

Abstract

We present a technical report explaining the theory and methodology of SOM-Self Organising Maps. Concretely, the we address the following:

- 1.Introduction to SOM and its brief history
- 2.Deconstructing SOMs
- 3.Working of SOM explained with an elementary example
- 4.Applying SOM on ‘World Happiness Dataset’
- 5.Code
- 6.Conclusion
- 7.References

In this process, we break down Self Organising Maps to its bare bone structure and then reconstruct it explaining each and every component for an exhaustive and thorough understanding.

1 Introduction

The concept of SOM (self-organizing map) was proposed by the Finnish professor Tuevo Kohonen in 1988. SOMs are therefore sometimes called Kohonen maps/Kohonen networks. Kohonen was inspired by neurobiology. He noted that in the brain, similar neurons are found together and therefore, regions of the cortex are specialized. It uses unsupervised learning. It has three main purposes—clustering, dimensionality reduction and data visualization. Its applications include machine state monitoring, fault identification, world poverty classification etc.

Self Organizing Maps are used to produce a low dimensional (typically a 2D or 3D) representation of a high dimensional feature space, it does this while preserving the topological structure of the data.

In the process of model development, SOM utilizes competitive learning techniques while the conventional neural network applies an error reduction process through backpropagation with a gradient descent algorithm.

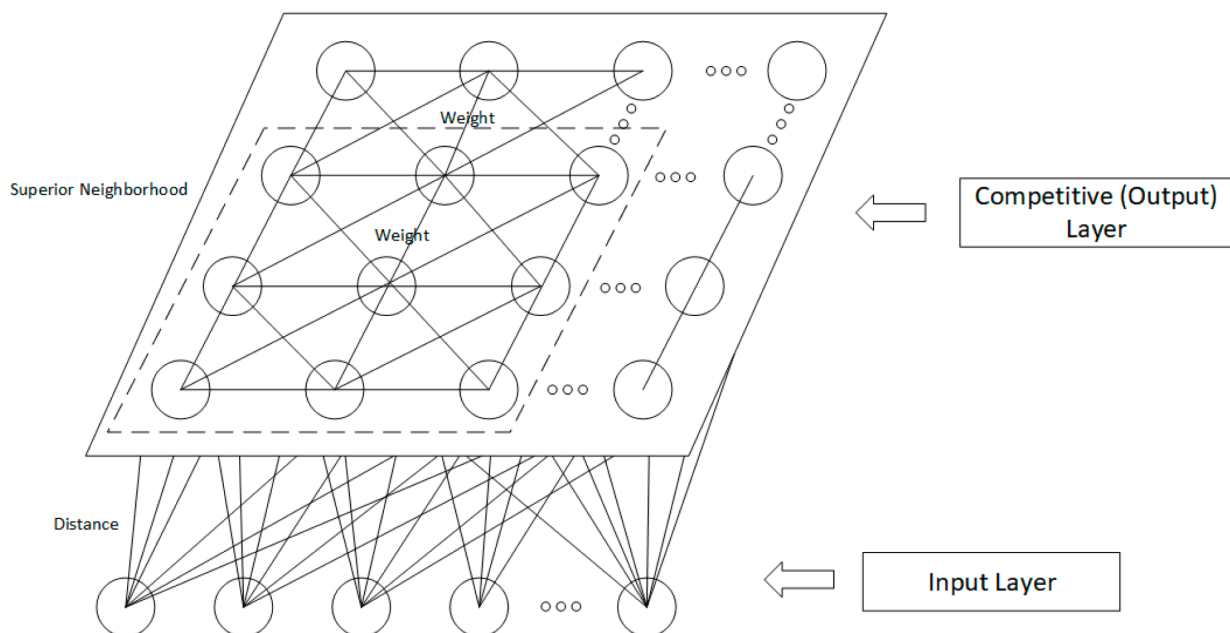
2 Deconstructing Self Organizing Maps (SOM)

Overview

The goal of SOM is to represent an input space with n dimensions as a map space with two/three dimensions (this is easier for visualization and thus is better for forming relations between independent and dependent variables). Due to such applicability SOMs can be considered a dimensionality reduction tool where the map space formed consists of nodes that are arranged in the form of a hexagonal/rectangular grid. All the nodes are associated with a certain weight vector which keeps getting updated with each epoch/iteration to better represent the data. The arrangement and number of nodes to be chosen are all flexible and are up to the programmers discretion, this decision is taken usually by a trial and error method depending upon the dataset.

Nodes in the mapped space remain fixed but it's the weight vector associated with them which gets updated in such a way so that they move closer to the input data. Most common method is the use of Euclidean Distance to measure the distance between nodes and input data.

Once training of the SOM is completed we can use the SOM for classifying new data points by assigning them to the formed clusters and hence performing a Classification operation.



In a SOM the input data is directly connected to the Competitive layer/Output Layer. In this Output layer all the nodes are fully connected. Each input data is iterated over all the nodes and the weight of the winning node i.e the node which is closest to the input data is updated using the weight updation formula which is discussed later in this report.

Learning Algorithm

The base algorithm running behind the training process of SOM follows:

1. Initialize the neural network weights
2. Randomly select an input
3. Select winning neuron by choosing minimum distance neuron(generally Euclidean Distance used)
4. Update the neuron with latest weights

1.Initialization

Initialization takes place either using: Random Initialization(RI) or Principal Component Initialization (PCI).

SOMs tend to be exceptionally sensitive to the initial weight and hence this tends to be an important feature of the model implemented. From research we have observed that for non linear data sets RI performs well.

2.Winning Neuron Selection/Best Matching Unit

During the first epoch after initialization of neuron weights we randomly select input data and calculate the Euclidean Distance with respect to the nodes. For each input data point the node which has shortest distance is assigned the winning neuron.

3.Updating weights for the neurons

Once the winning neurons are identified then the weights of these neurons are updated and this cycle is repeated for the specified number of epochs. This is the final step and is essential since it results in the formation of desired clusters of input data.

Formulas and Hyperparameters

There is a lot of mathematical background to the functioning of SOMs but we shall discuss the essential equations and also take a dive into the hyperparameter tuning which makes major impacts to the model performance.

Weight Updation Formula

Let us consider a single training example x to understand how the weights update.

We are aware that x affects all nodes present in a SOM grid since it pulls these nodes towards itself but the influence of x on the nodes decreases as we move away from the winning node.

$$w_{ij}^{(t+1)} \leftarrow w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$$

The weights w_{ij} is updated at epoch $(t+1)$ using the above formula. Where $\Delta w_{ij}^{(t)}$ is computed using

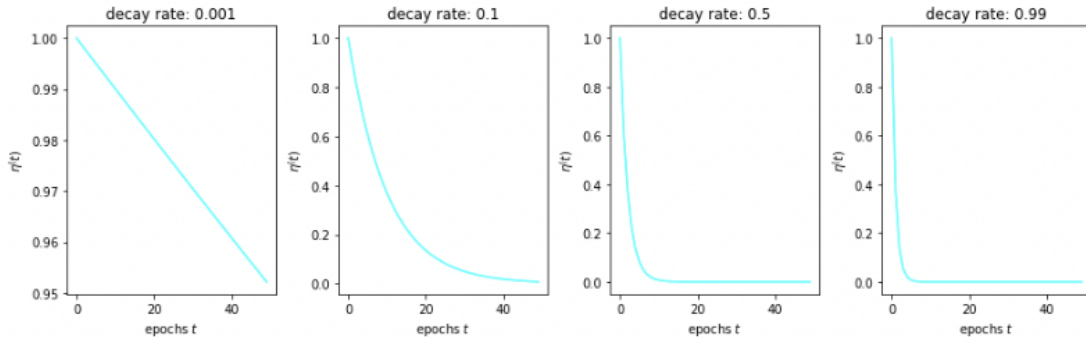
$$\Delta w_{ij}^{(t)} = \eta^{(t)} f_{i,j}(g, h, \sigma_t)(x - w_{ij}^{(t)})$$

- $\eta^{(t)}$ is the learning rate
- $f_{i,j}(g, h, \sigma_t)$ is the neighborhood distance function
- (g, h) are the coordinates of the winning neuron
- σ_t is the radius
- t is the epoch number

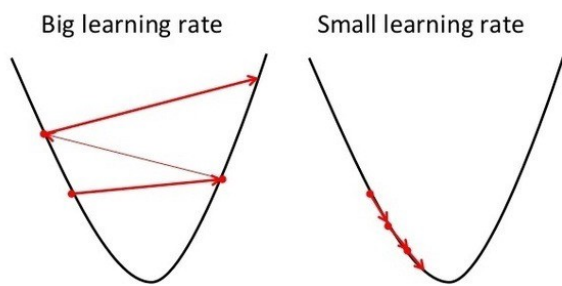
Learning Rate(η)

The learning rate η is in the range $(0,1)$. Its purpose is to control the step during the weight updation of nodes towards the input data. During the first few epochs the value of η is purposefully kept closer to 1 since we want the weights to change faster but as we are approaching the end of our iterations we choose our η value closer to 0 since we want smaller change as we would have already reached optimal weights for the neurons.

Let us try and understand the learning rate in a more comprehensive manner where we choose 4 different decay rates, and observe how the learning rate decreases with subsequent epochs.



In the first case we see that the learning rate reduces very slowly whereas in the fourth case there is a drastic drop in the learning rate. In most applications we want a smooth decrease in the learning rate and hence a decay rate= 0.1 is ideal.



Learning rate must be smaller during the later epochs since we do not want to oscillate away from the minima which corresponds to the ideal weights for our particular nodes in question.

Similarly a higher learning rate is preferred initially since we are really far away from the minima/the target weights.

Neighborhood Distance Function

The neighborhood distance function is denoted below. Where $d((i, j), (g, h))$ is the distance of x (located at (i, j)) from the winning node (located at (g, h)) and σ_t is the radius at epoch t .

$$f_{i,j}(g, h, \sigma_t) = e^{-d((i, j), (g, h))^2 / (2\sigma_t^2)}$$

For a training example x which is located at a large distance from the winning node the neighborhood distance function tends to zero and hence leads to a very small value of Δw_{ij} .

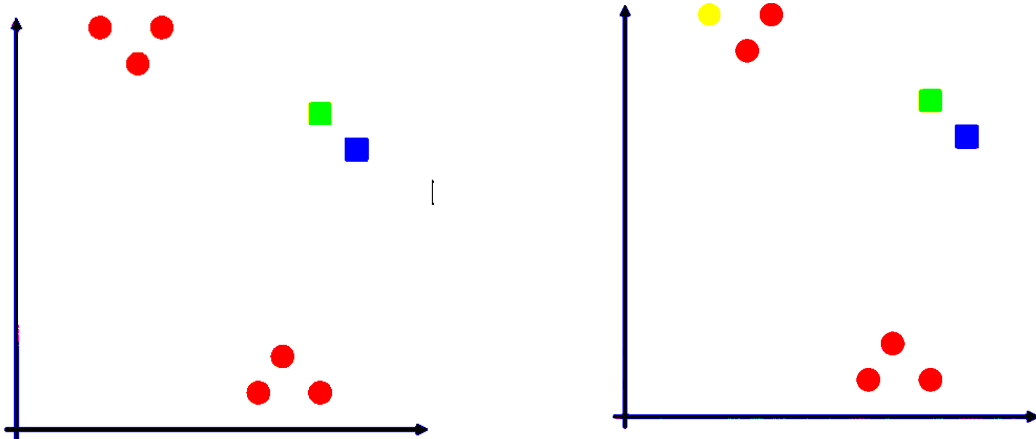
σ_t which denotes the radius determines the region of influence of a training example x . A larger radius affects a larger number of SOM cells and hence in turn the nodes present in these cells. Thus a common strategy followed is that the radius is larger during initial epochs but then slowly reduces as we want more definitive clustering.

3 Working of SOM explained with an elementary example.

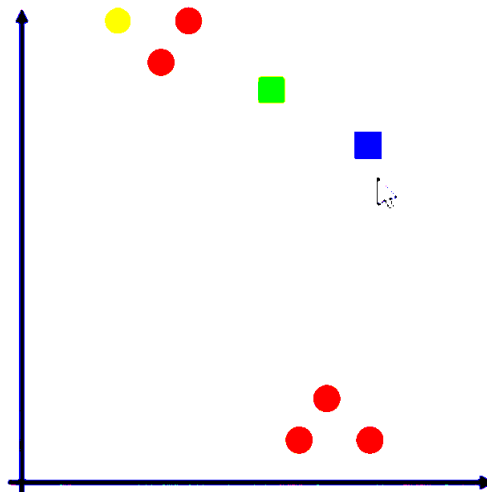
Self Organizing Maps are used for classification and it does this using the following algorithm/approach:

In SOMs, we want the output to have two/three dimensions (represented by the two axes of the graph). In the example shown below, we have 6 inputs (the red circles) and 2 neurons (squares). SOMs will pick a random input and check which neuron it is closest to. The algorithm used for calculating the distances is Euclidean distance calculation.

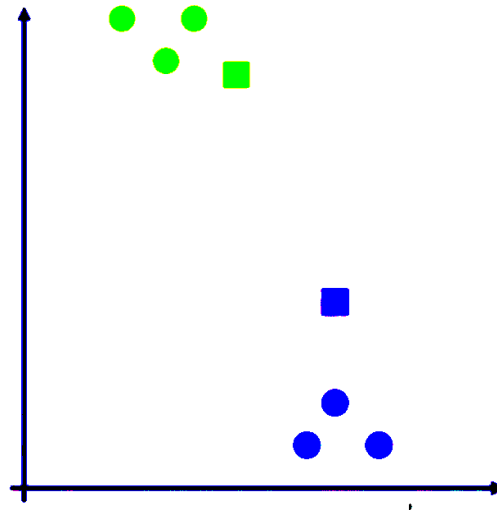
$$\sqrt{\sum_{i=0}^d (n_i - v_i)^2}$$



Next, we change the position of the neurons to account for this (we need to change the position of both as both neurons are interrelated).



After this process is repeated for each input, we obtain the final positions for each neuron. We classify the inputs based on which neuron is closer to the input.



The above example shows how SOMs allow people to visualize and interpret data that otherwise would be indecipherable. It is also evident that SOMs follow the principles of unsupervised learning—teacher is absent and no desired output is required to be fed to teach the model.

SOMs offer many advantages: they are extremely easy to understand and interpretation is simple, however, due to this they are computationally expensive because of the repetitive nature of the algorithm.

4 Applying SOM on ‘World Happiness Dataset’

About the dataset

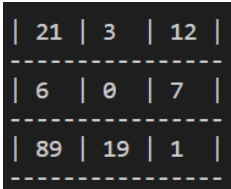
The World Happiness Report is a survey of the state of global happiness and continues to gain global recognition as governments, organizations and civil society increasingly use happiness indicators to make informative decisions. This report reviews the state of happiness in the world and shows how the new science of happiness explains personnel and national variations in happiness.

The report consists of data from 5 years i.e 2015-19 and spans across various countries. The data collected in the report which form our feature space spans across various indicators such as life expectancy, corruption rate, per capita GDP etc and covers over 150 countries. The list of features provided is as follows:

1. Country
2. Region
3. Happiness rank
4. Happiness score
5. Standard error
6. Economy (GDP per capita)
7. Family
8. Health (Life expectancy)
9. Freedom
10. Trust (Government corruption)
11. Generosity
12. Dystopia residual

Applying SOM on the dataset and inferring the Results

We used the World Happiness dataset for the year 2015 and applied it to our code. We observed results similar to any clustering application where countries which had similar happiness scores were grouped together and those with different features were assigned somewhat separate clusters. In our application we used a 3x3 grid to classify the data, i.e. 9 clusters for 158 countries. All features except for Country, Region and Happiness rank were used on the SOM we created. Different numbers of epochs and values for learning rate and radius (along with their decay rates) were tested to see how the parameters modified the clustering output.

	Number of Epochs	Initial Learning Rate	Initial Sigma	Clustering Result
1.	5	0.1	0.1	

2.	5	0.1	7	<table><tr><td>6</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>3</td><td>149</td></tr></table>	6	0	0	0	0	0	0	3	149
6	0	0											
0	0	0											
0	3	149											
3.	5	0.9	2	<table><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>52</td><td>106</td></tr></table>	0	0	0	0	0	0	0	52	106
0	0	0											
0	0	0											
0	52	106											
4.	10	0.9	2	<table><tr><td>41</td><td>1</td><td>94</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>22</td></tr></table>	41	1	94	0	0	0	0	0	22
41	1	94											
0	0	0											
0	0	22											
5.	5	0.1	5	<table><tr><td>9</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>28</td><td>0</td><td>121</td></tr></table>	9	0	0	0	0	0	28	0	121
9	0	0											
0	0	0											
28	0	121											
6.	10	0.1	5	<table><tr><td>0</td><td>110</td><td>43</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>5</td></tr></table>	0	110	43	0	0	0	0	0	5
0	110	43											
0	0	0											
0	0	5											
7.	10	0.01	5	<table><tr><td>9</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>94</td><td>55</td></tr></table>	9	0	0	0	0	0	0	94	55
9	0	0											
0	0	0											
0	94	55											

As can be seen from entries 1 and 2, increasing the value of initial sigma drastically changes the clustering output - with a higher sigma value, only 2 clusters are populated by the end of 5 epochs (much larger separation of clusters). From entries 3 and 4, at a very high initial learning rate, increasing the number of epochs spreads out the data points more than before, i.e. has the opposite effect as before when the initial sigma value was changed. This kind of focused clustering behavior is also observed with moderate values of learning rate and initial sigma, where number of epochs is increased from 5 to 10, resulting in only 3 final clusters with 110, 43 and 5 data points each.

We took our final results after running the SOM for 10 epochs, with initial sigma = 5 and initial learning rate = 0.01. The cluster labels were added to the csv file of the dataset using a small python script for ease of implementation (using Pandas). We then tried to study the resulting file, cluster-wise, focusing on the features that did not show too much variation within clusters.

Three clusters were created as follows:

Cluster 0 characteristics: Moderate Happiness Score (4.8 - 5.4) but with low Dystopian Residual

Cluster 1 characteristics: Moderate to high Happiness Score and high Family Score

Cluster 2 characteristics: Low Happiness Score

5 Code

A Self Organising Map was implemented from scratch in C without the use of any external libraries other than the standard C libraries.

Structs

Structs were defined for both the SOM itself that needs to be initialised and trained, along with the data that is being read from the csv file used for the dataset, as with C we are not afforded the ease of use of packages like pandas that aid dataset manipulation in Python. The struct for the SOM itself includes the following fields, with their descriptions:

Field	Description	Type
<code>grid</code>	3D matrix of dimensions (height x width x data dimensions)	<code>double***</code>
<code>height</code>	First dimension of the SOM grid, its value is set by the user based on the application	<code>int</code>
<code>width</code>	Second dimension of the SOM grid, its value is set by the user based on the application	<code>int</code>
<code>alpha0</code>	The initial value of the learning rate chosen, which decreases with iterations based on the value of learning rate decay	<code>double</code>
<code>sigma0</code>	The initial value of the radius chosen for the neighborhood function, which decreases with iterations based on the value of radius decay	<code>double</code>
<code>learnDecay</code>	Learning rate decay	<code>double</code>
<code>distDecay</code>	Radius decay	<code>double</code>
<code>epochs</code>	Number of epochs for which the SOM should be trained	<code>int</code>

The values from the csv files are similarly extracted and stored in a struct for the data, along with some related information:

Field	Description	Type
<code>examples</code>	Training data as a matrix of dimensions (num x data dimensions)	<code>double**</code>
<code>dims</code>	Number of dimensions i.e. features (input by the user)	<code>int</code>
<code>num</code>	Number of training examples (calculated by the program)	<code>int</code>
<code>cat</code>	Array of feature indices that are categorical i.e. restricted to integer values (input by the user)	<code>int**</code>
<code>numCat</code>	Number of categorical features (input by the user)	<code>int</code>

ranges	2D matrix of dimensions (dims x 2) that stores the minimum and maximum value for each feature (calculated by the program)	double**
names	Array of length num of names of countries corresponding to each training example	char**

Functions

Apart from some simple helper functions, 11 functions have been defined for the process:

1. **Data* readFile(FILE* fp, int* catOrig, int dims, int numCatOrig)**: Takes a file pointer and reads and stores the data from the csv file into the Data struct
2. **void getRanges(Data* data)**: Calculates and stores the minimum and maximum values for the features involved
3. **double getDistance(double* currRec, double* currUnit, int dims)**: Calculates the Euclidean distance between a given training example and a SOM unit
4. **int* getBMU(int index, double** examples, Som* som)**: Iterates over all SOM units to get the BMU (Best Matching Unit) for the given training example
5. **double getSigma(double sigma, double distDecay, int currIter)**: Calculates the value of the radius using the current iteration number
6. **double getAlpha(double alpha, double learnDecay, int currIter)**: Calculates the value of the learning rate based on the current iteration number
7. **double getNDF(Som* som, int a, int b, int i, int j, double sigma)**: Calculates the value of the neighborhood distance function for 2 given SOM units
8. **void initialiseGrid(Data* data, Som* som)**: Initialises the SOM units using random training examples
9. **void initialiseGridRand(Data* data, Som* som)**: Initialises the SOM units using random values between the ranges calculated for each feature
10. **void update(int index, Data* data, Som* som, int step, double alpha, double sigma, int a, int b)**: For a specific training example, updates the weights for the SOM units within the step value of the BMU
11. **Som* trainSom(Data* data, int epochs, double alpha, double sigma, int height, int width, double learnDecay, double distDecay, int dims, int num, int gridInitialise)**: Trains the SOM for the desired number of epochs by calling the related functions after initialising the grid

Initial Grid Values

The 2 ways in which a SOM grid can be initialised i.e. either by taking random training examples or by generating random numbers for each field are useful for different applications. Since the updation of weights depends heavily on the previous weights of a SOM unit, the final clustering output of a SOM also depends heavily on the choice of initial weights. While random initialisation favours non-linear datasets, initialisation by training examples may favour quasi-linear datasets. Hence, in our implementation, this choice of initialisation mode is left up to the user, wherein the variable **gridInitialise** is input as 0 if training examples are to be directly used and is input as 1 if random initialisation is desired (in the function **trainSom**).

Categorical Features

Our implementation of SOM focuses mainly on features whose range lies in the Real numbers. For the application we have chosen i.e. the World Happiness Dataset, all features chosen are real valued only and hence the model runs smoothly. We have included some support for features that take only integer values, by ensuring that the value returned after weight updation is an **int** and not **double**, but we have not created functions for handling string features or other categorical variables (like gender, which would take the values M/F) by converting them into integer valued features using encoding.

6 Conclusion

It may be concluded that using Self Organizing Maps (SOMs) to solve clustering problems is a promising approach. The results of running the SOM on the World Happiness dataset reveal that it was able to identify clusters and their features accurately.

Before conducting this analysis we had read about how SOMs are extremely sensitive to hyperparameter tuning and our results just prove the same. We were able to achieve quite diverse results by varying the number of epochs, learning rate, and initial sigma value.

We used a 3x3 grid for clustering the 158 countries in the dataset and made some very basic observations regarding the changes influenced by tuning some of these hyperparameters.

We combined these observations to perform a focused cluster analysis which divided the countries into groups of 110, 43 and 5 points each and was done neglecting only the Country, Region and Happiness Rank data from the dataset.

Note: The 3 clusters with their characteristic features have been elaborately defined in section 4 of the report.