**ML EXPERIMENT 7**

import numpy as np

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

def hebbian\_learning(inputs, targets, learning\_rate=0.1, num\_iterations=100):

    # Initialize weights and bias to zero

    num\_inputs = inputs.shape[1]

    weights = np.zeros(num\_inputs)

    bias = 0

    for \_ in range(num\_iterations):

        for i in range(len(inputs)):

            # Set activations for input units

            activations = inputs[i]

            # Set the corresponding output value to the target

            output = targets[i]

            # Update weights and bias using Hebbian learning rule

            weights += learning\_rate \* np.dot(activations, output)

            bias += learning\_rate \* output

    return weights, bias

# Input vectors for OR gate

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

# Target output pairs for OR gate

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

# Apply Hebbian learning

weights, bias = hebbian\_learning(X, y)

# Print learned weights and bias

print("Learned weight for input 1 (w1):", weights[0])

print("Learned weight for input 2 (w2):", weights[1])

print("Learned bias:", bias)

# Create a directed graph

G = nx.DiGraph()

# Add nodes

G.add\_nodes\_from(['X1', 'X2', 'B', 'Y', 'Output'])

# Add edges

G.add\_edge('X1', 'Y', weight=weights[0])

G.add\_edge('X2', 'Y', weight=weights[1])

G.add\_edge('B', 'Y', weight=bias)

G.add\_edge('Y', 'Output')

# Draw the graph

pos = {'X1': (0, 1), 'X2': (0, 0), 'B': (0, -1), 'Y': (1, 0), 'Output': (2, 0)}

nx.draw(G, pos, with\_labels=True, node\_size=1000, node\_color='skyblue', font\_size=15, arrows=True)

labels = nx.get\_edge\_attributes(G, 'weight')

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)

plt.title('Hebbian Learning Graph')

plt.show()



