**Національний технічний університет України**

**«Київський політехнічний університет імені Ігоря Сікорського»**

**Інститут прикладного системного аналізу**

**Кафедра Системного проектування**

Лабораторна рoбота № 3

з предмету «Штучний інтелект»

на тему: **“Розпізнавання візуальних образів за допомогою мережі Хопфілда”**

Виконала:

Студентка 4-го курсу

групи ДА-81

Желєзнова В.С.

Варіант № 10

Київ – 2021

***Мета роботи:***

Здобути навички програмної реалізації машинного навчання на базі нейронних мереж.

***Завдання:***

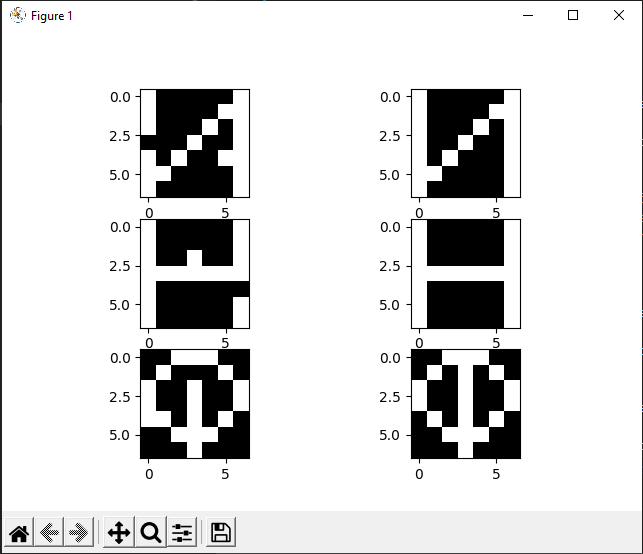
Дослідити методологію машинного навчання в задачі класифікації візуальних образів за допомогою нейронної мережі Хопфілда на прикладі чорно-білих образів літер абетки.

***Варіант №10:***

|  |  |
| --- | --- |
| 10 | И, Н, Ф |

**Хід роботи:**

**Результат роботи програми:**



**Лістинг програми з коментарями:**

**net.py**

import numpy as np

from random import randint, shuffle

class InvalidWeightsException(Exception): #ексепшн неправильних вагів

    pass

class InvalidNetworkInputException(Exception): #ексепшн неправильного вводу

    pass

class HopfieldNetwork(object):

    def \_\_init\_\_(self, num\_inputs):

        self.\_num\_inputs = num\_inputs

        self.\_weights = np.random.uniform(-1.0, 1.0, (num\_inputs, num\_inputs))

    def set\_weights(self, weights):

        """

        Оновлюємо масив вагів

        """

        if weights.shape != (self.\_num\_inputs, self.\_num\_inputs):

            raise InvalidWeightsException()

        self.\_weights = weights

    def get\_weights(self):

        """

        Повертаємо масив вагів

        """

        return self.\_weights

    def calculate\_neuron\_output(self, neuron, input\_pattern):

        """

        Підраховуємо вивід нейрона

        """

        num\_neurons = len(input\_pattern)

        s = 0.0

        for j in range(num\_neurons):

            s += self.\_weights[neuron][j] \* input\_pattern[j]

        return 1.0 if s > 0.0 else -1.0

    def run\_once(self, update\_list, input\_pattern):

        """

        Ітеруємо по кожному нейрону та оновлюємо його вивід

        """

        result = input\_pattern.copy()

        changed = False

        for neuron in update\_list:

            neuron\_output = self.calculate\_neuron\_output(neuron, result)

            if neuron\_output != result[neuron]:

                result[neuron] = neuron\_output

                changed = True

        return changed, result

    def run(self, input\_pattern, max\_iterations=10):

        """

        Запускаємо мережу, використовуючи вхідні дані поки вихідний стан не зміниться

        чи максимальне число ітерацій не буде досягнуто.

        """

        iteration\_count = 0

        result = input\_pattern.copy()

        while True:

            update\_list = list(range(self.\_num\_inputs))

            shuffle(update\_list)

            changed, result = self.run\_once(update\_list, result)

            iteration\_count += 1

            if not changed or iteration\_count == max\_iterations:

                return result

**trainers.py**

import numpy as np

def calculate\_weight(i, j, patterns):

    """

    Порахувати вагу між даними нейронами

    """

    num\_patterns = len(patterns)

    s = 0.0

    for mu in range(num\_patterns):

        s += patterns[mu][i] \* patterns[mu][j]

    w = (1.0 / float(num\_patterns)) \* s

    return w

def calculate\_neuron\_weights(neuron\_index, input\_patterns):

    """

    Порахувати ваги для даного нейрону

    """

    num\_patterns = len(input\_patterns)

    num\_neurons = len(input\_patterns[0])

    weights = np.zeros(num\_neurons)

    for j in range(num\_neurons):

        if neuron\_index == j: continue

        weights[j] = calculate\_weight(neuron\_index, j, input\_patterns)

    return weights

def hebbian\_training(network, input\_patterns):

    """

    Тренуємо мережу використовуючи навчальне правило Хебіана

    """

    n = len(input\_patterns)

    num\_neurons = len(input\_patterns[0])

    weights = np.zeros((num\_neurons, num\_neurons))

    for i in range(num\_neurons):

        weights[i] = calculate\_neuron\_weights(i, input\_patterns)

    network.set\_weights(weights)

**hopfield.py**

from random import randint

import numpy as np

from hopfieldnet.net import HopfieldNetwork

from hopfieldnet.trainers import hebbian\_training

from matplotlib import pyplot as plt

'''

Ініціалізуємо букви И, Н, Ф у вигляді матриць нулів та одиниць

'''

i\_pattern = np.array([[1, 0, 0, 0, 0, 0, 1],

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

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

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

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

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

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

n\_pattern = np.array([[1, 0, 0, 0, 0, 0, 1],

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

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

                      [1, 1, 1, 1, 1, 1, 1],

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

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

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

f\_pattern = np.array([[0, 0, 1, 1, 1, 0, 0],

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

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

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

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

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

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

'''

Робимо перетворення матриць, аби отримати матриці, що будуть містити значення 1 та -1

'''

i\_pattern \*= 2

i\_pattern -= 1

n\_pattern \*= 2

n\_pattern -= 1

f\_pattern \*= 2

f\_pattern -= 1

'''

Перетворюємо матриці, що втілюють букви у один одностроковий масив.

'''

input\_patterns = np.array([i\_pattern.flatten(), n\_pattern.flatten(), f\_pattern.flatten()])

'''

Створюємо мережу та тренуємо її за допомогою хебіана

'''

network = HopfieldNetwork(49)

hebbian\_training(network, input\_patterns)

'''

Створюємо тестові патерни, використовуючи тренувальні паттерни та додаємо трохи шуму до них

та використовуємо нейронну мережу, аби прибрати шум

'''

i\_test = i\_pattern.flatten()

for i in range(2):

    p = randint(0, 34)

    i\_test[p] \*= -1

i\_result = network.run(i\_test)

i\_result.shape = (7, 7)

i\_test.shape = (7, 7)

n\_test = n\_pattern.flatten()

for i in range(2):

    p = randint(0, 34)

    n\_test[p] \*= -1

n\_result = network.run(n\_test)

n\_result.shape = (7, 7)

n\_test.shape = (7, 7)

f\_test = f\_pattern.flatten()

for i in range(2):

    p = randint(0, 34)

    f\_test[p] \*= -1

f\_result = network.run(f\_test)

f\_result.shape = (7, 7)

f\_test.shape = (7, 7)

'''

Зберігаємо результати у вигляді графіків

'''

plt.subplot(3, 2, 1)

plt.imshow(i\_test, interpolation="nearest", cmap='gray')

plt.subplot(3, 2, 2)

plt.imshow(i\_result, interpolation="nearest", cmap='gray')

'''

interpolation='nearest' - просто відображає зображення без спроби інтерполяції між

пікселями, якщо роздільна здатністсь дисплею не така сама, як роздільна здатність зображення

У результаті ми отримаємо зображення, у якому пікселі відображені як квадрат з множиною пікселів

'''

plt.subplot(3, 2, 3)

plt.imshow(n\_test, interpolation="nearest", cmap='gray')

plt.subplot(3, 2, 4)

plt.imshow(n\_result, interpolation="nearest", cmap='gray')

plt.subplot(3, 2, 5)

plt.imshow(f\_test, interpolation="nearest", cmap='gray')

plt.subplot(3, 2, 6)

plt.imshow(f\_result, interpolation="nearest", cmap='gray')

plt.show()

**Висновок:**

В ході виконання лабораторної роботи я ознайомилася з реалізацією машинного навчання на базі нейронних мереж, побудувавши мережу Хопфілда. Мережу, що була реалізована, було навчено розпізнавати буква И, Н, Ф.