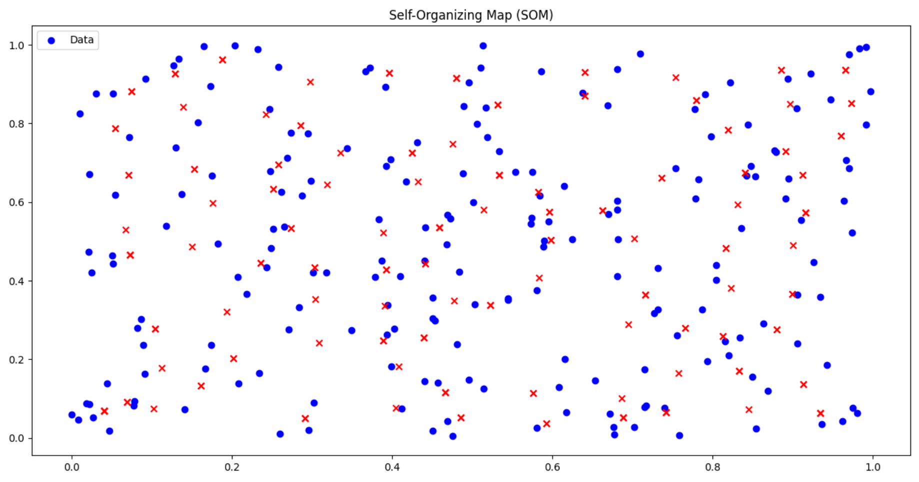
```python
import numpy as np

import matplotlib.pyplot as plt

class SelfOrganizingMap:

  def __init__(self, m, n, dim, learning_rate=0.5, radius=None, epochs=1000):

    self.m = m

    self.n = n

    self.dim = dim

    self.learning_rate = learning_rate

    self.epochs = epochs

    self.radius = max(m, n) / 2 if radius is None else radius

    self.weights = np.random.rand(m, n, dim)


  def find_bmu(self, x):

    distances = np.linalg.norm(self.weights - x, axis=2)

    bmu_index = np.unravel_index(np.argmin(distances), (self.m, self.n))

    return bmu_index

  def train(self, data):

    time_constant = self.epochs / np.log(self.radius)

    for epoch in range(self.epochs):

      for x in data:

        bmu_index = self.find_bmu(x)

        lr = self.learning_rate * np.exp(-epoch / self.epochs)

        rad = self.radius * np.exp(-epoch / time_constant)

        for i in range(self.m):

          for j in range(self.n):

            dist_to_bmu = np.linalg.norm(np.array([i, j]) - np.array(bmu_index))

            if dist_to_bmu <= rad:

              influence = np.exp(-(dist_to_bmu**2) / (2 * (rad**2)))

              self.weights[i, j] += lr * influence * (x - self.weights[i, j])

      if epoch % (self.epochs // 10) == 0:

        print(f"Epoch {epoch}/{self.epochs}")


  def map_vects(self, data):

    return [self.find_bmu(x) for x in data]


data = np.random.rand(200, 2)

som = SelfOrganizingMap(m=10, n=10, dim=2, learning_rate=0.5, epochs=100)

som.train(data)

mapped = som.map_vects(data)


plt.figure(figsize=(6,6))

plt.scatter(data[:,0], data[:,1], c="blue", label="Data")

for i, m in enumerate(mapped):

  plt.scatter(som.weights[m[0], m[1], 0], som.weights[m[0], m[1], 1], c="red", marker="x")

plt.title("Self-Organizing Map (SOM)")

plt.legend()

plt.show()
```

OUTPUT

```
Epoch 0/100

Epoch 10/100

Epoch 20/100

Epoch 30/100

Epoch 40/100

Epoch 50/100

Epoch 60/100

Epoch 70/100

Epoch 80/100

Epoch 90/100
```


Self-Organizing Map (SOM)
● Data