|  |  |
| --- | --- |
| **Batch: C1 Roll No.: 16010122221**  **Experiment No. : 04**  **Grade: AA / AB / BB / BC / CC / CD /DD**  **Signature of the Staff In-charge with date** | |
| **Title:** Perceptron net for an AND function with bipolar inputs and targets. | |
|  | |
| **Objective:** To write a program to implement the perceptron learning rule | |
| **Expected Outcome of Experiment:**  CO2 : Understand perceptron’s and counter propagation networks | |
| **Books/ Journals/ Websites referred:** |  |
| **Pre Lab/ Prior Concepts:**  **Learning**   * **Definition**: Learning in the context of neural networks refers to the process of adjusting weights based on input data to improve the performance of the network on a given task.   **Types of learning**   * + **Supervised Learning**: The model is trained on a labeled dataset, where the correct output is provided, and the model learns by comparing its predictions to the actual labels. Perceptron learning is an example of supervised learning.   + **Unsupervised Learning**: The model learns from an unlabeled dataset by identifying patterns and structures within the input data.   + **Reinforcement Learning**: The model learns by interacting with an environment and receiving feedback in the form of rewards or penalties. | |



# Perceptron learning rule.

* Overview: The perceptron learning rule is used to adjust the weights and bias in a perceptron. The rule is applied iteratively during the training process to minimize the difference between the actual output and the desired output.
* Learning Rule:
  + For each training example, the perceptron calculates the output by taking the weighted sum of inputs plus the bias. If the output matches the target, no changes are made. If there is a difference (error), the weights and bias are adjusted using the following formulas:
    - wi ← wi + Δwi
    - Δwi = η ⋅ (target−output) ⋅inputi
    - bias ← bias + η ⋅ (target−output)
  + Here, η is the learning rate, which controls how much the weights are adjusted per update.

# Steps of Perceptron learning algorithm/approach for binary classification

* *Step 1: Initialization*
  + Initialize the weights and bias to small random values or zeros.
* *Step 2: Input Processing*
  + For each input vector in the dataset, calculate the net input by taking the dot product of the input vector and the weight vector, then adding the bias.
* *Step 3: Activation Function*
  + Apply an activation function to the net input to produce the output. For binary classification, a common activation function is the step function:
    - Output = 1 if net input ≥ 0
    - Output = -1 (or 0 in binary) if net input < 0
* *Step 4: Weight and Bias Update*
  + If the output does not match the target, update the weights and bias using the perceptron learning rule.



* *Step 5: Iteration*
  + Repeat the process for multiple epochs (iterations over the entire dataset) until the weights converge (i.e., no more significant changes in weights) or a specified number of epochs is reached.
* *Step 6: Final Model*
  + After training, the final weights and bias can be used to classify new input data.

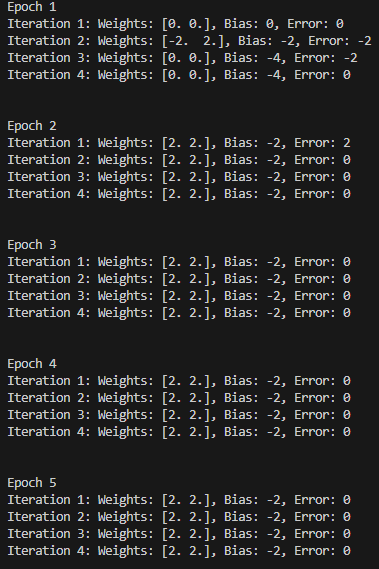
# Single layer perceptron network for AND logic function

* Network Structure: A single-layer perceptron consists of an input layer and an output layer with no hidden layers. The input layer receives the features, and the output layer produces the final prediction.
* AND Function: The AND logic function is a binary operation where the output is true (1) if and only if both inputs are true (1). For bipolar inputs and targets, the inputs and outputs are usually encoded as 1 and -1 instead of 1 and 0.
  + Example:
    - Input: [1, 1] → Output: 1
    - Input: [1, -1] → Output: -1
    - Input: [-1, 1] → Output: -1
    - Input: [-1, -1] → Output: -1
* Training: The perceptron is trained on these inputs and corresponding outputs to adjust the weights such that the network accurately models the AND function.

|  |  |
| --- | --- |
| **K. J. Somaiya College of Engineering, Mumbai-77**  **Department of Computer Engineering** | |
| **Implementation Details:**  **Task to be done:**  Write a program to implement a Perceptron network for the AND logic function using bipolar inputs and targets. Test the Perceptron with different learning rates and initial weights. Additionally, explore multiple epochs until the weights converge  #insert the code and snapshot of the output here  ***CODE :*** | |
| **Department of Computer Engineering**  Page No : SC/ Sem V/ 2024 |  |

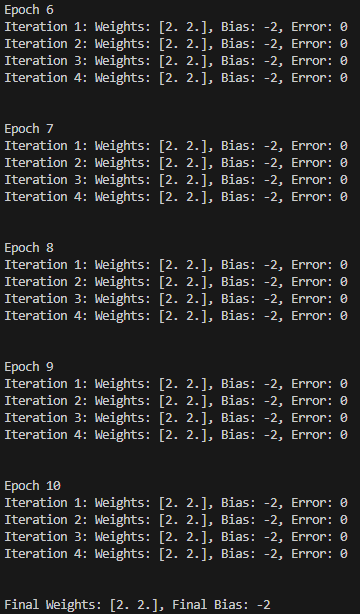






## OUTPUT :





.



# Conclusion:

The experiment successfully implements a perceptron for the AND logic function using bipolar inputs and targets. The perceptron learning rule effectively updates the weights and bias over multiple epochs until the network can correctly classify all input patterns.

# Post Lab Descriptive Questions :

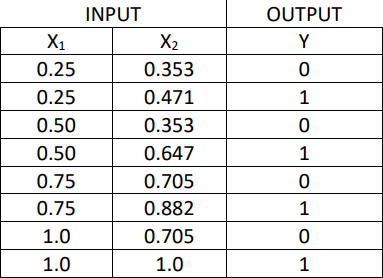
## How is the linear separability concept implemented using perceptron network

The perceptron works on the principle of linear separability, meaning it can only solve problems where the classes can be separated by a single hyperplane. If a dataset is linearly separable, the perceptron algorithm will find a solution where the classes are divided by a straight line (or hyperplane in higher dimensions). For the AND logic function, the perceptron successfully separates the output classes with appropriate weights and bias.

## Mention the application of the perceptron network.

The perceptron is the foundation of neural networks and is used in simple binary classification tasks. Applications include early image recognition systems, spam detection, and other binary classification problems where the classes are linearly separable.

1. ***The following is a training set for a 2-class (as 0 and 1) classification problem. Iterate the perception using the perceptron learning algorithm through the training set and obtain the weights. You may make a reasonable assumption if any.***





# Perceptron Learning Algorithm:

**Initialize Weights and Bias:**

Assume initial weights w1=0, w2=0 and bias b=0. Learning Rate η=1 (commonly chosen value).

# Activation Function:

If net input w1⋅x1+w2⋅x2+b≥0, output = 1. Otherwise, output = 0.

# Weight and Bias Updates:

If error (target - output) exists, update the weights and bias using: wi=wi+η⋅(target−output)⋅xi

# Iterations:

**Epoch 1:**

* + **Input [0.25, 0.353], Target: 0** Net input = 0⋅0.25+0⋅0.353+0=0 Output = 1 (since net input ≥0) Error = 0 - 1 = -1

Update weights: w1=0+1⋅(−1)⋅0.25=−0.25 w2=0+1⋅(−1)⋅0.353=−0.353 b=0+1⋅(−1)=−1

# Input [0.25, 0.471], Target: 0

Net input = −0.25⋅0.25+(−0.353)⋅0.471−1=−1.20675

Output = 0 (since net input < 0) Error = 0 - 0 = 0 (No update needed)

# Input [0.50, 0.353], Target: 0

Net input = −0.25⋅0.50+(−0.353)⋅0.353−1=−1.245409

Output = 0 (since net input < 0) Error = 0 - 0 = 0 (No update needed)

# Input [0.50, 0.647], Target: 1

Net input = −0.25⋅0.50+(−0.353)⋅0.647−1=−1.418191

Output = 0 (since net input < 0) Error = 1 - 0 = 1



Update weights: w1=−0.25+1⋅1⋅0.50=0.25 w2=−0.353+1⋅1⋅0.647=0.294 b=−1+1=0

# Input [0.75, 0.705], Target: 1

Net input = 0.25⋅0.75+0.294⋅0.705+0=0.456030.25

Output = 1 (since net input ≥0) Error = 1 - 1 = 0 (No update needed)

# Input [0.75, 0.882], Target: 1

Net input = 0.25⋅0.75+0.294⋅0.882+0=0.5155380.

Output = 1 (since net input ≥0) Error = 1 - 1 = 0 (No update needed)

# Input [1.0, 0.705], Target: 1

Net input = 0.25⋅1.0+0.294⋅0.705+0=0.457370.25

Output = 1 (since net input ≥0) Error = 1 - 1 = 0 (No update needed)

* + **Input [1.0, 1.0], Target: 1**

Net input = 0.25⋅1.0+0.294⋅1.0+0=0.5440.25

Output = 1 (since net input ≥0) Error = 1 - 1 = 0 (No update needed)

# Final Weights and Bias:

Weights: w1=0.25 w2=0.294

Bias: b=0

.

# Date: Signature of faculty in-charge