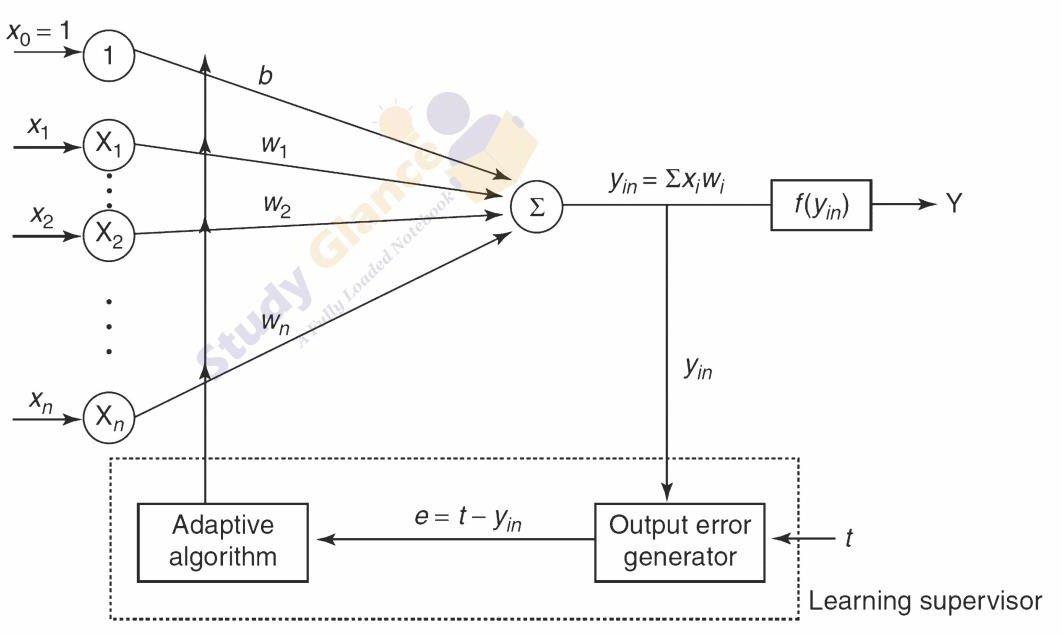
|  |  |  |  |
| --- | --- | --- | --- |
| **K. J. Somaiya College of Engineering, Mumbai-77**  **Department of Computer Engineering** | | | |
|  | **Batch: C1 Roll No.: 16010122221**  **Experiment No. 05**  **Grade: AA / AB / BB / BC / CC / CD /DD**  **Signature of the Staff In-charge with date** | |  |
| **Title:** Implementation of OR function with bipolar inputs and targets using Adaline network. Assume the required parameters for training of the network. | | |  |
|  | |  | |
| **Objective:** To learn Adaline network. | |
| **Expected Outcome of Experiment:**  CO2: To understand the features of neural networks and different learning methods. | |
| **Books/ Journals/ Websites referred:** | | | |
| **Pre Lab/ Prior Concepts:**  **Adaptive Linear Neuron (Adaline):**  Adaline which stands for Adaptive Linear Neuron, is a network having a single linear unit. It was developed by Widrow and Hoff in 1960.  Some important points about Adaline are as follows −   * It uses bipolar activation function. * It tries to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.   The weights and the bias are adjustable  **Architecture:** | | | |



# Algorithm:

Step 1: Initialize the following to start the training −Weights, Bias, Learning rate *α*

Step 2: While the stopping condition is False do steps 3 to 7. Step 3: for each training set perform steps 4 to 6.

Step 4: Set activation of input unit xi = si for (i=1 to n).

Step 5: compute net input to output unit 

Here, b is the bias and n is the total number of neurons.

Step 6: Update the weights and bias for i=1 to n



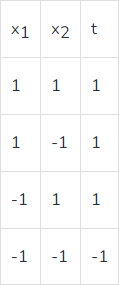
and calculate

Step 7: Test the stopping condition. The stopping condition may be when the weight changes at a low rate or no change.



# Implementation Details:

Implementation of OR function with bipolar inputs and targets using Adaline network.



***Code:***

import numpy as np

def bipolar\_activation(x):

return 1 if x >= 0 else -1

def net\_input(weights, inputs, bias):

return np.dot(weights, inputs) + bias

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

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

weights = np.random.rand(2)

bias = np.random.rand(1)

learning\_rate = 0.1

epochs = 10

for epoch in range(epochs):

print(f"Epoch {epoch + 1}/{epochs}")

for input\_vec, target in zip(inputs, targets):

net = net\_input(weights, input\_vec, bias)

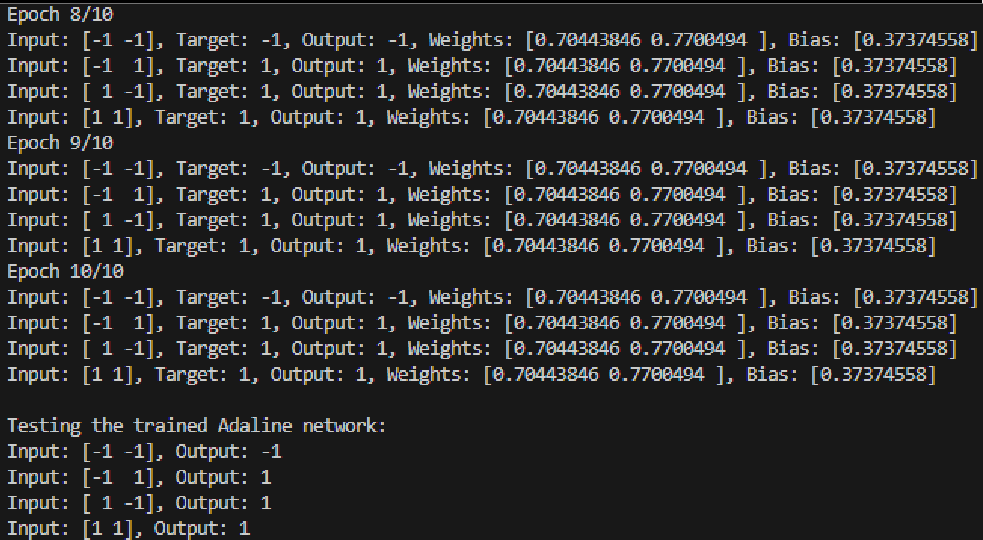
output = bipolar\_activation(net)

error = target - output

weights += learning\_rate \* error \* input\_vec

bias += learning\_rate \* error

print(f"Input: {input\_vec}, Target: {target}, Output: {output}, Weights: {weights}, Bias: {bias}")



print("Testing the trained Adaline network:")

for input\_vec in inputs:

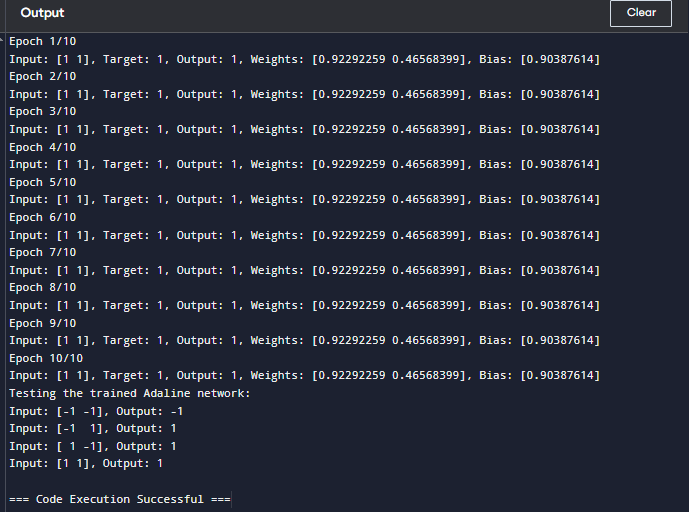
net = net\_input(weights, input\_vec, bias)

output = bipolar\_activation(net)

print(f"Input: {input\_vec}, Output: {output}")



***Output:***





**Conclusion:** Summarize the results of your experiment. Discuss whether the Adaline network successfully implemented the OR function, any observations about the learning process, and how well the network minimized the Mean-Squared Error (MSE). You could also reflect on the effectiveness of the Adaline network for this task and potential improvements.

# Post Lab Descriptive Questions:

Use Adaline network to train AND NOT function with bipolar inputs and targets. Perform 1 epoch of training

***Code:***

***import numpy as np***

***def bipolar\_activation(x):***

***return 1 if x >= 0 else -1***

***def net\_input(weights, inputs, bias):***

***return np.dot(weights, inputs) + bias***

***inputs\_and\_not = np.array([[1, -1], [-1, 1], [1, 1], [1, 1]])***

***targets\_and\_not = np.array([1, -1, -1, 1])***

***weights\_and\_not = np.random.rand(2)***

***bias\_and\_not = np.random.rand(1)***

***learning\_rate = 0.1***

***for input\_vec, target in zip(inputs\_and\_not, targets\_and\_not):***

***net = net\_input(weights\_and\_not, input\_vec, bias\_and\_not)***

***output = bipolar\_activation(net)***

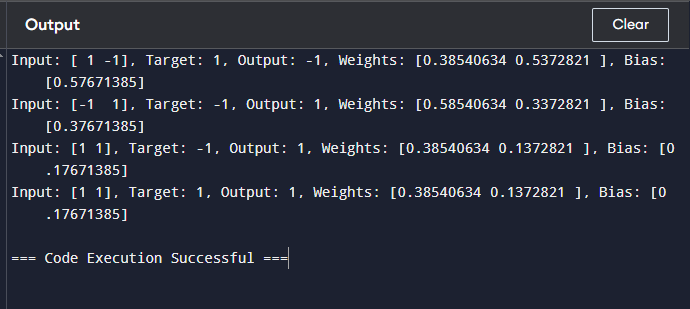
***error = target - output***

***weights\_and\_not += learning\_rate \* error \* input\_vec***

***bias\_and\_not += learning\_rate \* error***

***print(f"Input: {input\_vec}, Target: {target}, Output: {output}, Weights: {weights\_and\_not}, Bias: {bias\_and\_not}")***

***Output:***



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