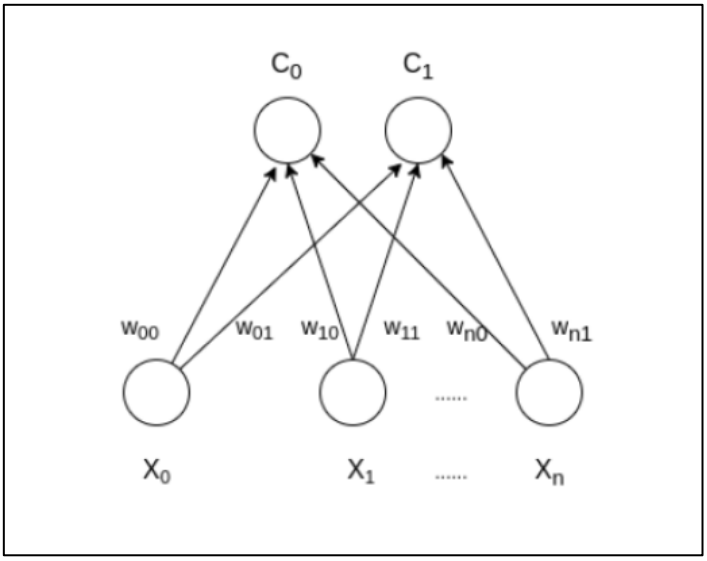
EXERCISE	8. SELF ORGANIZING MAPS
DATE	17.09.2024

AIM:
To implement Self Organizing Map (SOM) to cluster the given data points.

DESCRIPTION: It follows an unsupervised learning approach and trained its network through a competitive learning algorithm. SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation. SOM has two layers, one is the Input layer and the other one is the Output layer.



Weight updation rule is given by: $w_{ij} = w_{ij}(old) + alpha(t) * (x_i^k - w_{ij}(old))$

Training:

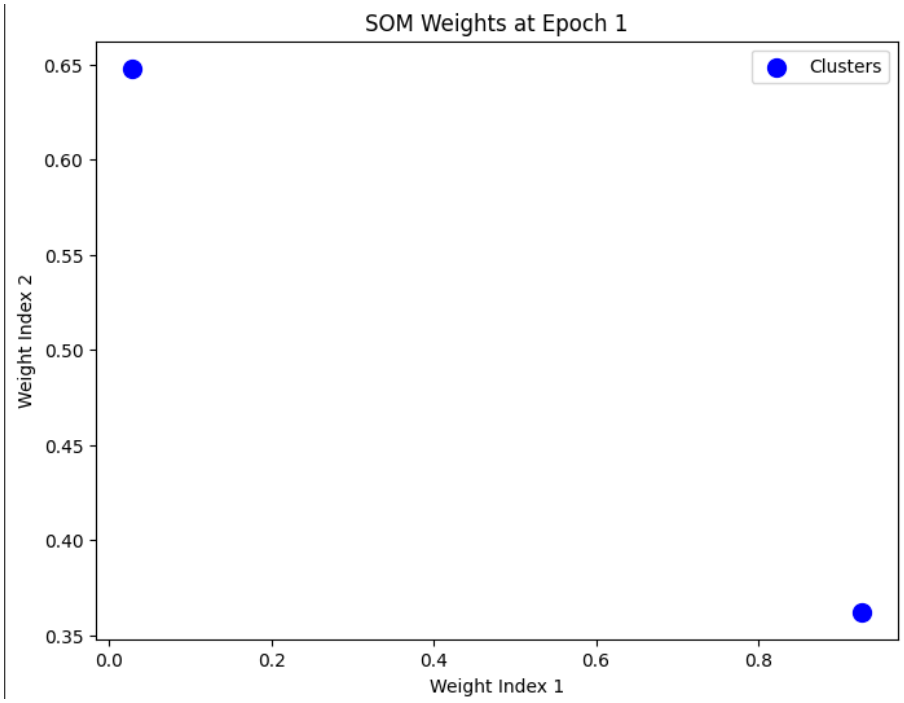
Step 1: Initialize the weights w_{ij} random value may be assumed. Initialize the learning rate α.
Step 2: Calculate squared Euclidean distance.
D(j) = Σ (wij – xi)^2 where i=1 to n and j=1 to m
Step 3: Find index J, when D(j) is minimum that will be considered as winning index.
Step 4: For each j within a specific neighborhood of j and for all i, calculate the new weight.
wij(new)=wij(old) + α[xi – wij(old)]
Step 5: Update the learning rule by using :
*α(t+1) = 0.5 * t*
Step 6: Test the Stopping Condition.

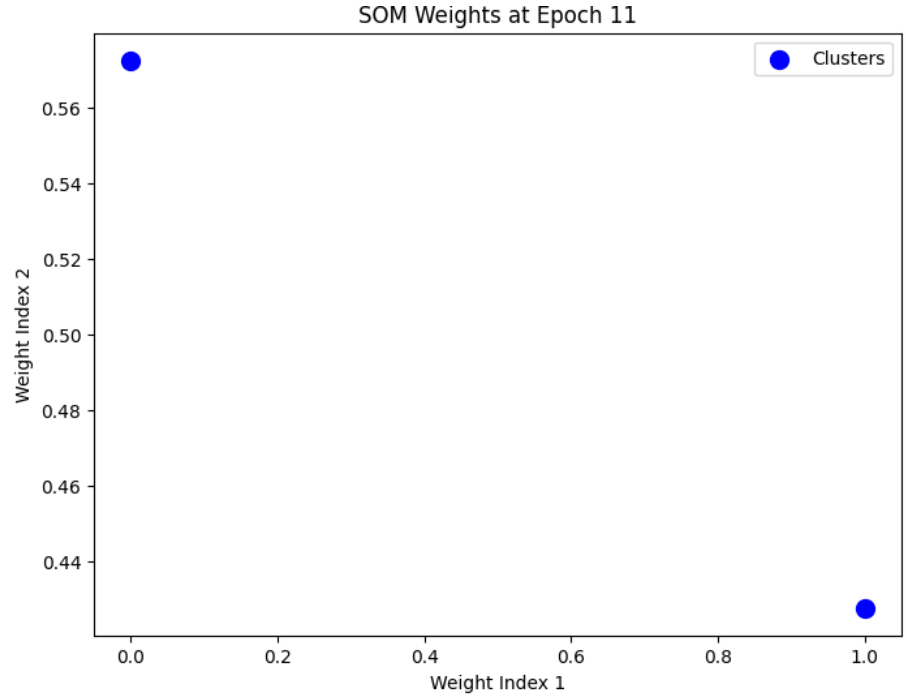
PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
import math

class SOM:
    def __init__(self, m, n, num_clusters, alpha=0.5, epochs=100):
        """
        Initialize the SOM with the given dimensions, number of clusters,
        learning rate, and number of epochs.
        """
        self.m = m  # Rows (number of neurons in the grid)
        self.n = n  # Columns (number of neurons in the grid)
        self.num_clusters = num_clusters  # Number of neurons/clusters
        self.alpha = alpha  # Learning rate
        self.epochs = epochs  # Number of epochs

        # Initialize weights (random values between 0 and 1)
        # Now initializing weights for 2 features (same as input data)
        self.weights = np.random.rand(num_clusters, 2)
        self.grid_shape = (m, n)

    def winner(self, sample):
        """
        Compute the Euclidean distance between the sample and each weight vector.
        The weight vector with the minimum distance is the winning neuron (cluster).
        """
        distances = np.linalg.norm(self.weights - sample, axis=1)
        return np.argmin(distances)  # Index of the minimum distance

    def update(self, sample, winner_index):
        """
        Update the weights of the winning neuron.
        """
        self.weights[winner_index] += self.alpha * (sample - self.weights[winner_index])

    def train(self, data):
        """
        Train the SOM using the given data for a number of epochs.
        """
        for epoch in range(self.epochs):
            # For each sample, find the winning neuron and update the weights
```

```
            for sample in data:
                winner_index = self.winner(sample)
                self.update(sample, winner_index)

            # Optionally decay the learning rate over time
            self.alpha = self.alpha * (1.0 - epoch / float(self.epochs))

            # Plot the weights at the end of each epoch to visualize the training
            if epoch % 10 == 0:  # Visualize every 10 epochs
                self.plot_weights(epoch)

    def plot_weights(self, epoch):
        """
        Plot the weights of the SOM at each epoch.
        """
        plt.figure(figsize=(8, 6))
        plt.title(f'SOM Weights at Epoch {epoch + 1}')
        plt.scatter(self.weights[:, 0], self.weights[:, 1], s=100, c='blue', label='Clusters')
        plt.xlabel('Weight Index 1')
        plt.ylabel('Weight Index 2')
        plt.legend()
        plt.show()

# Driver code to demonstrate SOM

def main():
    # Sample training data (4 samples, 2 features)
    data = np.array([[1, 1], [0, 0], [1, 0], [0, 1]])

    # Define the size of the SOM grid (2x2 grid)
    m = 2  # Rows
    n = 2  # Columns
    num_clusters = 2  # Number of clusters
    som = SOM(m, n, num_clusters, alpha=0.5, epochs=50)
    som.train(data)
    test_sample = np.array([0.8, 0.8])  # Example test sample
    winner_index = som.winner(test_sample)
    print(f'Test sample {test_sample} belongs to Cluster: {winner_index}')
if __name__ == "__main__":
    main():
```

OUTPUT:





RESULT:

The above code is executed successfully using self- organizing maps.