PRACTICAL 5

A. 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]]
```

Original Pattern:
```
[[ 1  1  1]
 [-1 -1 -1]
 [ 1  1  1]]
```
Recalled Pattern:
```
[[ 1  1  1]
 [-1 -1 -1]
 [ 1  1  1]]
```

Original Pattern:
```
[[ 1 -1  1]
 [ 1 -1  1]
 [ 1 -1  1]]
```
Recalled Pattern:
```
[[ 1 -1 -1]
 [ 1 -1  1]
 [ 1 -1  1]]
```

PRACTICAL 5

B. 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()
```