Київський національний університет   
імені Т.Шевченка

**Звіт**

до лабораторної роботи 3-4  
з предмету Нейронні мережі та нейрообчислення «Мережі Кохонена»

***Студента четвертого курсу   
Групи ТК-42   
Факультету комп’ютерних наук   
та кібернетики   
Мальованого Дмитра***

***Київ****2023*

Алгоритм роботи:

Нехай є набор із M образів, який необхідно класифікувати, тобто, розбити на K класів.

Кожний образ будем описувати деяким вектором

Xm=[xm1,xm2,...,xmN],

де xmi - дійсні числа, m=1,2…M.

Навчати будемо вагові коефіцієнти

Wk=[wk1,wk2,...,wkN],

k=1,2,...,K, де K – кількість класів на які розбиваємо початкові образи.

Нормування початкових векторів.

Xm=[xm1,xm2,...,xmN]

Обчислюємо Maxn=maxm(xmn), Minn=minm(xmn).

Позначимо an=(Maxn-Minn)-1, bn=-Minn(Maxn-Minn)-1.   
Нормуємо xmn=anxmn+bn, n=1,...,N.

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

Wk=[wk1,wk2,...,wkN],

1. Вибираємо коефіцієнт навчання λ=0.3, Δλ=0.05.
2. Поки λ>0 виконуємо кроки 4-5-6
3. Повторити 10 раз кроки 5-6
4. Для кожного Xm шукаємо найближчий вектор Wk і для знайденого вектора Wk корегуємо компоненти:

wkn=wkn+λ(xmn-wkn).

1. Зменшуємо коефіцієнти навчання λ=λ-Δλ, Δλ=0.05. На крок 4.
2. Кінець роботи.

Мережа навчена.

**Тепер будь-який вектор X потрібно нормувати та знайти найближчий вектор Wk, де номер k і буде номером класу.**

Оскільки ініціалізували ваги випадковим чином, то номери класів ролі не грають, важливе лише групування образів по класах.

Якщо до ваг Wk примінити процедуру обернену до нормування, то коефіцієнти ваг для кожного класу вкажуть середні значення компонент для класу.

Реалізація



Результат:

Протестуємо реалізацію на декількох випакдових данних:

Випадкові данні: [[3.97330747e-01 2.65730902e-01 9.88834598e-01 5.73564815e-01]

[4.42934210e-01 5.95093460e-01 8.97158615e-01 2.13418214e-01]

[9.64468451e-01 4.23218981e-01 7.79164542e-01 8.81917640e-01]

[3.84158472e-01 2.60696397e-01 3.63357898e-04 9.07857068e-01]

[8.45427427e-02 6.58306062e-01 4.98740152e-01 9.55782537e-01]

[1.98921317e-01 3.35334306e-01 1.46191037e-01 8.54921692e-02]

[7.83739061e-01 1.75995529e-01 5.84216761e-01 3.20443933e-01]

[9.17377469e-01 4.47453477e-01 8.57098694e-01 7.91859112e-01]

[1.67668382e-01 3.92807797e-02 1.83432535e-01 7.17448931e-01]

[3.25163298e-01 3.67955247e-01 8.41640282e-01 6.02905263e-01]

[5.82074352e-01 8.23017842e-01 5.58111403e-01 7.65692010e-01]

[6.29040981e-01 1.53321485e-01 9.58519939e-01 8.25773144e-02]

[1.11367449e-01 8.39636521e-01 4.77342322e-02 9.10825348e-01]

[1.40527017e-01 8.89225275e-01 8.03963514e-01 8.42866154e-01]

[9.58758497e-01 2.24035284e-01 9.98763135e-01 8.76823064e-01]

[4.50043045e-01 1.31949573e-03 6.56454570e-01 3.83902060e-01]

[9.78289013e-01 1.74605254e-01 7.37695140e-01 3.72253543e-03]

[6.41813707e-01 9.88544521e-01 8.76500987e-01 9.44652513e-02]

[4.25737805e-01 9.79302092e-01 6.69054349e-01 2.46371379e-02]

[1.57245087e-02 3.83247621e-01 2.13837498e-01 7.90600474e-01]

[3.06188192e-01 3.42082033e-01 7.43098466e-01 9.60959673e-01]

[3.93423123e-01 8.61105771e-01 9.35520436e-01 7.74853468e-01]

[2.59228853e-01 4.61512812e-01 7.95703423e-01 2.80042347e-01]

[5.12857162e-01 4.29168375e-01 6.09732762e-01 8.82709102e-01]

[2.14157178e-01 9.07953352e-01 8.16936814e-01 6.53494402e-01]

[3.24052447e-01 8.80113292e-02 3.52939091e-01 9.76419627e-01]

[7.71852013e-01 3.84541472e-02 4.60724268e-01 6.54816770e-01]

[9.61571010e-01 8.00279196e-01 2.63597462e-01 9.88925987e-01]

[6.04475953e-01 1.08888302e-01 3.16025196e-01 4.50820018e-01]

[4.51725184e-01 2.12969731e-01 3.02084932e-01 4.07088386e-01]

[6.07403394e-01 3.19687407e-01 3.01875628e-01 1.95568444e-01]

[3.74292901e-03 9.83422771e-01 4.40245608e-01 7.56980733e-01]

[8.95491104e-01 8.70803186e-01 1.24229107e-01 7.92814871e-01]

[6.77572444e-01 9.53329447e-01 4.06347579e-01 8.24918779e-02]

[3.09275096e-01 5.77792073e-01 3.36256068e-01 7.30904034e-01]

[1.89756956e-01 4.11550656e-01 8.57538507e-01 6.16440156e-01]

[3.42667098e-01 7.90836360e-02 2.60094070e-01 1.63410377e-01]

[4.60262875e-01 1.29361411e-01 8.73003752e-01 4.68424206e-01]

[6.65421735e-01 5.12062100e-01 7.43949252e-01 6.40861517e-01]

[3.74569636e-01 4.72976385e-01 5.45316150e-01 3.82736079e-01]

[3.67268918e-01 8.29973173e-01 3.80780759e-01 8.02012357e-01]

[8.47233723e-01 1.84927492e-01 6.91994788e-01 3.69943142e-01]

[6.35386365e-01 1.22207193e-01 3.87063539e-01 6.05535276e-01]

[6.15381047e-01 9.93122994e-01 1.17105798e-02 1.73389783e-01]

[1.46914857e-01 7.46701975e-01 6.15801529e-01 6.27792579e-01]

[2.13955371e-01 5.71152081e-01 4.42088273e-01 4.97442407e-01]

[7.96897870e-01 3.64850840e-01 9.12609066e-01 6.86863242e-01]

[7.20901764e-02 3.85268138e-01 7.16703115e-01 4.52348010e-01]

[5.02676841e-01 1.05952800e-01 1.57473873e-01 8.85537515e-01]

[5.70305675e-01 5.48621652e-01 7.58072969e-01 5.03245834e-01]]

Класи: [2. 2. 3. 2. 2. 2. 2. 0. 2. 0. 0. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 2. 0.

2. 2. 3. 2. 3. 0. 2. 2. 2. 2. 1. 2. 2. 2. 0. 0. 1. 2. 3. 2. 2. 1. 2. 2.

3. 0.]

Самоорганізуючі карти Кохонена

Алгоритм роботи:

Нехай вихідні вектора, за якими ми будемо будувати карту представляють собою N-мірні вектора

X=[x1,x2,...,xN].

Карта Кохонена є набором нейронів, які розташовані на площині, наприклад, прямокутної сітки. При цьому важливо, що для кожної пари цих нейронів можна обчислити геометричну відстань. Будемо вважати, що таких нейронів є M.

Кожен нейрон, крім свого геометричного положення, описується N-мірним числовим вектором

Wm=[wm1,wm2,...,wmN],

де wmn - дійсні числа.

Алгоритм навчання мережі:

1. Ініцілізуємо вектора Wm 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 Wm\*, де m\*- номер цього нейрона.
5. Корегуємо всі коефіцієнти за формулою

wmn=wmn + η(t)h(t, ρ(m, m\*))[xn-wmn],

де η(t) = η0exp(-at),

h(t, ρ)=exp[-(ρ2) / (2σ(t))],

σ(t)=σ0exp(-bt).

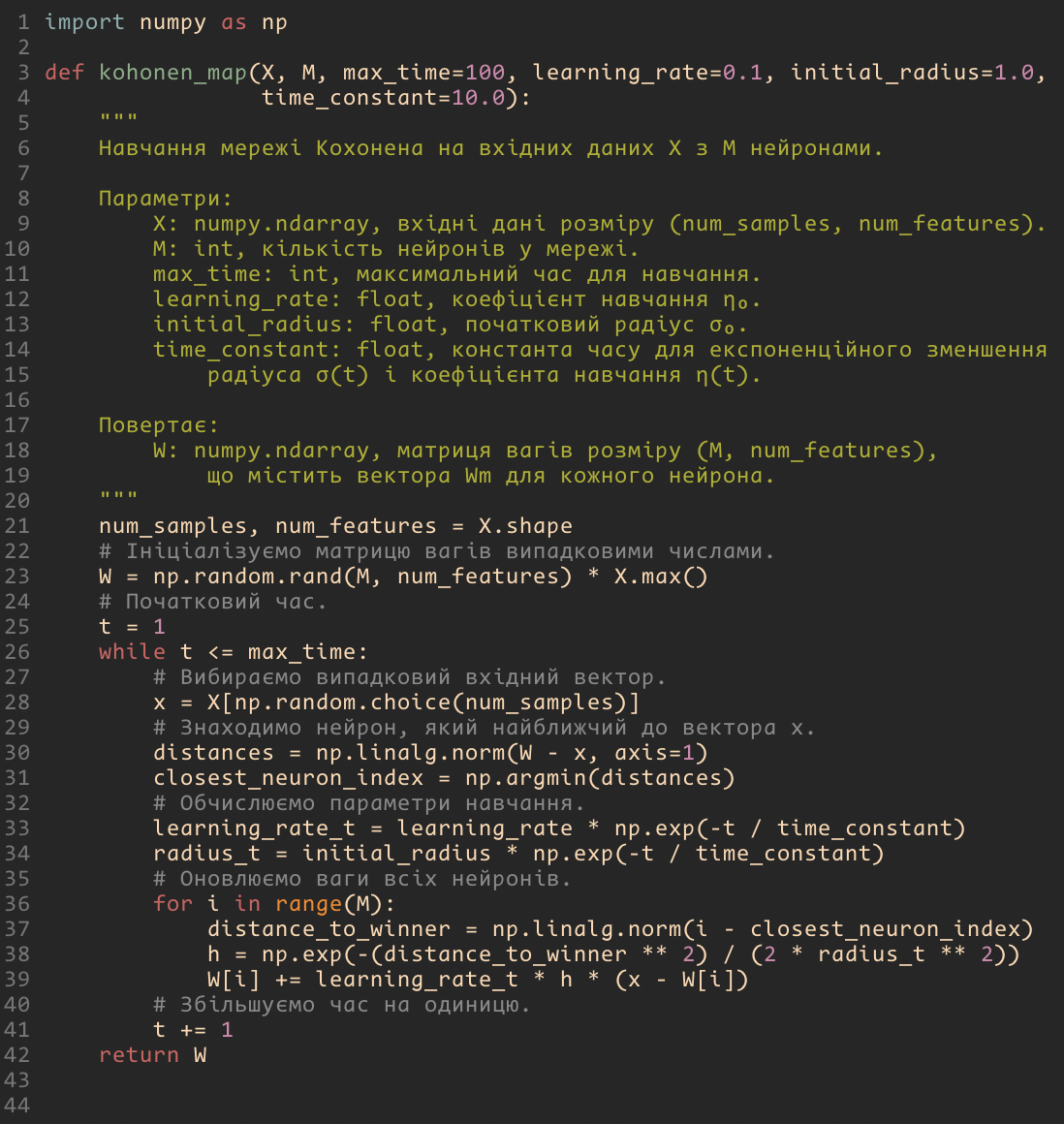
Через ρ(m, m\*) позначено відстань в геометрії розміщення нейронів на площині для нейрона с номером m и m\*.

1. t = t + 1
2. Якщо перевищено максимальний час, то виходимо із алгоритму.
3. Перехід до уроку 3.

Після побудови карти SOM її потрібно градуювати. Для цього потрібно вибрати еталонні значення вхідних даних (необов'язково з тих, що використовувалися в навчанні) і накласти їх на карту. Накласти на карту означає знайти нейрон, який буде ближче всього еталонного значення.

Після градуювання карти її можна використовувати. Для будь-якого вектора X визначаємо найближчий нейрон і його розташування покаже область, до якої відноситься цей вектор

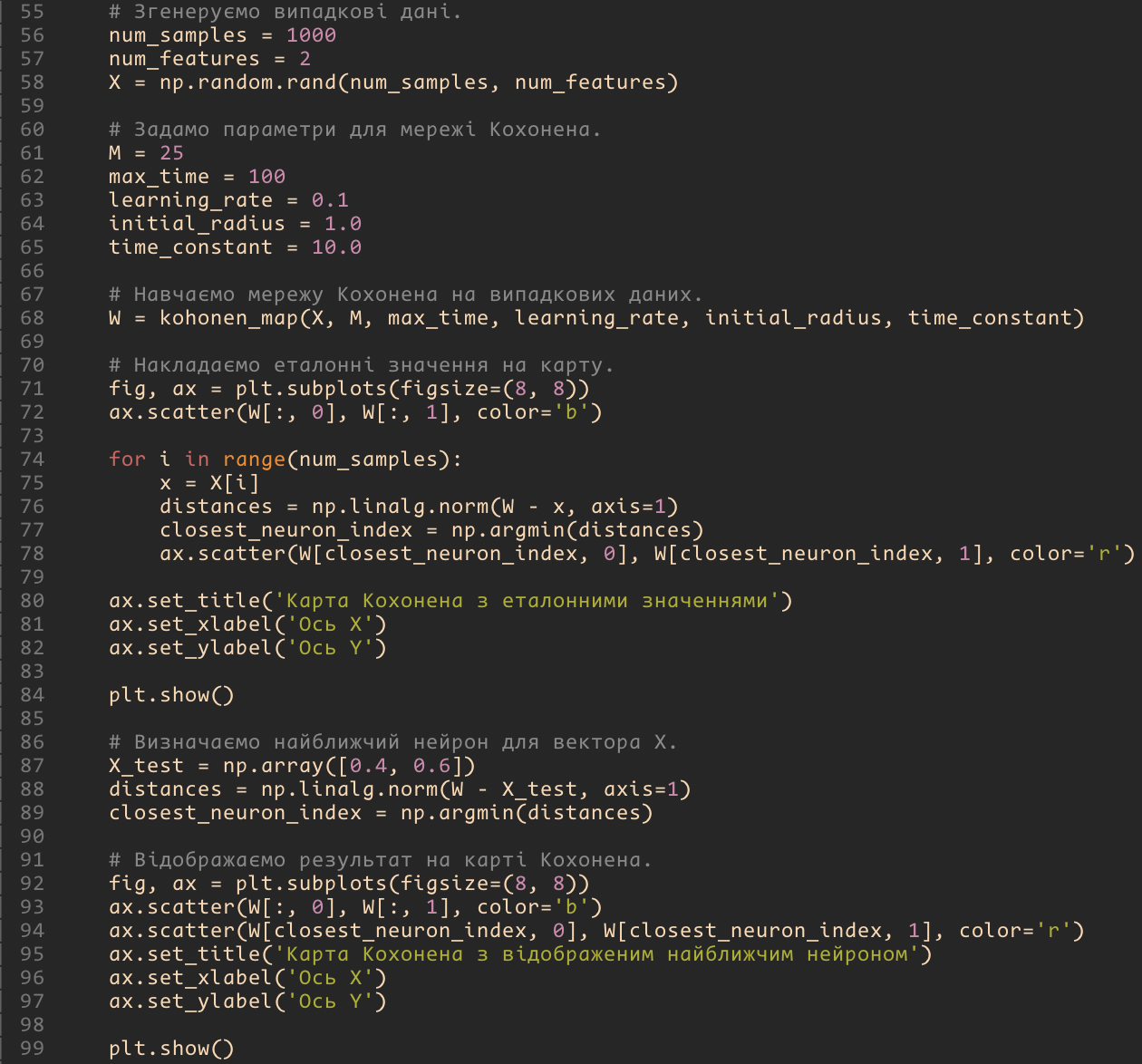
Реалізація:



Функція kohenen\_map будує карту Кохонена для заданого набору даних. Ця функція ініціалізує нейронну мережу з випадковими значеннями ваг, після чого вона тренує мережу за допомогою алгоритму навчання Кохонена. Після закінчення тренування функція повертає розташування нейронів на карті Кохонена.

Значення ваг нейронів на карті Кохонена відображають структуру даних, на яких тренувалась мережа. Градація кольорів на карті Кохонена може вказувати на зони, в яких знаходяться подібні дані. Наприклад, у зеленій зоні можуть знаходитися дані, що містять велику кількість зелених компонент.

Градуюємо SOM:



Результат роботи:

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Весь код за посиланням: <https://github.com/DiMalovanyy/University_Term9/tree/main/NeurNet/Lab3>