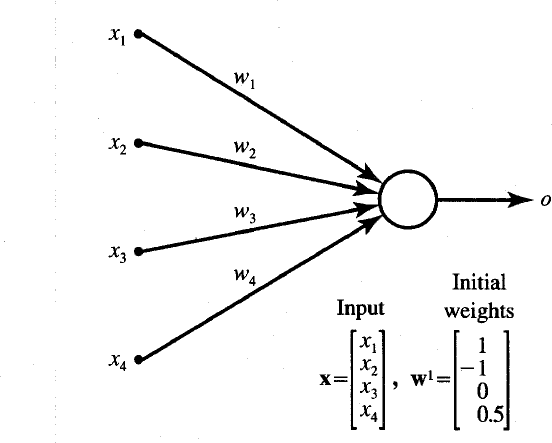
|  |  |  |  |
| --- | --- | --- | --- |
| **K. J. Somaiya College of Engineering, Mumbai-77**  **Department of Computer Engineering** | | | |
|  | **Batch: C1 Roll No.: 16010122221**  **Experiment No. 03**  **Grade: AA / AB / BB / BC / CC / CD /DD**  **Signature of the Staff In-charge with date** | |  |
| **Title: Implement Hebbian learning Rule** | | |  |
|  | |  | |
| **Objective:** To implement Hebbian learning with binary bipolar activation functions.. | |
| **Expected Outcome of Experiment:**  CO3: Understand perceptron’s and counter-propagation networks | |
| **Books/ Journals/ Websites referred:** | | | |
| **Pre Lab/ Prior Concepts:**  **Hebbian Learning Rule:**  This is a learning algorithm that updates the synaptic weights based on the principle that "cells that fire together, wire together." In this context, it is necessary to understand the fundamental concepts of neural networks, specifically perceptrons and counter-propagation networks. The experiment aims to solidify your understanding of how Hebbian learning can be implemented and its effect when using different activation functions. | | | |
|  | | | |



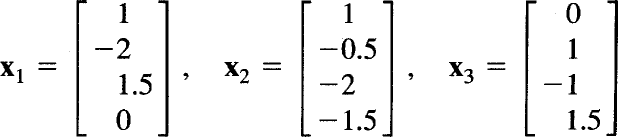
**Implementation Details:**

**#** Implement Hebbian learning for the given network

#Compare the learning using Binary and Bipolar activation functions.

**Network for implementation**

**Input vector:**





**CODE:**

import numpy as np

def binary\_activation(x):

return 1 if x > 0 else 0

def bipolar\_activation(x):

return 1 if x > 0 else -1

def hebbian\_learning(input\_vectors, activation\_function, initial\_weights):

weights = initial\_weights.copy()

print("Initial weights:", weights)

learning\_rate = 1

for i, x in enumerate(input\_vectors):

net\_input = np.dot(weights, x)

output = activation\_function(net\_input)

weights += learning\_rate \* output \* x

print(f"Iteration {i+1} - Input: {x}, Net input: {net\_input}, Output: {output}, Updated weights: {weights}")

return weights

input\_vectors = np.array([

[1, -2],

[2, 3],

[1, -1]

])

initial\_weights = np.array([1, -1])

print("Hebbian Learning with Binary Activation Function:")

final\_weights\_binary = hebbian\_learning(input\_vectors, binary\_activation, initial\_weights)

print("Final weights (Binary):", final\_weights\_binary)

print("\n")

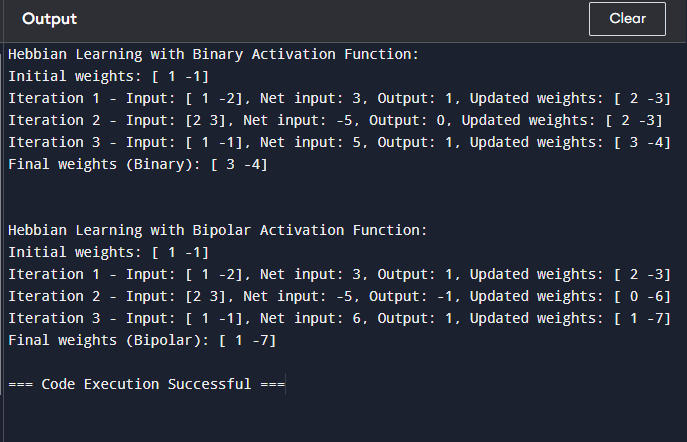
print("Hebbian Learning with Bipolar Activation Function:")

final\_weights\_bipolar = hebbian\_learning(input\_vectors, bipolar\_activation, initial\_weights)

print("Final weights (Bipolar):", final\_weights\_bipolar)



**OUTPUT:**



1. **Activation Functions:**
   * binary\_activation(x): Returns 1 if the net input is positive, otherwise 0.
   * bipolar\_activation(x): Returns 1 if the net input is positive, otherwise -1.
2. **Hebbian Learning Process:**
   * The initial weights are set to [1, -1].
   * For each input vector, the net input (dot product of weights and input vector) is computed.
   * The activation function (Binary or Bipolar) is applied to determine the output.
   * The weights are then updated according to the Hebbian rule: Wnew=Wold+C×output×XW\_{\text{new}} = W\_{\text{old}} + C \times

\text{output} \times XWnew=Wold+C×output×X

* + This process is repeated for each input vector.

1. **Comparison:**
   * The code compares the final weights obtained after applying Hebbian learning using both Binary and Bipolar activation functions. By running the code, you will observe how the choice of activation function impacts the learning process and the final set of weights.



**Conclusion:**

Hebbian learning with binary activation functions results in more abrupt and less stable weight updates due to the discrete nature of the output. In contrast, bipolar activation functions provide smoother and more stable weight adjustments because they capture both positive and negative signals. Consequently, bipolar activation tends to offer better convergence and effectiveness in learning tasks compared to binary activation.

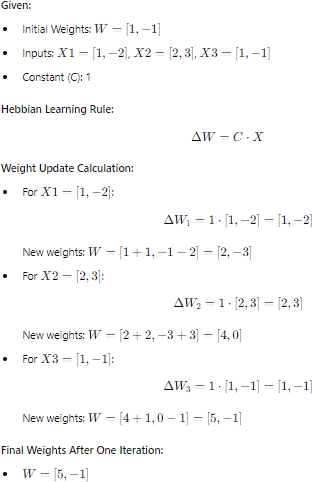
**Post Lab Descriptive Questions :**

1. ***Compare the Hebbian learning and competitive learning.***

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Hebbian Learning** | **Competitive Learning** |
| Principle | Strengthens connections based on  co-activation. | Only the winning neuron updates  weights. |
| Update | All neurons’ weights are updated. | Only the winning neuron’s weights are  updated. |
| Activation | Can use binary or bipolar functions. | Uses a winner-takes-all approach. |
| Convergence | Slower and less stable. | Generally faster and more stable. |
| Application | Learning associations and patterns. | Clustering and pattern recognition. |



1. ***Find the weights after one iteration for hebbian learning of a single neuron network. Start with initial weights W=[1,-1] and Inputs as X1=[1,-2] X2=[2,3], x3[1,-1] and C=1.***



.

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