Чернівецький національний університет імені Юрія Федьковича

Факультет математики та інформатики

Кафедра математичного моделювання

**Лабораторна робота №2**

з навчальної дисципліни «**Системи штучного інтелекту**»

Варіант №4

Виконав: студент 3 курсу

311 групи

спеціальності «Комп’ютерні науки»

Вайнагій Д. В.

Перевірив: доцент Юрченко І. В.

Чернівці - 2024

Завдання до лабораторної роботи

1. Вивчити теоретичний матеріал щодо різних видів штучних нейронних мереж.

2. Ознайомитися із бібліотекою Neurolab мови Python, її основними класами та методами.

3. Виконати, згідно з варіантом(мережа LVQ), один з вище наведених прикладів використання нейромереж, описати постановку задачі, використані засоби бібліотеки Neurolab, обґрунтувати та проаналізувати отримані результати. Змінити початкові дані самостійно, провести порівняльний аналіз отриманих в обох дослідженнях результатів.

Мережа Learning Vector Quantization (LVQ) представляє собою специфічний тип штучної нейронної мережі, яка здебільшого використовується для завдань класифікації. LVQ була розроблена Тево Кохоненом.

Процес роботи алгоритму LVQ включає наступні етапи:

1. Ініціалізація: Ваги мережі початково ініціалізуються випадковим чином.

2. Конкуренція: Для кожного вхідного вектора обчислюється відстань до всіх нейронів. Нейрон з найменшою відстанню визначається як переможець.

3. Оновлення ваг: Ваги переможця оновлюються на основі різниці між вхідним вектором та вагами переможця. Якщо вхідний вектор та переможець належать до одного класу, ваги переможця збільшуються, в іншому випадку - зменшуються.

4. Повторення: Кроки 2 та 3 повторюються для заданої кількості епох або до тих пір, поки помилка не стане меншою за певний поріг.

Цей алгоритм дозволяє мережі "вчитися" шляхом адаптації ваг до вхідних даних, що забезпечує ефективну класифікацію.

Код який наведений в прикладі дозволяє класифікувати точки за 2 класами та чітко відобразити їх на графіку використовуючи мережу LVQ

Код програми:

import numpy as np

import neurolab as nl

# Створення вхідних данних

input = np.array([[-3, 0], [-2, 1], [-2, -1], [0, 2], [0, 1], [0, -1], [0, -2],

                                                        [2, 1], [2, -1], [3, 0]])

target = np.array([[1, 0], [1, 0], [1, 0], [0, 1], [0, 1], [0, 1], [0, 1],

                                                        [1, 0], [1, 0], [1, 0]])

net = nl.net.newlvq(nl.tool.minmax(input), 4, [.6, .4])

error = net.train(input, target, epochs=1000, goal=-1)

# Plot result

import pylab as pl

xx, yy = np.meshgrid(np.arange(-3, 3.4, 0.2), np.arange(-3, 3.4, 0.2))

xx.shape = xx.size, 1

yy.shape = yy.size, 1

i = np.concatenate((xx, yy), axis=1)

o = net.sim(i)

grid1 = i[o[:, 0]>0]

grid2 = i[o[:, 1]>0]

class1 = input[target[:, 0]>0]

class2 = input[target[:, 1]>0]

pl.plot(class1[:,0], class1[:,1], 'bo', class2[:,0], class2[:,1], 'go')

pl.plot(grid1[:,0], grid1[:,1], 'b.', grid2[:,0], grid2[:,1], 'gx')

pl.axis([-3.2, 3.2, -3, 3])

pl.xlabel('Input[:, 0]')

pl.ylabel('Input[:, 1]')

pl.legend(['class 1', 'class 2', 'detected class 1', 'detected class 2'])

pl.show()

Ініціалізація *net = nl.net.newlvq(nl.tool.minmax(input), 4, [.6, .4])*

1. **nl.tool.minmax(input):** **nl.tool.minmax(input)** використовується для визначення мінімального та максимального значень у вхідних даних **input** для нормалізації. Це важливий крок, оскільки нормалізація даних допомагає забезпечити стабільне та швидке навчання мережі.
2. **Кількість нейронів у конкурентному шарі (Competitive Layer):** У вказаному коді кількість нейронів у конкурентному шарі встановлено на 4. Це означає, що мережа матиме 4 нейрони, які будуть змагатися за право визначати класи для вхідних даних.
3. **Швидкість навчання:** В останньому аргументі **[.6, .4]** передаються значення швидкості навчання для кожного з нейронів у конкурентному шарі. Перше значення (**.6**) відповідає за переможця (Winner), а друге значення (**.4**) - за програвача (Loser). Ці значення впливають на те, наскільки значущо змінюються ваги нейронів під час навчання, що впливає на ефективність навчання та точність класифікації.

*error = net.train(input, target, epochs=1000, goal=-1)*

Після створення мережі LVQ проводиться її навчання за допомогою вхідних даних input та відповідних цільових значень target. Параметри цієї функції включають:

epochs=1000: Кількість епох навчання, тобто кількість повторень навчального процесу.

goal=-1: Ціль для втрати під час навчання. У цьому випадку -1 означає, що навчання буде тривати, поки не буде досягнута мінімальна можлива втрата.

Генерація меш-сітки (Meshgrid): За допомогою np.meshgrid створюється меш-сітка для всіх можливих комбінацій значень x та y. В даному випадку, np.arange(-3, 3.4, 0.2) генерує значення від -3 до 3.2 з кроком 0.2 для обох вхідних змінних x та y.

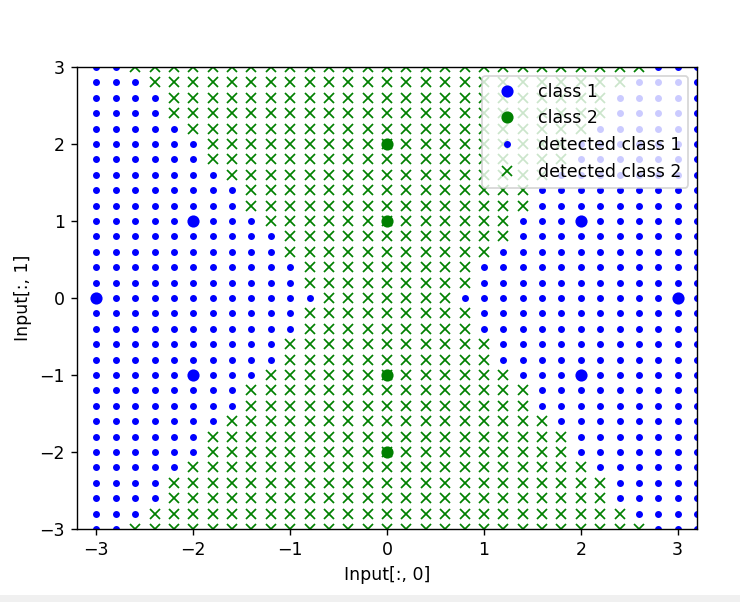
Об'єднання вхідних значень меш-сітки: Значення xx та yy об'єднуються разом за допомогою np.concatenate для створення вхідних даних i, які передаються до нейромережі для передбачення.

Використання навченої мережі для передбачення: За допомогою net.sim(i) застосовується навчена мережа до вхідних даних i. Результати, які повертаються, представлені в масиві o.

Визначення зон визнання класів: За допомогою масиву o визначаються області, в яких мережа визнала класи. Так, grid1 представляє точки, де мережа визнала клас 1, а grid2 - точки, де мережа визнала клас 2.

Визначення точок для кожного класу: Також визначаються точки вхідних даних, які відносяться до кожного з класів. class1 містить вхідні дані, що належать до класу 1, а class2 - до класу 2.

Далі відбувається виведення інформації на графік:



При зміні вихідних даних програма навчатиме нейронну мережу за новими даними і використовуватиме цю модель для класифікації точок за 2 класами. Наприклад з наступними вхідними даними отримаємо такий результат

input = np.array([[-3, 0], [-2, 1], [-2, -1],  [0, 1], [0, -1],

                                                        [2, 1], [2, -1], [3, 0]])

target = np.array([[1, 0], [0, 1], [1, 0], [0, 1], [1, 0], [0, 1], [1,0], [0, 1]])

