.

- A sigmoid function as the activation function.
- A training function to update weights based on the error.
- A testing function to predict the output of test inputs.
- A plotting function to visualize decision boundaries.

4 Results

As the result of the experiment, the truth table of both AND gate and OR gate were obtained as result. The result was as shown in the Figure 4-1.



```
amrit@amrit-mint:/media/amrit/sda
Model trained
AND Gate table

(0, 0) -->  0
(0, 1) -->  0
(1, 0) -->  0
(1, 1) -->  1
Model trained
OR Gate table

(0, 0) -->  0
(0, 1) -->  1
(1, 0) -->  1
(1, 1) -->  1
```

Figure 4-1: Result of experiment

The decision boundaries of both gates were also obtained as given in Figure 4-2 and Figure 4-3.

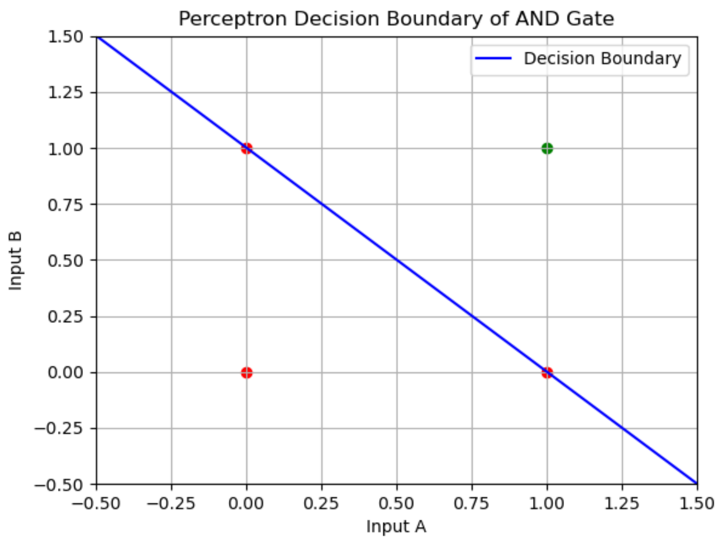


Figure 4-2: Decision boundary of AND gate

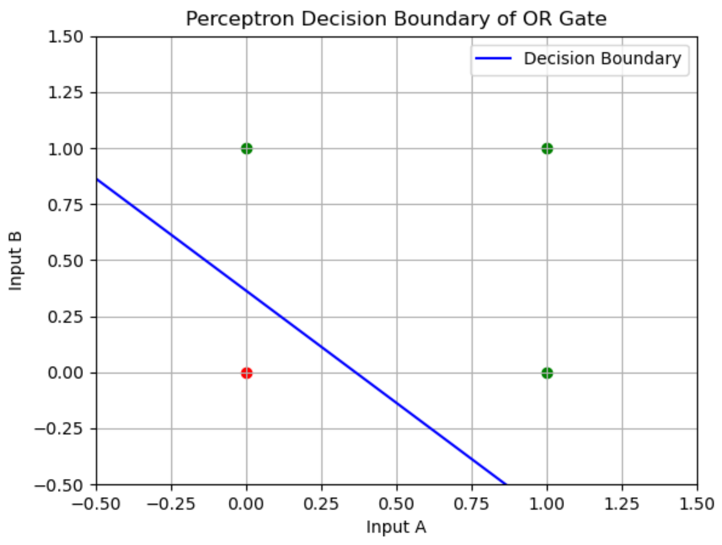


Figure 4-3: Decision boundary of OR gate

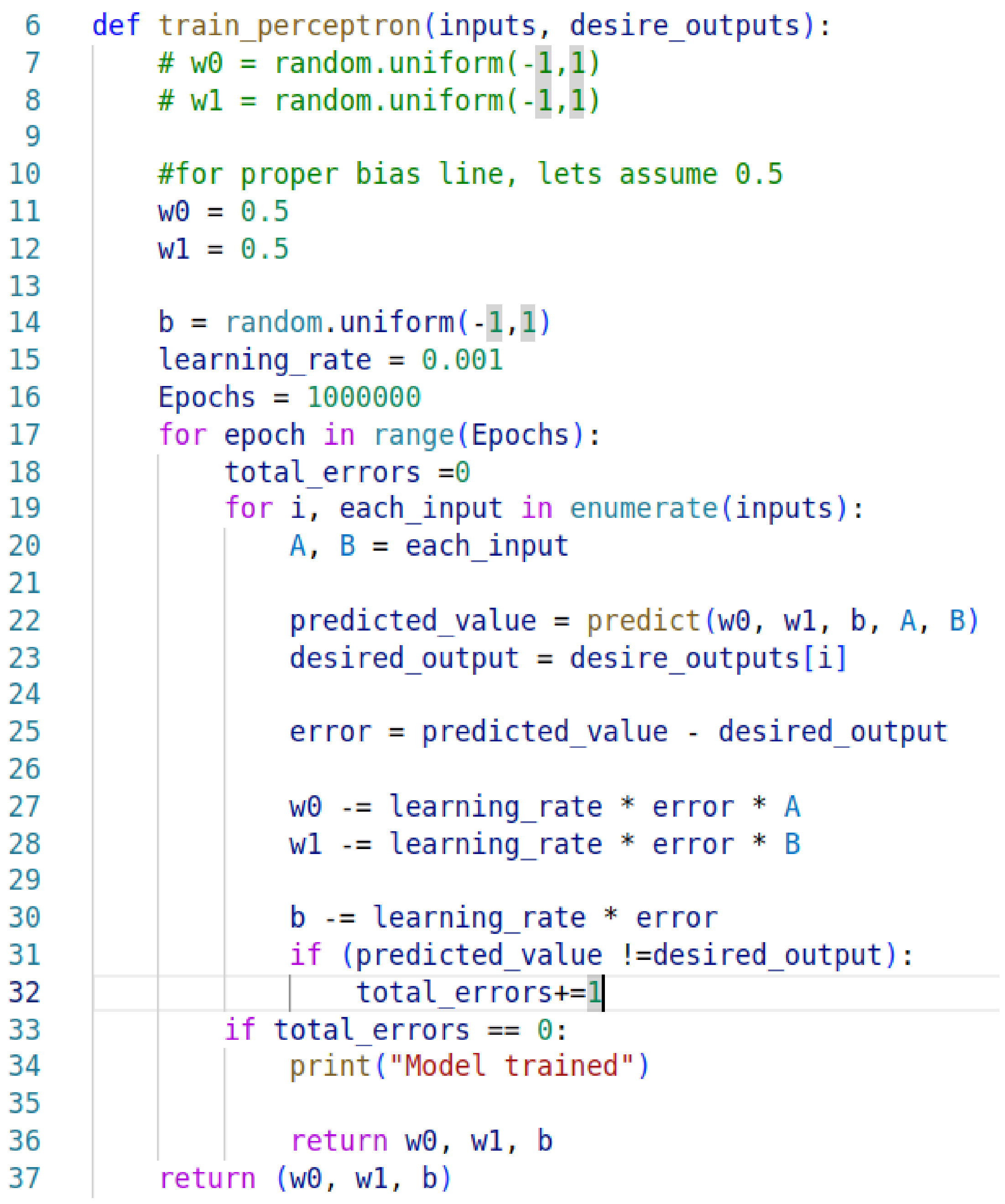
5 Discussion

In this experiment, we used the Perceptron Learning Algorithm to understand how a simple neural network can learn to classify data into two categories. We used the sigmoid (logistic) function as the activation function because it gives outputs between 0 and 1, which makes it suitable for deciding between two options like 0 or 1. We used a threshold of 0.5 to make decisions—if the output was 0.5 or more, we took it as 1; otherwise, we took it as 0. This is a common choice because 0.5 is the middle point of the sigmoid curve, and it helps make fair and balanced decisions. The perceptron worked well with the AND gate, which has linearly separable data. However, it could not learn the XOR gate properly because XOR is not linearly separable. This shows a limitation of single-layer perceptrons—they cannot solve problems where the data can't be separated by a straight line. To solve such problems, we would need to use more advanced models like multi-layer perceptrons. From this experiment, we also learned how important it is to choose the right activation function, threshold value, learning rate, and how useful visualizing decision boundaries can be to understand how the model works.

6 Conclusion

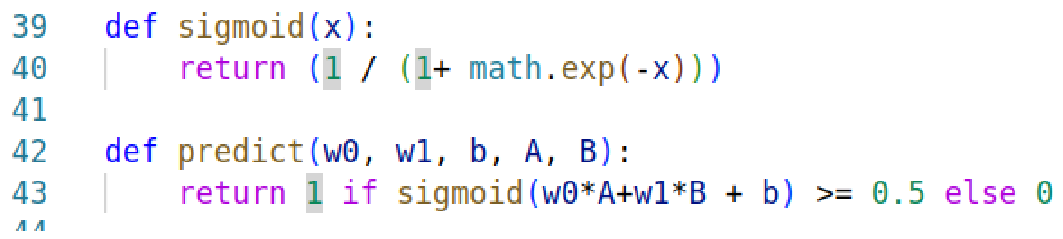
In this lab, the fundamental concept of Artificial Neural Network (ANN) and single layer perceptron was understood. The single layer perceptron was implemented to realize AND gate and OR gate. ALso, it was found that XOR gate can not be realized with single-layer perceptron.

7 Source Codes



```python
def train_perceptron(inputs, desire_outputs):
    # w0 = random.uniform(-1,1)
    # w1 = random.uniform(-1,1)

    #for proper bias line, lets assume 0.5
    w0 = 0.5
    w1 = 0.5

    b = random.uniform(-1,1)
    learning_rate = 0.001
    Epochs = 1000000
    for epoch in range(Epochs):
        total_errors =0
        for i, each_input in enumerate(inputs):
            A, B = each_input

            predicted_value = predict(w0, w1, b, A, B)
            desired_output = desire_outputs[i]

            error = predicted_value - desired_output

            w0 -= learning_rate * error * A
            w1 -= learning_rate * error * B

            b -= learning_rate * error
            if (predicted_value !=desired_output):
                total_errors+=1
        if total_errors == 0:
            print("Model trained")

            return w0, w1, b
    return (w0, w1, b)
```

Figure 7-1: Function definition to train the perception



```python
def sigmoid(x):
    return (1 / (1+ math.exp(-x)))

def predict(w0, w1, b, A, B):
    return 1 if sigmoid(w0*A+w1*B + b) >= 0.5 else 0
```

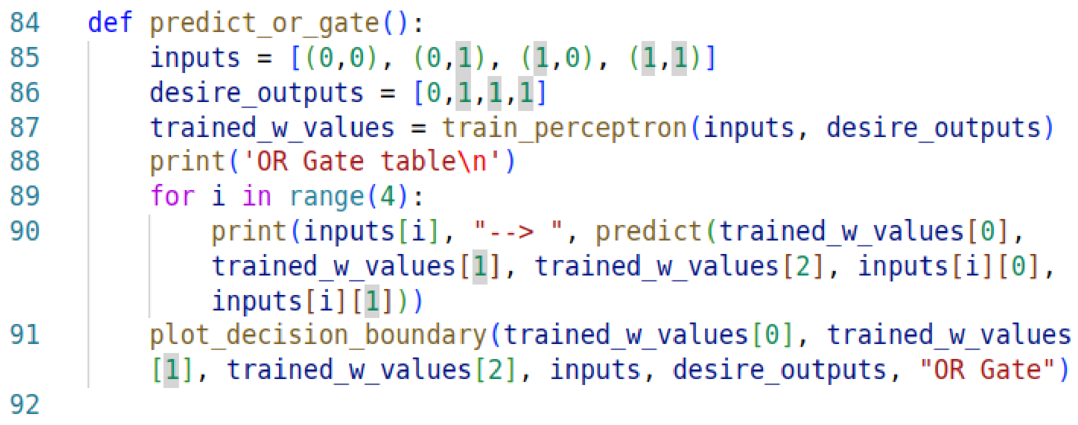
Figure 7-2: Definition of sigmoid and predict functions

```python
def plot_decision_boundary(w0, w1, b, inputs, desire_outputs, gate_name):
    # Prepare input data for plotting
    x_vals = [i[0] for i in inputs]
    y_vals = [i[1] for i in inputs]

    # Plot the points
    for i in range(len(inputs)):
        color = 'red' if desire_outputs[i] == 0 else 'green'
        plt.scatter(inputs[i][0], inputs[i][1], color=color)

    # Create the decision boundary line: w0*x + w1*y + b = 0 → y = -(w0*x + b)/w1
    x = np.linspace(-0.5, 1.5, 100)
    if w1 != 0:
        y = -(w0 * x + b) / w1
        plt.plot(x, y, '-b', label='Decision Boundary')
    else:
        # Special case: vertical line if w1 == 0
        x_val = -b / w0
        plt.axvline(x=x_val, color='blue', label='Decision Boundary')

    plt.xlim(-0.5, 1.5)
    plt.ylim(-0.5, 1.5)
    plt.xlabel('Input A')
    plt.ylabel('Input B')
    plt.legend()
    plt.title(f'Perceptron Decision Boundary of {gate_name}')
    plt.grid(True)
    plt.show()

```

Figure 7-3: function definition to plot the decision boundaries

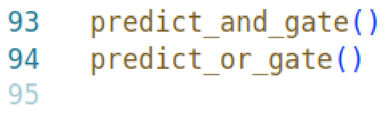
```python
def predict_and_gate():
    inputs = [(0,0), (0,1), (1,0), (1,1)]
    desire_outputs = [0,0,0,1]
    trained_w_values = train_perceptron(inputs, desire_outputs)
    print("AND Gate table\n")
    for i in range(4):
        print(inputs[i], "--> ", predict(trained_w_values[0], trained_w_values[1], trained_w_values[2], inputs[i][0], inputs[i][1]))
    plot_decision_boundary(trained_w_values[0], trained_w_values[1], trained_w_values[2], inputs, desire_outputs, "AND Gate")

```

Figure 7-4: Function definition to predict the AND gate



```python
def predict_or_gate():
    inputs = [(0,0), (0,1), (1,0), (1,1)]
    desire_outputs = [0,1,1,1]
    trained_w_values = train_perceptron(inputs, desire_outputs)
    print('OR Gate table\n')
    for i in range(4):
        print(inputs[i], "--> ", predict(trained_w_values[0], trained_w_values[1], trained_w_values[2], inputs[i][0], inputs[i][1]))
    plot_decision_boundary(trained_w_values[0], trained_w_values[1], trained_w_values[2], inputs, desire_outputs, "OR Gate")

```

Figure 7-5: Function definition to predict the OR gate



```python
predict_and_gate()
predict_or_gate()

```

Figure 7-6: Calling the predict_and_gate and predict_or_gate functions