Self-Organizing Map (SOM) for Clustering Geographic Coordinates

Grok

July 29, 2025

1 Introduction to Self-Organizing Maps

A Self-Organizing Map (SOM), also known as a Kohonen map, is an unsupervised neural network algorithm used for clustering and dimensionality reduction. It maps high-dimensional data onto a lower-dimensional (typically 2D) grid while preserving the topological properties of the input space, making it ideal for visualizing and clustering data such as geographic coordinates.

1.1 Key Concepts

- **Neurons and Grid**: The SOM consists of a grid of neurons, each associated with a weight vector matching the input data's dimensions.
- **Training Process**:
  - *Initialization*: Neuron weights are initialized randomly or based on the data.
  - *Competition*: For each input, the neuron with the closest weight vector (Best Matching Unit, BMU) is identified using a distance metric (e.g., Euclidean).
  - *Update*: The BMU and its neighbors update their weights to become closer to the input, with the update magnitude decreasing with distance and time.
  - *Topology Preservation*: Nearby neurons represent similar data points, enabling clustering.
- **Applications**: SOMs are used for clustering, visualization, and pattern recognition, especially for spatial data.
- **Visualization**: The SOM grid can be visualized using a Unified Distance Matrix (U-Matrix), where darker shades indicate dissimilarities between neurons and lighter shades indicate similarities.

2 Dataset and Approach

The example uses the "World Cities Database" from Kaggle, containing city names, latitudes, and longitudes. A subset of 1000 cities is used for clustering based on their coordinates. The approach involves:

1. Normalizing latitude and longitude to [0,1].
2. Training a 10x10 SOM using the MiniSom library.
3. Assigning each city to a cluster based on its BMU.
4. Visualizing clusters on a map using Folium, with colors indicating different clusters, and a U-Matrix to show similarities/dissimilarities.

3 Python Code

The following Python code implements the SOM, clusters the data, and visualizes the results. It uses MiniSom for the SOM, pandas for data handling, scikit-learn for normalization, and Folium for map visualization.

```
import numpy as np
import pandas as pd
from minisom import MiniSom
from sklearn.preprocessing import MinMaxScaler
import folium
import matplotlib.pyplot as plt
from matplotlib.patches import RegularPolygon
import matplotlib.cm as cm

# Load dataset (World Cities Database from Kaggle)
# For simplicity, we'll create a sample dataset here; replace with actual dataset path
# Download from: https://www.kaggle.com/datasets/juanmah/world-cities
data = pd.read_csv('https://raw.githubusercontent.com/datasets/world-cities/maste
# Select relevant columns and a subset for demonstration
data = data[['name', 'lat', 'lng']].sample(1000, random_state=42)  # Use 1000 cities for speed

# Preprocess: Normalize latitude and longitude
scaler = MinMaxScaler()
coords = scaler.fit_transform(data[['lat', 'lng']])

# Initialize and train SOM
som_size = (10, 10)  # 10x10 grid
som = MiniSom(som_size[0], som_size[1], 2, sigma=1.0, learning_rate=0.5, random_seed=42)
som.random_weights_init(coords)
som.train_random(coords, num_iteration=1000)

# Get U-Matrix for visualization (shows distances between neurons)
u_matrix = som.distance_map().T  # Transpose for correct orientation

# Assign each city to a cluster (BMU)
```

```
clusters = []
for i, x in enumerate(coords):
    bmu = som.winner(x)
    clusters.append(bmu)
data['cluster'] = clusters

# Visualize SOM grid with U-Matrix
plt.figure(figsize=(10, 10))
plt.title('SOM U-Matrix (Darker = More Dissimilar)')
plt.imshow(u_matrix, cmap='bone_r')
plt.colorbar(label='Distance')
plt.grid(False)
plt.show()

# Create a folium map centered on the mean coordinates
m = folium.Map(location=[data['lat'].mean(),
   data['lng'].mean()], zoom_start=2)

# Color map for clusters
colors = cm.rainbow(np.linspace(0, 1, som_size[0] * som_size[1]))

# Add points to the map
for _, row in data.iterrows():
    cluster_idx = row['cluster'][0] * som_size[1] +
       row['cluster'][1]  # Convert 2D index to 1D
    folium.CircleMarker(
        location=[row['lat'], row['lng']],
        radius=5,
        color='#%02x%02x%02x' %
           (int(colors[cluster_idx][0]*255),
           int(colors[cluster_idx][1]*255),
           int(colors[cluster_idx][2]*255)),
        fill=True,
        fill_opacity=0.7,
        popup=row['name']
    ).add_to(m)

# Save the map
m.save('world_cities_som_clusters.html')

# Print sample of clustered data
print(data[['name', 'lat', 'lng', 'cluster']].head())
```

4 Notes

- **Dependencies**: Install required libraries using `pip install minisom pandas sklearn folium matplotlib numpy`.
- **Running the Code**: Save the code as `som_clustering.py` and execute it. Ensure internet access to load the dataset, or download it locally from Kaggle.

- **Output**: The code generates a U-Matrix plot showing similarities/dissimilarities, an interactive HTML map (`world_cities_som_clusters.html`) with colored cluster markers, and a printed sample of clustered data.
- **Customization**: Adjust `som_size`, `sigma`, or `learning_rate` to tune clustering granularity. For larger datasets, increase the SOM grid size and iterations.

noto