

| **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**

**Types of learning**

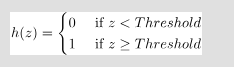
**Perceptron learning rule.**

Steps of Perceptron learning algorithm/approach for binary classification

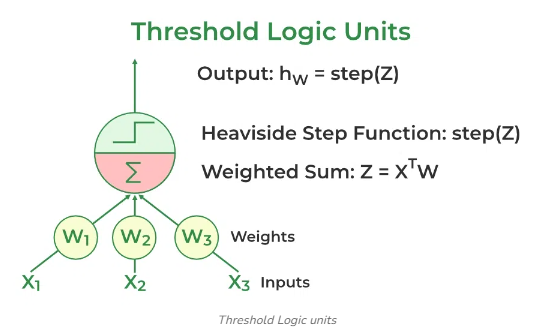
A weight is assigned to each input node of a perceptron, indicating the significance of that input to the output. The perceptron’s output is a weighted sum of the inputs that have been run through an activation function to decide whether or not the perceptron will fire. it computes the weighted sum of its inputs as:

z = w1x1 + w1x2 + ... + wnxn = XTW

The step function compares this weighted sum to the threshold, which outputs 1 if the input is larger than a threshold value and 0 otherwise, is the activation function that perceptrons utilize the most frequently. The most common step function used in perceptron is the Heaviside step function:



A perceptron has a single layer of **threshold logic units**with each TLU connected to all inputs.



When all the neurons in a layer are connected to every neuron of the previous layer, it is known as a fully connected layer or dense layer.

The output of the fully connected layer can be:



where X is the input W is the weight for each inputs neurons and b is the bias and h is the step function.

During training, The perceptron’s weights are adjusted to minimize the difference between the predicted output and the actual output. Usually, supervised learning algorithms like the delta rule or the perceptron learning rule are used for this.

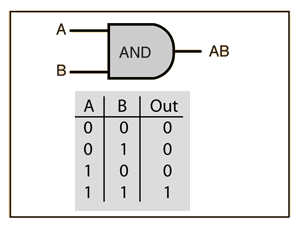


Here wi,j is the weight between the ith input and jth output neuron, xi is the ith input value, and yj and  is the jth actual and predicted value is  the learning rate.

Single layer perceptron network for AND logic function

**AND Gate**

From our knowledge of logic gates, we know that an AND logic table is given by the diagram below



The question is, what are the weights and bias for the AND perceptron?

First, we need to understand that the output of an AND gate is 1 only if both inputs (in this case, x1 and x2) are 1. So, following the steps listed above;

**Row 1**

* From w1\*x1+w2\*x2+b, initializing w1, w2, as 1 and b as –1, we get;

*x1(1)+x2(1)–1*

* Passing the first row of the AND logic table (x1=0, x2=0), we get;

*0+0–1 = –1*

* From the Perceptron rule, if Wx+b≤0, then y`=0. Therefore, this row is correct, and no need for Backpropagation.

**Row 2**

* Passing (x1=0 and x2=1), we get;

*0+1–1 = 0*

* From the Perceptron rule, if Wx+b≤0, then y`=0. This row is correct, as the output is 0 for the AND gate.
* From the Perceptron rule, this works (for both row 1, row 2 and 3).

**Row 4**

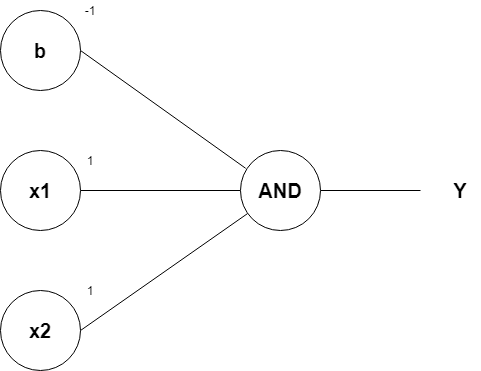
* Passing (x1=1 and x2=1), we get;

*1+1–1 = 1*

* Again, from the perceptron rule, this is still valid.

Therefore, we can conclude that the model to achieve an AND gate, using the Perceptron algorithm is;

*x1+x2–1*



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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

import numpy as np

# Step Activation Function (bipolar)

def step\_function(weighted\_sum):

    return 1 if weighted\_sum >= 0 else -1

# Perceptron Training Function

def train\_perceptron(X, Y, learning\_rate, epochs, initial\_weights, initial\_bias):

    weights = np.array(initial\_weights)

    bias = initial\_bias

    print("\nInitial Weights:", weights)

    print("Initial Bias:", bias)

    for epoch in range(epochs):

        print(f"\nEpoch {epoch + 1}")

        for i in range(X.shape[0]):

            # Calculate weighted sum

            weighted\_sum = np.dot(X[i], weights) + bias

            # Get the predicted output using the step function

            y\_pred = step\_function(weighted\_sum)

            # Update weights and bias based on the error

            error = Y[i] - y\_pred

            weights += learning\_rate \* error \* X[i]

            bias += learning\_rate \* error

            print(f"Input: {X[i]}, Target: {Y[i]}, Predicted: {y\_pred}, Error: {error}")

            print(f"Updated Weights: {weights}")

            print(f"Updated Bias: {bias}")

    return weights, bias

# Testing the trained perceptron

def test\_perceptron(X, weights, bias):

    results = []

    for i in range(X.shape[0]):

        weighted\_sum = np.dot(X[i], weights) + bias

        y\_pred = step\_function(weighted\_sum)

        results.append((X[i], y\_pred))

    return results

# Define the inputs and targets for the bipolar AND logic function

X = np.array([[-1, -1],

              [-1,  1],

              [ 1, -1],

              [ 1,  1]])

Y = np.array([-1, -1, -1, 1])  # Bipolar targets

# User inputs for learning rate, epochs, initial weights, and bias

learning\_rate = float(input("Enter the learning rate (e.g., 0.1): "))

epochs = int(input("Enter the number of epochs: "))

initial\_weights = [float(x) for x in input("Enter initial weights separated by commas (e.g., 0.5,-0.5): ").split(',')]

initial\_bias = float(input("Enter initial bias (e.g., 0.0): "))

# Train the perceptron

final\_weights, final\_bias = train\_perceptron(X, Y, learning\_rate, epochs, initial\_weights, initial\_bias)

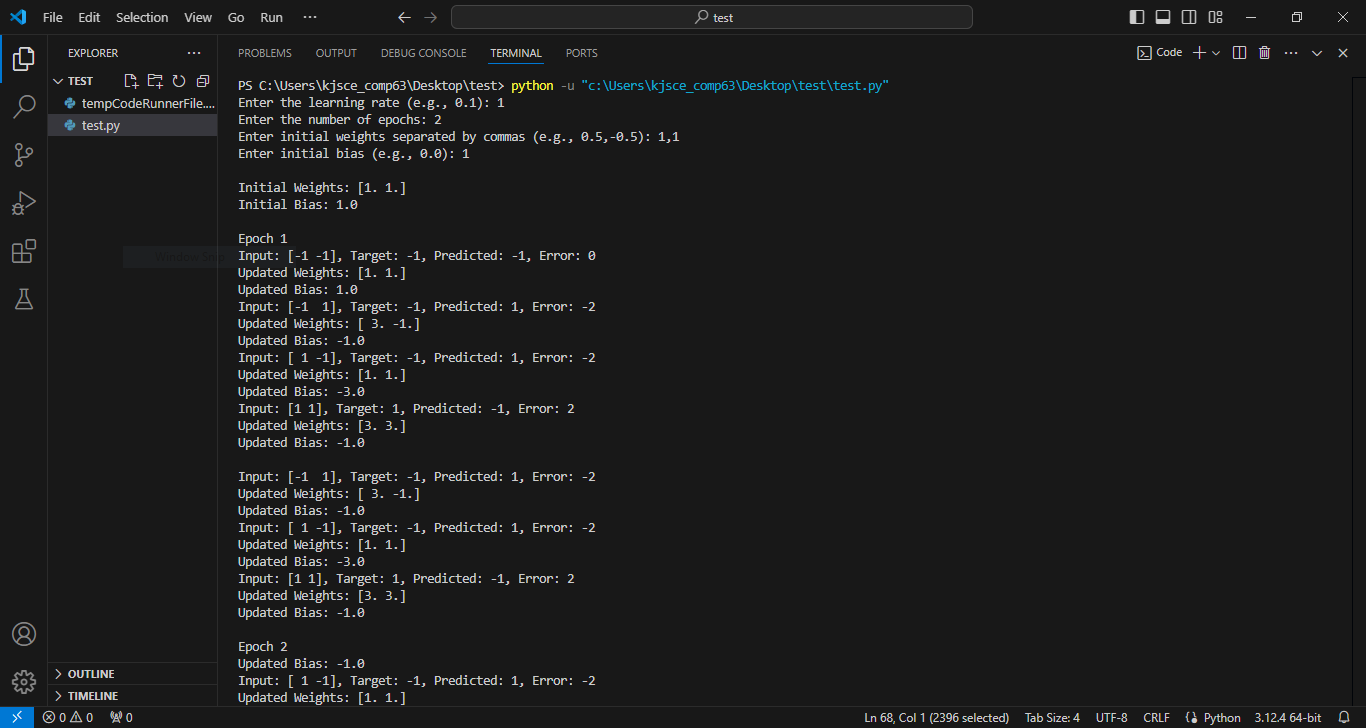
# Test the perceptron

print("\nTesting the Perceptron:")

results = test\_perceptron(X, final\_weights, final\_bias)

for x, y\_pred in results:

    print(f"Input: {x}, Predicted Output: {y\_pred}")



**Conclusion:**

**Post Lab Descriptive Questions :**

1. How is the linear separability concept implemented using perceptron network.

**Ans:**

**Concept of Linear Separability: Linear separability refers to the ability to separate two classes of data points using a linear decision boundary (a hyperplane). For a dataset to be linearly separable, there must exist a straight line (in two dimensions), a plane (in three dimensions), or a hyperplane (in higher dimensions) that can divide the data points into two distinct classes.**

**Implementation in Perceptron Network:**

* **A perceptron is a simple linear classifier that models a binary classification problem. It attempts to find a hyperplane that separates the input data into two classes.**
* **The perceptron learns by iteratively adjusting its weights based on the input features and their corresponding class labels. The learning rule is based on minimizing the error between the predicted output and the actual class label.**
* **During training, if the input data is linearly separable, the perceptron will eventually find the correct weights that define the decision boundary, ensuring that all points of one class are on one side of the boundary and all points of the other class are on the opposite side.**

1. Mention the application of the perceptron network.

**Ans:**

**Binary Classification:**

* Perceptrons are widely used for binary classification tasks where the goal is to classify input data into one of two categories (e.g., spam vs. non-spam emails).

**Pattern Recognition:**

* They are used in recognizing patterns within data, such as handwriting recognition, where the input features represent pixel intensities of characters.

**Decision Making:**

* In robotics and automation, perceptrons can be used for decision-making processes where the input features guide the robot's actions based on certain rules.

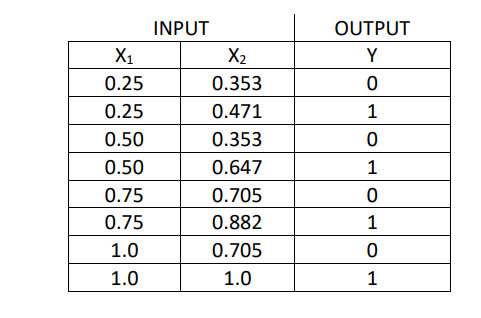
**Feature Selection:**

* Perceptrons can be used to determine the importance of various features in a dataset by analyzing the learned weights after training.

**Signal Processing:**

* In signal processing, perceptrons can be used to classify different types of signals or to filter out noise from relevant data.

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.



**Code:**

import numpy as np

# Training data

X = np.array([[0.25, 0.353],

              [0.25, 0.471],

              [0.50, 0.353],

              [0.50, 0.647],

              [0.75, 0.705],

              [0.75, 0.882],

              [1.0, 0.705],

              [1.0, 1.0]])

# Corresponding class labels

Y = np.array([0, 1, 0, 1, 0, 1, 0, 1])

# Initialize weights and bias

W = np.array([0.0, 0.0])

bias = 0.0

# Learning rate

learning\_rate = 0.1

# Function to predict output

def predict(X, W, bias):

    weighted\_sum = np.dot(X, W) + bias

    return 1 if weighted\_sum >= 0 else 0

# Perceptron Learning Algorithm

for epoch in range(10):  # Assuming 10 epochs

    print(f"\nEpoch {epoch + 1}")

    for i in range(X.shape[0]):

        prediction = predict(X[i], W, bias)

        error = Y[i] - prediction

        W += learning\_rate \* error \* X[i]

        bias += learning\_rate \* error

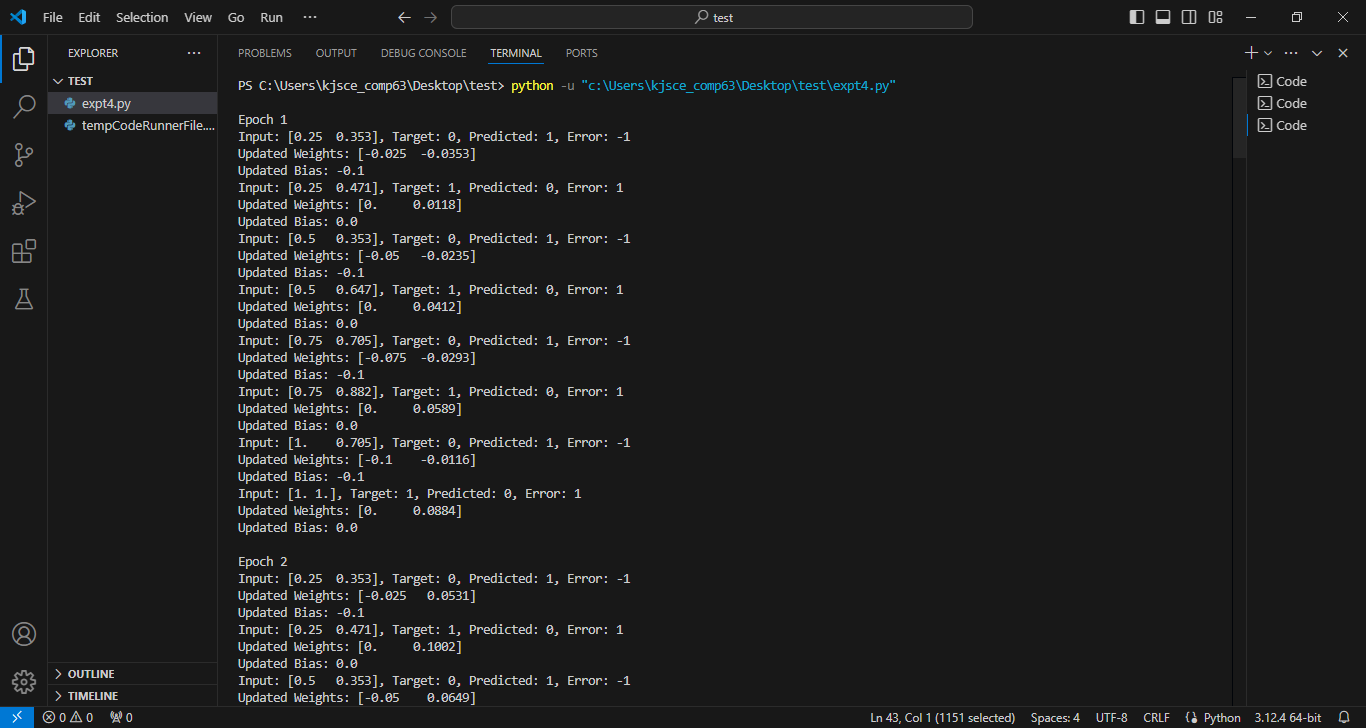
        print(f"Input: {X[i]}, Target: {Y[i]}, Predicted: {prediction}, Error: {error}")

        print(f"Updated Weights: {W}")

        print(f"Updated Bias: {bias}")

print("\nFinal Weights:", W)

print("Final Bias:", bias)



.

**Date: 20/08/2024 Signature of faculty in-charge**