Київський національний університет
імені Т.Шевченка

Звіт

до лабораторних робіт 3-4
з предмету Нейронні мережі та нейрообчислення
«Мережі Конохена»

Студента четвертого курсу
Групи ТК-41
Факультету комп'ютерних наук
та кібернетики
Некряча Владислава

Київ
2023

Постановка задачі

В таблиці 1 приведені параметри деяких об’єктів.

Розіб’ємо (класифікуємо) ці об’єкти на два класи.

Розмірність вектора образів N=5.

Назва	Параметр1	Параметр2	Параметр3	Параметр4	Параметр5
Об’єкт 1	5.4	4020	2060	57	37
Об’єкт 2	8.9	4810	2223	140	40
Об’єкт 3	19.5	5380	2910	285	60
Об’єкт 4	25	5890	2880	300	40
Об’єкт 5	44.8	6870	3270	700	55
Об’єкт 6	56	6316	3705	700	44
Об’єкт 7	68	7380	3755	700	38
Об’єкт 8	13.8	5200	2470	300	60
Об’єкт 9	9.2	4285	2348	140	42
Об’єкт 10	30	5920	3000	500	54
Об’єкт 11	31.8	6070	3180	500	60
Об’єкт 12	47.5	6675	3320	600	34
Об’єкт 13	44.2	6770	3070	520	37

Назва	Параметр1	Параметр2	Параметр3	Параметр4	Параметр5
Об’єкт 14	46	6770	3070	520	37
Об’єкт 15	49	6900	3150	520	40

Алгоритм навчання мережі Конохена.

1. Ініціалізуємо ваги випадковим чином значеннями від 0.1 до 0.3.
$$W^k=[w^k_1,w^k_2,...,w^k_N],$$
2. Вибираємо коефіцієнт навчання $\lambda=0.3$, $\Delta\lambda=0.05$.
3. Поки $\lambda>0$ виконуємо кроки 4-5-6
4. Повторити 10 раз кроки 5-6

5. Для кожного X^m шукаємо найближчий вектор W^k і для знайденого вектора W^k корегуємо компоненти:
$$w^k_n=w^k_n+\lambda(x^m_n-w^k_n).$$

6.Зменшуємо коефіцієнти навчання $\lambda=\lambda-\Delta\lambda$, $\Delta\lambda=0.05$. На крок 4.

7.Кінець роботи.

SOM

1. Ініцілізуємо вектора W^m випадковими числами, бажано, щоб ці числа були порядку тих, що в початкових даних.
2. Покладемо параметр часу t = 1.
3. Вибираємо довільний вектор з початкових даних X.
4. Знаходим нейрон, який найближчий до вектора X. Найближчий по метриці N-мірного вектора.

Позначимо цей нейрон через W^{m*}, де m*- номер цього нейрона.

5. Корегуємо всі коефіцієнти за формулою

$w^m_n=w^m_n+\eta(t)h(t,\rho(m,m^*))[x_n-w^m_n]$,

де $\eta(t)=\eta_0\exp(-at)$,

$h(t,\rho)=\exp[-(\rho^2)/(2\sigma(t))]$,

$\sigma(t)=\sigma_0\exp(-bt)$.

Через $\rho(m,m^*)$ позначено відстань в геометрії розміщення нейронів на площині для нейрона с номером m и m*.

6. t = t + 1

7. Якщо перевищено максимальний час, то виходимо із алгоритму.
8. Перехід до уроку 3.

Опис (завдання 3)

Імпортуємо бібліотеки та вводимо початкові дані.

```
import math
from pprint import pprint
from typing import List
import numpy as np


data = [
    [5.4, 4020, 2060, 57, 37],
    [8.9, 4810, 2223, 140, 40],
    [19.5, 5380, 2910, 285, 60],
    [25, 5890, 2880, 300, 40],
    [44.8, 6870, 3270, 700, 55],
    [56, 6316, 3705, 700, 44],
    [68, 7380, 3755, 700, 38],
    [13.8, 5200, 2470, 300, 60],
    [9.2, 4285, 2348, 140, 42],
    [30, 5920, 3000, 500, 54],
    [31.8, 6070, 3180, 500, 60],
    [47.5, 6675, 3320, 600, 34],
    [44.2, 6770, 3070, 520, 37]
]

predict_data = [
    [46, 6770, 3070, 520, 37],
    [49, 6900, 3150, 520, 40]
]
```

Алгоритм: нормалізуємо дані, тренуємо мережу, перевіряємо натреновану мережу на даних, які використовувалися для тренування та на нових даних.

```
if __name__ == '__main__':
    result = normalize(data)
    W = generate_random_weights()
    train(result, W)
    print("Train data verification:")
    for data_point in data:
        closest_weight_vector_index = find_closest_vector(data_point, W)
        print(closest_weight_vector_index)
    print("Predictions:")
    for data_point in predict_data:
        closest_weight_vector_index = find_closest_vector(data_point, W)
        print(closest_weight_vector_index)
```

Алгоритм в деталях:

Рандомно ініціалізуємо ваги:

```
def generate_random_weights():
    W = np.random.uniform(
        low=0.1,
        high=0.3,
        size=[
            2,
            len(data[0])
        ]
    )
    return W
```

Нормування початкових векторів.

$X^m=[x^m_1,x^m_2,...,x^m_N]$

Обчислюємо $Max_n=max_{m(}x^m_{n)}$, $Min_n=min_{m(}x^m_{n)}$.

Позначимо $a_n=(Max_n-Min_n)^{-1}$, $b_n=-Min_n(Max_n-Min_n)^{-1}$.

Нормуємо $x^m_n=a_nx^m_n+b_n$, $n=1,...,N$.

Функція нормалізації даних:

```
def normalize(data: List[List[float]]):
    for column_index in range(len(data[0])):
        max_value = float("-inf")
        min_value = float("inf")
        for data_point in data:
            if data_point[column_index] > max_value:
                max_value = data_point[column_index]
            if data_point[column_index] < min_value:
                min_value = data_point[column_index]
        a_n = 1 / (max_value - min_value)
        b_n = - min_value / (max_value - min_value)
        for data_point in data:
            data_point[column_index] = a_n * data_point[column_index] + b_n
    return data
```

Навчання мережі

Виставляємо початкові значення для швидкості навчання та дельти навчання.
Тренуємо мережу за алгоритмом:

1. Ініціалізуємо ваги випадковим чином значеннями від 0.1 до 0.3.

$$W^k=[w^k_1,w^k_2,...,w^k_N],$$

2. Вибираємо коефіцієнт навчання $\lambda=0.3$, $\Delta\lambda=0.05$.
3. Поки $\lambda>0$ виконуємо кроки 4-5-6
4. Повторити 10 раз кроки 5-6

5. Для кожного X^m шукаємо найближчий вектор W^k і для знайденого вектора W^k корегуємо компоненти:

$$w^k_n=w^k_n+\lambda(x^m_n-w^k_n).$$

6.Зменшуємо коефіцієнти навчання $\lambda=\lambda-\Delta\lambda$, $\Delta\lambda=0.05$. На крок 4.

7.Кінець роботи.

```
def train(data, W):
    learning_rate = 0.3
    learning_rate_delta = 0.05
    while learning_rate > 0:
        for _ in range(10):
            for data_point in data:
                closest_weight_vector_index = find_closest_vector(data_point, W)
                for weight_index in range(len(W[closest_weight_vector_index])):
                    W[closest_weight_vector_index][weight_index] += \
                        learning_rate * (data_point[weight_index] -
W[closest_weight_vector_index][weight_index])
            learning_rate -= learning_rate_delta
```

Опис (завдання 4)

Імпортуємо бібліотеки та вводимо початкові дані.

```
import math
import random
from pprint import pprint
from typing import List
import numpy as np
```

Вводимо дані.

```
data = [
    [5.4, 4020, 2060, 57, 37],
    [8.9, 4810, 2223, 140, 40],
    [19.5, 5380, 2910, 285, 60],
    [25, 5890, 2880, 300, 40],
    [44.8, 6870, 3270, 700, 55],
    [56, 6316, 3705, 700, 44],
    [68, 7380, 3755, 700, 38],
    [13.8, 5200, 2470, 300, 60],
    [9.2, 4285, 2348, 140, 42],
    [30, 5920, 3000, 500, 54],
    [31.8, 6070, 3180, 500, 60],
    [47.5, 6675, 3320, 600, 34],
    [44.2, 6770, 3070, 520, 37]
]

predict_data = [
    [46, 6770, 3070, 520, 37],
    [49, 6900, 3150, 520, 40]
]
```

Нормалізуємо дані за тим самим алгоритмом.

```
def normalize(data: List[List[float]]):
    for column_index in range(len(data[0])):
        max_value = float("-inf")
        min_value = float("inf")
        for data_point in data:
            if data_point[column_index] > max_value:
                max_value = data_point[column_index]
            if data_point[column_index] < min_value:
                min_value = data_point[column_index]
        a_n = 1 / (max_value - min_value)
        b_n = - min_value / (max_value - min_value)
        for data_point in data:
            data_point[column_index] = a_n * data_point[column_index] + b_n

    return data
```

Виконуємо наступний алгоритм:

1. Ініцілізуємо вектора W^m випадковими числами, бажано, щоб ці числа були порядку тих, що в початкових даних.
2. Покладемо параметр часу t = 1.
3. Вибираємо довільний вектор з початкових даних X.
4. Знаходим нейрон, який найближчий до вектора X. Найближчий по метриці N-мірного вектора.

   Позначимо цей нейрон через W^{m*}, де m*- номер цього нейрона.
5. Корегуємо всі коефіцієнти за формулою

   $w^m_n=w^m_n + \eta(t)h(t, \rho(m, m^*))[x_n-w^m_n]$,

   де $\eta(t) = \eta_0 \exp(-at)$,

   $h(t, \rho)=\exp[-(\rho^2) / (2\sigma(t))]$,

   $\sigma(t)=\sigma_0 \exp(-bt)$.

   Через $\rho(m, m^*)$ позначено відстань в геометрії розміщення нейронів на площині для нейрона с номером m и m*.
6. t = t + 1

7. Якщо перевищено максимальний час, то виходимо із алгоритму.
8. Перехід до уроку 3.

```
def learning_rate_decay(time):
    return mu_0 * np.exp(-a*time)

def sigma(time):
    return sigma_0 * np.exp(-b * time)

def neighbourhood_function(time, distance):
    return np.exp(-(distance ** 2) / (2 * sigma(time)))


time = 1

while time < 200:
    random_data_point = random.choice(data)
    closest_weight_vector_index =
find_closest_weight_vector(random_data_point, W)
    for vector_index, weight_vector in enumerate(W):
        if vector_index != closest_weight_vector_index:
            distance_between_closest_and_this_weight_vectors =
calc_eucledean_distance(
                W[closest_weight_vector_index],
                weight_vector
            )
            for column_index in range(len(weight_vector)):
                weight_vector[column_index] += \
                    learning_rate_decay(time) * \
                    neighbourhood_function(time,
distance_between_closest_and_this_weight_vectors) * \
                    (random_data_point[column_index] - weight_vector[column_index])
    time += 1
```

Приклад роботи:

Завдання 3 (мережа Конохена):

```
Train data verification:
1
1
1
0
0
0
0
1
1
0
0
0
0
Predictions:
0
0
```

Результати на тренувальних даних сходятяться з кластеризацією в лекції. 14/15 об’єкти додаються до кластеру з елементами 4,5,6,7,10,11,12,13

Завдання 4 (SOM):

```
Train data verification:
0
0
0
1
1
1
1
0
0
```

```
1
1
1
1
Predictions:
1
1
```

SOM кластеризує нові об’єкти в той же кластер (номера кластерів змінюються через різний початковий стан нейронів).