A Report on Self-Organizing Maps

Prepared for

Dr. Navneet Goyal

Instructor In-charge, Foundation of Data Science (CS F320)
Department of Computer Science & Information Systems
Birla Institute of Technology & Science
PILANI, 333031 Rajasthan, INDIA

By

Aryan Madaan (2020A1PS0222P)
Parth Sethia (2020A3PS)
Atharv (2020ABPS)

Birla Institute of Technology & Science
PILANI, 333031 Rajasthan, INDIA

29th November 2022

Acknowledgement

We are very grateful to our professor Dr Navneet Goyal who gave us a chance to work on this project. We would like to thank him for giving us valuable suggestions and ideas.We would also like to thank our college for providing all the necessary resources for the project. We would like to thank everyone involved in this project who helped us with their suggestions to improve the project.

/* Add Content Here as well */

Abstract

In the early 1980s, Professor Teivo Kohonen created the Self-Organizing Map data visualisation approach. SOMs translate multidimensional data onto lower-dimensional subspaces in which geometric correlations between points indicate similarity. The decrease in dimensionality provided by SOMs enables individuals to perceive and analyse data that would otherwise be incomprehensible. SOMs construct subspaces using a neural network trained with a competitive learning algorithm and unsupervised learning. The weights of neurons are modified depending on their proximity to "winning" neurons (i.e. neurons that most closely resemble a sample input). Training through several rounds of input data sets resulted in the grouping of comparable neurons and vice versa.

/* Add Content Here Later after finishing the report */

Table of Contents

1. Introduction to Self-Organizing Maps

Because humans can only view data in two or three dimensions, it may be quite challenging to analyse data with a high dimension. Before and after training machine learning models on big, high-dimensional datasets, it is preferable to have an intuitive grasp of the dataset's connections and patterns to guide the learning process and aid in the interpretation of findings. Self-Organizing Map (SOM) is a dimensionality reduction strategy that may provide us with insights into high-dimensional data with minimum processing. Self-Organizing Maps may be used for exploratory data analysis, clustering issues, and the presentation of high-dimensional datasets.
Teivo Kohonen created SOMs in 1980, and they have been the focus of several research articles.

1.1 Simple Algorithm for Self-Organizing Maps

```
Initialize weights
For 0 to X number of training epochs

    Select a sample from the input data set
    Find the "winning" neuron for the sample input
    Adjust the weights of nearby neurons


End for loop
```

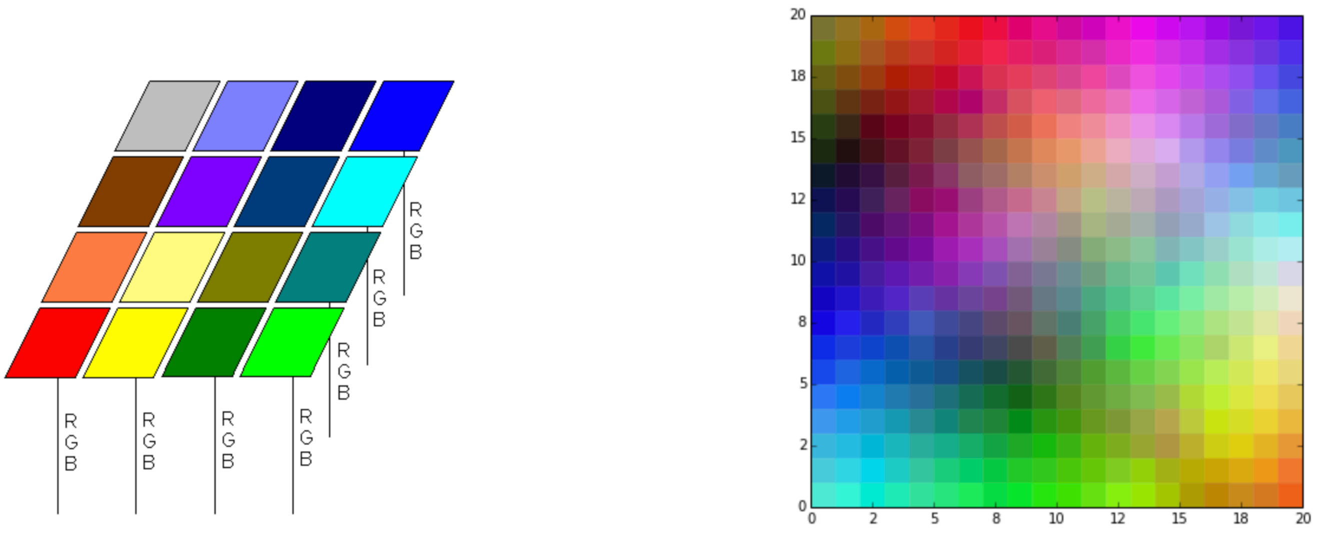

Figure 1:- An Image of SOM classifying RGB input data

2. Building a Self-Organizing Map

2.1 Structure of Self Organizing Map

Lattice Structure :- Both rectangular and hexagonal grids can be employed to define the lattice's structure.

Lattice Size :- The number of nodes in a SOM lattice has a significant impact on the resultant map. The recommendation of Kohonen is to use $5\sqrt{n}$ nodes, where n is the number of data points. If a rectangular map is utilised, Kohonen indicates that the best ratio of grid height to width is equal to the ratio of the two greatest eigenvalues of the autocorrelation matrix.

2.2 Methods for Initializing Weight Vector

Over a lengthy training period, the initialization of a Self-Organizing Map's weight vectors has minimal effect. However, in the interest of saving time, it is preferable to use a starting technique that rapidly achieves equilibrium.

Random Initialization :- Self-Organizing Maps may be started using random weight vector values. This displays the organising ability of the SOM when beginning from an arbitrary state. Random initialization may impede convergence.

Random Sampling Initialization :- Using random data points from the input data as values for the initial weight vectors is an additional straightforward approach for initialization This technique has the benefit that the distribution of the SOM's starting values will mirror the distribution of the data. It is believed that this strategy provides quicker convergence than random initialization.

Best Candidate Sampling Initialization :- The best candidate sampling method attempts to optimise the distance between samples while randomly placing samples. It has been used to construct data subsets with minimal bias and uniform spacing. The approach begins by randomly selecting one point from the input dataset. For each of the remaining weight vectors, a predetermined number of random points are then selected from the input dataset. Calculating the distance between each point in the sample and its closest neighbour. As the current weight vector's value, the point with the greatest closest neighbour distance is picked. This process is continued until each weight vector has been given a value.

Principal Component Initialization :- Principal Component Initialization, or Linear Initialization, might be a highly beneficial strategy for SOM initialization. This approach will often need fewer training iterations than previous methods. Before initialising the weight vectors, we construct the data covariance matrix's eigenvectors. The two greatest eigenvalues' associated eigenvectors are then extracted. The values of the weight vectors are then calculated by constructing a rectangular array along this data subspace. Using this strategy, initial weight vectors begin in an ordered state. For training a SOM started in this manner, a modest learning rate and neighbourhood function are advised. PCI is successful even if the data being modelled does not have a linear connection, since it begins weight vectors at the middle of the data.

3 Training Step For Self-Organizing Maps

3.1 Preprocessing of Data

Rescaling of input data is important as it ensures each feature has the same scale. Usually we scale features between 0 and 1. The data given to a SOM consists of all the data a network receives. If incorrect data are supplied into the SOM, the output is equally incorrect or of poor quality. Consequently, Self-Organizing Map and other neural network models adhere to the "garbage in, garbage out" concept. Therefore, erroneous data should be removed before training a SOM by using common sense and prior domain knowledge.

3.2 Updation of Weights

There are two ways to update the weights of SOM :-

A) Online Update
B) Batch Update

3.2.1 Online Update

In the online update stage, each iteration evaluates a random point from the input dataset. First, the point's best matching unit (BMU) is determined. The best matching unit of a data point x is the node in the input space that is closest to the data point. Let c(x) be the index of the unit that best corresponds to the point x.

$$c(x) = \arg\min_i \|x - w_i\|$$

Once the BMU is computed, we can calculate the updated value of the weight vectors for the k + 1-th iteration.

$$w_i(k+1) = w_i(k) + \alpha(k) h_{i,c(x)} * (x - w_i(k))$$

Here, the neighbourhood function is denoted by $h_{i,c(x)}$. This is a function of the distance from the best matching unit of x, to the i-th lattice node in the lattice space. We have also incorporated the learning rate, α(k), in the update function. The following criteria are defined for α(k):

$$\alpha(k + 1) \leq \alpha(k)$$

3.2.2 Batch Update

The batch update step is somewhat similar to the online update step however rather of employing a single data point for each iteration, the complete dataset is used. To compute it, two things must first be computed. Given the current $w_i(k)$ for I 1,2,3... n, the BMU must be computed for each data point. Also, for a given node j, we need to determine the number of data points whose BMU is node j. We refer to this value as n_j.

$$n_j = \sum_{x \in D} \left(\begin{cases} 1 & c(x) = j \\ 0 & \text{otherwise} \end{cases} \right)$$

The batch update step in its final form is:

$$w_i = \frac{\sum_{j=1}^{n} n_j h_{j,i} \bar{x}_j}{\sum_{j=1}^{n} n_j h_{j,i}}$$

In this instance x^{-}_{j} is the average of all data points closest to the j-th lattice node, in the input space. Also note that hi,j is the neighborhood distance from node i to node j. Note that the batch update step does not require a learning rate.

3.2.3 Batch vs Online

The batch update does not require a learning rate function. This is advantageous because it eliminates the number of needed parameters. The other advantage of batch computation is that it is completely deterministic.

3.3 Neighbourhood Functions

Because neighbourhood function $h_{i,j}$ is what establishes the link between the input space and the lattice space, it must be selected carefully. This relationship is responsible for the map's self-organizing quality.

Some important aspects of neighbourhood function are :-

- $h_{i,j}$ should attain its maximum when $\|l_i - l_j\| = 0$ and be symmetric about the winning node j.
- As $\| l_i - l_j \|$ increases, $h_{i,j}$ should decrease.

Some of the commonly used neighbourhood functions are :-

$$\text{Gaussian - } h_{i,c(x)} = \exp\left(-\frac{\|l_i - l_{c(x)}\|}{2\sigma(k)^2}\right)$$

$$\text{Step/Bubble - } h_{i,c(x)} = \begin{cases} 1 & \|l_i - l_{c(x)}\| \leq \sigma(k) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Triangle - } h_{i,c(x)} = \begin{cases} 1 - \frac{\|l_i - l_{c(x)}\|}{\sigma(k)} & \|l_i - l_{c(x)}\| \leq \sigma(k) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Gaussian with cutoff - } h_{i,c(x)} = \begin{cases} \exp\left(-\frac{\|l_i - l_{c(x)}\|}{2\sigma(k)^2}\right) & \|l_i - l_{c(x)}\| \leq \sigma(k) \\ 0 & \text{otherwise} \end{cases}$$

3.3.1 Choice of Neighbourhood Function

Due to the fact that the bulk of training iterations include a neighbourhood with one or no neighbours, the choice of neighbourhood function is almost irrelevant. As long as the radius is of a sufficient size during the organising phase, it is probable that the map will self-organize. Once the convergence phase has begun, the neighbourhood function is irrelevant so long as the neighbourhood size only catches one or zero neighbours of a particular node.

4 Evaluation of a Self-Organizing Map

In dimensions more than three, it is very difficult to determine if a map that has achieved equilibrium adequately represents the geometry of the dataset. Due of this, it may be quite difficult to judge the quality of a mapping. To assess a trained SOM, it is necessary to understand what defines a successful mapping. A well-fitted SOM should maintain the data's structure, fit the data properly, and eliminate modelling noise.

We use different types of Error to asses a trained SOM :-

4.1 Quantization Error

Quantization Error(EQ) is the average distance of data point to the nearest lattice node. Let c = arg $\min_i \|x - w_i\|$

$$E_Q = \sum_i \|x_i - w_c\|^2$$

EQ can evaluate how well the mapping matches the data distribution. It does not evaluate whether the lattice matches the data's topology or form. Although EQ does not assess topological preservation, it provides a decent indication of how well the map corresponds to the data. This measure is useful for determining if the map has nodes near to dense clusters.

4.2 Topographic Error

Topographic Error seeks to collect information that Quantization Error is unable to. It determines how effectively the data's form is kept in output space. In its most basic form, topographic error is estimated by determining the percentage of data points for which the best matching unit and the 2nd best matching unit are neighbours in the lattice space. Topographic error provides a general indication of how well the map preserves the data's topology.

Let c_i be equal to the index of the i-th best matching unit and let D be the input dataset.

$$E_T = \frac{1}{|D|} \sum_{x \in D} te(x)$$

$$te(x) = \begin{cases} 1 & c_1(x) \text{ and } c_2(x) \text{ are neighbors.} \\ 0 & \text{otherwise.} \end{cases}$$