

Problem Statement: Implement the Kohonen Self-Organizing Neural Network algorithm.

Introduction: Kohonen is a kind of Self-Organizing Neural Network. It is a kind of unsupervised learning method as there is no parameter which drives the learning process. A Self-Organizing Map in short SOM is not like the typical Artificial Neural Networks (ANN). The differences are in the properties of algorithm and their structure. In ANN there can be multiple layers each with multiple number of nodes but the SOM architecture contains a single layer linear 2D neuron grids. Each input remains connected to each of the nodes, but there is no connection in between the nodes which doesn't permit them to know their neighbor's values. The nodes update their weight value to the connections according to a given function of the input data. The grid also called the map organizes itself at every iterations as a function of the provided input data.

For every input, Euclidean distance is calculated from the initialized weights. The formula for calculating the distance is as follows:

Euclidean Distance measure: Suppose, in a rectangular coordinate system we have two vectors (X and Y) and we want to find the distance between them. According to the Euclidean distance measure technique the formula for calculating the distance is:

$$d(X, Y)_{euc} = \sqrt{\sum_{k=1}^n (X_k - Y_k)^2}$$

Where, n defines the dimensionality of the vector.

Now, let's take a two dimensional example. Then the formula will be:

$$d(X, Y)_{euc} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

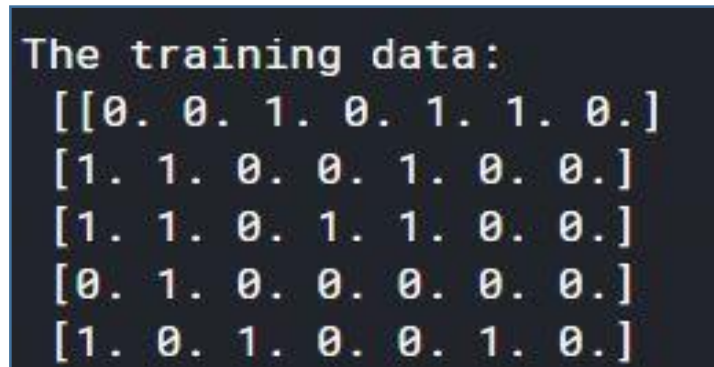
After calculating the distances from each node for a given input the weights of the closest node is updated. The formula for weight updating is given below:

$$w_{ij}(t + 1) = w_{ij}(t) + \alpha(t)(x_i(t) - w_{ij}(t))$$

Here, $w(t)$ is the new weight. α is the learning rate and $x(t)$ is the input elements.

The calculation continues and the inputs becomes the member of the cluster nodes closest to them. The calculation stops when certain criteria is fulfilled.

Dataset Description and Preprocessing: The dataset for Kohonen Self-Organizing Neural Network algorithm was randomly generated. For this training purpose 100 input data were taken each of them is of 6 bits. The values were in between 0 and 1. After that an extra column of zero was appended along with the dataset. This was done to store the cluster number for each input in future. A sample of the dataset is demonstrated below:



```
The training data:
[[0. 0. 1. 0. 1. 1. 0.]
 [1. 1. 0. 0. 1. 0. 0.]
 [1. 1. 0. 1. 1. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 0. 1. 0.]
```

Methodology: For this task the number of nodes taken was 5. An initial value of 0.7 was taken as the learning rate. Then the calculation was iterated for 3000 epochs but a breaking condition was induced in the code which states if the average weight becomes smaller the certain threshold value then stop the training. The threshold value was chosen to be 0.49.

Code:

```
import numpy as np, random

import math

import matplotlib.pyplot as plt

import scipy

from PIL import Image

from scipy import ndimage

#from dnn_app_utils_v3 import *

%matplotlib inline

plt.rcParams['figure.figsize'] = (5.0, 4.0) # set default size of plots

plt.rcParams['image.interpolation'] = 'nearest'

plt.rcParams['image.cmap'] = 'gray'

k = 5 #number of neurons or clusters

p = 0
```

```
alpha = 0.7 # Initial learning rate
```

```
random.seed(1)
```

```
X = np.array([[random.randint(0,1) for i in range(6)] for j in range(100)]) #fix the random value range between 0 and 1
```

```
labels = np.array([[random.randint(0,4) for i in range(1)] for j in range(n)])
```

```
# Print the number of data and dimension
```

```
n = len(X)
```

```
d = len(X[0])
```

```
addZeros = np.zeros((n, 1))
```

```
X = np.append(X, addZeros, axis=1) #initializing column to hold the cluster number later
```

```
print("The SOM algorithm: \n")
```

```
print("The training data: \n", X)
```

```
print("\nTotal number of data: ",n)
```

```
print("Total number of features: ",d)
```

```
print("Total number of Clusters: ",k)
```

```
# Create an empty array of centers
```

```
#C = np.zeros((k,d+1))
```

```
weight = np.random.rand(d,k) #initializing weight
```

```
print("\nThe initial weight: \n", np.round(weight,2))
```

```
#print("Average of weights:", np.round(np.sum(weight)/(d*k), 3))
```

```
def euclidean_distance(row1, row2):
```

```
    distance = 0.0
```

```
    for i in range(len(row1)-1):
```

```
        distance += (row1[i] - row2[i])**2
```

```
    return math.sqrt(distance)
```

```
accuracy = []
```

```
for it in range(3000):# Total number of iterations
```

```
    for i in range(n):
```

```
        distMin = 99999999
```

```
        for j in range(k):
```

```

        #print("I J",i,j)

        dist = euclidean_distance(weight[:,j], X[i,0:d])

        if distMin>dist:

            distMin = dist

            jMin = j

            X[i,-1] = jMin

            #y.append(jMin)

        #if (it == 0):

            #y.append(jMin)

        #print("jMin dist",jMin,dist)

        weight[:,jMin] = weight[:,jMin]*(1-alpha) + alpha*X[i,0:d]  #weight updating

        #print(weight)

    counter=0

    for i in range(n):

        if(X[i,-1]==labels[i,0]):

            counter+=1

    acc=(counter/n)*100

    accuracy.append(acc)

    #print(accuracy)

    if ((np.sum(weight)/(d*k))<0.49):

        break

    alpha = 0.5*alpha


plt.plot(accuracy)

plt.ylabel('accuracy')

plt.xlabel('iterations')

plt.title("Accuracy Curve")

plt.show()

print("\n\nThe final weight: \n",np.round(weight,4))

print("Average of final weights:", np.round(np.sum(weight)/(d*k), 3))

#y=np.array(y)

#y=y.reshape(n,1)

#print(y)

print("\n\nThe data with cluster number: \n", X)

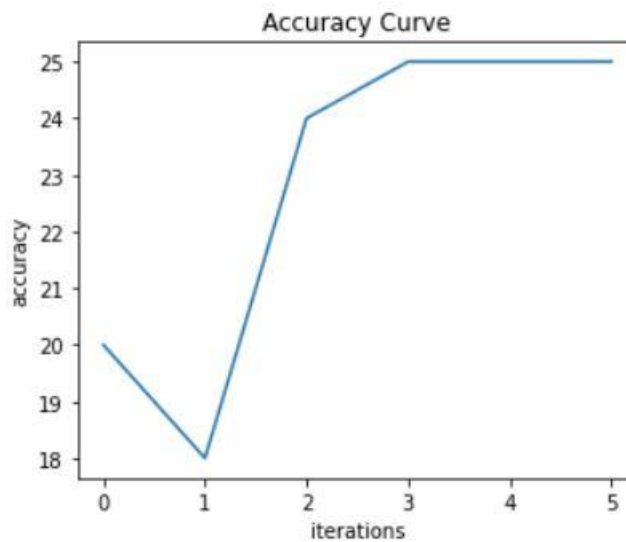
print("Accuracy: ",np.round(accuracy[-1], 2))

```

Output: A sample of the output dataset along with their respective cluster number is demonstrated below:

```
The data with cluster number:  
[[0. 0. 1. 0. 1. 1. 1.]  
 [1. 1. 0. 0. 1. 0. 3.]  
 [1. 1. 0. 1. 1. 0. 3.]  
 [0. 1. 0. 0. 0. 0. 0.]  
 [1. 0. 1. 0. 0. 1. 4.]
```

The accuracy curve is given below:



Conclusion: In this unsupervised learning technique the accuracy was very low. The main cause behind this is the randomly generated data and labels. The performance can be improved using a well-organized dataset and experimentally selected parameter values.