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

Кафедра вычислительной математики и программирования

Лабораторная работа №6
по курсу «Нейроинформатика»

Сети Кохонена

Выполнил: С.А. Красоткин
Группа: 8О-408Б
Вариант: 17
Преподаватели: Тюменцев Ю.В.
Рожлейс И. А.
Оценка:

Москва, 2022

Лабораторная №6 "Сети Кохонена"

Вариант № 17

Красоткин Семён (М80-408Б-19)

Цель работы

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.

```
import matplotlib.pyplot as plt
import io
import imageio
import numpy as np

from tqdm import tqdm
from IPython.display import Image
```

Класс слоя Кохоннена

```
class SOM():
  def __init__(self, in_features: int, width, height):
    self.nodes = np.random.randn(width * height, in_features)
    self.indices = np.array([[x,y] for x in range(height) for y in
range(width)])

  def update(self, input, radius, _lr):
    BMU_id = np.argmin(np.linalg.norm(self.nodes - input, axis = 1))
    distances = np.linalg.norm(self.indices - self.indices[BMU_id],
axis = 1)
    for distance, node in zip(distances, self.nodes):
      if distance < radius:
        influence = np.exp(-distance / (2 * radius))
        node += _lr * influence * (input - node)
```

Задаю размер карты Кохонена, начальный радиус и learning rate.

```
image_height = 2**6
image_width = 2**6
kox_radius = max(image_width, image_height) // 2
lr = 1e-1
global_epochs = 750
```

Функция нормализации.

```
def normalize(data): return (data - np.min(data)) / (np.max(data) -
np.min(data))
```

Функция визуализации.

```
def visualise(data, epoch, images):
  image_to_show = normalize(data).reshape((image_height, image_width,
3))
  plt.imshow(image_to_show)

  plt.savefig(f'./epoch_{epoch}.png', transparent = False,
facecolor='white')
  images.append(imageio.imread(f'./epoch_{epoch}.png'))

  plt.close()
```

Функция обучения

```
def fit(model, train_loader, radius, _lr, epochs):
  _radius = []
  _lr_ = []
  images = []

  start_radius = radius
  start_lr = _lr

  with tqdm(desc="learning", total=epochs) as pbar_outer:
    for epoch in range(epochs):
      for inputs in train_loader:
        model.update(inputs, radius, _lr)

      visualise(model.nodes, epoch + 1, images)

      radius = start_radius * np.exp(-epoch / (epochs /
np.log(start_radius)))
      _radius.append(radius)

      _lr = start_lr * np.exp(-epoch / epochs)
      _lr_.append(_lr)

      pbar_outer.update()

  imageio.mimsave('som.gif', images)
  return _radius, _lr_

def get_train_data(X, Y, lables):
  return [[x, y, 1] for x, y, l in zip (X, Y, lables)]

x = [-1.3, 0, 0.6, 0, 1, -0.6, 0.7, -1.2, 0, 0.6, -0.7, 1.2]
y = [-0.6, -1.4, 0.1, 0.9, 0.8, -0.2, -1.2, -0.7, 1.4, -0.6, 1, 0.4]
lables = [-1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, 1]

train_data = get_train_data(x, y, lables)
```

Даю виктор фич и размеры карты.

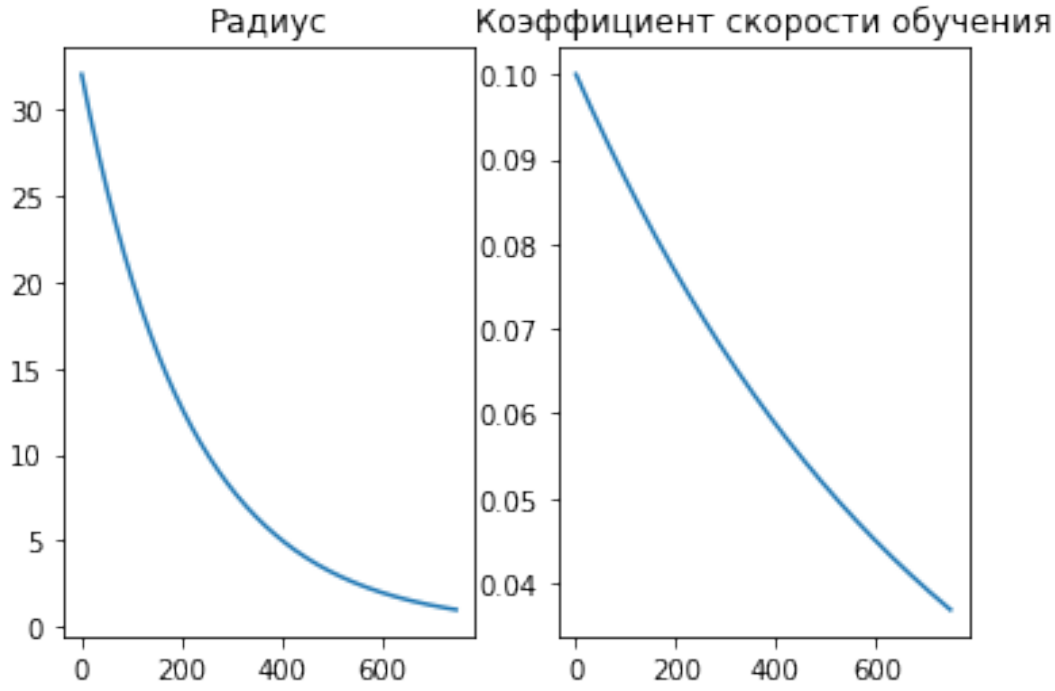
```
_SOM = SOM(3, image_width, image_height)
```

Обучение модели:

```
radiusus, lrs = fit(_SOM, train_data, kox_radius, lr, global_epochs)
```

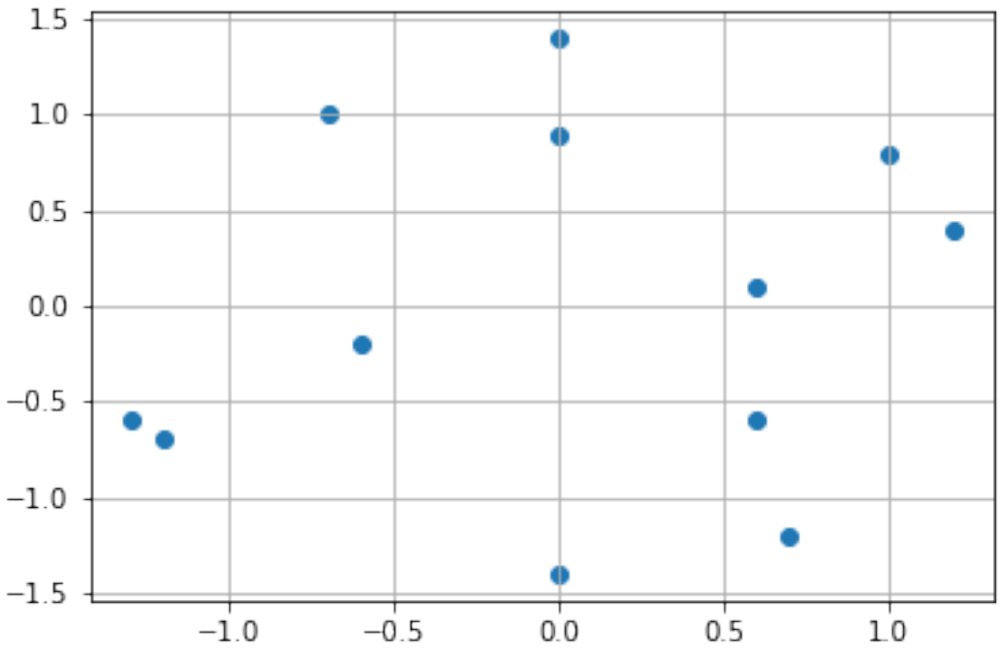
```
learning: 100%|████████| 750/750 [01:58<00:00,  6.32it/s]
```

```
plt.subplot(1, 2, 1)
plt.plot(radiusus)
plt.title("Радиус")
plt.subplot(1, 2, 2)
plt.plot(lrs)
plt.title("Коэффициент скорости обучения")
plt.show()
```

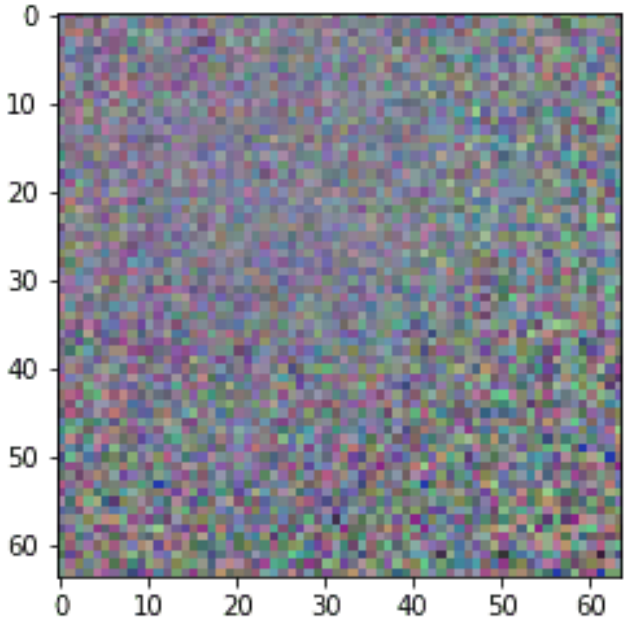



```
plt.scatter(x, y)
plt.grid(True, which='both')

plt.show()
```




```
Image(open('som.gif', 'rb').read())
```



Выводы

Ознакомился с самоорганизующейся картой Кохонена и реализовал её для кластеризации множества точек.