

| **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** in the context of machine learning refers to the process by which a system (algorithm, model) improves its performance on a task based on experience (data). There are different types of learning:

**Types of learning**

Supervised Learning: In supervised learning, the algorithm is trained on a labeled dataset, where the input data is paired with the correct output or target. The goal is to learn a mapping from inputs to outputs.

Unsupervised Learning: Unsupervised learning involves learning from unlabeled data. The algorithm tries to find patterns, structures, or relationships within the data without explicit target values.

Reinforcement Learning: Reinforcement learning involves an agent that learns to interact with an environment to maximize a reward signal. The agent takes actions and learns from the consequences of those actions.

**Perceptron learning rule.**

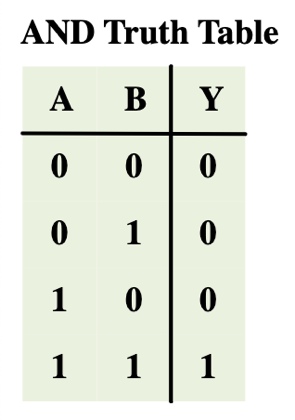
The Perceptron Learning Rule is an algorithm for training a single-layer perceptron, which is a simple neural network used for binary classification. The perceptron learning rule aims to adjust the weights of the perceptron's inputs to correctly classify input data into one of two classes.

Steps of Perceptron learning algorithm/approach for binary classification

* 1. Initialization: Initialize the weights and bias to small random values.
  2. Input Processing: For each input vector, append a bias term (usually set to 1) to the beginning of the input vector.
  3. Weighted Sum: Calculate the weighted sum of inputs and weights.
  4. Weighted Sum = (Weight 1 \* Input 1) + (Weight 2 \* Input 2) + ... + Bias \* Bias Weight
  5. Activation Function: Apply an activation function (often a step function) to the weighted sum. The perceptron output is 1 if the activation condition is met, otherwise, it's 0.
  6. Error Calculation: Calculate the error by subtracting the predicted output from the true target value.
  7. Weight Update: Update the weights and bias using the following update rule:
  8. New Weight = Old Weight + Learning Rate \* Error \* Input
  9. New Bias Weight = Old Bias Weight + Learning Rate \* Error \* Bias
  10. Repeat: Repeat steps 3-6 for a certain number of epochs or until the error becomes sufficiently small.
  11. Termination: Stop training when the error is minimized or after a fixed number of epochs.

Single layer perceptron network for each logic functions

**AND Logic:**

****

**Activation Function:** Step function (1 if weighted sum >= 0, else 0)

**Explanation:**

* For the AND logic, the perceptron uses a step function as the activation function.
* The weighted sum of inputs and weights is calculated.
* If the weighted sum is greater than or equal to 0, the perceptron outputs 1; otherwise, it outputs 0.

**OR Logic:**

**A white and black rectangular object with black letters

Description automatically generated with medium confidence**

**Activation Function:** Step function (1 if weighted sum >= 0, else 0)

**Explanation:**

* For the OR logic, similar to the AND logic, a step function is used as the activation function.
* The perceptron calculates the weighted sum of inputs and weights.
* If the weighted sum is greater than or equal to 0, the perceptron outputs 1; otherwise, it outputs 0.

**NOT Logic:**

**A white square with black text

Description automatically generated**

**Activation Function:** Step function (1 if weighted sum >= 0, else 0)

**Explanation:**

* For the NOT logic, a step function is also used as the activation function.
* The perceptron computes the weighted sum of the input and weight.
* If the weighted sum is greater than or equal to 0, the perceptron outputs 1; otherwise, it outputs 0.

In each case, the activation function determines the output of the perceptron based on the calculated weighted sum. The perceptron learning algorithm adjusts the weights and bias to minimize the error between the predicted output (determined by the activation function) and the target output. This iterative process helps the perceptron learn the correct parameters to classify inputs according to the specified logic function.

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

**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

Code:

import numpy as np

X = np.array([

[-1, -1],

[-1, 1],

[ 1, -1],

[ 1, 1]

])

y = np.array([-1, -1, -1, 1])

# Initialize weights and bias

weights = np.zeros(X.shape[1])

bias = 0

learning\_rate = 1

# Sign activation function

def activation\_function(x):

return 1 if x >= 0 else -1

# Perceptron learning algorithm

def train\_perceptron(X, y, weights, bias, learning\_rate, epochs):

for epoch in range(epochs):

print(f'Epoch {epoch+1}')

for inputs, target in zip(X, y):

# net input

net\_input = np.dot(inputs, weights) + bias

# output

output = activation\_function(net\_input)

# error

error = target - output

# Update

weights += learning\_rate \* error \* inputs

bias += learning\_rate \* error

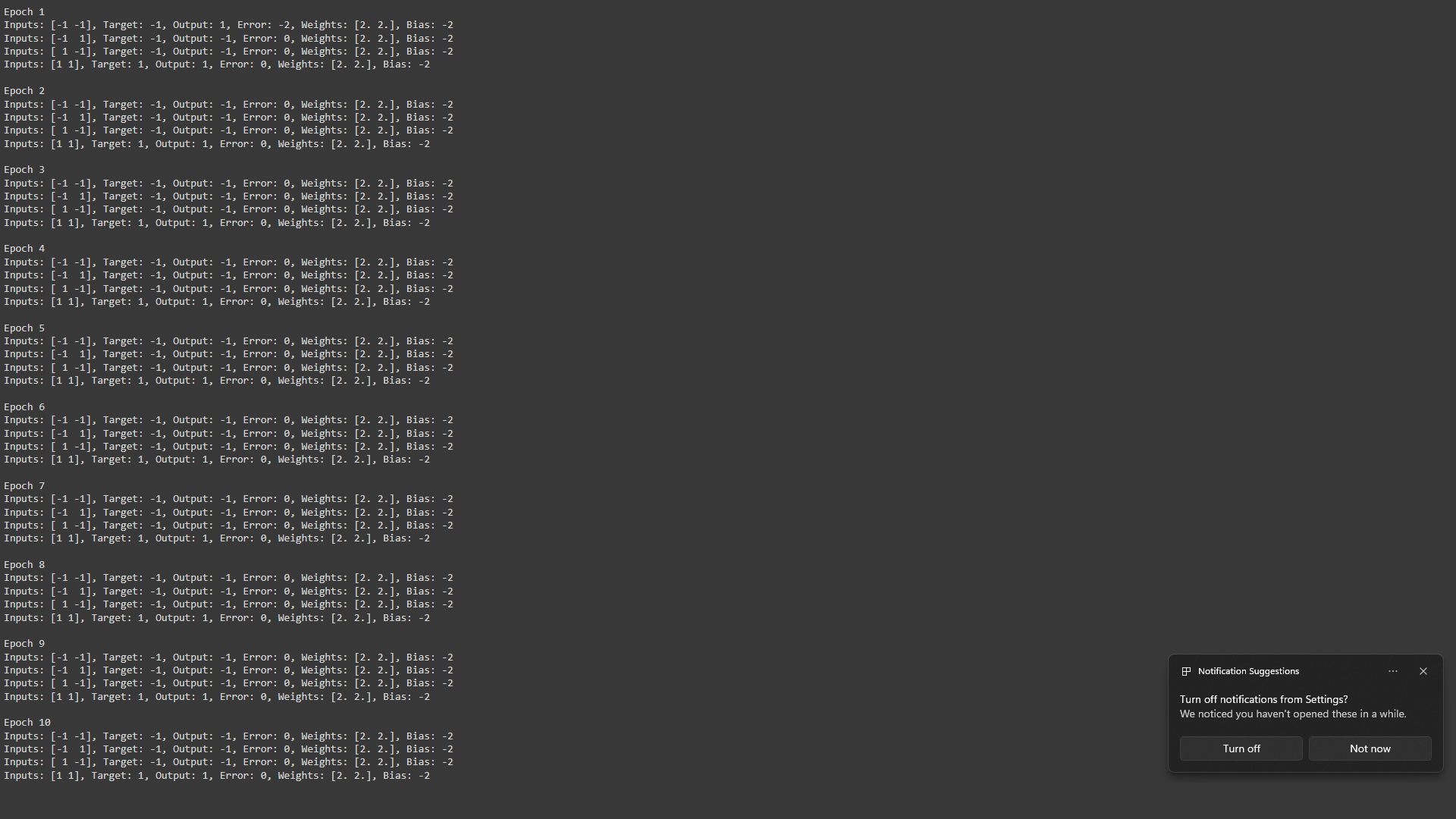
print(f'Inputs: {inputs}, Target: {target}, Output: {output}, Error: {error}, Weights: {weights}, Bias: {bias}')

print('')

# Training

epochs = 10

train\_perceptron(X, y, weights, bias, learning\_rate, epochs)



**Conclusion:** Learnt about perceptron learning network.

**Post Lab Descriptive Questions :**

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

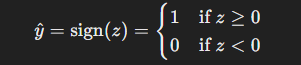
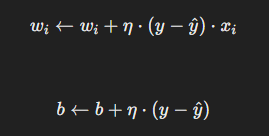
### Linear Separability and Perceptron Network

**Linear Separability Concept:**

Linear separability is the idea that a set of data points can be separated into different classes using a linear decision boundary. In the case of a binary classification problem, this means finding a straight line (or hyperplane in higher dimensions) that can divide the data into two distinct classes.

**Implementing Linear Separability with a Perceptron Network:**

A perceptron network is a simple type of neural network used to determine whether a dataset is linearly separable. It consists of a single layer of neurons and uses a linear decision boundary to classify input data. Here's how the perceptron network implements linear separability:

1. **Initialization:**
   * **Weights and Bias:** Start with small random values for the weights and bias of the perceptron.
2. **Forward Pass:**
   * **Linear Combination:** For each input vector x compute the weighted sum plus bias: 
   * **Activation Function:** Apply the activation function (usually a step function) to z to get the output. For binary classification, this is typically: 
   * **Update Rule:**
   * **Error Calculation:** Calculate the error by comparing the predicted output y^​ with the actual target y.
   * **Weight Adjustment:** Update the weights and bias based on the error using the learning rule:
   * 
   * where η is the learning rate.
3. **Iteration:**
   * Repeat the forward pass and update rule for each training example until the weights and bias converge, meaning the perceptron correctly classifies all training examples or a set number of epochs is reached.

By iteratively adjusting the weights and bias based on classification errors, the perceptron network attempts to find a linear boundary that separates the data points of different classes. If the dataset is linearly separable, the perceptron will eventually converge to a set of weights that correctly classifies all training examples.

1. Mention the application of the perceptron network.

### Applications of the Perceptron Network:

1. **Binary Classification Tasks:**
   * **Spam Detection:** Classifying emails as spam or not spam.
   * **Sentiment Analysis:** Determining whether the sentiment of a text is positive or negative.
2. **Pattern Recognition:**
   * **Handwritten Digit Recognition:** Identifying handwritten digits (though more complex networks are typically used for this now).
3. **Medical Diagnosis:**
   * **Disease Classification:** Classifying patient data to predict the presence or absence of diseases.
4. **Simple Object Recognition:**
   * **Basic Image Classification:** Identifying simple objects in images.
5. **Basic Speech Recognition:**
   * **Phoneme Classification:** Classifying spoken phonemes in a speech signal.

The perceptron network is a fundamental building block in machine learning and neural networks. While it is limited to linearly separable problems, it provides a foundation for understanding more complex models such as multi-layer perceptrons (MLPs) and deep neural networks.

.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**