Visualization Clusters in Data Using SOM

Nimesha Asintha Muthunayake
CS5613 - Neural Networks
209359G

Question 1

U-Matrix Visualization

U-Matrix gives insights into the local distance structures of the data set U-Matrix can be used for clustering and it has advantage of a nonlinear disentanglement of complex cluster structures. The distance between adjacent neurons can be calculate ed , using their trained vector. If the input dimension was 4, then each neuron in the trained map also corresponds to a 4-dimensional vector. Let's say we have a 3x3 hexagonal map. The U-matrix will be a 5x5 matrix with interpolated elements for each connection between two neurons like this.

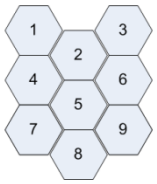


Figure 1: 3x3 hexagonal

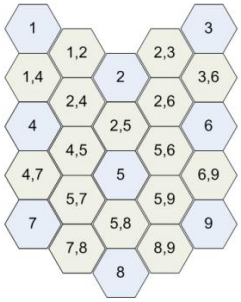


Figure 2: 5x5 hexagonal

The {x,y} elements are the distance between neuron x and y, and the values in {x} elements are the mean of the surrounding values . Then light gray color can be assigned to the largest of these values and a dark gray to the smallest and the other values to corresponding shades of gray. Basically, we can use these colors to paint the cells of the U-matrix and have a visualized representation of the distances between neurons.

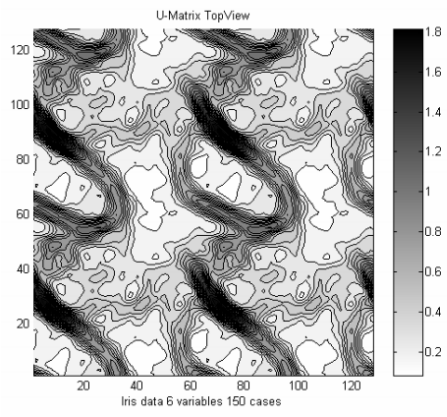


Figure 3: U-Matrix Top

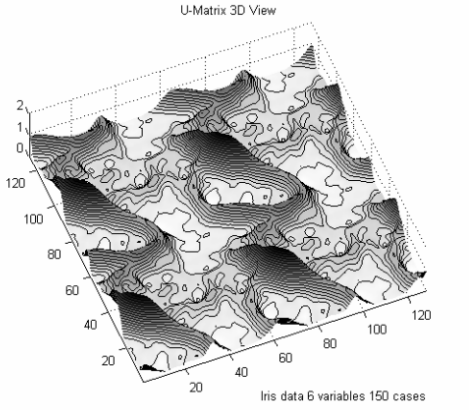


Figure 4: U-Matrix 3D View

Vector Field Visualization

Gradient field visualization is displayed as a vector field on top of the map lattice and the method turns out to be most useful for SOMs with high numbers of map units. One of the concepts that is required is the neighborhood kernel that is used by the SOM training algorithm. The commonly used Gaussian kernel has the well-known form of the Gaussian Bell-Shaped Curve, formally. The Self Organizing Map is a valuable tool in exploratory data analysis. Also, SOM is a popular unsupervised neural network algorithm that has been used in a wide range of industrial applications. The projection of a SOM can be visualized in numerous ways in order to reveal the characteristics of the underlying input data. In this paper, authors propose a novel visualization method based upon vector field plotting.

It is based on the neighborhood kernel function and on aggregation of distances in the proximity of each codebook vector. It requires a parameter ¾ that determines the smoothness and the level of detail of the visualization. There are two choices for depicting it, either as gradient field where arrows point towards the closest cluster center, or as border visualization that indicates how grave a transition is between neighboring regions. Their experiments have shown that this method is especially useful for maps with high numbers of units, and that the neighborhood radius has a major impact on the outcome.

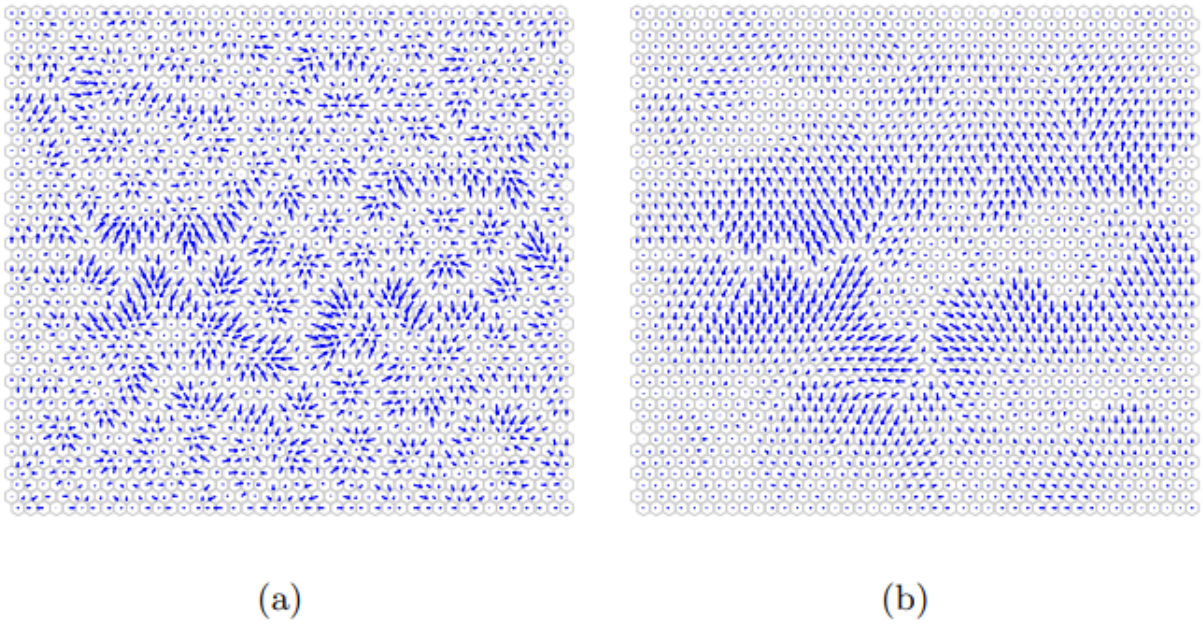
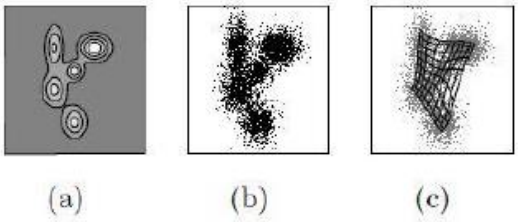


Figure 5: 44 × 44 SOM: Vector Field with (a) sigma = 2, (b) sigma = 7, (c) sigma = 15, (d) component plane “stimulation costs”

Smoothed Data Histogram

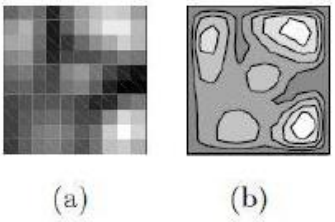
The SOM consists of units which are usually ordered on a rectangular 2-dimensional grid which is referred to as map. A model vector in the high-dimensional data space is assigned to each of the units. During the training process the model vectors are fitted to the data in such a way that the distances between the data items and the corresponding closest model vectors are minimized under the constraint that model vectors which belong to units close to each other on the map, are also close to each other in the data space.

The bin centers in the data space are defined by the model vectors and the varying bin widths are defined through the distances between the model vectors. The membership degree of a data item to a specific bin is governed by the smoothing parameter *s* and calculated based on the rank of the distances between the data item and all bin centers. In particular, the membership degree is *s=cs* to the closest bin, (*s¡*1)*=cs* to the second, (*s¡*2)*=cs* to the third, and so forth. The membership to all but the closest *s* bins is 0. The constant *cs* = P*s¡*1 *i*=0 *s ¡ i* ensures that the total membership of each data item adds up to 1.



(a) (b) (c)

Fig. 1. The 2-dimensional data space. (a) The probability distribution from which (b) the sample was drawn and (c) the model vectors of the SOM.



(a) (b)

Fig. 2. The SDH ($s=8$) visualized using (a) gray shadings and (b) contours.

(a) $s=1$ (b) $s=3$ (c) $s=5$ (d) $s=15$ (e) $s=20$ (f) $s=50$

Fig. 3. Different values for the smoothing parameters s and their effects on the SDH.

Question 2 – Implement a SOM

GitHub URL - https://github.com/nimeshacs/Visualization-Clusters-in-Data-Using-SOM

U-Matrix approach which was proposed by Alfred Ultsch was selected to implement a SOM for Iris dataset.

Data file loading to the program and process

```
# Load the Iris data in to program
data_file = "/content/drive/My Drive/Nimesha/iris_data_012.txt"
data_x = np.loadtxt(data_file, delimiter=",", usecols=range(0,4),dtype=np.float64)
data_y = np.loadtxt(data_file, delimiter=",", usecols=[4],dtype=np.int)
```

Generated 30 x 30 SOM for IRIS data

```
map = np.random.random_sample(size=(Rows,Cols,Dim))
for s in range(StepsMax):
if s % (StepsMax/10) == 0: print("step = ", str(s))
pct_left = 1.0 - ((s * 1.0) / StepsMax)
curr_range = (int)(pct_left * RangeMax)
curr_rate = pct_left * LearnMax
t = np.random.randint(len(data_x))
(bmu_row, bmu_col) = closest_node(data_x, t, map, Rows, Cols)
for i in range(Rows):
for j in range(Cols):
if manhattan_dist(bmu_row, bmu_col, i, j) < curr_range:
map[i][j] = map[i][j] + curr_rate * \
(data_x[t] - map[i][j])
```

Constructing U-Matrix from SOM

```
u_matrix = np.zeros(shape=(Rows,Cols), dtype=np.float64)
for i in range(Rows):
  for j in range(Cols):
    v = map[i][j]  # a vector
    sum_dists = 0.0; ct = 0

    if i-1 >= 0:    # above
      sum_dists += euc_dist(v, map[i-1][j]); ct += 1
    if i+1 <= Rows-1:   # below
      sum_dists += euc_dist(v, map[i+1][j]); ct += 1
    if j-1 >= 0:   # left
      sum_dists += euc_dist(v, map[i][j-1]); ct += 1
    if j+1 <= Cols-1:   # right
      sum_dists += euc_dist(v, map[i][j+1]); ct += 1

    u_matrix[i][j] = sum_dists / ct
```

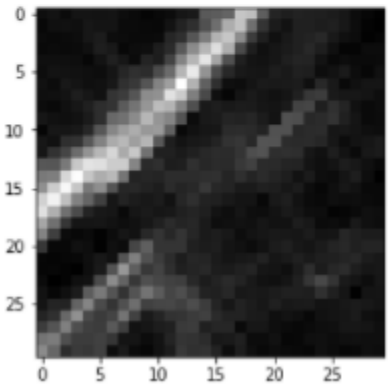
```
  print("U-Matrix is generated \n")


  # display U-Matrix
  plt.imshow(u_matrix, cmap='gray')
  plt.show()
  # display reduced data

if __name__=="__main__":
  main()
```

```
Constructing a 30x30 SOM for data set
step =  0
step =  500
step =  1000
step =  1500
step =  2000
step =  2500
step =  3000
step =  3500
step =  4000
step =  4500
SOM is gnerated

Constructing U-Matrix from SOM
U-Matrix is generated
```



Complete python code for U-Matrix approach

```
"""
Fomatted @09-09-2020
@Nimesha Muthunayake
Visualization Clusters in Data Using SOM
Nimesha Asintha Muthunayake
CS5613 - Neural Networks
Assignment 3
"""
import numpy as np
import matplotlib.pyplot as plt

def closest_node(data, t, map, m_rows, m_cols):
      result = (0,0)
      small_dist = 1.0e20
      for i in range(m_rows):
        for j in range(m_cols):
          ed = euc_dist(map[i][j], data[t])
          if ed < small_dist:
            small_dist = ed
            result = (i, j)
      return result

def euc_dist(v1, v2):
  return np.linalg.norm(v1 - v2)

def manhattan_dist(r1, c1, r2, c2):
  return np.abs(r1-r2) + np.abs(c1-c2)

def most_common(lst, n):
  # lst is a list of values 0 . . n
  if len(lst) == 0: return -1
  counts = np.zeros(shape=n, dtype=np.int)
  for i in range(len(lst)):
    counts[lst[i]] += 1
  return np.argmax(counts)

def main():
  # 0. get started
  np.random.seed(1)
  Dim = 4
  Rows = 30; Cols = 30
  RangeMax = Rows + Cols
  LearnMax = 0.5
  StepsMax = 5000

  # Load the Iris data in to program
  data_file = "/content/drive/My Drive/Nimesha/iris_data_012.txt"
  data_x = np.loadtxt(data_file, delimiter=",", usecols=range(0,4),dtype=np.float64)
  data_y = np.loadtxt(data_file, delimiter=",", usecols=[4],dtype=np.int)

  #Construct the SOM network  , Constructing a 30x30 SOM from the iris data
  print("Constructing a 30x30 SOM for data set")
  map = np.random.random_sample(size=(Rows,Cols,Dim))
  for s in range(StepsMax):
    if s % (StepsMax/10) == 0: print("step = ", str(s))
    pct_left = 1.0 - ((s * 1.0) / StepsMax)
    curr_range = (int)(pct_left * RangeMax)
    curr_rate = pct_left * LearnMax
```

```
59.
60.     t = np.random.randint(len(data_x))
61.     (bmu_row, bmu_col) = closest_node(data_x, t, map, Rows, Cols)
62.     for i in range(Rows):
63.       for j in range(Cols):
64.         if manhattan_dist(bmu_row, bmu_col, i, j) < curr_range:
65.           map[i][j] = map[i][j] + curr_rate * \
66. (data_x[t] - map[i][j])
67.   print("SOM is gnerated \n")
68.
69.   #U-Matrix Visualization
70.   print("Constructing U-Matrix from SOM")
71.   u_matrix = np.zeros(shape=(Rows,Cols), dtype=np.float64)
72.   for i in range(Rows):
73.     for j in range(Cols):
74.       v = map[i][j]  # a vector
75.       sum_dists = 0.0; ct = 0
76.
77.       if i-1 >= 0:    # above
78.         sum_dists += euc_dist(v, map[i-1][j]); ct += 1
79.       if i+1 <= Rows-1:   # below
80.         sum_dists += euc_dist(v, map[i+1][j]); ct += 1
81.       if j-1 >= 0:   # left
82.         sum_dists += euc_dist(v, map[i][j-1]); ct += 1
83.       if j+1 <= Cols-1:   # right
84.         sum_dists += euc_dist(v, map[i][j+1]); ct += 1
85.
86.       u_matrix[i][j] = sum_dists / ct
87.   print("U-Matrix is generated \n")
88.
89.   # display U-Matrix
90.   plt.imshow(u_matrix, cmap='gray')
91.   plt.show()
92.   # display reduced data
93.
94. if __name__=="__main__":
95.   main()
```

Reference

Exploiting the Structures of the U-Matrix (Jorn Lotsch , Alfred Ultsch)

A SOM-view of oilfield data: A novel vector field visualization for Self-Organizing Maps and its applications in the petroleum industry (Georg Polzlbauer, Andreas Rauber)

Smoothed Data Histograms for Cluster Visualization in Self-Organizing Maps by Elias Pampalk, Andreas Rauber, and Dieter Merkl2

- ---------- END ----------