# Self Organizing Map

## Install MiniSom Package

```
!pip install MiniSom

     Requirement already satisfied: MiniSom in /usr/local/lib/python3.6/dist-packages (2.2.7)
```

## Importing the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

## Importing the dataset

```
dataset = pd.read_csv('Credit_Card_Applications.csv')
# Austrailian Credit Approval Data set
# all attributes names have been put as meaningless symbols
# so we have to use SOM to extract features
# to get insights and segment customers

# the frauds would be the outliers in dataset, therefore they will be the
# outlier neurons that will be distant from other neurons from its neighborhood

x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1]
```

## Feature Scaling

```
from sklearn.preprocessing import MinMaxScaler
# NNs not effected by the distribution of the data
# so normal scaling is used
sc = MinMaxScaler(feature_range = (0, 1))
x = sc.fit_transform(x)
```

## Training the SOM

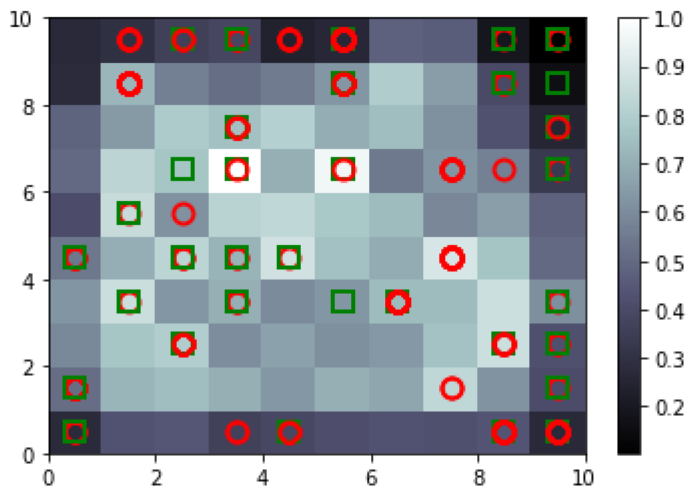
```
from minisom import MiniSom
som = MiniSom(x = 10, y = 10, input_len=15, sigma=1.0, learning_rate= 0.5)
# initialize the weights
som.random_weights_init(x)
som.train_random(data=x, num_iteration=100) #100 epochs
```

## Visualizing the results

```
from pylab import bone, pcolor, colorbar, plot, show
bone()
# display mean distances for all the units by color
# (further they are, the brigther they get)
# takes the transpose of the matrix
pcolor(som.distance_map().T)
colorbar()
# therefore the custsomers that are close to the brightest
# nodes are most likely to commit fraud

# highlight the most suspicious customers who got approval
markers = ['o', 's']
colors = ['r', 'g']
```

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



## Finding the frauds

```
# get list of all co-ordinate for winning nodes
mappings = som.win_map(x)
# coordinates of outliar nodes (3,6) and (5,6)
frauds = np.concatenate( (mappings[(3,6)], mappings[(5,6)]))
frauds = sc.inverse_transform(frauds)
```

## Printing the Fraud Client IDs

```
print(frauds[:, 0])
```

```
[15699963. 15667934. 15789611. 15668679. 15738487. 15773421. 15682686.
 15781875. 15809837. 15636521. 15761554. 15707602. 15811690. 15815095.
 15720725. 15672912. 15694677. 15759387. 15712483. 15698522.]
```

## Using ANN to predict fraud

```
# matrix of features
customers = dataset.iloc[:, 1:].values

# from the SOM we know the customers that potentially commited fraud,
# we use that to create our depedent variable
is_fraud = [0]*len(customers)
for i in range(len(customers)):
  if (dataset.iloc[i, 0] in frauds):
    is_fraud[i] = 1
```

```
# scale the customers
sc_2 = MinMaxScaler(feature_range = (0, 1))
customers = sc_2.fit_transform(customers)
```

```
is_fraud = np.array(is_fraud)
```

## Build the ANN

```
import tensorflow as tf

#initialize ann
ann = tf.keras.models.Sequential()
```

```
# this dataset is very small so we'll create a simple ann

# Hidden + output layer
ann.add(tf.keras.layers.Dense(units=2, activation='relu'))

# add output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

#compile the model
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

## Train the ANN

```
# tried, 3,4,10 all seem to coverge to 96% very quickly
ann.fit(customers, is_fraud, batch_size=1, epochs=2)
```

```
     Epoch 1/2
     690/690 [==============================] - 1s 839us/step - loss: 0.3189 - accuracy: 0.9790
     Epoch 2/2
     690/690 [==============================] - 1s 891us/step - loss: 0.1239 - accuracy: 0.9731
     <tensorflow.python.keras.callbacks.History at 0x7fd71d89f828>
```

```
y_pred = ann.predict(customers)
# attach the customer id to prediction
y_pred = np.concatenate((dataset.iloc[:, 0:1].values, y_pred), axis=1)
# sort the customers by most likely to commit fraud
y_pred = y_pred[y_pred[:, 1].argsort()[::-1]]
print(y_pred)
```

```
     [[1.56232100e+07 1.77971184e-01]
      [1.56114090e+07 1.56050056e-01]
      [1.56963610e+07 1.43202126e-01]
      ...
      [1.57238270e+07 3.68654728e-03]
```

```
 [1.56712930e+07 2.70923972e-03]
 [1.55805790e+07 1.40452385e-03]]
```