Self-Organizing Feature Maps (SOM)

- **Self Organizing Map (or Kohonen Map or SOM)** is a type of Artificial Neural Network which is also inspired by biological models of neural systems from the 1970s. 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.
- It helps in visualizing and understanding high-dimensional data by mapping it to a lower-dimensional (usually 2D) grid. SOM was introduced by **Teuvo Kohonen**, so it is also called the **Kohonen Network**.

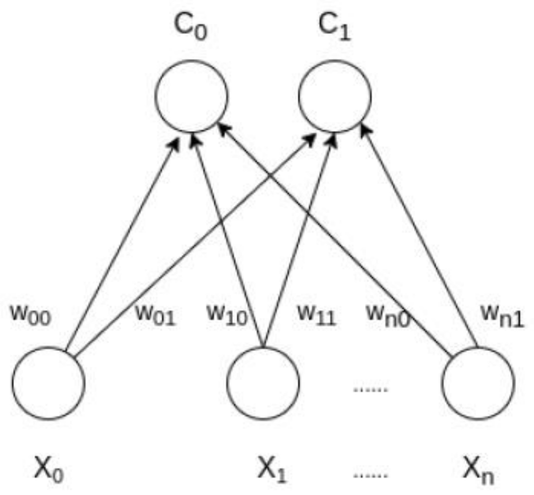
Key features:

- SOMs organize data based on **similarity**.
- The result is a **map** where similar data points are placed closer together.
- It is often used for **clustering**, **dimensionality reduction**, and **visualization**.

Structure of a Self-Organizing Feature Map

- **Input Layer**: Each input node represents a feature of the data. For example, if data has 3 features, the input layer will have 3 nodes.
- **Output Layer (Feature Map)**: The output layer is a grid of neurons arranged in 1D, 2D, or sometimes even 3D. Each neuron has a **weight vector** equal in dimension to the input layer.

The architecture of the Self Organizing Map with two clusters and n input features of any sample is



Structure Explanation:

- Each neuron in the map has a **weight vector** that gets updated during training.
- Neurons are connected to their neighbours, maintaining a topological relationship.
- The size of the map is determined based on the data's complexity.

1. Components of SOM

- **Input Layer**:
  - ➢ Represents the features of the input data.
  - ➢ Each input corresponds to a node in the input layer.
  - ➢ Example: For a dataset with 3 features (age, income, spending score), the input layer has 3 nodes.

- **Output Layer (Feature Map):**
  - ➢ A grid (usually 2D) of neurons.
  - ➢ The grid can have various shapes, such as rectangular or hexagonal.
  - ➢ Each neuron is connected to its neighbors, maintaining a structured relationship.

- **Weights:**
  - ➢ Each neuron in the grid has a weight vector of the same dimension as the input data.
  - ➢ These weights adjust during training to match input patterns.

2. Properties of the SOM Structure

Neighborhood Structure:
- ➢ Neurons in the grid are linked to their neighbors.
- ➢ This connection ensures that similar inputs adjust not just the Best Matching Unit (BMU) but also nearby neurons.

Topological Preservation:
- Data points that are close in the input space are mapped to nearby neurons on the SOM grid.

3. Example of SOM Structure

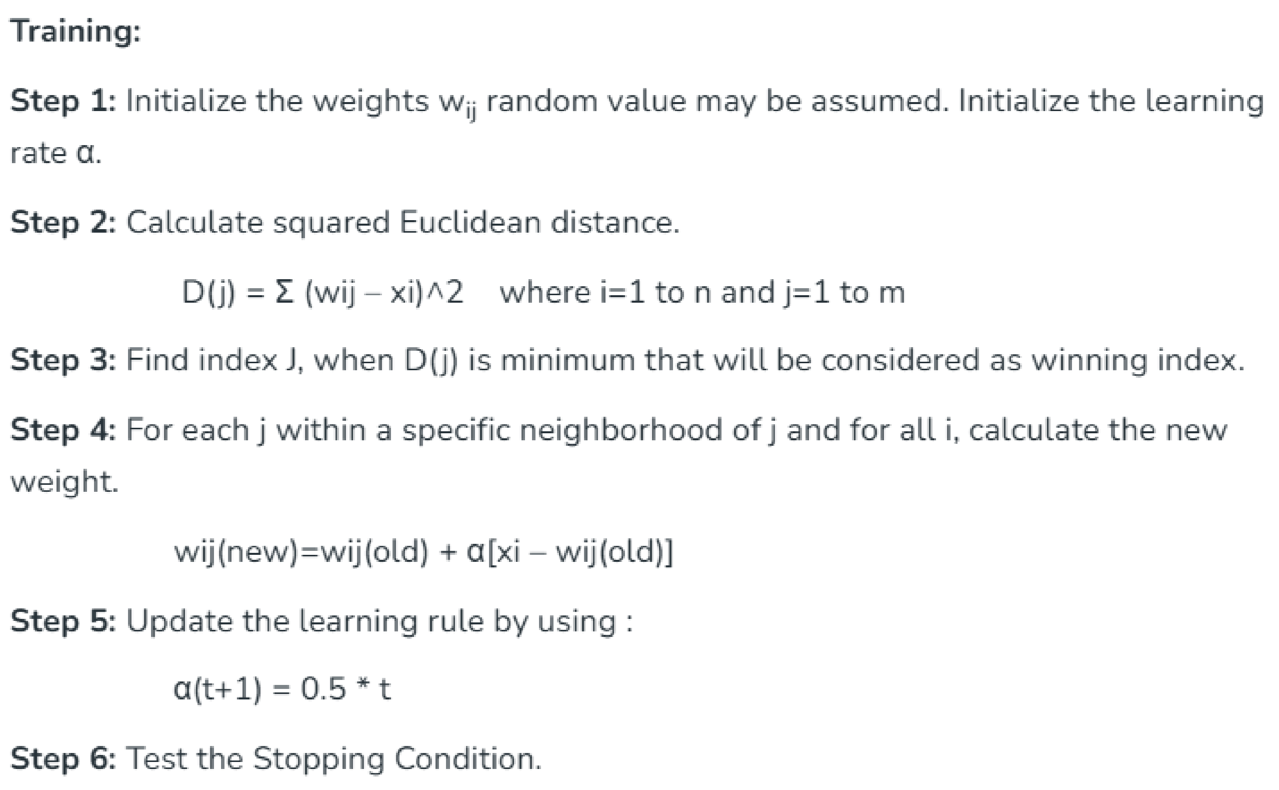
Suppose we have a dataset with customer features:
- Input features: Age, annual income, spending score.
- The SOM will map these inputs onto a grid, where:
  - Neurons in one area of the grid may represent "high-income, high-spending" customers.
  - Neurons in another area represent "low-income, budget-conscious" customers.

How do SOM works?

Let's say an input data of size (m, n) where m is the number of training examples and n is the number of features in each example. First, it initializes the weights of size (n, C) where C is the number of clusters. Then iterating over the input data, for each training example, it updates the winning vector (weight vector with the shortest distance (e.g Euclidean distance) from training example). Weight updation rule is given by :

$$w_{ij} = w_{ij}(old) + alpha(t) * (x_i^k - w_{ij}(old))$$

where alpha is a learning rate at time t, j denotes the winning vector, i denotes the ith feature of training example and k denotes the kth training example from the input data. After training the SOM network, trained weights are used for clustering new examples. A new example falls in the cluster of winning vectors.



Training:

Step 1: Initialize the weights w_{ij} random value may be assumed. Initialize the learning rate α.

Step 2: Calculate squared Euclidean distance.

$$D(j) = \Sigma (wij - xi)^2 \quad \text{where } i=1 \text{ to } n \text{ and } j=1 \text{ to } m$$

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.

$$wij(new)=wij(old) + \alpha[xi - wij(old)]$$

Step 5: Update the learning rule by using :

$$\alpha(t+1) = 0.5 * t$$

Step 6: Test the Stopping Condition.