# **Aim:** Write a program for Hopfield Network.

import numpy as np

class HopfieldNetwork:

    def \_\_init\_\_(self, pattern\_size):

        self.pattern\_size = pattern\_size

        self.weights = np.zeros((pattern\_size, pattern\_size))

    def train(self, patterns):

        for i in range(self.pattern\_size):

            for j in range(i, self.pattern\_size):

                if i != j:

                    weight = 0

                    for pattern in patterns:

                        weight += pattern[i] \* pattern[j]

                    self.weights[i][j] = weight

                    self.weights[j][i] = weight

    def recall(self, pattern):

        for \_ in range(10):

            for i in range(self.pattern\_size):

                activation = 0

                for j in range(self.pattern\_size):

                    activation+=self.weights[i][j]\*pattern[j]

                pattern[i] = 1 if activation > 0 else -1

        return pattern

if \_\_name\_\_ == "\_\_main\_\_":

    pattern\_size = 9

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

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

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

    hopfield\_net = HopfieldNetwork(pattern\_size)

    hopfield\_net.train(patterns)

    for pattern in patterns:

        recalled\_pattern = hopfield\_net.recall(pattern.copy())

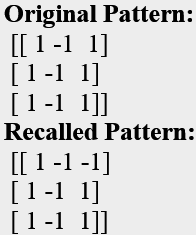
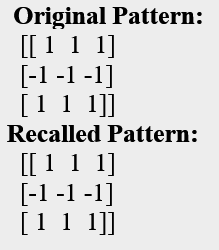
        print("Original Pattern:")

        print(pattern.reshape(3, 3))

        print("Recalled Pattern:")

        print(recalled\_pattern.reshape(3, 3), "\n")

**Output:**

**Original Pattern:**

[[-1 1 1]

[-1 1 -1]

[-1 1 -1]]

**Recalled Pattern:**

[[-1 1 -1]

[-1 1 -1]

[-1 1 -1]]

# **Aim:** Write a program for Radial Basis function

import numpy as np

import matplotlib.pyplot as plt

def radial\_basis\_function(x, c, sigma):

    return np.exp(-((x - c) \*\* 2) / (2 \* sigma \*\* 2))

x = np.linspace(0, 2 \* np.pi, 100)

y\_target = np.sin(x)

num\_centers = 5

centers = np.linspace(0, 2 \* np.pi, num\_centers)

sigma = (max(centers) - min(centers)) / (2 \* num\_centers)

rbf\_activations = np.zeros((len(x), num\_centers))

for i in range(len(x)):

    for j in range(num\_centers):

        rbf\_activations[i, j] = radial\_basis\_function(x[i], centers[j], sigma)

weights = np.linalg.pinv(rbf\_activations).dot(y\_target)

y\_approximated = rbf\_activations.dot(weights)

plt.figure()

plt.plot(x, y\_target, label="Target Function (sin(x)")

plt.plot(x, y\_approximated, label="RBF Approximation")

plt.scatter(centers, np.sin(centers), c='red', marker='o', label="RBF Centers")

plt.legend()

plt.title("Radial Basis Function Approximation")

plt.show()

