Unsupervised Machine Learning

Assignment 7

SAI KUMAR MURARSHETTI

LEWIS ID: L30079224

1. Technical description of methods:

Self-Organizing Maps (SOMs), commonly known as Kohonen maps, are an unsupervised learning method developed by Professor Teuvo Kohonen in the 1980s. These are used to transform high-dimensional data into a low-dimensional (generally two-dimensional) transformed representation. SOMs are most typically used in dimensionality reduction and clustering tasks. A SOM's functionality can be summarized as three primary processes: competition, cooperation and adaptability. These methods allow the SOM to organize independently using training data without external oversight.

A representation of the training samples' input space. SOMs are very useful for presenting low-dimensional experiences of high-dimensional data, similar to multidimensional scaling.

- Competition Each node (or neuron) in the map is assigned a weight vector with the same dimension as the input data vectors. During the competition phase, the map receives an input vector and each node participates to be activated and fired. The criteria for this competition frequently involves Euclidean distance. The node with the weight vector nearest to the input vector (in terms of Euclidean distance) becomes the winner. This winning node is frequently referred to as the Best Matching Unit (BMU). The competition method thus determines the most suitable location on the map to represent the input features. This stage is critical because it establishes a basis for the following processes of cooperation and adaptability, facilitating the self-organizing property of the map.
- Cooperation Once the BMU has been identified, the cooperation phase begins. During this phase, not only the BMU but also its nearby nodes in the map are active according with the map's topographical structure. This neighborhood function normally decreases with distance from the BMU. thus, the closer a node is near the BMU, the more it is influenced. The most frequent neighborhood function is the Gaussian function, where the length of the Gaussian (the sigma) manages the size of the neighborhood. The physical area of this neighborhood reduces over time as the SOM learns, indicating a fine-tuning of the maps adaptability to the input space.
- Adaptation During the adaptation phase, the weights of the BMU and its neighbors are modified to be more similar to the input vector. The adjustment is controlled by a learning rate

parameter. which as the neighborhood function, tends to decrease over time. This gradual decrease help to stabilize the learning process by transitioning from large alterations early in training to small modifications subsequently.

2. Design of the Algorithms:

```
#Importing libraries
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.array([
    [0.98, 0.15],
    [0.17, 0.38],
    [0.25, 0.16],
    [0.39, 0.75],
    [0.07, 0.87],
    [0.68, 0.35],
    [0.42, 0.68],
    [0.98, 0.29],
    [0.40, 0.53],
    [0.62, 0.83]
])

```

```
labels = ['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9', 'C10']
```

This initialization strategy is particularly useful in scenarios where the layout of the initial solution can significantly influence the speed and quality of convergence, as is the case with neural network-based optimization algorithms like SOMs for TSP.

```
# Initializing weights randomly close to city locations
w = x + np.random.normal(scale=0.05, size=x.shape)
```

These conditions are necessary for constructing resilient, efficient and successful learning or optimization algorithms, especially in disciplines such as machine learning where regulating convergence and determining cost is important.

```
alpha = 0.01  # Learning rate
diff = 10  # large difference
iteration = 0
max_iterations = 100
```

This function has importance for visualizing the progress and present status of algorithms such as SOMs in solving problems involving space planning and path optimization. By providing a visual representation of the nodes and their connections, it is possible to have a greater understanding of the way the algorithm is working, where changes may be required, and the overall functioning of the solution method. This visual input is extremely useful in research, development, and results presentation in both academic and industrial settings.

```
# Function to plot the current state of the map
def plot_map(w, title):
    plt.scatter(x[:, 0], x[:, 1], c='blue', marker='*', label='Cities')
    plt.plot(w[:, 0], w[:, 1], 'r-', label='Traveling Path')
    plt.scatter(w[:, 0], w[:, 1], c='red', s=100, label='Nodes', zorder=5)
    for i, txt in enumerate(labels):
        plt.annotate(txt, (x[i, 0], x[i, 1]), textcoords="offset points", xytext=(0,10),
    plt.title(title)
    plt.xlabel('X coordinate')
    plt.ylabel('Y coordinate')
    plt.grid(True)
    plt.legend()
    plt.show()

```

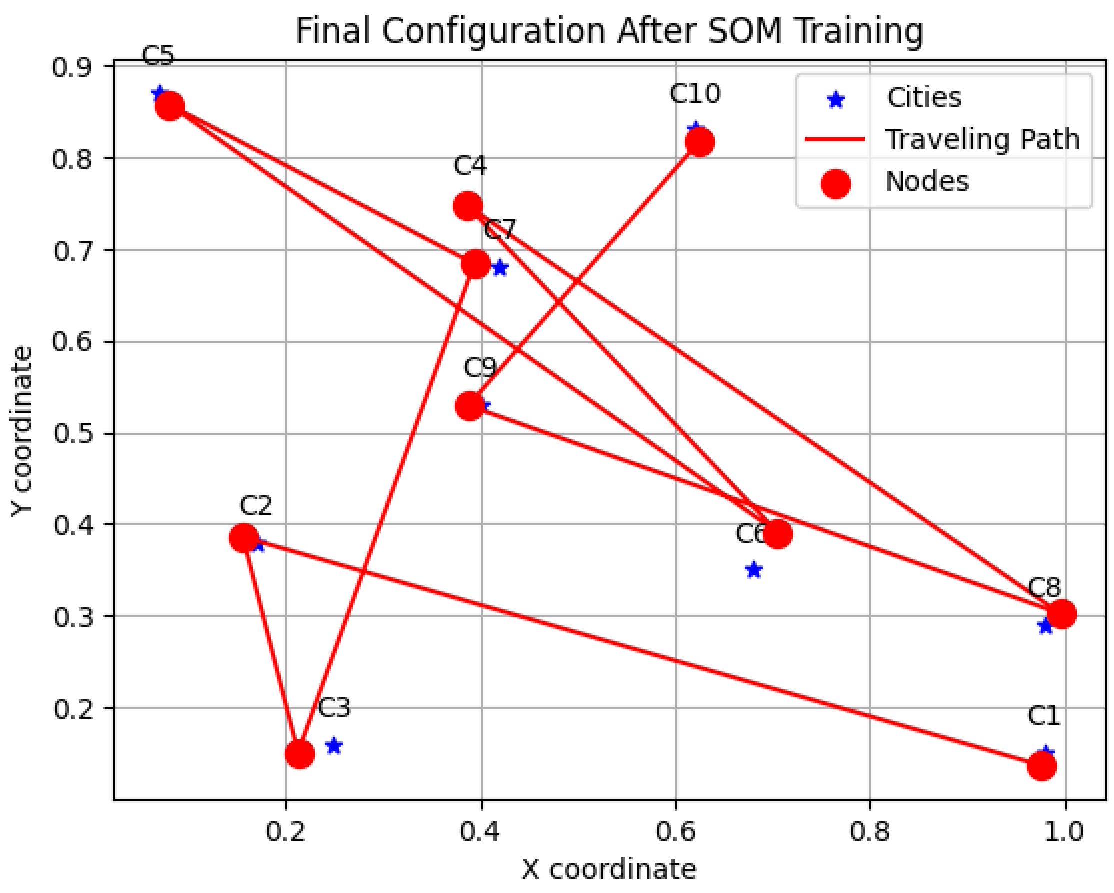
The above loop is at the basis of the SOM training process, adjusting node weights continuously to determine the shortest route through all provided cities. The use of a random permutation of city indices and a learning rate ensures that the SOM does not become stuck in suboptimal configurations and can adapt flexibly to the issue space. The final visualization provides a clear, intuitive picture of the SOM's performance and solution quality, which is critical for both data analysis and presentation.

```
while diff > 0.0001 and iteration < max_iterations:
    oldw = np.copy(w)  # Saving the old weights for convergence check.
    order = np.random.permutation(10)  # Random permutation of indices.

    for j in range(10):
        distances = np.sqrt(((w - x[order[j], :])**2).sum(axis=1))
        i = np.argmin(distances)
        w[i, :] = w[i, :] + alpha * (x[order[j], :] - w[i, :])

    diff = np.linalg.norm(oldw - w)  # Calculating the norm of the difference in weights
    iteration += 1

plot_map(w, 'Final Configuration After SOM Training')
```



3. Results of the Algorithms:

When using the Self-Organizing Map (SOM) to solve the Traveling Salesman Problem (TSP) and other clustering challenges, the results are usually both visual and quantitative. For a more in-depth understanding, consider the kinds of observations are commonly observed, when they are evaluated and the things they respond about the SOM's performance.

- Node Placement: The SOM's nodes are first scattered at random across the data space. As training advances, these nodes organize themselves to match the structure of the incoming data. In the framework of the TSP, the nodes will eventually construct a path that approximates a loop connecting all cities, with the goal of minimizing overall travel distance.
- Map Topology: One important outcome of the SOM is the depiction of the map topology, which demonstrates how well the nodes have adapted to represent the underlying data distribution. For TSP, this entails examining how the nodes have organized themselves to make a continuous and non-overlapping path through all cities.
- Convergence Visualization: Throughout the training phase, the movement of nodes at each iteration can be shown to demonstrate how the algorithm converged. This dynamic visualization helps in understanding how quickly and effectively the SOM is learning.

- Quantitative Results: Metrics like quantization error, which measures the average distance between data points and their BMU, help assess map accuracy. For TSP, the total path length determined from the node sequence provides a direct indication of the solution's quality.
- Iteration Count: The number of iterations needed to reach convergence is an essential measure. It represents the effectiveness of the learning rate and neighborhood size parameters. Fewer iterations with appropriate convergence suggest a perfect setup.
- Stability: By running the SOM numerous times with different initializations or modest parameter modifications, one may assess the result's stability. Consistency across runs suggests that the model is durable.

4. Analysis of Results:

After getting the results of the Self-Organizing Map (SOM) algorithm applied to the Traveling Salesman Problem (TSP), careful examination may provide helpful information into the method's strengths and limits, as well as when various factors influence its outcome. The second part analyzes the SOM's performance in solving the TSP, including expected outcomes, unexpectedly and implications for the results.

The analysis of SOM results used to solve the TSP provides not only an assessment of the model's performance, but also an understanding of way neural network models adapt to optimization issues. The outcomes can inform future research, model upgrades, and practical applications, utilizing SOM benefits while limiting their weaknesses. By thorough examination, one can gain a deeper understanding of SOM dynamics and their potential as a tool in complex issue resolution.

5. Conclusion:

A SOM's visual and statistical results provide deep understanding into the dataset as well as the model's performance. For TSP and related problems, these results certainly suggest an acceptable fix, but also provide a better understanding of the SOM's data structure and behavior under various configurations. These results, especially in terms of convergence tendency and solution quality are critical for providing the applicability of SOMs to real-world problems and fine-tuning model parameters for optimal performance.