Title: Implementation of Kohonen Network Algorithm

Objectives:

1. Clustering for unsupervised data
2. Find the closest matching unit to a training input
3. Increase the similarity of this unit and those in the neighboring proximity, to the input.

Methodology:

Kohonen Network Algorithm is an unsupervised technique used for clustering for unsupervised data. Moreover, it gives output in real time and don't need any calculation while predicting the output or class like other algorithms for example Hopfield algorithm.

The learning algorithm organizes the nodes in the grid into local neighbors that act as feature classifiers on the input data. The topographic map is autonomously organized by a cyclic process of comparing input patterns to vectors "stored" at each node. No training response is specified for any training input. Where inputs match the node vectors, that area of the map is selectively optimized to represent an average of the training data for that class. From a randomly organized set of nodes the grid settles into a feature map that has local representation and is self-organized. The algorithm itself is notionally very simple.

1. **Initialize network:** Define $w_{ij}(t)(0 \leq i \leq n-1)$ to be the weight from input i to node j at time t. Initialize weights from the n inputs to the nodes to small random values. Set the initial radius of the neighborhood around node j, $N_j(0)$, to be large.

2. **Present input:** Present input $x_0(t), x_1(t), x_2(t), \dots, x_{n-1}(t)$, where $x_i(t)$ is the input to node i at time t.

3. **Calculate distances:** Compute the distance d_j between the input and each output node j, given by
$$d_j = \sum_{i=0}^{n-1} \left(x_i(t) - w_{ij}(t)\right)^2$$

4. **Select minimum distance:** Designate the output node with minimum d_j to be $j*$.

5. **Update weights:** Update weights for node $j*$ and its neighbors, defined by the neighborhood size $N_{j*}(t)$. New weights are
$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(x_i(t) - w_{ij}(t))$$
For j in $N_{j*}(t), 0 \leq i \leq n-1$

The term $\eta(t)$ is a gain term $(0 < \eta(t) < 1)$ that decreases in time, so slowing the weight adaptation. Notice that the neighborhood $N_{j*}(t)$ decreases in size as time goes on thus localizing the area of maximum activity.

6. **Repeat by going to 2.**

Is there any biological justification for such a learning rule? As we have already seen, Kohonen has based most of his works on close studies of the topology of the brain's cortex region, and indeed there would appear to be a good deal of biological evidence to support this idea.

Activation in a nervous cell is propagated to other cells via axon links (which may have an inhibitory or excitatory effect at the input of another cell). However, we have not considered the question of how the axon links are affected by lateral distance from the propagating neuron. A simplified yet plausible model of the effect is illustrated by the Mexican hat function shown in the following picture.

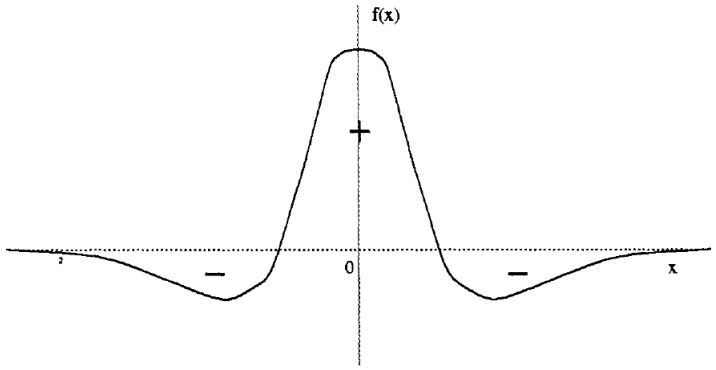


Figure: The Mexican hat function describes the effect of lateral interconnections.

We can see that cells physically close to active cell have the strongest links. Those at a certain distance actually switch to inhibitory links. It is this phenomenon to which Kohonen attributes to the development (at least in part) of localized topological mapping in the brain. He has modelled this effect by using only locally interconnected networks and restricting the adaptation of weight values to localized "neighborhoods".

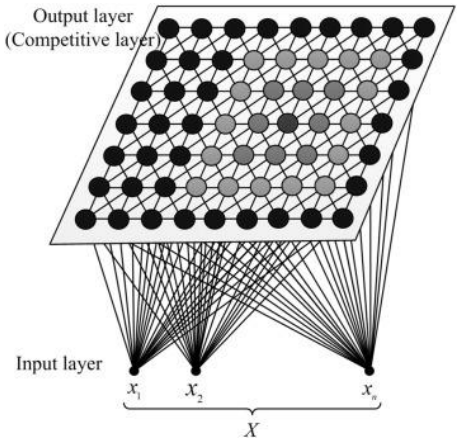


Figure: A Kohonen feature map.

Now, let's focus on the model of Kohonen network. There are two layers in Kohonen network. One is the input layer and the other one is the output layer. All the nodes of input layer are connected with all the nodes of output layer. The model has been showed in the previous figure. The colors in the figure indicates the main cluster (dark gray and semi-dark gray) and its neighbors (light gray). Now, let's look at the flowchart of the algorithm:

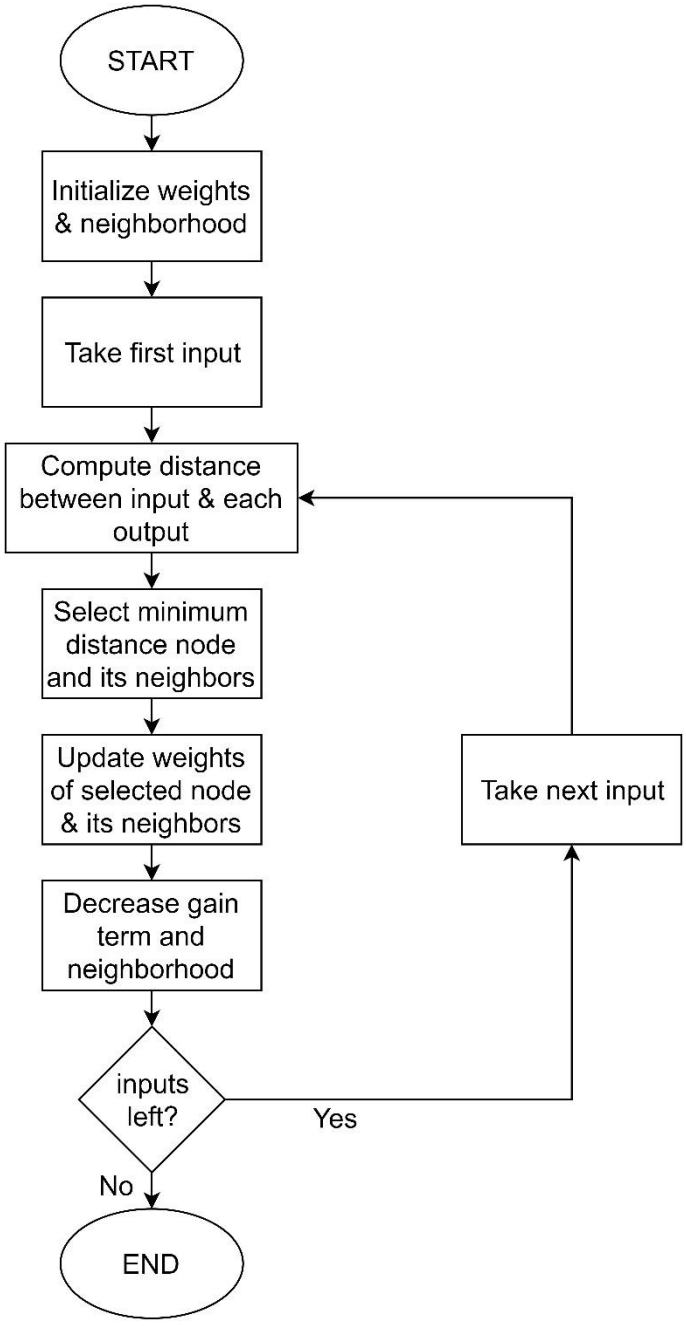


Figure: Flowchart for the Kohonen Network Algorithm.

Every time the same node is selected as win node, Kohonen Network Algorithm decrease the neighborhood size of the winning node. Also, in each step the gain factor decreases. How much the decrease will be – is a matter of tuning and there are a lot of approaches to do that.

Implementation:

To implement Kohonen Network Algorithm, I divided the whole part into two basic segments:

1. Data Generation
2. Main algorithm implementation

To generate data, all the code needs is the number of input lines. If the number of input lines is 5, then this code will generate 32 inputs and save them into a csv file called "Final-Data.csv". Thus, the user has the power to generate as much dataset as needed. As Kohonen Algorithm is an unsupervised technique, there's no need for train-test set generation. That's why this part has been skipped.

Now, in the main algorithm, the csv file is read and Kohonen algorithm is implemented on this. The number of nodes in the output layer is fixed as 20. This number can be varied with the will of user. The number of neighbors is set as 10 and eta value is set as 0.9 as well. All these values can be modified by user. These values have been determined on the basis of a research that says, if there are n number of desired clusters then the number of nodes needed at least in output layer is $10 * n$ and the value of neighbors of each output node should be 10. Eta is set to a larger value in the beginning of the algorithm. So, it has been set to 0.9 and after each iteration of sample it is decreased by 0.02 which can also be modified as well.

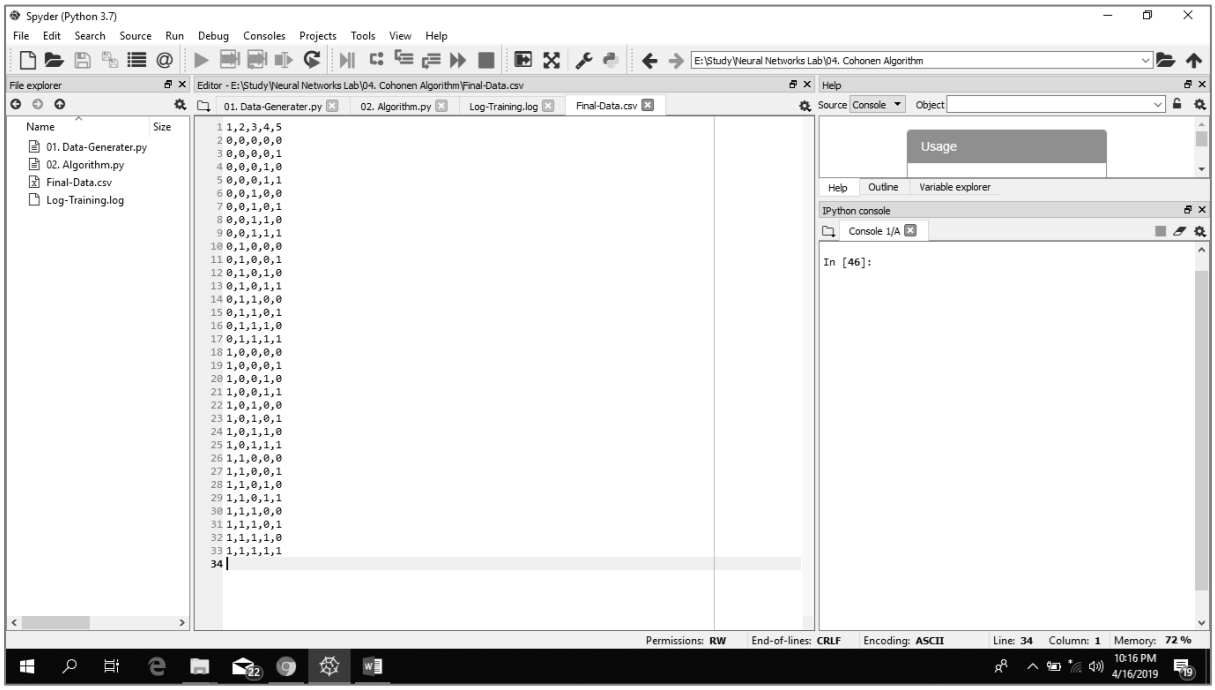
A log file has been generated to show which output node has been predicted for the corresponding input nodes. Without further due, let's jump into the codes.

Code-1: Data Generation

```
"""
Code: Kohonen Network Algorithm
Title: Dataset Generater
Author: Azmain Yakin Srizon
"""

from numpy import binary_repr # Adding binary conversion library

filename = 'Final-Data.csv'
log = open(filename, 'w')  # creating csv file

data = []

input_line = 5 # define input lines

for i in range (0,int(pow(2,input_line))): # generating data as binary form
    tmp = list(binary_repr(i,input_line))
    data.append(tmp) # adding to data

for i in range(0,input_line-1): # generating column names of csv file
    log.write(str(i+1)+',')
log.write(str(i+2)+'\n')

c = 0
for i in data: # adding data to csv file
    for j in i:
        c = c+1
        log.write(str(j)) # adding value in each column for a row
```

```
        if c!=input_line:
            log.write(',')
        else:
            c=0
    log.write('\n') # go to next row

log.close() # closing the csv file
```

Output-1: Data Generation

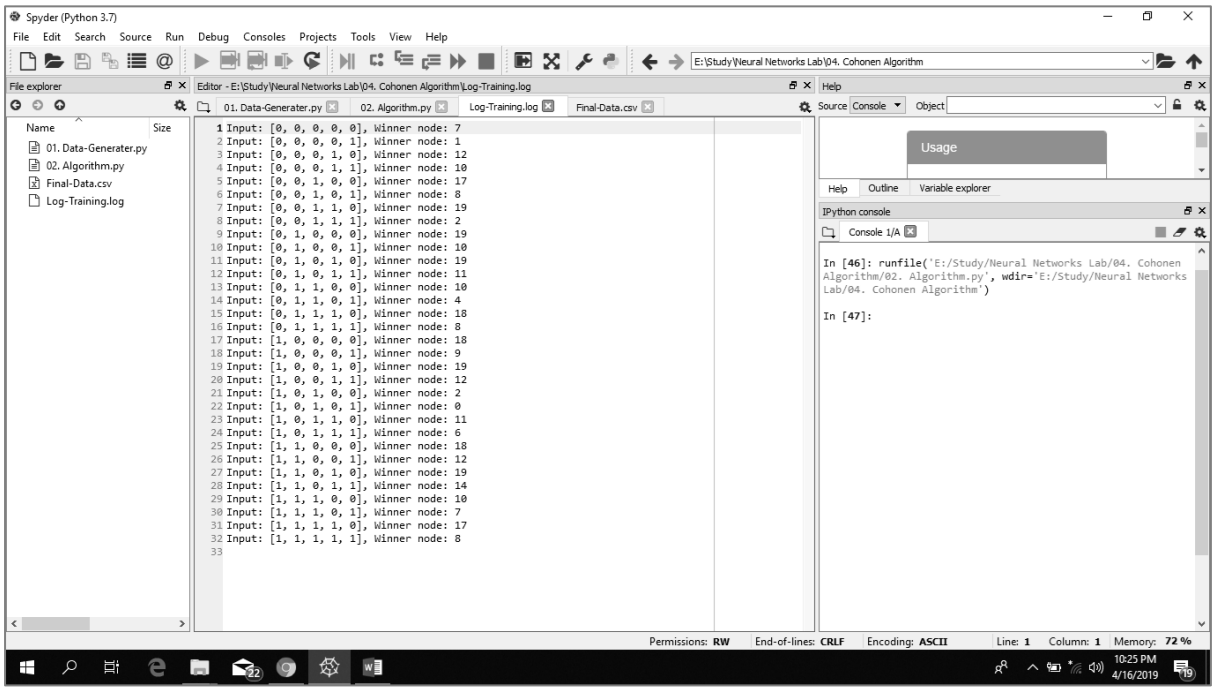


Code-2: Main Algorithm

```
"""
Code: Kohonen Network Algorithm
Title: Algorithm Implementation
Author: Azmain Yakin Srizon
"""

# adding necessary libraries
import pandas
import random

df = pandas.read_csv('Final-Data.csv') # reading csv file

input_nodes = int(df.size/len(df)) # calculating number of input nodes
output_nodes = 20 # calculating numbe rof output nodes
eta = 0.9 #define eta

# initializing weights
weights = []
for i in range(0,input_nodes):
    tmp = []
    for j in range(0,output_nodes):
        tmp.append(0.1*random.randint(1,10))
    weights.append(tmp)

# initializing neighbors
neighbor = []
for i in range(0, output_nodes):
    neighbor.append(10)

# reading the inputs
inputs = []
for j in range(0, len(df)):
    tmp=[]
    for i in range(0,input_nodes):
        tmp.append(df[str(i+1)][j])
    inputs.append(tmp)

filename = 'Log-Training.log'
log = open(filename, 'w') # creating log file
winner_nodes = [] # set of winner nodes

for k in inputs: # for each input
    minimum_distance = 99999999999999999999999
    winning_node = 0
    for j in range(0,output_nodes): # selection of output node
        distance = 0
        for i in range(0,input_nodes): # selection of input node
            distance  =  distance  +  (k[i]-weights[i][j])*(k[i]-weights[i][j])  #
calculating distance
        if distance < minimum_distance: # find out minimum distance node
            minimum_distance = distance
            winning_node = j # winner node
            winner_nodes.append(winning_node)
    log.write('Input: ' + str(k) + ', Winner node: ' + str(winning_node) + '\n') #
writing the result

    # updating weights for winner node
    for i in range(0,input_nodes):
        weights[i][winning_node]    =    weights[i][winning_node]    +    eta*(k[i]-
weights[i][winning_node])

    # updating weights for neighbors in left of winner nodes
    for j in range(1, neighbor[winning_node]+1):
        for i in range(0,input_nodes):
            if winning_node-j>=0:
```

```
                weights[i][winning_node-j] = weights[i][winning_node-j] + eta*(k[i]-
weights[i][winning_node-j])

    # updating weights for neighbors in right of winner nodes
    for j in range(1, neighbor[winning_node]+1):
        for i in range(0,input_nodes):
            if winning_node+j<output_nodes:
                weights[i][winning_node+j] = weights[i][winning_node+j] + eta*(k[i]-
weights[i][winning_node+j])

    # decreasing value of neighbour and eta
    if neighbor[winning_node]>1:
        neighbor[winning_node] = neighbor[winning_node]-1
    eta = eta-0.02

log.close() # closing log file
```

Output-2: Main Algorithm



Performance Analysis:

As there are a lot of parameters here, hence it can be understood that by changing the values of these parameters we can change the performance of the algorithm. The main parameters here are the number of input and output nodes, the value of neighborhood and eta.

Increasing the number of inputs will give a large number of samples to the algorithm. Hence the algorithm will have enough samples to feed the kernel and cluster it more efficiently. However, if the number of inputs is increasing the number of output nodes is not necessary to be increased if the number of desired class is small. Otherwise an increasing value of output nodes will not help in clustering at all.

The number of output nodes however has a pretty amazing impact on the algorithm. Number of output nodes should be higher in terms of clustering. It can be illustrated with an example. Say, we want to recognize 60 students from 60 picture samples. Now if the number of output nodes is small (say 60), then there is a chance that same output node will predict for more than one samples. To avoid this problem, the output nodes should be taken this much large so that nodes are not repeated in general as winning nodes every time. Research shows that if 60 samples are needed to be clustered then at least 600 nodes are needed in output layer.

The value of neighborhood is decreased with time. The reason behind is to have a lower impact on neighbors as the time goes on. And it does make sense. If all the neighbors get affected every time, then same neighbor will predict two cluster for same category inputs. That's why it's necessary to decrease the value. However, how much to decrease? - depends on tuning.

The value of eta is also decreased with time. The main reason is to make the adaptation slower, so that the solution doesn't get stuck in local minima. Slower adaptation will give a solution which is either a global solution or near to global solution. So, the value is decreased.

One thing should be in mind that eta should not be zero or the predicted node will always be zero. Apart from these, this algorithm also has some limitations:

1. This works for unsupervised data only
2. For all datasets it doesn't give a better cluster
3. Sometimes local solutions dictate the clusters

Apart from these limitation Kohonen Network Algorithm is considered as one of the finest inventions for unsupervised data in neural networks.