SOM Anomaly Detection

Architecture of Implementation

Self Organizing Maps

Self-organizing maps are technique for performing unsupervised dimensionality reduction using neural network nodes. They can thought of as a non-parametric competitor to classical dimensionality reduction techniques like principal component analysis or linear discriminant analysis.

Working: The input is connected to every node, and the nodes form a multidimensional lattice over a space (typically a two-dimensional lattice). Nodes are competitive: input is fed one at a time, and the node that has the strongest reaction to the input is declared the "winning node" and gets to assign its weights to the input and return that as output. This is known as a **topographical map**.

Node winningness is determined using a discriminant function. The discriminant function of choice is simple Euclidean distance between the node weights and the input vector:

$$d_j(x) = \sum_{i=1}^{D} (x_i - w_{ji})^2$$

Training is performed using topological ordering. A training sample is presen n then picks the node whose weights are closest to that input. The node shifts its weights towards the input value by a certain learning rate. The neighborhood of nodes closest to that node also shift their location, less strongly than the winning node does, with the strength of the movement dependent on that node's distance to the winning node, subject to a learning rate and to exponential decay. The learning rate is annealed over time.

Self Organizing Maps

Self-organizing maps are technique for performing unsupervised dimensionality reduction using neural network nodes. They can thought of as a non-parametric competitor to classical dimensionality reduction techniques like principal component analysis or linear discriminant analysis.

Working: The input is connected to every node, and the nodes form a multidimensional lattice over a space (typically a two-dimensional lattice). Nodes are competitive: input is fed one at a time, and the node that has the strongest reaction to the input is declared the "winning node" and gets to assign its weights to the input and return that as output. This is known as a **topographical map**.

Node winningness is determined using a discriminant function. The discriminant function of choice is simple Euclidean distance between the node weights and the input vector:

$$d_j(x) = \sum_{i=1}^{D} (x_i - w_{ji})^2$$

Training is performed using topological ordering. A training sample is presen n then picks the node whose weights are closest to that input. The node shifts its weights towards the input value by a certain learning rate. The neighborhood of nodes closest to that node also shift their location, less strongly than the winning node does, with the strength of the movement dependent on that node's distance to the winning node, subject to a learning rate and to exponential decay. The learning rate is annealed over time.

Self Organizing Maps

Each update can be thought of as a shift in the topology of a neighborhood local to the output point. With sufficiently many iterations through this process, the nodes will form the dimension-reduced feature space over the original input which best preserves local topology:

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

where alpha is a learning rate at time t, j denotes the winning vector, i denotes the i^{th} feature of training example and k denotes the k^{th} 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 vector

Algorithm: Steps involved are :

- Weight initialization
- The input vector is selected from the dataset and used as an input for the network
- BMU is calculated[Best Matching Unit]
- The radius of neighbors that will be updated is calculated
- Each weight of the neurons within the radius are adjusted to make them more like the input vector
- Steps from 2 to 5 are repeated for each input vector of the dataset

Implementation of above approach Using Tensorflow

Sample code : https://github.com/ashinkrishnan/SOM/blob/main/som.py