SAP ID:

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

NAME:DIVYESH KHUNT **SAPID:60009210116** **BATCH:D12**

COURSE CODE: DJ19DSC501 **DATE:**

COURSE NAME: Machine Learning - II **CLASS: AY 2023-24**

LAB EXPERIMENT NO. 7

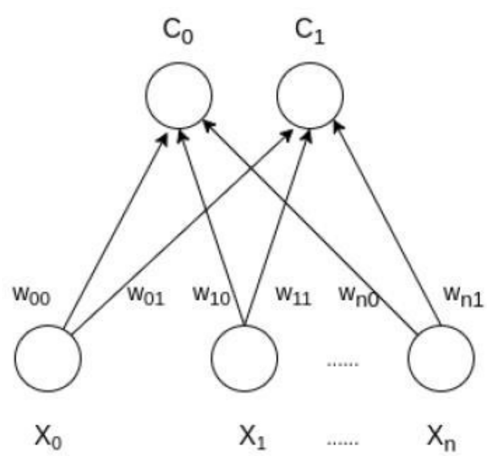
AIM :

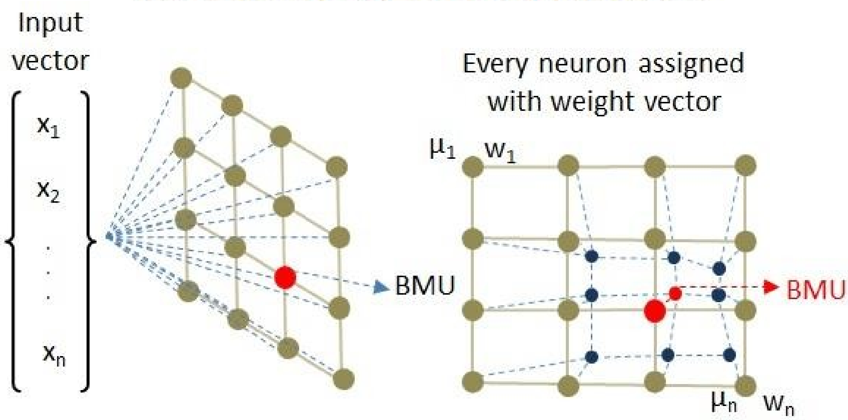
Anomaly detection using Self-Organizing Network.

THEORY:

Self-Organizing Maps:

Self Organizing Map (or Kohonen Map or SOM) is a type of Artificial Neural Network that follows an unsupervised learning approach and trains its network through a competitive learning algorithm. SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation. SOM has two layers, one is the Input layer and the other one is the Output layer. The architecture of the Self Organizing Map with two clusters and n input features of any sample is given below:





The underlying idea of the SOMs training process is to examine every node and find the one node whose weight is most like the input vector. The winning neuron is known as Best Matching Unit(BMU). The weights of the neighbouring neuron are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its weights get altered. The training is carried out in a few steps and over many iterations. The output of the SOMs is a two-dimensional map and color-coding is used to identify any specific group of data points.

Hyperparameters:

SOMs are a two-dimensional array of neurons. So, to define SOMs it is required to know how many rows and columns and neurons are needed in order of the x and y dimensions. The parameters of SOM are:

[1] x: som_grid_rows, is the number of rows
[2] y: som_grid_columns, is the number of columns
[3] Sigma is the neighborhood radius - All the nodes that fall in the radius of the BMU get updated according to their respective distance from the BMU.
[4] learning_rate – weight adjustment at each step

Tasks to be performed:

1. Use **Credit Card Applications DATASET:**

   Source: https://www.kaggle.com/datasets/ujjwal9/credit-card-applications

   The data has 690 records and 16 features along with a class label and customerID. Since SOMs are an unsupervised technique, don't use the class column and also drop the customerID column.

2. **Detect fraud customers in the dataset using SOM and perform hyperparameter tuning. Show map and use markers to distinguish frauds.**
3. **List Applications of Self-Organizing Networks.**
4. **What do you think is the loss function that needs to be computed for SOMs?**

5. **State disadvantages of Kohonen Maps**

For reference:

https://www.superdatascience.com/blogs/the-ultimate-guide-to-self-organizing-maps-soms

```
!pip install minisom
```

```
Collecting minisom
  Downloading MiniSom-2.3.1.tar.gz (10 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: minisom
  Building wheel for minisom (setup.py) ... done
  Created wheel for minisom: filename=MiniSom-2.3.1
  Stored in directory: /root/.cache/pip/wheels/c7/9
Successfully built minisom
Installing collected packages: minisom
Successfully installed minisom-2.3.1
```

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data = pd.read_csv('Credit_Card_Applications.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   CustomerID  690 non-null    int64  
 1   A1          690 non-null    int64  
 2   A2          690 non-null    float64
 3   A3          690 non-null    float64
 4   A4          690 non-null    int64  
 5   A5          690 non-null    int64  
 6   A6          690 non-null    int64  
 7   A7          690 non-null    float64
 8   A8          690 non-null    int64  
 9   A9          690 non-null    int64  
 10  A10         690 non-null    int64  
 11  A11         690 non-null    int64  
 12  A12         690 non-null    int64  
 13  A13         690 non-null    int64  
 14  A14         690 non-null    int64  
 15  Class       690 non-null    int64  
dtypes: float64(3), int64(13)
memory usage: 86.4 KB
```

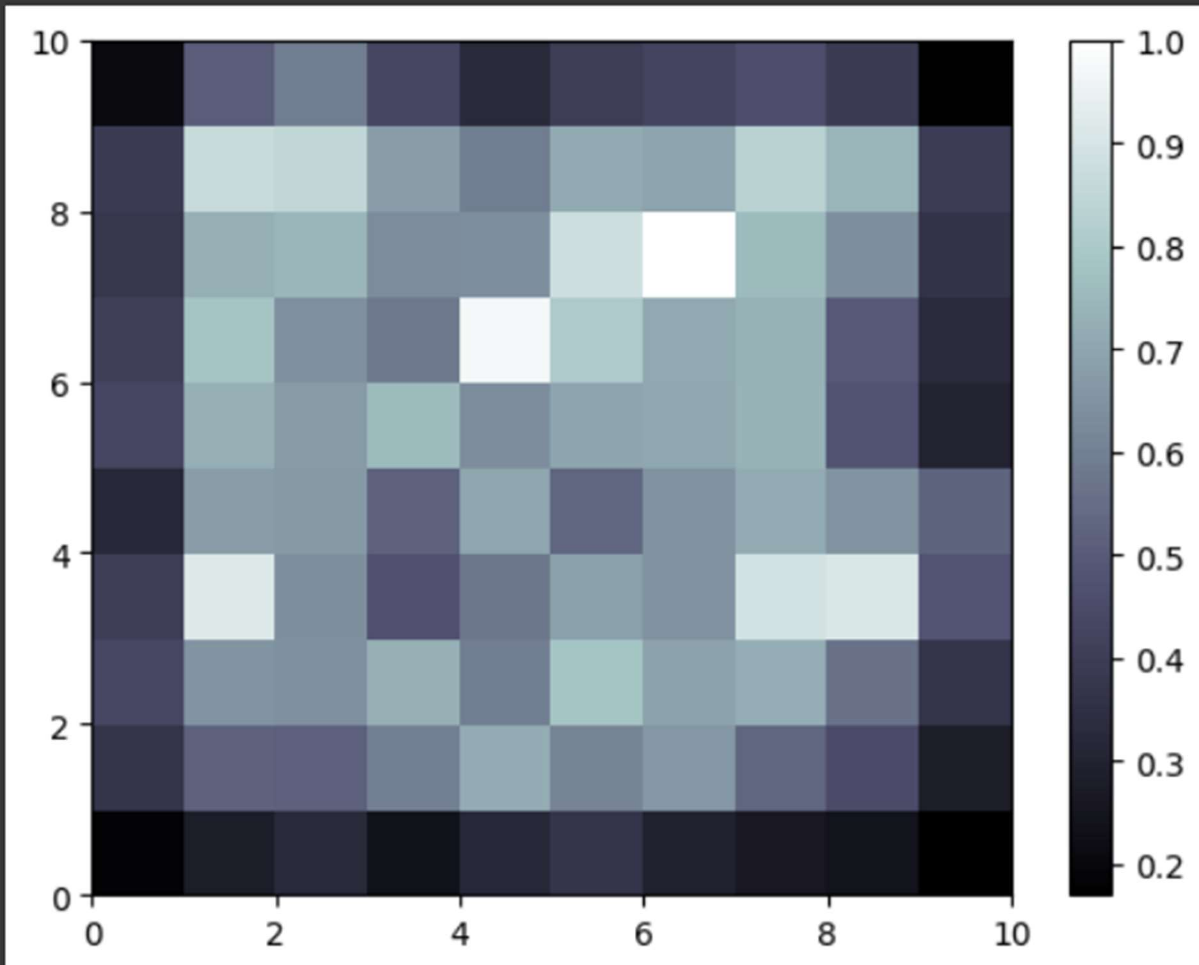
```
X = data.iloc[:, 1:14].values
y = data.iloc[:, -1].values
pd.DataFrame(X)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1.0	22.08	11.460	2.0	4.0	4.0	1.585	0.0	0.0	0.0	1.0	2.0	100.0
1	0.0	22.67	7.000	2.0	8.0	4.0	0.165	0.0	0.0	0.0	0.0	2.0	160.0
2	0.0	29.58	1.750	1.0	4.0	4.0	1.250	0.0	0.0	0.0	1.0	2.0	280.0
3	0.0	21.67	11.500	1.0	5.0	3.0	0.000	1.0	1.0	11.0	1.0	2.0	0.0
4	1.0	20.17	8.170	2.0	6.0	4.0	1.960	1.0	1.0	14.0	0.0	2.0	60.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	1.0	31.57	10.500	2.0	14.0	4.0	6.500	1.0	0.0	0.0	0.0	2.0	0.0
686	1.0	20.67	0.415	2.0	8.0	4.0	0.125	0.0	0.0	0.0	0.0	2.0	0.0
687	0.0	18.83	9.540	2.0	6.0	4.0	0.085	1.0	0.0	0.0	0.0	2.0	100.0
688	0.0	27.42	14.500	2.0	14.0	8.0	3.085	1.0	1.0	1.0	0.0	2.0	120.0
689	1.0	41.00	0.040	2.0	10.0	4.0	0.040	0.0	1.0	1.0	0.0	1.0	560.0

690 rows × 13 columns

```
[ ] from sklearn.preprocessing import MinMaxScaler
    from minisom import MiniSom
    sc = MinMaxScaler(feature_range = (0, 1))
    X = sc.fit_transform(X)
    pd.DataFrame(X)
    som = MiniSom(x = 10, y = 10, input_len=13, sigma=1, learning_rate=0.5)
    som.random_weights_init(X)
    som.train_random(X,20000)
    win=som.distance_map()
```

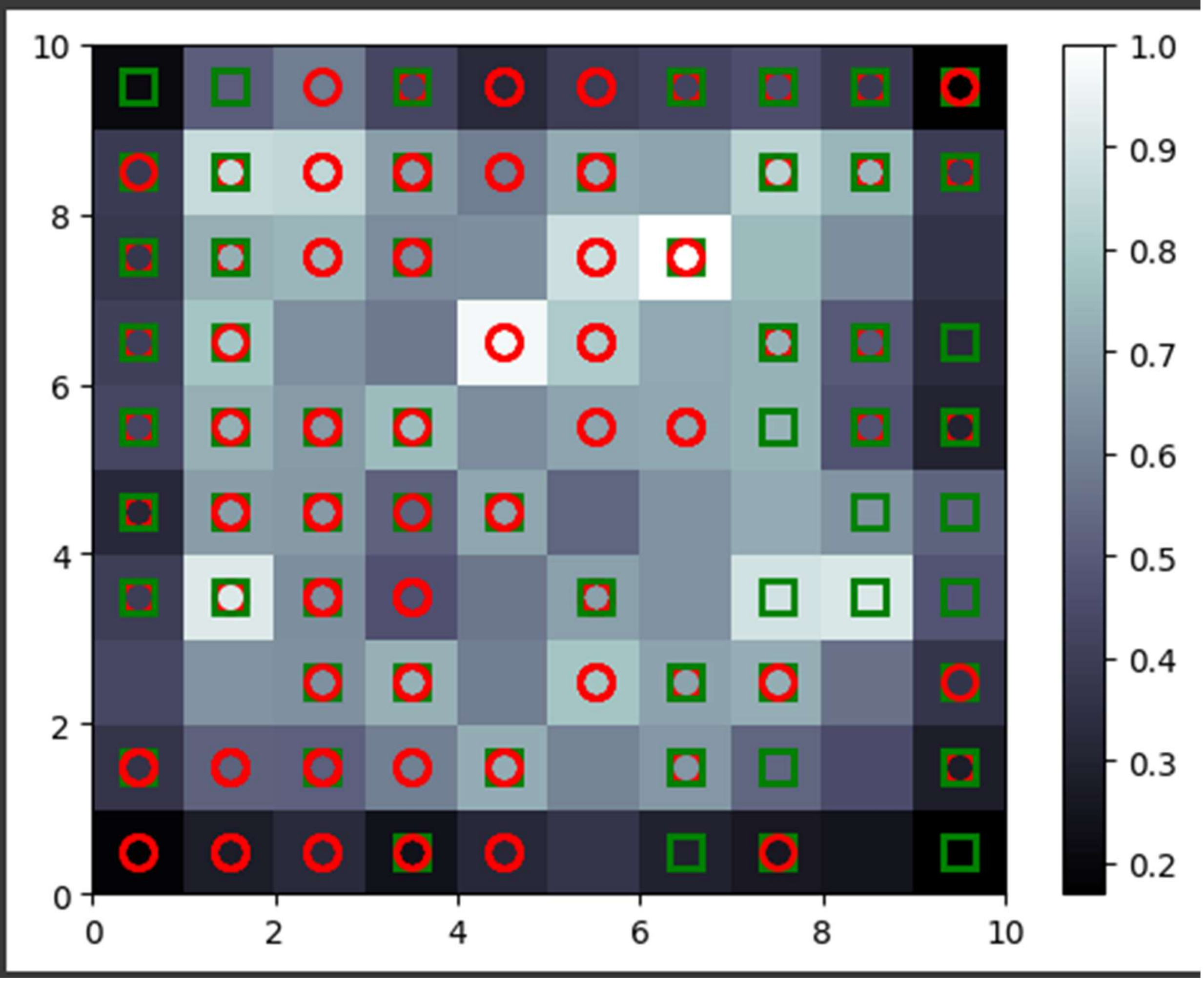
```python
from pylab import plot, axis, show, pcolor, colorbar, bone
bone()
pcolor(som.distance_map().T)
colorbar()
show()
```



_

```python
bone()
pcolor(som.distance_map().T)
colorbar()
markers = ['o', 's']
colors = ['r', 'g']

for i, x in enumerate(X):
    w = som.winner(x)
    plot(w[0] + 0.5,
         w[1] + 0.5,
         markers[y[i]],
         markeredgecolor = colors[y[i]],
         markerfacecolor = 'None',
         markersize = 10,
         markeredgewidth = 2)
show()
```



–

```
[ ] whitebox=[]
    for i in range(10):
      for j in range(10):
        if win[i][j]==1:
          whitebox.append((i,j))
    print(whitebox)
```

```
[(4, 1)]
```

```
mappings = som.win_map(X)
mappings
mappings.keys()
len(mappings.keys())
mappings[(9,8)]
frauds = np.concatenate((mappings[(0,9)], mappings[(8,9)]), axis = 0)
frauds
frauds1 = sc.inverse_transform(frauds)
pd.DataFrame(frauds1)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.0	20.75	10.335	2.0	13.0	8.0	0.335	1.0	1.0	1.0	1.0	2.0	80.0
1	0.0	31.25	3.750	2.0	13.0	8.0	0.625	1.0	1.0	9.0	1.0	2.0	181.0
2	0.0	22.83	2.290	2.0	11.0	8.0	2.290	1.0	1.0	7.0	1.0	2.0	140.0
3	0.0	23.00	11.750	2.0	14.0	8.0	0.500	1.0	1.0	2.0	1.0	2.0	300.0
4	0.0	46.67	0.460	2.0	13.0	8.0	0.415	1.0	1.0	11.0	1.0	2.0	440.0
5	0.0	41.00	2.040	1.0	11.0	8.0	0.125	1.0	1.0	23.0	1.0	2.0	455.0
6	1.0	23.17	0.000	2.0	13.0	4.0	0.085	1.0	0.0	0.0	0.0	2.0	0.0
7	1.0	26.75	1.125	2.0	14.0	8.0	1.250	1.0	0.0	0.0	0.0	2.0	0.0
8	1.0	23.42	0.585	2.0	8.0	8.0	0.085	1.0	0.0	0.0	0.0	2.0	180.0
9	1.0	22.67	10.500	2.0	11.0	8.0	1.335	1.0	0.0	0.0	0.0	2.0	100.0
10	1.0	41.33	0.000	2.0	8.0	5.0	15.000	1.0	0.0	0.0	0.0	2.0	0.0
11	1.0	22.08	11.000	2.0	13.0	4.0	0.665	1.0	0.0	0.0	0.0	2.0	100.0
12	1.0	41.75	0.960	2.0	14.0	4.0	2.500	1.0	0.0	0.0	0.0	2.0	510.0
13	1.0	29.92	1.835	2.0	8.0	8.0	4.335	1.0	0.0	0.0	0.0	2.0	260.0
14	1.0	51.42	0.040	2.0	14.0	8.0	0.040	1.0	0.0	0.0	0.0	2.0	0.0
15	1.0	25.75	0.750	2.0	8.0	5.0	0.250	1.0	0.0	0.0	0.0	2.0	349.0
16	1.0	23.33	1.500	2.0	8.0	8.0	1.415	1.0	0.0	0.0	0.0	2.0	422.0