Московский авиационный институт
(Национальный исследовательский университет)
Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 6

по курсу «Нейроинформатика».

Тема: «Сети Кохонена».

Студент: Вельтман Л.Я.
Группа: 80-407Б
Преподаватели: Тюменцев Ю.В.
Аносова Н.П.

Вариант: 7

Оценка:

Москва, 2020

Цель работы.

Целью работы является исследование свойств слоя Кохонена, карты Кохонена, а также сетей векторного квантования, обучаемых с учителем, алгоритмов обучения, а также применение сетей в задачах кластеризации и классификации.

Основные этапы работы.

1. Использовать слой Кохонена для выполнения кластеризации множества точек. Проверить качество разбиения.
2. Использовать карту Кохонена для выполнения кластеризации множества точек.
3. Использовать сеть векторного квантования, обучаемую с учителем, (LVQ-сеть) для классификации точек в случае, когда классы не являются линейно разделимыми.

Оборудование.

Операционная система: macOS Catalina version 10.15.5

Процессор: 2,3 GHz 2-ядерный процессор Intel Core i5

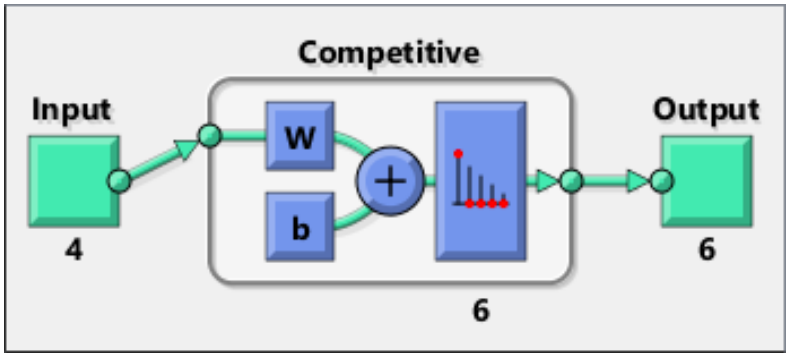
Оперативная память: 8 ГБ 2133 MHz LPDDR3

Программное обеспечение.

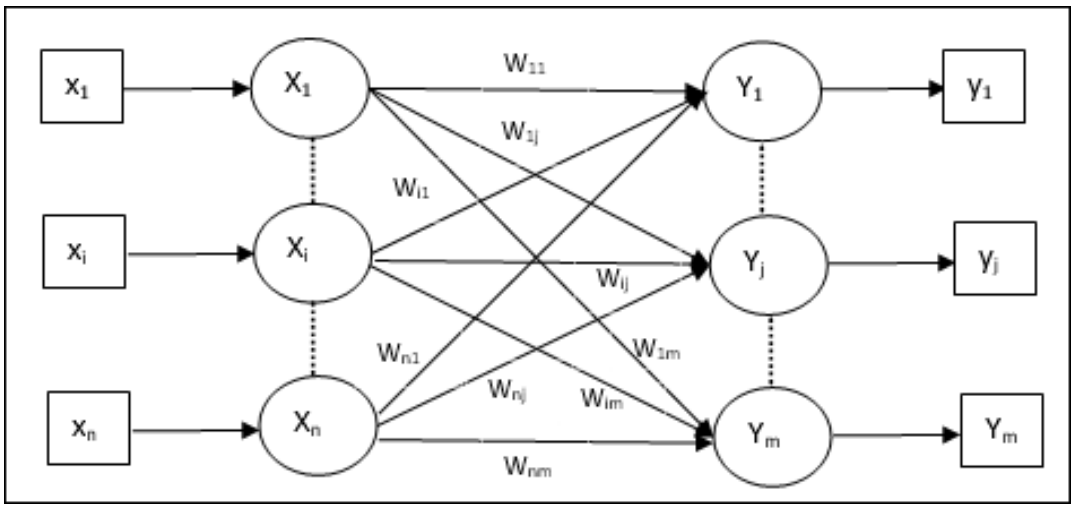
Работа выполнена на Python3 с применением библиотек numpy (для вычислений), pandas и matplotlib (для графиков) при помощи командной оболочки Jupyter Notebook.

Сценарий выполнения работы.

Сеть Кохонена



Сеть векторного квантования



Нейронные сети Кохонена — класс нейронных сетей, основным элементом которых является слой Кохонена. Слой Кохонена состоит из адаптивных линейных сумматоров («линейных формальных нейронов»). Как правило, выходные сигналы слоя Кохонена обрабатываются по правилу «Победитель получает всё»: наибольший сигнал превращается в единичный, остальные обращаются в ноль.

По способам настройки входных весов сумматоров и по решаемым задачам различают много разновидностей сетей Кохонена. Наиболее известные из них: сети векторного квантования сигналов, тесно связанные с простейшим базовым алгоритмом кластерного анализа (метод динамических ядер или К-средних); самоорганизующиеся карты Кохонена; сети векторного квантования, обучаемые с учителем .

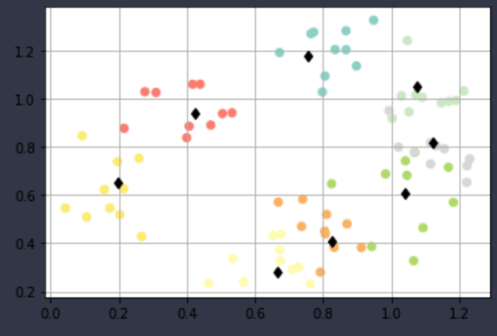
Входные данные и результаты

Задание 1

Сформируем множество точек, которые изначально сгруппированы в 8 кластеров

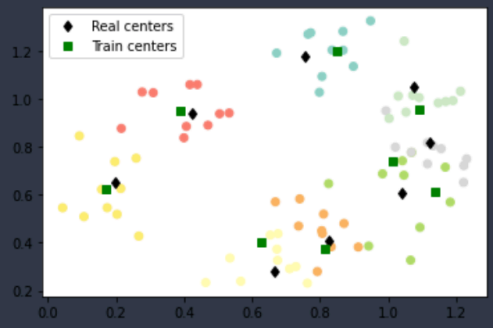
In [2]:
```
x, y, centers = make_blobs(n_samples=80, cluster_std=0.1, n_features=2, centers=8, center_box=(0,
```

In [3]:
```
plt.scatter(x[:, 0], x[:, 1], c=y, cmap='Set3')
plt.plot(centers[:,0], centers[:, 1], 'dk')
plt.grid(True)
```



In [7]:

```
plt.scatter(x[:, 0], x[:, 1], c=y, cmap='Set3')
plt.plot(centers[:,0], centers[:, 1], 'dk')
plt.plot(weightsAreCenter[:,0], weightsAreCenter[:,1], 'gs')
plt.legend(['Real centers', 'Train centers'])
plt.show()
```



Подаем их в сеть и на выходе получим номера кластеров

In [9]:

```
pred = competNet.sim(randPoints)
```

Таким образом пять точек принадлежат следующим классам

In [10]:

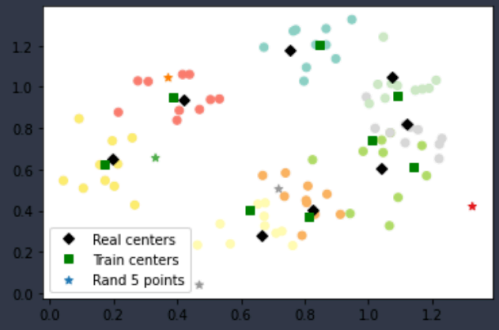
```
classPred = np.argmax(pred, axis=1)

classPred
```

```
array([3, 2, 6, 6, 4])
```

In [11]:

```
plt.scatter(x[:, 0], x[:, 1], c=y, cmap='Set3')
plt.plot(centers[:,0], centers[:, 1], 'Dk', label='Real centers')
plt.plot(weightsAreCenter[:,0], weightsAreCenter[:,1], 'gs', label='Train centers')
plt.scatter(randPoints[:, 0], randPoints[:, 1], c=classPred,
            cmap='Set1', marker='*', label='Rand 5 points')
plt.legend()
plt.show()
```



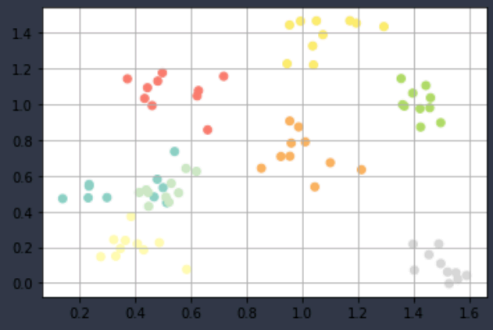
Задание 2

Задание 2

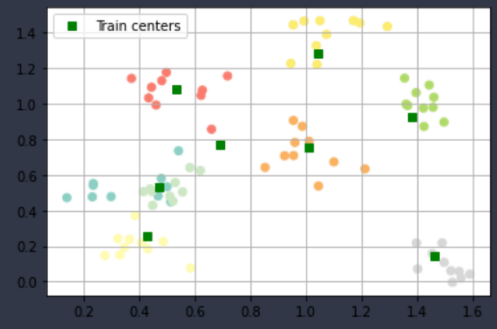
Нужно построить и обучить карту Кохонена размера 2x4 с гексагональной сеткой

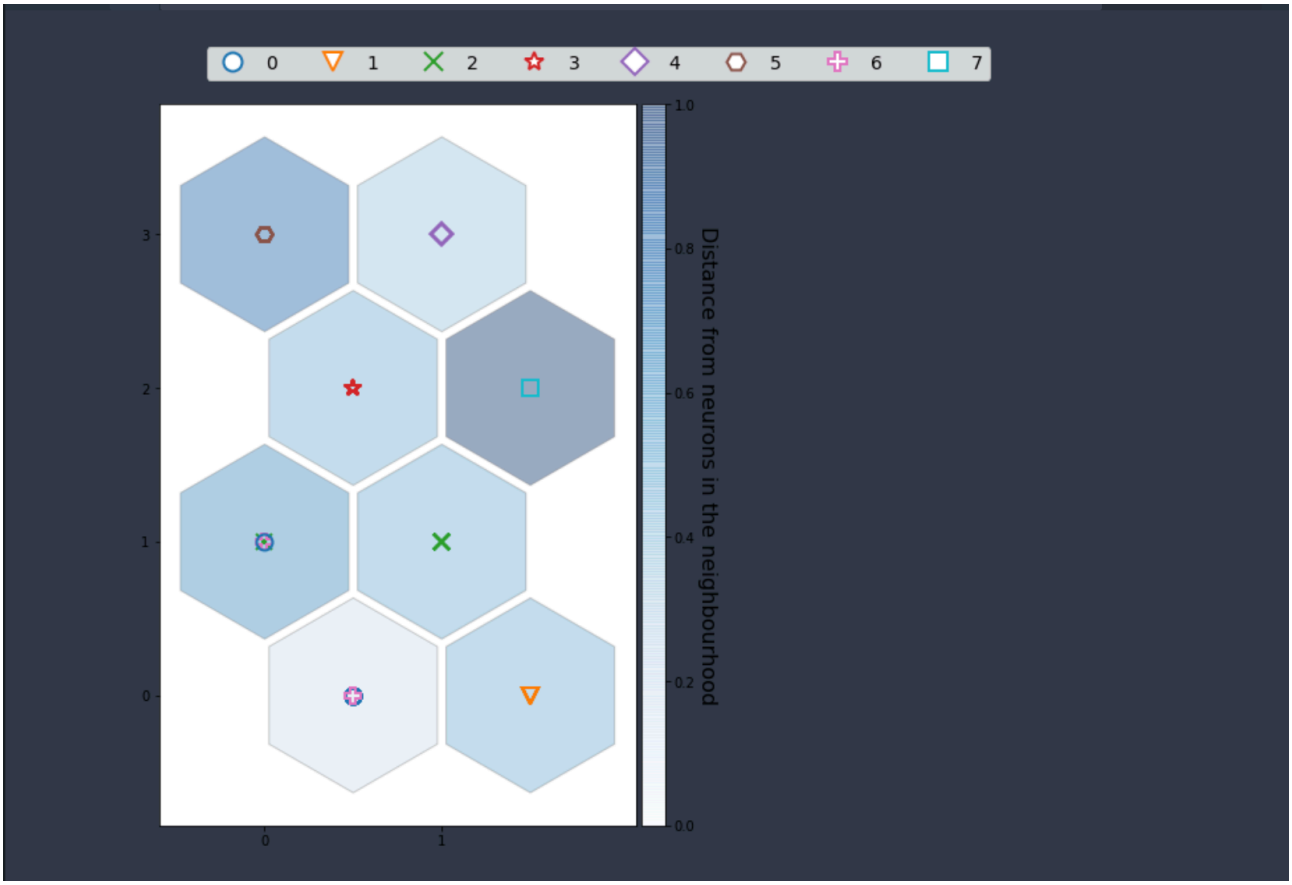
In [12]:
```
x2, y2 = make_blobs(n_samples=80, cluster_std=0.1, n_features=2,
                    centers=8, center_box=(0, 1.5), random_state=91)
```

In [13]:
```
plt.scatter(x2[:, 0], x2[:, 1], c=y2, cmap='Set3')
plt.grid(True)
```



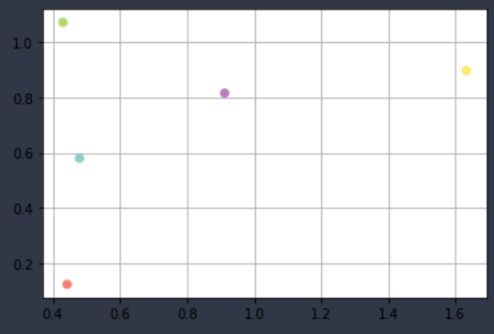
In [16]:
```
plt.scatter(x2[:, 0], x2[:, 1], c=y2, cmap='Set3')
plt.plot(weights[0][:,0], weights[0][:,1], 'gs', label='Train centers')
plt.plot(weights[1][:,0], weights[1][:,1], 'gs')
plt.legend()
plt.grid(True)
```





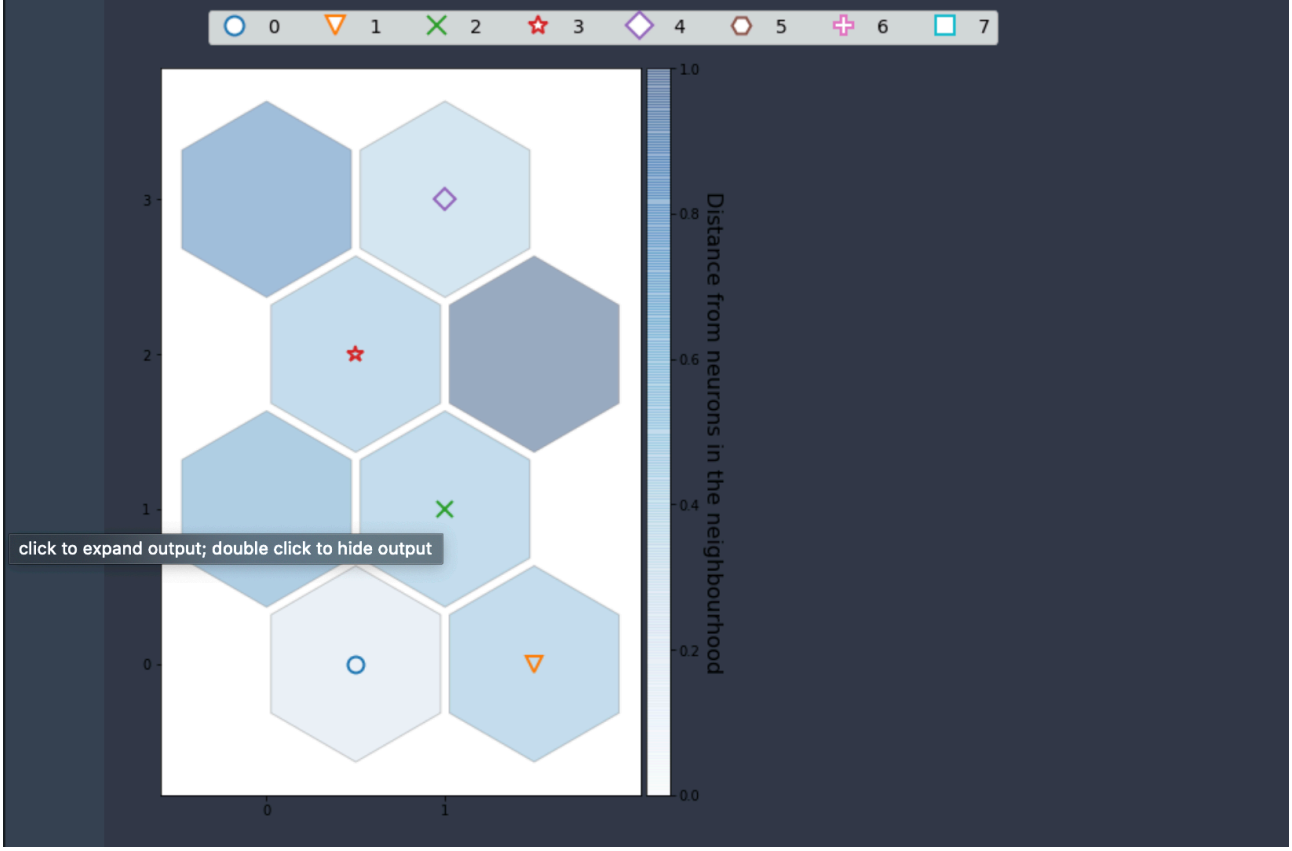
In [19]:

```
plt.scatter(randx[:, 0], randx[:, 1], c=randy, cmap='Set3')
plt.grid(True)
```



In [20]:

```
1 DrawHex(randx, randy, xx, yy, umatrix, weights)
```



Задание 3

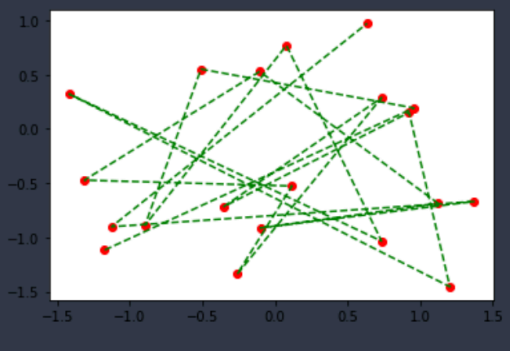
Задание 3

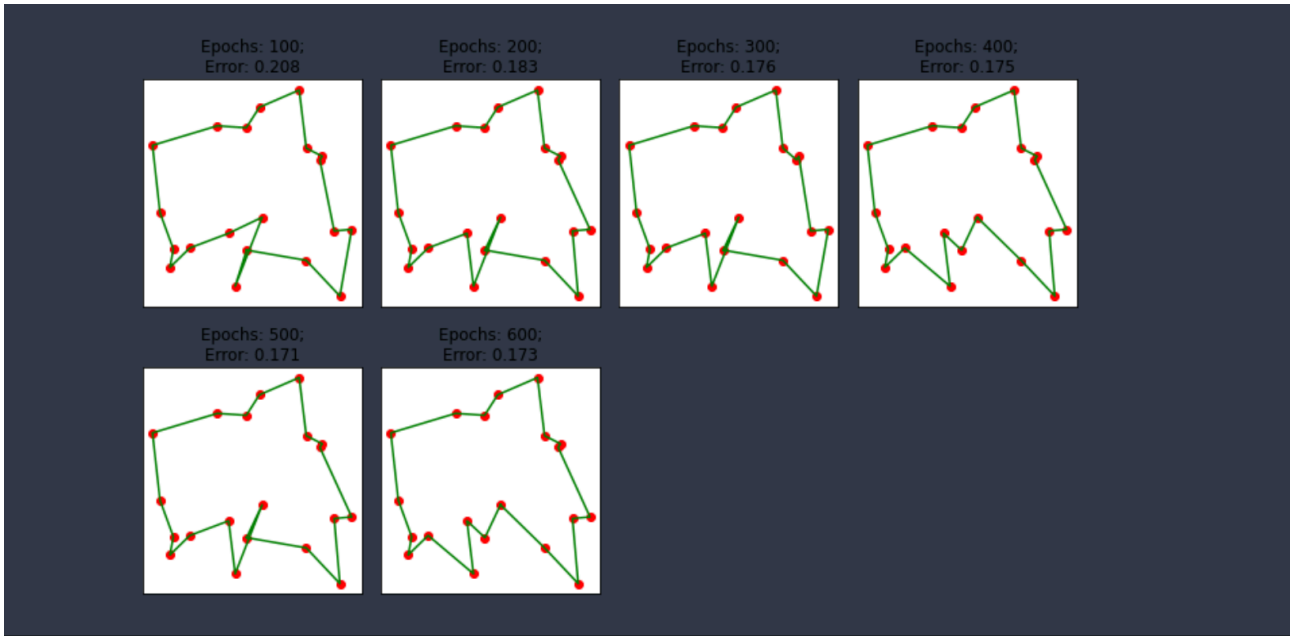
Построим и обучим карту Кохонена, которая будет находить одно из решений задачи коммивояжера.

Сгенерируем набор из 20 случайных точек из диапазона [-1.5, 1.5].

In [21]:

```
1 z = np.array([[np.random.uniform(-1.5, 1.5), np.random.uniform(-1.5, 1.5)] for _ in range(20)])
2 plt.plot(z[:, 0], z[:, 1], '--', c='green')
3 plt.scatter(z[:, 0], z[:, 1], c='red');
```





Задание 4

4 задание

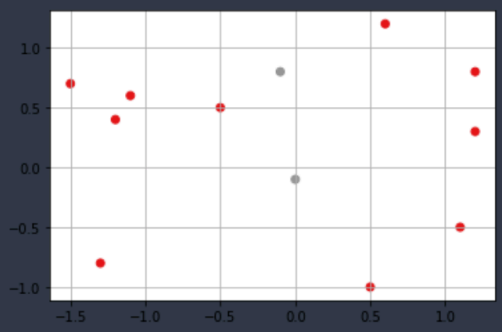
Для обучающей выборки построить LVQ-сеть, которая будет правильно соотносить точки к двум классам. Классы не являются линейно разделимыми.

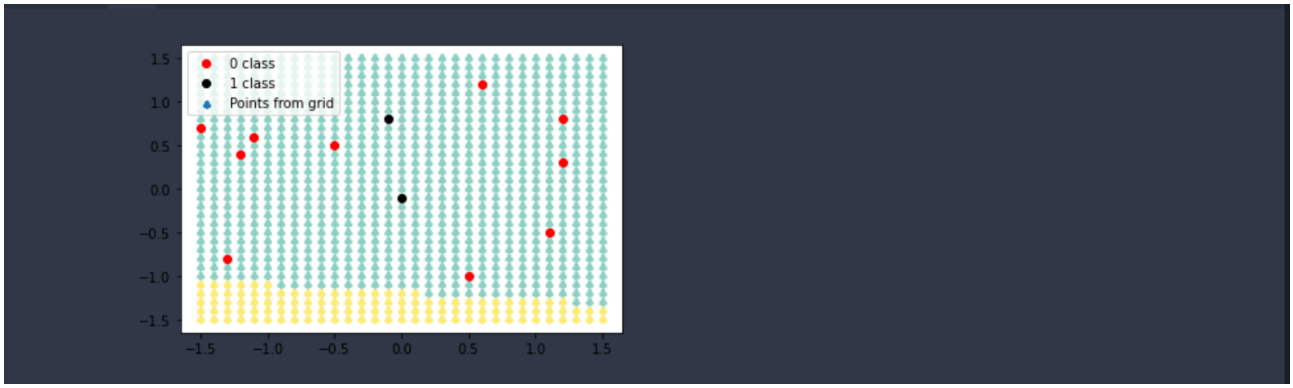
In [24]:

```
points = np.array([[1.2, -1.3, -0.5, -1.1, -1.2, -0.1, 0.6, 1.1, 0.5, -1.5, 0, 1.2],
                   [0.8, -0.8, 0.5, 0.6, 0.4, 0.8, 1.2, -0.5, -1, 0.7, -0.1, 0.3]])
target = np.array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0])
pointsT = points.T
```

In [25]:

```
plt.scatter(pointsT[:, 0], pointsT[:, 1], c=target, cmap='Set1')
plt.grid(True)
```





Код программы.

```
import numpy as np
from minisom import MiniSom
from neupy.algorithms import SOFM, LVQ
import neurolab as nl
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from matplotlib.patches import RegularPolygon, Ellipse
from mpl_toolkits.axes_grid1 import make_axes_locatable
from matplotlib import cm, colorbar
from matplotlib.lines import Line2D


# Сформируем множество точек, которые изначально сгруппированы в 8 кластеров

x, y, centers = make_blobs(n_samples=80, cluster_std=0.1, n_features=2, centers=8,
center_box=(0, 1.5), return_centers=True, random_state=177)

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='Set3')
plt.plot(centers[:,0], centers[:, 1], 'dk')
plt.grid(True)


# Построим и обучим слой Кохонена

clusters = 8
competNet = nl.net.newc([[0.0, 0.8],[0.0, 1.3]], clusters)
error = competNet.train(x, epochs=500, show=100)


# Весовые коэффициенты первого слоя, которые являются центрами кластеров.

weightsAreCenter = competNet.layers[0].np['w']
weightsAreCenter
```

```
plt.title('Classification Problem')
plt.plot(error)
plt.xlabel('Epoch number')
plt.ylabel('Error (default MAE)')

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='Set3')
plt.plot(centers[:,0], centers[:, 1], 'dk')
plt.plot(weightsAreCenter[:,0], weightsAreCenter[:,1], 'gs')
plt.legend(['Real centers', 'Train centers'])
plt.show()


# Проверим качество разбиения. Создадим случайным образом 5 точек

randPoints = np.array([[np.random.uniform(0, 1.5),
                        np.random.uniform(0, 1.5)] for _ in range(5)])


# Подаем их в сеть и на выходе получим номера кластеров

pred = competNet.sim(randPoints)


# Таким образом пять точек принадлежат следующим классам

classPred = np.argmax(pred, axis=1)

classPred

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='Set3')
plt.plot(centers[:,0], centers[:, 1], 'Dk', label='Real centers')
plt.plot(weightsAreCenter[:,0], weightsAreCenter[:,1], 'gs', label='Train centers')
plt.scatter(randPoints[:, 0], randPoints[:, 1], c=classPred,
            cmap='Set1', marker='*', label='Rand 5 points')
plt.legend()
plt.show()


# Задание 2
#
# Нужно построить и обучить карту Кохонена размера 2x4 с гексагональной сеткой


x2, y2 = make_blobs(n_samples=80, cluster_std=0.1, n_features=2,
                    centers=8, center_box=(0, 1.5), random_state=91)

plt.scatter(x2[:, 0], x2[:, 1], c=y2, cmap='Set3')
plt.grid(True)

epochs = 150
som = MiniSom(2, 4, x2.shape[1], sigma=0.66, learning_rate=0.8, activation_distance='euclidean',
              topology='hexagonal', neighborhood_function='gaussian', random_seed=10)
som.train(x2, epochs, verbose=True)

xx, yy = som.get_euclidean_coordinates()
umatrix = som.distance_map()
```

```
weights = som.get_weights()

plt.scatter(x2[:, 0], x2[:, 1], c=y2, cmap='Set3')
plt.plot(weights[0][:,0], weights[0][:,1], 'gs', label='Train centers')
plt.plot(weights[1][:,0], weights[1][:,1], 'gs')
plt.legend()
plt.grid(True)

def DrawHex(x2, y2, xx, yy, umatrix, weights):
    f = plt.figure(figsize=(10,10))
    ax = f.add_subplot(111)

    ax.set_aspect('equal')

    # iteratively add hexagons
    for i in range(weights.shape[0]):
        for j in range(weights.shape[1]):
            wy = yy[(i, j)] * 2 / np.sqrt(3) * 3 / 4
            hex = RegularPolygon((xx[(i, j)], wy),
                                 numVertices=6,
                                 radius=0.95 / np.sqrt(3),
                                 facecolor=cm.Blues(umatrix[i, j]),
                                 alpha=0.4,
                                 edgecolor='gray')
            ax.add_patch(hex)

    markers = ['o', 'v', 'x', '*', 'D', 'H', 'P', 's']
    colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink',
'tab:cyan']
    for cnt, x in enumerate(x2):
        # getting the winner
        w = som.winner(x)
        # place a marker on the winning position for the sample xx
        wx, wy = som.convert_map_to_euclidean(w)
        wy = wy * 2 / np.sqrt(3) * 3 / 4
        plt.plot(wx, wy,
                 markers[y2[cnt]],
                 markerfacecolor='None',
                 markeredgecolor=colors[y2[cnt]],
                 markersize=12,
                 markeredgewidth=2)

    xrange = np.arange(weights.shape[0])
    yrange = np.arange(weights.shape[1])
    plt.xticks(xrange-0.5, xrange)
    plt.yticks(yrange * 2 / np.sqrt(3) * 3 / 4, yrange)

    divider = make_axes_locatable(plt.gca())
    ax_cb = divider.new_horizontal(size="5%", pad=0.05)
    cb1 = colorbar.ColorbarBase(ax_cb, cmap=cm.Blues,
                                orientation='vertical', alpha=.4)
    cb1.ax.get_yaxis().labelpad = 16
    cb1.ax.set_ylabel('Distance from neurons in the neighbourhood',
                      rotation=270, fontsize=16)
    plt.gcf().add_axes(ax_cb)
```

```
    legend_elements = [Line2D([0], [0], marker='o', color='tab:blue', label='0',
                              markerfacecolor='w', markersize=14, linestyle='None', markeredgewidth=2),
                       Line2D([0], [0], marker='v', color='tab:orange', label='1',
                              markerfacecolor='w', markersize=14, linestyle='None', markeredgewidth=2),
                       Line2D([0], [0], marker='x', color='tab:green', label='2',
                              markerfacecolor='w', markersize=14, linestyle='None', markeredgewidth=2),
                       Line2D([0], [0], marker='*', color='tab:red', label='3',
                              markerfacecolor='w', markersize=14, linestyle='None', markeredgewidth=2),
                       Line2D([0], [0], marker='D', color='tab:purple', label='4',
                              markerfacecolor='w', markersize=14, linestyle='None', markeredgewidth=2),
                       Line2D([0], [0], marker='H', color='tab:brown', label='5',
                              markerfacecolor='w', markersize=14, linestyle='None', markeredgewidth=2),
                       Line2D([0], [0], marker='P', color='tab:pink', label='6',
                              markerfacecolor='w', markersize=14, linestyle='None', markeredgewidth=2),
                       Line2D([0], [0], marker='s', color='tab:cyan', label='7',
                              markerfacecolor='w', markersize=14, linestyle='None', markeredgewidth=2)]
    ax.legend(handles=legend_elements, bbox_to_anchor=(0.1, 1.08), loc='upper left',
              borderaxespad=0., ncol=8, fontsize=14)

    plt.show()

DrawHex(x2, y2, xx, yy, umatrix, weights)
randx, randy = make_blobs(n_samples=5, cluster_std=0.1, n_features=2,
                          centers=8, center_box=(0, 1.5), random_state=91)

plt.scatter(randx[:, 0], randx[:, 1], c=randy, cmap='Set3')
plt.grid(True)

DrawHex(randx, randy, xx, yy, umatrix, weights)


# Задание 3
#
# Построим и обучим карту Кохонена, которая будет находить одно из решений задачи коммивояжера.
#
# Сгенерируем набор из 20 случайных точек из диапазона [-1.5, 1.5].

z = np.array([[np.random.uniform(-1.5, 1.5), np.random.uniform(-1.5, 1.5)] for _ in range(20)])
plt.plot(z[:, 0], z[:, 1], '--', c='green')
plt.scatter(z[:, 0], z[:, 1], c='red');

np.random.RandomState(10)
neurons = 80

som = MiniSom(1, neurons, z.shape[1], sigma=8, learning_rate=0.4,
              neighborhood_function='gaussian', random_seed=0)
som.random_weights_init(z)

plt.figure(figsize=(10, 9))
for i, iterations in enumerate(range(100, 601, 100)):
    som.train(z, iterations, verbose=False, random_order=False)
    plt.subplot(3, 4, i+1)
    plt.scatter(z[:, 0], z[:, 1], c='red')
    visit_order = np.argsort([som.winner(p)[1] for p in z])
    visit_order = np.concatenate((visit_order, [visit_order[0]]))
```

```
    plt.plot(z[visit_order][:,0], z[visit_order][:,1], c='green')
    plt.title("Epochs: {i};\nError: {e:.3f}".format(i=iterations,
                                          e=som.quantization_error(z)))
    plt.xticks([])
    plt.yticks([])
plt.tight_layout()
plt.show()


# 4 задание
#
# Для обучающей выборки построить LVQ-сеть, которая будет правильно соотносить точки
к двум классам. Классы не являются линейно разделимыми.

points = np.array([[1.2, -1.3, -0.5, -1.1, -1.2, -0.1, 0.6, 1.1, 0.5, -1.5, 0, 1.2],
                   [0.8, -0.8, 0.5, 0.6, 0.4, 0.8, 1.2, -0.5, -1, 0.7, -0.1, 0.3]])
target = np.array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0])
pointsT = points.T

plt.scatter(pointsT[:, 0], pointsT[:, 1], c=target, cmap='Set1')
plt.grid(True)


# Строим LVQ-сеть и обучаем на 300 эпохах.

lvqnet = LVQ(n_inputs=2, n_classes=2, step=0.1)
lvqnet.train(pointsT, target, epochs=300)


# Проверяем качество обучения.
#
# Для этого классифицируем точки области [-1.5;1.5]x[-1.5;1.5] с шагом 0.1
xx, yy = np.meshgrid(np.arange(-1.5, 1.51, 0.1), np.arange(-1.5, 1.51, 0.1))
xx.shape = xx.size, 1
yy.shape = yy.size, 1
sample = np.concatenate((xx, yy), axis=1)


# Рассчитаем выходы сети

pred = lvqnet.predict(sample)

plt.scatter(sample[:, 0], sample[:, 1], c=pred,
            cmap='Set3', marker=r'$\clubsuit$', label='Points from grid')
plt.plot([pointsT[i][0] for i in range(12) if target[i] == 0],
         [pointsT[i][1] for i in range(12) if target[i] == 0], 'ro', label='0 class')
plt.plot([pointsT[i][0] for i in range(12) if target[i] == 1],
         [pointsT[i][1] for i in range(12) if target[i] == 1], 'ko', label='1 class');
plt.legend()
plt.show()
```

Выводы:

Полученную самоорганизующуюся карту Кохонена можно использовать как средство визуализации при анализе данных. В результате обучения карта Кохонена классифицирует входные примеры на кластеры (группы схожих примеров) и визуально отображает многомерные входные данные на плоскости нейронов. Уникальность метода самоорганизующихся карт состоит в преобразовании n-мерного пространства в двухмерное. Применение двухмерных сеток связано с тем, что существует проблема отображения пространственных структур большей размерности. Имея такое представление данных, можно визуально определить наличие или отсутствие взаимосвязи во входных данных.