EXPERIMENT 1.3

Student Name: Samuel **UID: 24MAI10018**

Branch: AI&ML **Section/Group: 24MAI-1**

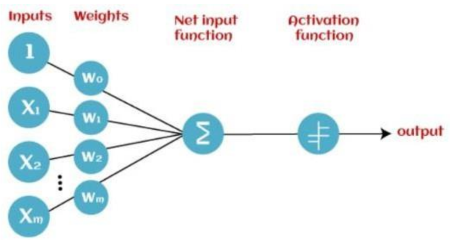
Semester: 2nd **Date of Performance: /01/2025**

Subject Name: SC Lab **Subject Code: 24CSH–668**

AIM: Implementation of Perceptron Learning Algorithm.

SOFTWARE USED: Visual Studio

THEORY: Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence.
Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., input values, weights and Bias, net sum, and an activation function.



- Input Nodes or Input Layer:

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- Wight and Bias:

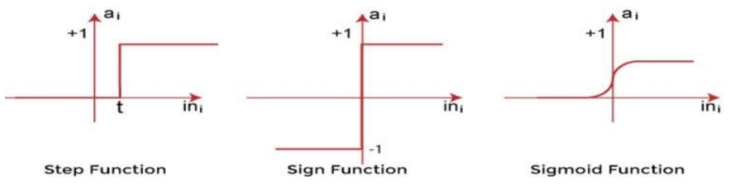
Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- Activation Function:

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function



Based on the layers, Perceptron models are divided into two types. These are as follows:
1. Single Layer Perceptron Model:This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

2. Multi-Layered Perceptron Model:Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.
The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- Forward Stage: Activation functions start from the input layer in the forward stage and terminate on the output layer.
- Backward Stage: In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

ALGORITHM:

The perceptron learning rule was originally developed by Frank Rosenblatt in the late 1950s. Training patterns are presented to the network's inputs; the output is computed. Then the connection weightsw$_j$are modified by an amount that is proportional to the product of

- the difference between the actual output, y, and the desired output, d,and
- the input pattern, x.

The algorithm is as follows:

1. Initialize the weights and threshold to small random numbers.
2. Present a vector x to the neuron inputs and calculate the output.
3. Update the weights according to:
   where
   · d is the desiredoutput,
   · t is the iteration number,and
   · eta is the gain or step size, where 0.0 < n <1.0
4. Repeat steps 2 and 3 until:

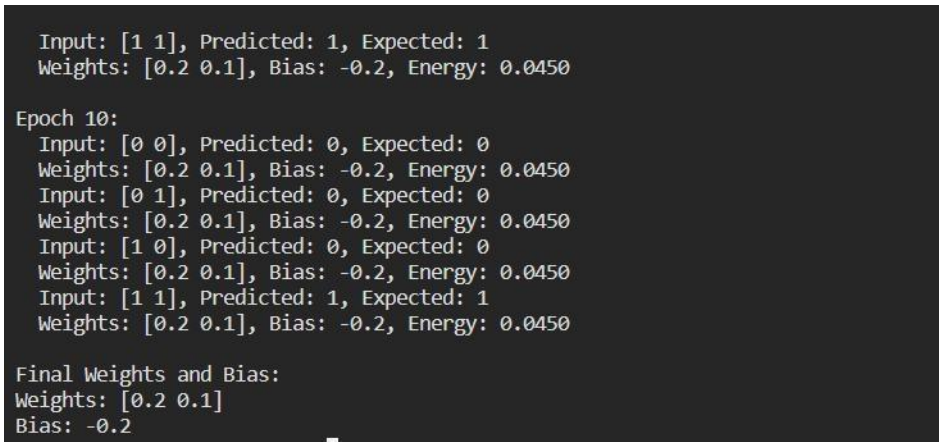
1. the iteration error is less than a user-specified error thresholder
2. a predetermined number of iterations have been completed.

*Learning only occurs when an error is made; otherwise the weights are left unchanged.

SOURCE CODE:

```
import numpy as np

def perceptron_and_gate():
    # Input data for AND gate (X1, X2) and expected output (Y)
    inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    expected_output = np.array([0, 0, 0, 1])  # AND gate truth table

    # Initialize weights, bias, and learning rate
    weights = np.array([0.1, 0.1])  # Initial weights
    bias = 0.2  # Initial bias (threshold)
    learning_rate = 0.1

    # Function to calculate energy
    def calculate_energy(weights, bias):
        return 0.5 * np.sum(weights**2) + 0.5 * bias**2

    # Perceptron learning algorithm
    for epoch in range(10):  # Run for a fixed number of epochs
        print(f"Epoch {epoch + 1}:")
        for i, input_vector in enumerate(inputs):
            linear_output = np.dot(input_vector, weights) + bias
            predicted_output = 1 if linear_output > 0 else 0

            # Calculate error
            error = expected_output[i] - predicted_output

            # Update weights and bias
            weights += learning_rate * error * input_vector
            bias += learning_rate * error

            # Display state
            print(f"  Input: {input_vector}, Predicted: {predicted_output}, Expected:
{expected_output[i]}")
            print(f"  Weights: {weights}, Bias: {bias}, Energy: {calculate_energy(weights, bias):.4f}")

        print()

    print("Final Weights and Bias:")
    print(f"Weights: {weights}")
    print(f"Bias: {bias}")

# Run the Perceptron Learning Algorithm for an AND gate
perceptron_and_gate()
```

SCREENSHOT OF OUTPUTS:



```
  Input: [1 1], Predicted: 1, Expected: 1
  Weights: [0.2 0.1], Bias: -0.2, Energy: 0.0450

Epoch 10:
  Input: [0 0], Predicted: 0, Expected: 0
  Weights: [0.2 0.1], Bias: -0.2, Energy: 0.0450
  Input: [0 1], Predicted: 0, Expected: 0
  Weights: [0.2 0.1], Bias: -0.2, Energy: 0.0450
  Input: [1 0], Predicted: 0, Expected: 0
  Weights: [0.2 0.1], Bias: -0.2, Energy: 0.0450
  Input: [1 1], Predicted: 1, Expected: 1
  Weights: [0.2 0.1], Bias: -0.2, Energy: 0.0450

Final Weights and Bias:
Weights: [0.2 0.1]
Bias: -0.2
```

LEARNING OUTCOME:

1. Understand the Perceptron Algorithm: Gain a clear understanding of how the perceptron algorithm updates weights and biases iteratively to minimize error and achieve convergence.
2. Implement Logical Gates with Machine Learning: Demonstrate the ability to implement a perceptron to simulate basic logical operations like the AND gate, reinforcing the foundational concepts of neural networks.
3. Apply Learning Parameters Effectively: Learn the impact of initial weights, bias, and the learning rate on the training process and the convergence behavior of the perceptron model.
4. Analyze Energy Function: Develop the ability to calculate and analyze the energy (or cost) function to monitor the optimization process and understand the model's progression during training