UNIT 4

Self-Organization Maps (SOM):

Understanding Self-Organizing Maps (SOMs)

- **Core Function:**
  - SOMs are unsupervised learning neural networks that aim to discover patterns and relationships within data.
  - They achieve this by mapping high-dimensional input data onto a lower-dimensional grid (the "map").
  - This mapping preserves the topological properties of the input data, meaning that data points close to each other in the high-dimensional space will also be close to each other on the map.
- **Key Concepts:**
  - **Input Space:** The high-dimensional space containing the input data.
  - **Map Space:** The lower-dimensional grid (usually 2D) where the input data is mapped.
  - **Neurons (Nodes):** The units within the map space, each associated with a weight vector.
  - **Best Matching Unit (BMU):** The neuron whose weight vector is most similar to the current input vector.
  - **Neighborhood Function:** A function that defines the region around the BMU that will be updated during training.

Feature Mapping Models within SOMs

The "feature mapping" in SOMs essentially refers to how the network transforms the input data into a representation on the map. This process involves:

1. **Competitive Learning:**
   - This is the fundamental principle of SOMs.
   - When an input vector is presented to the network, each neuron's weight vector is compared to the input vector.
   - The neuron with the closest weight vector (the BMU) "wins" the competition.

- This process effectively maps the input vector to the location of the BMU on the map.

2. **Topological Mapping:**

- This is the key characteristic that distinguishes SOMs.
- During training, not only is the BMU's weight vector updated, but also the weight vectors of its neighboring neurons.
- The neighborhood function determines the extent of this update.
- This process ensures that neurons that are close to each other on the map will have similar weight vectors, reflecting the topological relationships in the input data.
- This is what creates the feature mapping. Data with similar features, will be mapped to similar locations on the map.

In essence:

- SOMs create a "feature map" by organizing neurons on the map so that they represent different clusters or patterns in the input data.
- The location of a neuron on the map indicates the characteristics of the data points that it represents.

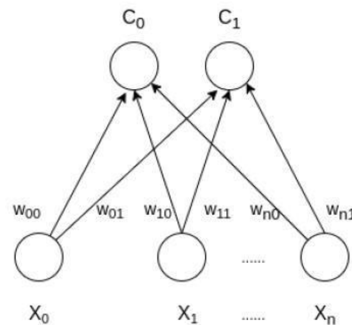
Where SOMs are Useful:

- **Data Visualization:** SOMs are excellent for visualizing high-dimensional data in a lower-dimensional space.
- **Clustering:** SOMs can be used to cluster data points based on their similarity.
- **Feature Extraction:** SOMs can extract meaningful features from data by mapping it onto the map.

Self-Organization Map,SOM ALGORITHM:

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.

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



Algorithm

Training:

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

Step 2: Calculate squared Euclidean distance.

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.

$$w_{ij}(new)=w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$$

Step 5: Update the learning rule by using

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

Properties of Feature Map:

Key Properties:

- **Topological Ordering:**
  - This is the most crucial property. The feature map preserves the topological relationships of the input data. This means that data points that are close to each other in the high-dimensional input space will also be close to each other on the lower-dimensional map.
  - This property is what makes SOMs so valuable for visualization. It allows us to see how data points relate to each other in terms of similarity.
- **Dimensionality Reduction:**

  - SOMs effectively reduce the dimensionality of the input data. High- dimensional data is mapped onto a lower-dimensional grid, typically a 2D plane.
  - This simplification makes it easier to visualize and analyze complex data.
- **Data Clustering:**
  - The feature map naturally forms clusters of similar data points. Neurons that are close to each other on the map tend to respond to similar input patterns.
  - This clustering property allows us to identify groups of related data points.
- **Feature Extraction:**
  - The SOM extracts essential features from the input data. The position of a neuron on the map represents a particular combination of features.
  - By analyzing the arrangement of neurons on the map, we can gain insights into the underlying structure of the data.
- **Visualization:**
  - The feature map provides a visual representation of the input data, making it easier to understand patterns and relationships.
  - Various visualization techniques, such as U-matrices and component planes, can be used to further enhance the interpretability of the map.
- **Non-linear mapping:**
  - SOMs are capable of non-linear mapping of the data, this allows for the analysis of data that does not have linear relationships.

In essence:

The feature map in a SOM is designed to:

- Organize data in a meaningful way.
- Reveal hidden patterns and relationships.
- Simplify complex data for easier analysis.

These properties make SOMs a powerful tool for a wide range of applications, including data mining, pattern recognition, and image processing.

Computer Simulations:

```
class SOM:

    # Function here computes the winning vector
    # by Euclidean distance
    def winner(self, weights, sample):

        D0 = 0
        D1 = 0

        for i in range(len(sample)):

            D0 = D0 + math.pow((sample[i] - weights[0][i]), 2)
            D1 = D1 + math.pow((sample[i] - weights[1][i]), 2)

        # Selecting the cluster with smallest distance as winning cluster

        if D0 < D1:
            return 0
        else:
            return 1

    # Function here updates the winning vector
    def update(self, weights, sample, J, alpha):
        # Here iterating over the weights of winning cluster and modifying them
        for i in range(len(weights[0])):
            weights[J][i] = weights[J][i] + alpha * (sample[i] - weights[J][i])

        return weights

# Driver code
```

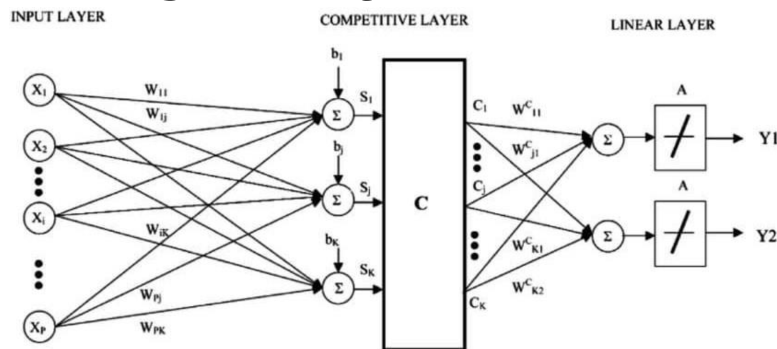
```
def main():

    # Training Examples ( m, n )
    T = [[1, 1, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 1]]
```

```
if __name__ == "__main__":
    main()
```

Output:

Test Sample s belongs to Cluster : 0 ;Trained weights :
[[0.6000000000000001, 0.8, 0.5, 0.9], [0.3333984375, 0.0666015625, 0.7, 0.3]]

Learning Vector Quantization:



- Learning Vector Quantization (LVQ) is a supervised learning algorithm that builds upon the principles of vector quantization. While Self-Organizing Maps (SOMs) are unsupervised, LVQ uses labeled data to refine the prototypes (codebook vectors) and create a classification system

Understanding Learning Vector Quantization (LVQ)

- **Supervised Learning:**
  - Unlike SOMs, LVQ requires labeled data. This means that each input data point has a corresponding class label.
- **Prototype-Based Classification:**
  - LVQ works by defining a set of prototype vectors, each representing a class.
  - These prototypes are adjusted during training to better represent the distribution of data within each class.
- **Competitive Learning:**
  - Similar to SOMs, LVQ uses a competitive learning process. When an input data point is presented, the algorithm finds the closest prototype vector.
- **Prototype Adjustment:**
  - The key difference is that LVQ adjusts the winning prototype based on whether it correctly classifies the input data

Key Algorithm Steps:

1. **Initialization:**
   - Initialize the prototype vectors and their class labels.
2. **Sampling:**

- Randomly select an input data point and its class label.

2. **Matching:**
   - Find the closest prototype vector.
3. **Updating:**
   - Adjust the winning prototype's position based on the class label.
4. **Iteration:**
   - Repeat steps 2-4 for many

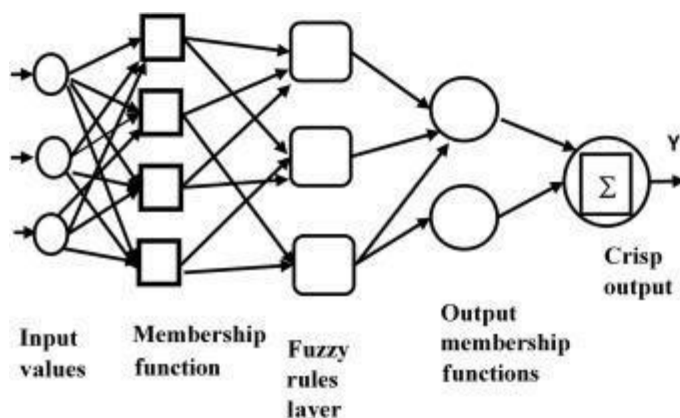
iterations. Relationship to SOMs:

- LVQ can be seen as a supervised extension of SOMs.
- While SOMs create a topological map of the input data, LVQ refines the map by incorporating class labels.
- LVQ is often used after training a SOM to fine-tune the classification boundaries.

Applications:

- Pattern Recognition:
- Classification Tasks:
- Image Classification:

Adaptive Pattern Classification:



Adaptive pattern classification refers to systems that can learn and adjust their classification rules over time, often in response to changing data or environments. This is a crucial concept in machine learning, particularly when dealing with non- stationary data.

Core Concepts:

- **Adaptability:**
  - The system's ability to modify its internal parameters (e.g., weights, decision boundaries) as new data becomes available.
- **Dynamic Environments:**
  - Adaptive systems are designed to handle situations where the underlying data distribution may change over time.
- **Online Learning:**
  - Often, adaptive classification involves online learning, where the system updates its model with each new data point.
- **Real-time Processing:**
  - Many adaptive systems are used in real-time applications, where decisions must be made quickly.

Adaptive pattern classification refers to systems that can learn and adjust their classification rules over time, often in response to changing data or environments. This is a crucial concept in machine learning, particularly when dealing with non- stationary data.

Core Concepts:

- **Adaptability:**
  - The system's ability to modify its internal parameters (e.g., weights, decision boundaries) as new data becomes available.
- **Dynamic Environments:**
  - Adaptive systems are designed to handle situations where the underlying data distribution may change over time.
- **Online Learning:**
  - Often, adaptive classification involves online learning, where the system updates its model with each new data point.
- **Real-time Processing:**
  - Many adaptive systems are used in real-time applications, where decisions must be made quickly.

Key Techniques and Considerations:

- **Incremental Learning:**
  - Algorithms that can update their models without requiring access to the entire training dataset.
- **Concept Drift Detection:**
  - Methods for detecting changes in the data distribution, triggering model updates.

- **Adaptive Learning Rates:**
  - Adjusting the learning rate of algorithms to account for changes in data variability.
- **Ensemble Methods:**
  - Using multiple classifiers and dynamically adjusting their weights to improve performance.

Equations and Concepts:

- **Incremental Weight Updates:**
  - In many adaptive algorithms, weight updates are performed incrementally. A basic form of this can be expressed as:

```
w(t+1) = w(t) + η(t) * error(t) * x(t)
```

- Where:
  - `w(t)` is the weight vector at time `t`.
  - `w(t+1)` is the updated weight vector.
  - `η(t)` is the learning rate at time `t`.
  - `error(t)` is the classification error at time `t`.
  - `x(t)` is the input data vector at time `t`.

Adaptive learning rates, means that the $\eta(t)$ value is not a constant, but a value that changes over time, based on the performance of the algorithm.

Concept Drift:

- Concept drift is a change in the statistical properties of the target variable, which the model is trying to predict, over time.
- Detecting concept drift often involves monitoring the error rate of the classifier. When the error rate exceeds a certain threshold, it indicates that a concept drift has occurred.

Distance metrics:

- Many adaptive pattern classification algorithms utilize distance metrics, such as Euclidean distance, to determine the similarity between data points. These metrics are used in algorithms like k-nearest neighbors, which can be adapted to handle changing data by dynamically updating the set of nearest neighbors.

Applications:

- **Financial Trading:**
  - Adapting to changing market conditions.
- **Network Security:**
  - Detecting evolving cyber threats.
- **Medical Diagnosis:**
  - Adjusting diagnostic models to account for new patient data.
- **Robotics:**
  - Allowing robots to learn and adapt to changing environments.