ЛАБОРАТОРНА РОБОТА № 5

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовувати прості нейронні мережі.

Хід роботи:

Завдання 1

Лістинг програми:

```
!pip install neurolab
import neurolab as nl
import numpy as np

def sigmoid(x):
    # Наша функція активації: f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))

class Neuron:
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def feedforward(self, inputs):
        # Вхідні дані про вагу, додавання зміщення
        # і подальше використання функції активації
        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)

weights = np.array([0, 1])  # w1 = 0, w2 = 1
bias = 4  # b = 4
n = Neuron(weights, bias)
x = np.array([2, 3])  # x1 = 2, x2 = 3
print(n.feedforward(x))
```

```
0.9990889488055994
```

Рис.1. Результат виконання програми.

					ДУ «Житомирська політехніка».23.122.4.000 – Лр5		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Дяченко В.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		.					
Керівник						ФІКТ Гр. КН-20-1(1)	
Н. контр.							
Зав. каф.							

Даний код представляє простий нейрон, який використовує сигмоїдну функцію активації для обчислення виходу на основі вхідних даних і вагових коефіцієнтів. З урахуванням, що вхід був x = [2, 3], вихід дорівнює 0.999.

Завдання 2

Лістинг програми:

```
import numpy as np
import neurolab as nl

def sigmoid(x):
    # Наша функція активації: f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))

class Neuron:
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def feedforward(self, inputs):
        # Вхідні дані про вагу, додавання зміщення
        # і подальше використання функції активації
        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)


class DiachenkoNeuralNetwork:
    def __init__(self):
        weights = np.array([0, 1])
        bias = 0
        self.h1 = Neuron(weights, bias)
        self.h2 = Neuron(weights, bias)
        self.o1 = Neuron(weights, bias)

    def feedforward(self, x):
        out_h1 = self.h1.feedforward(x)
        out_h2 = self.h2.feedforward(x)
        out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))
        return out_o1

network = DiachenkoNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x))  # 0.721632560951842

import numpy as np

def mse_loss(y_true, y_pred):
```

```
        # y_true і y_pred є масивами numpy з однаковою довжиною
        return ((y_true - y_pred) ** 2).mean()

    y_true = np.array([1, 0, 0, 1])
    y_pred = np.array([0, 0, 0, 0])
    print(mse_loss(y_true, y_pred))  # 0.5

    import numpy as np

    def sigmoid(x):
        return 1 / (1 + np.exp(-x))

    def deriv_sigmoid(x):
        fx = sigmoid(x)
        return fx * (1 - fx)

    def mse_loss(y_true, y_pred):
        return ((y_true - y_pred) ** 2).mean()

    class DiachenkoNeuralNetwork:
        """
        Нейронна мережа, у якої:
        - 2 входи
        - прихований шар з двома нейронами (h1, h2)
        - шар виходу з одним нейроном (o1)
        """
        def __init__(self):
            self.w1 = np.random.normal()
            self.w2 = np.random.normal()
            self.w3 = np.random.normal()
            self.w4 = np.random.normal()
            self.w5 = np.random.normal()
            self.w6 = np.random.normal()
            self.b1 = np.random.normal()
            self.b2 = np.random.normal()
            self.b3 = np.random.normal()

        def feedforward(self, x):
            h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
            h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
            o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
            return o1

        def train(self, data, all_y_trues):
            learn_rate = 0.1
            epochs = 1000
            for epoch in range(epochs):
                for x, y_true in zip(data, all_y_trues):
                    sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
                    h1 = sigmoid(sum_h1)
```

```
                sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2
                h2 = sigmoid(sum_h2)
                sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
                o1 = sigmoid(sum_o1)
                y_pred = o1

                d_L_d_ypred = -2 * (y_true - y_pred)
                d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
                d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
                d_ypred_d_b3 = deriv_sigmoid(sum_o1)
                d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
                d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

                d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
                d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
                d_h1_d_b1 = deriv_sigmoid(sum_h1)
                d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
                d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
                d_h2_d_b2 = deriv_sigmoid(sum_h2)

                self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 *
d_h1_d_w1
                self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 *
d_h1_d_w2
                self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 *
d_h1_d_b1
                self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 *
d_h2_d_w3
                self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 *
d_h2_d_w4
                self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 *
d_h2_d_b2
                self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
                self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
                self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3

            if epoch % 10 == 0:
                y_preds = np.apply_along_axis(self.feedforward, 1, data)
                loss = mse_loss(all_y_trues, y_preds)
                print("Epoch %d loss: %.3f" % (epoch, loss))

    # Задання набору даних
    data = np.array([
        [-2, -1],  # Alice
        [25, 6],   # Bob
        [17, 4],   # Charlie
        [-15, -6], # Diana
    ])

    all_y_trues = np.array([1, 0, 0, 1])
```

```
# Тренуємо вашу нейронну мережу!
network = DiachenkoNeuralNetwork()
network.train(data, all_y_trues)

# Робимо передбачення
emily = np.array([-7, -3]) # 128 фунтов, 63 дюйма
frank = np.array([20, 2]) # 155 фунтов, 68 дюймов
print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F
print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M
```

```
0.7216325609518421
```

Рис.2. Результат роботи класу DiachenkoNeuralNetwork.

```
0.5
```

Рис.3. Результат підрахунку втрат.

```
# Робимо передбачення
emily = np.array([-7, -3]) # 128 фунтов, 63 дюйма
frank = np.array([20, 2]) # 155 фунтов, 68 дюймов
print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F
print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M

Emily: 0.950
Frank: 0.040
```

Рис.4. Результат передбачення статі.

Висновки: виконавши дане завдання ми розглянули навчальний код нейронної мережі, та можемо сказати наступне:

- **Функція активації sigmoid** використовується для перетворення ваговані сум вхідних сигналів у вихідні значення нейронів від 0 до 1. Це ймовірності активації нейронів, де значення 0 означає низьку активацію, а близьке до 1 – високу.
- **Нейронні мережі прямого поширення можуть використовуватись для** вирішення задач класифікації (визначають клас об'єкта на основі вхідних даних), вирішення задач регресії (навчаються передбачати числові значення на основі вхідних даних), зображення та обробки сигналів (розпізнавання облич, аналізу звуку та інших сигналів), прогно-

зування (нпариклад майбутніх подій на основі історичних даних), а також задач оптимізації (де потрібно знайти найкращий набір параметрів або шлях для досягнення певної цілі).

У нашому коді ми бачимо приклад нейронної мережі прямого поширення, яка навчається класифікувати об'єкти на основі вхідних.

Завдання 3

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Завантаження вхідних даних
text = np.loadtxt('data_perceptron.txt')

# Поділ точок даних та міток
data = text[:, :2]
labels = text[:, 2].reshape((text.shape[0], 1))

# Побудова графіка вхідних даних
plt.figure()

plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Параметр 1')
plt.ylabel('Параметр 2')
plt.title("Вхідні дані")


# Визначення максимального та мінімального значень для кожного виміру
dim1_min, dim1_max, dim2_min, dim2_max = 0, 1, 0, 1

# Кількість нейронів у вихідному шарі
num_output = labels.shape[1]

# Визначення перцептрону з двома вхідними нейронами (оскільки вхідні дані
- двовимірні)
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]

perceptron = nl.net.newp([dim1, dim2], num_output)

# Тренування перцептрону з використанням наших даних
error_progress = perceptron.train(data, labels, epochs=100, show=20,
lr=0.03)
```

```
# Побудова графіка процесу навчання
plt.figure()

plt.plot(error_progress)
plt.xlabel('Кількість епох')
plt.ylabel('Помилка навчання')
plt.title('Зміна помилки навчання')
plt.grid()

plt.show()
```

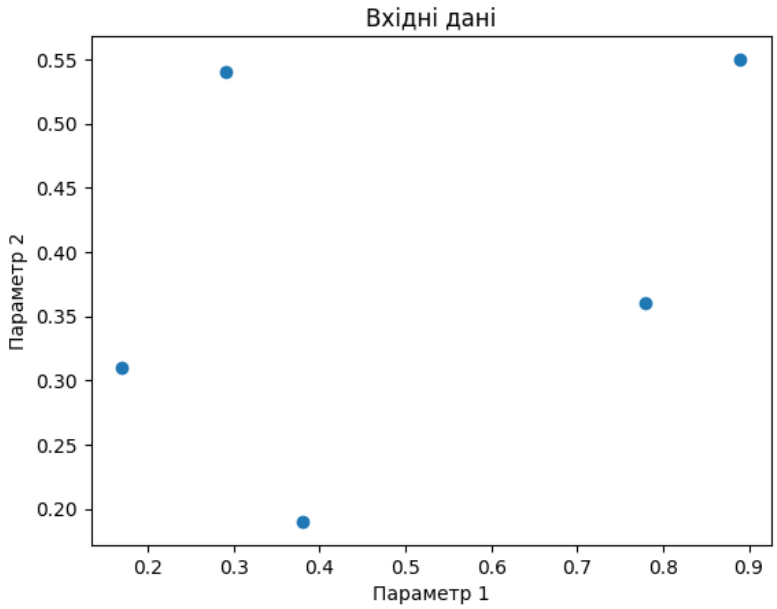

Рис.5. Графік вхідних даних.

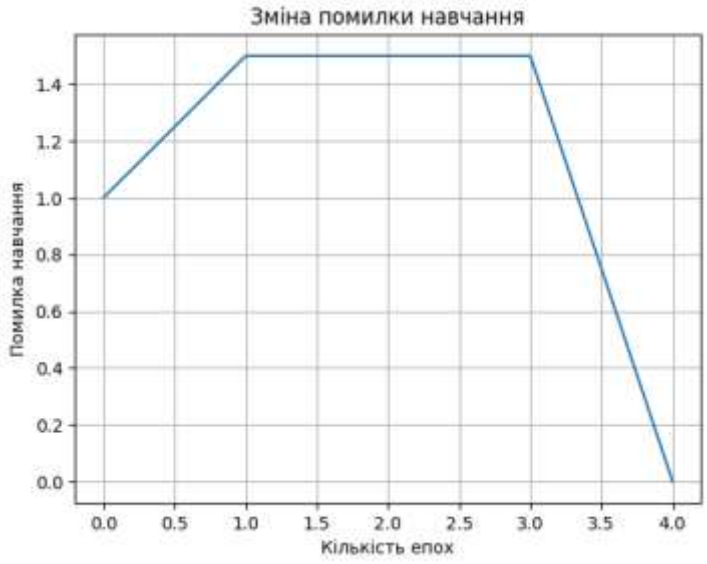

Рис.6. Графік процесу навчання.

Висновки: Другий графік, який побудований після тренування перцептрону, містить інформацію про зміну помилки навчання з плином часу (кількість епох). По горизонталі відображена кількість епох, тобто кількість ітерацій навчання, а по вертикалі - значення помилки навчання. Так як з ітераціями помилка навчання спала - це свідчить про успішність навчання перцептрону.

Завдання 4

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Завантаження вхідних даних
text = np.loadtxt('data_simple_nn.txt')

# Поділ даних на точки даних та мітки
data = text[:, 0:2]
labels = text[:, 2:4]  # Виберіть останні два стовпчики як мітки

# Побудова графіка вхідних даних
plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Паспортність 1')
plt.ylabel('Паспортність 2')
plt.title('Вхідні дані')
plt.show()

# Мінімальне та максимальне значення для кожного виміру
dim1_min, dim1_max = data[:, 0].min(), data[:, 0].max()
dim2_min, dim2_max = data[:, 1].min(), data[:, 1].max()

# Визначення кількості нейронів у вихідному шарі (2 нейрони, оскільки два класи)
num_output = 2

# Визначення одношарової нейронної мережі
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)

# Навчимо мережу на тренувальних даних
error_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)

# Побудова графіка просування процесу навчання
```

```
plt.figure()
plt.plot(error_progress)
plt.xlabel('К-ть епох')
plt.ylabel('Помилка навчання')
plt.title('Зміна помилки навчання')
plt.grid()
plt.show()

# Виконання класифікатора на тестових точках даних
print('\nTest results:')
data_test = [[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]]
for item in data_test:
    print(item, '-->', nn.sim([item])[0])
```

```
Epoch: 20; Error: 4.0;
Epoch: 40; Error: 4.0;
Epoch: 60; Error: 4.0;
Epoch: 80; Error: 4.0;
Epoch: 100; Error: 4.0;
The maximum number of train epochs is reached
```

```
Test results:
[0.4, 4.3] --> [0. 0.]
[4.4, 0.6] --> [1. 0.]
[4.7, 8.1] --> [1. 1.]
```

Рис.7. Дані з вікна терміналу.

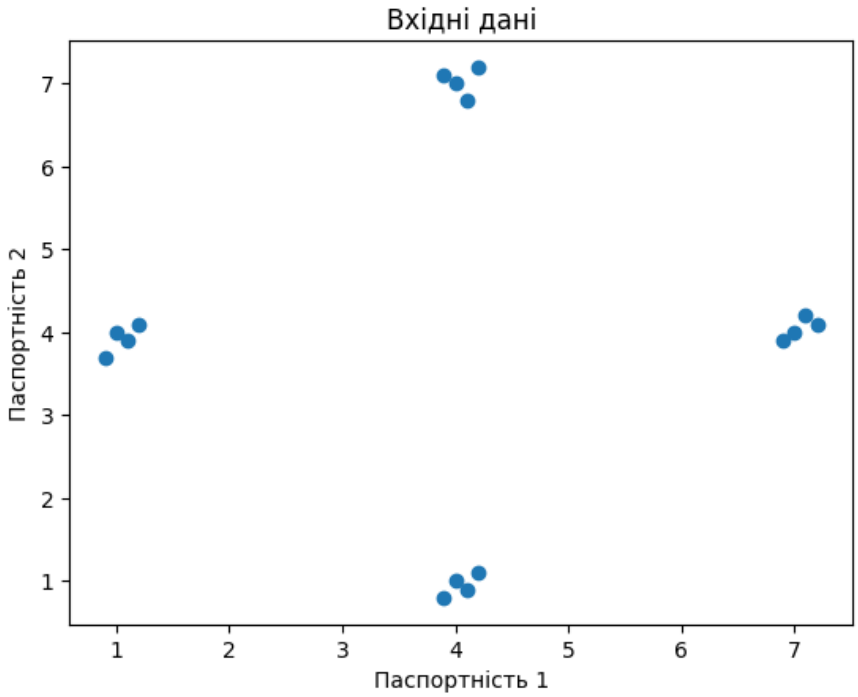

Рис.8. Графік вхідних даних.

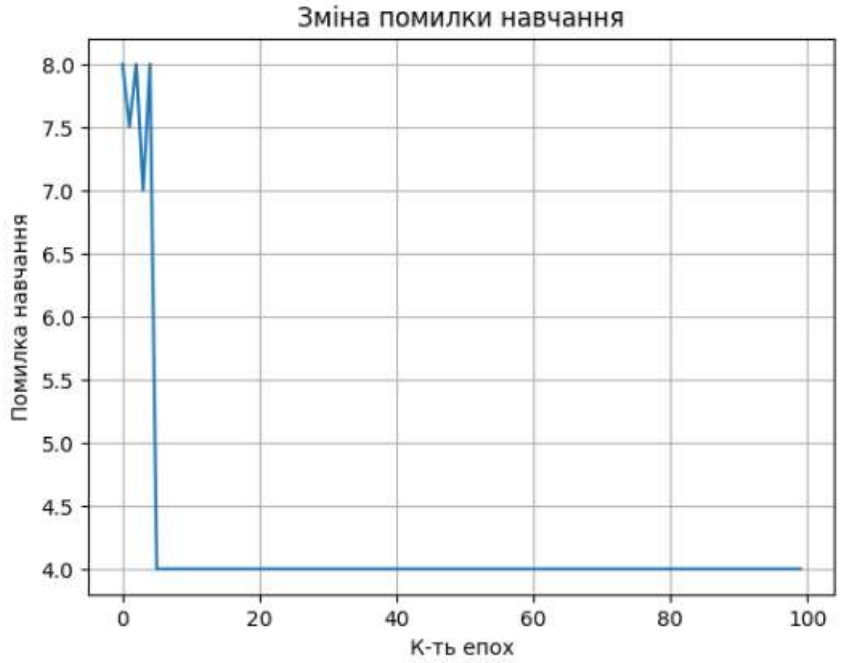

Рис.9. Графік процесу навчання.

Висновки: на основі результатів навчання та отриманого графіку, можна сказати, що:

- Помилка навчання залишилася сталою на протязі більшості епох навчання, і становила 4.0. Це свідчить про те, що одношарова мережа не здатна адекватно навчитися цим даним. Така мережа не може вирішувати складні задачі класифікації, де класи не є лінійно роздільними.
- Проте, у даному випадку, мережа віднесла результати до різних класів, що близькі з дійсністю.

З цими результатами можна зробити висновок, що одношарова нейронна мережа не є найкращим інструментом для цієї задачі класифікації.

Завдання 5

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Генерація тренувальних даних
min_val = -15
```

```
    max_val = 15
    num_points = 130

    x = np.linspace(min_val, max_val, num_points)
    y = 3 * np.square(x) + 5

    y /= np.linalg.norm(y)

    # Створення даних та міток
    data = x.reshape(num_points, 1)
    labels = y.reshape(num_points, 1)

    # Побудова графіка вхідних даних
    plt.figure()
    plt.scatter(data, labels)
    plt.xlabel('Пареметр 1')
    plt.ylabel('Параметр 2')
    plt.title('Вхідні дані')
    plt.show()
    # Визначення багатошарової нейронної мережі з двома прихованими
    # шарами. Перший прихований шар складається із десяти нейронів.
    # Другий прихований шар складається з шести нейронів.
    # Вихідний шар складається з одного нейрона
    nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])

    # Завдання градієнтного спуску як навчального алгоритму
    nn.trainf = nl.train.train_gd

    # Тренування нейронної мережі
    error_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)

    # Виконання нейронної мережі на тренувальних даних
    y_pred = nn.sim(data).reshape(num_points)

    # Побудова графіка помилки навчання
    plt.figure()
    plt.plot(error_progress)
    plt.xlabel("К-ть епох")
    plt.ylabel("Помилка навчання")
    plt.title('Зміна помилки навчання')

    # Побудова графіка результатів
    x_dense = np.linspace(min_val, max_val, num_points * 2)
    y_dense_pred = nn.sim(x_dense.reshape(x_dense.size,
1)).reshape(x_dense.size)
    plt.figure()
    plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
    plt.title('Апроксимація функції за допомогою багатошарової нейронної
мережі')
```

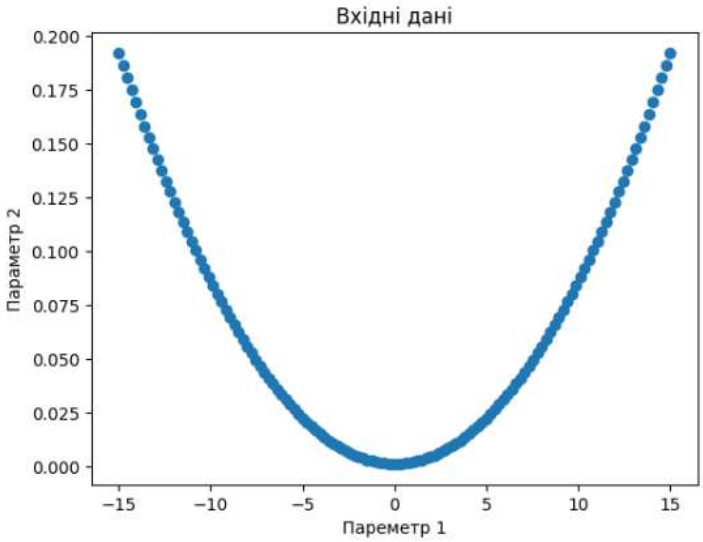

Рис.10. Графік вхідних даних.

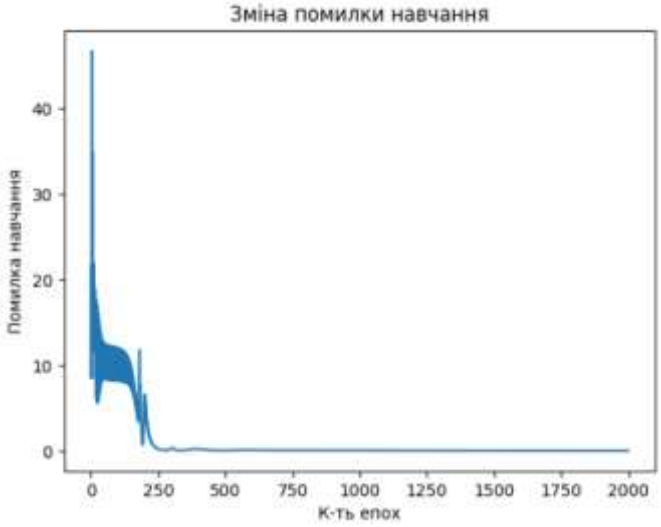

Рис.11. Графік процесу навчання.

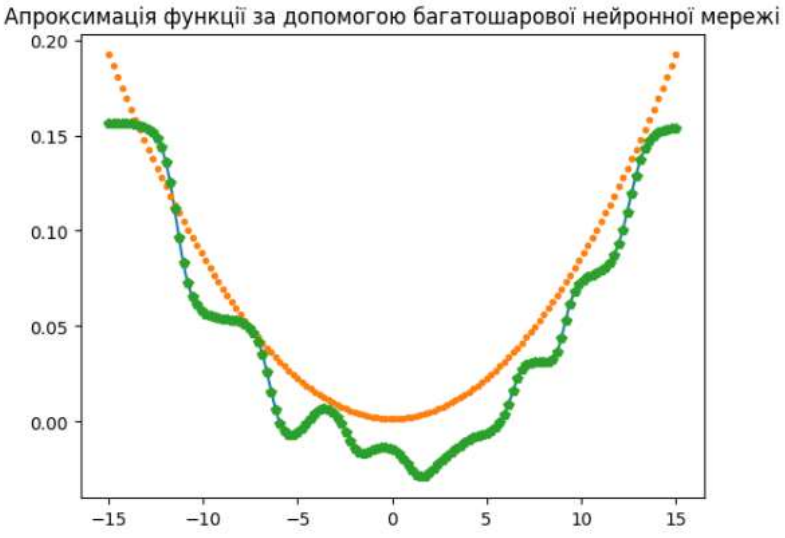

Рис.12. Графік передбачуваних даних.

Висновки: Початкова помилка навчання була високою (40+), але під час тренування поступово зменшувалася з кожною епохою. Після 300 епох навчання вона зменшилася до значення близького нулю. Це свідчить про те, що нейронна мережа успішно навчилася апроксимувати функцію. З графіку передбачуваних даних видно, що мережа має непогані результати. Після повторного навчання точність тільки поліпшується.

Завдання 6

Варіант 4	$y = 2x^2+8$	Варіант 19	$y = 3x^2+2x+1$
	4	2	5-1

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Генерація тренувальних даних
min_val = -15
max_val = 15
num_points = 130

x = np.linspace(min_val, max_val, num_points)
y = 2 * np.square(x) + 8

y /= np.linalg.norm(y)

# Створення даних та міток
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)

# Побудова графіка вхідних даних
plt.figure()
plt.scatter(data, labels)
plt.xlabel('Параметр 1')
plt.ylabel('Параметр 2')
plt.title('Вхідні дані')
plt.show()

# Визначення багатошарової нейронної мережі з двома прихованими
# шарами. Перший прихований шар складається із п'яти нейронів,
# другий прихований шар містить один нейрон.
nn = nl.net.newff([[min_val, max_val]], [5, 1])
```

```
    # Завдання методу оптимізації Adam
    nn.trainf = nl.train.train_gdm

    # Тренування нейронної мережі
    error_progress = nn.train(data, labels, epochs=5000, show=500, lr=0.01,
goal=0.01)

    # Виконання нейронної мережі на тренувальних даних
    y_pred = nn.sim(data).reshape(num_points)

    # Побудова графіка помилки навчання
    plt.figure()
    plt.plot(error_progress)
    plt.xlabel("К-ть епох")
    plt.ylabel("Помилка навчання")
    plt.title('Зміна помилки навчання')

    # Побудова графіка результатів
    x_dense = np.linspace(min_val, max_val, num_points * 2)
    y_dense_pred = nn.sim(x_dense.reshape(x_dense.size,
1)).reshape(x_dense.size)
    plt.figure()
    plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
    plt.title('Апроксимація функції за допомогою багатошарової нейронної
мережі')

    plt.show()
```

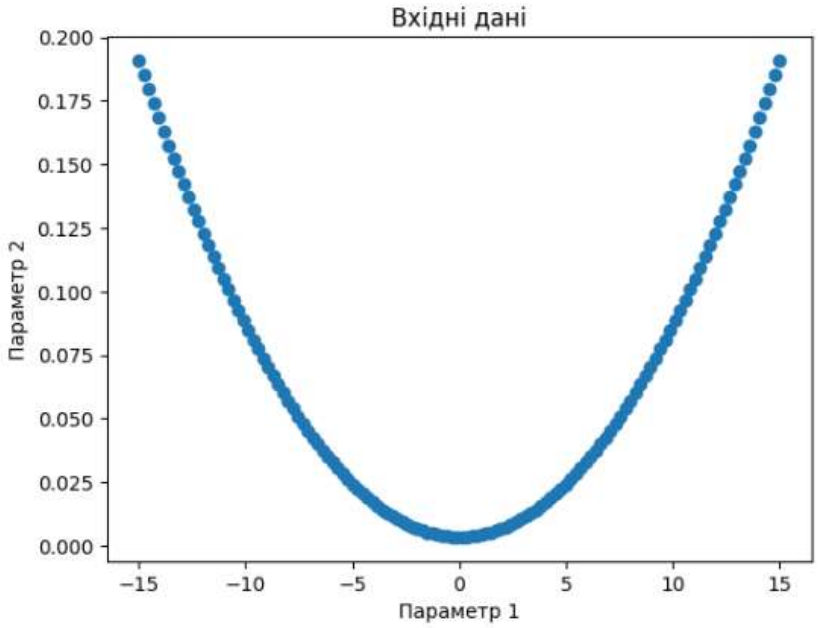

Рис.13. Графік вхідних даних.

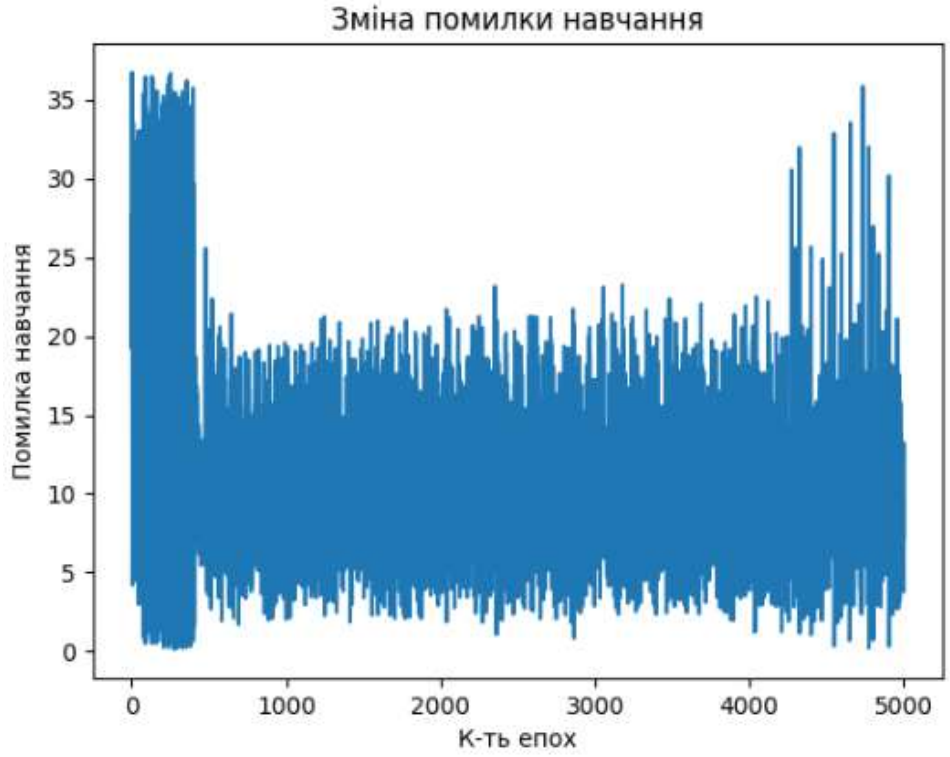

Рис.14. Графік процесу навчання. (gd)

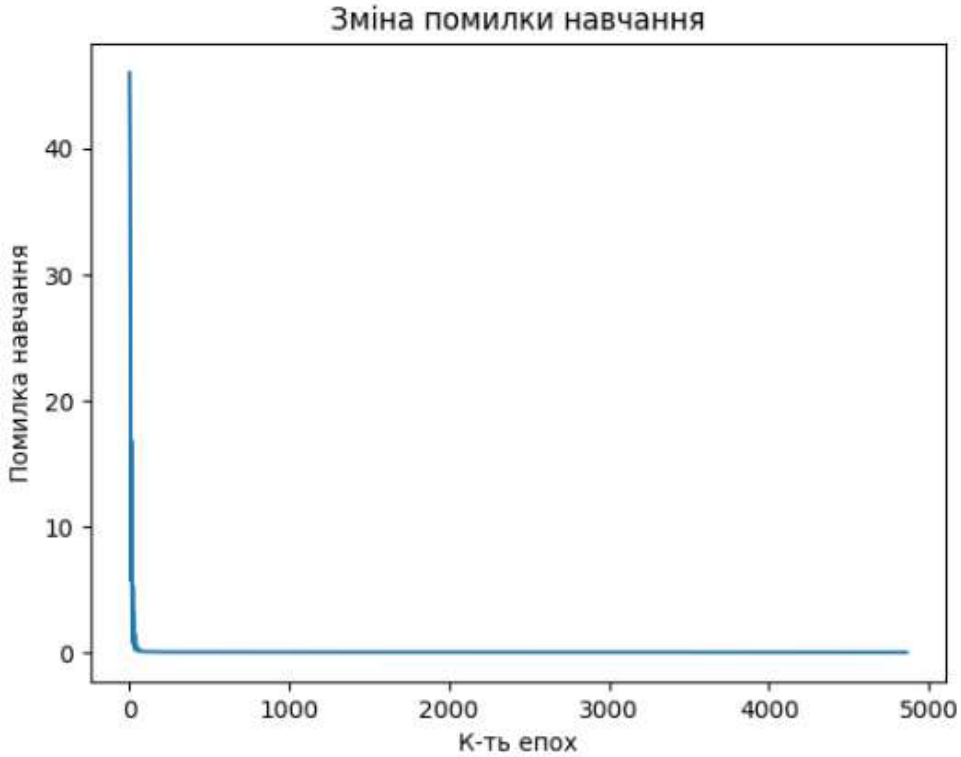

Рис.15. Графік процесу навчання. (gdm)

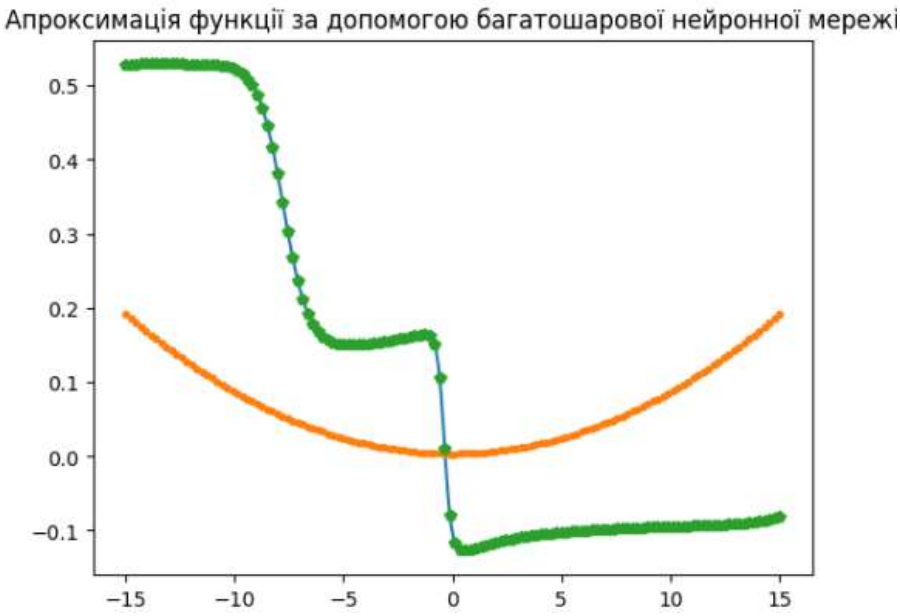

Рис.16. Графік передбачуваних даних. (gd)

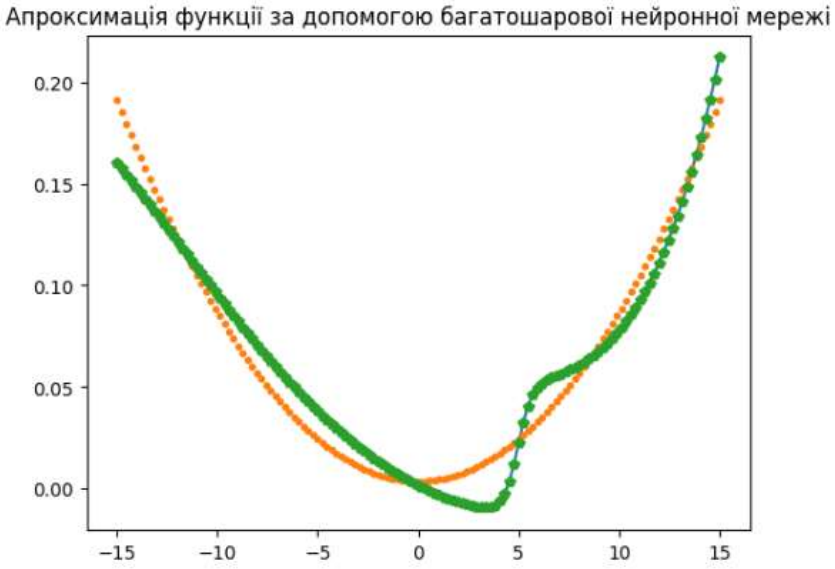

Рис.17. Графік передбачуваних даних. (gdm)

Висновки: при виконанні даного завдання було використано не стандартний градієнтний спуск (Gradient Descent) для оновлення ваг нейронної мережі під час навчання, а спуск з моментом (Gradient Descent with Momentum). Цей метод враховує попередні зміни ваг і використовує момент для прискорення навчання та уникнення локальних мінімумів. Таке рішення було прийнято через те, що стандартний метод оптимізації вимагає багато епох навчання (відповідно ресурсів), і

все одно не дає настільки точного результату. Натомість отримана мережа успішно навчилася апроксимувати функцію, а помилка зменшилась з 40+ до менше ніж 0.1.

Завдання 7

Лістинг програми:

```
import numpy as np
import neurolab as nl
import numpy.random as rand
skv = 0.05
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5]])
rand_norm = skv * rand.randn(100, 4, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 4, 2)
rand.shuffle(inp)
# Create net with 2 inputs and 4 neurons
net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 4)
# train with rule: Conscience Winner Take All algoritm (CWTA)
error = net.train(inp, epochs=200, show=20)
# Plot results:
import pylab as pl
pl.title('Classification Problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('error (default MAE)')
w = net.layers[0].np['w']
pl.subplot(212)
pl.plot(inp[:,0], inp[:,1], '.', \
 centr[:,0], centr[:, 1] , 'yv', \
 w[:,0], w[:,1], 'p')
pl.legend(['train samples', 'real centers', 'train centers'])
pl.show()
```

```
Epoch: 20; Error: 33.69038385590757;
Epoch: 40; Error: 31.250294488951287;
Epoch: 60; Error: 31.150666623593033;
Epoch: 80; Error: 31.137548921471357;
Epoch: 100; Error: 31.135501510130634;
Epoch: 120; Error: 31.13516659206445;
Epoch: 140; Error: 31.135110634871737;
Epoch: 160; Error: 31.135101080820693;
Epoch: 180; Error: 31.135099414334995;
Epoch: 200; Error: 31.135099117705476;
The maximum number of train epochs is reached
```

Рис.18. Дані з вікна терміналу.

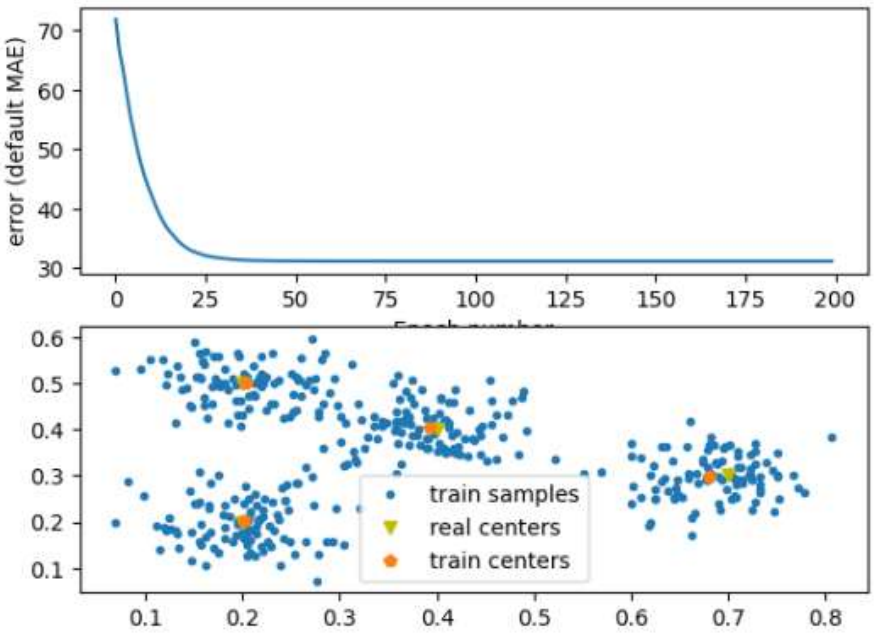

Рис.19. Отримані графіки.

Висновки: MAE (Mean Absolute Error) - це метрика, яка використовується для оцінки точності моделі. Вона вимірює середню абсолютну різницю між прогнозованими значеннями моделі і фактичними. MAE вимірює, наскільки в середньому прогнози моделі відрізняються від фактичних даних.

Під час навчання нашої мережі виводиться значення помилки (MAE) на кожній епосі. Помітно, що MAE зменшується з 33.69 на 20-й епосі до приблизно 31.14 на 100-й. Подальше навчання не призводить до зниження помилки, тому ми отримуємо повідомлення "The maximum number of train epochs is reached." Чим менше значення MAE, тим краще модель відповідає даним. У нашому випадку, хоча MAE і зменшилася, вона все ще досить велика.

Щодо графіків, перший графік візуалізує зменшення помилки (MAE) під час навчання на протязі епох (тобто показує дані з вікна терміналу). На другому графіку видно вхідні дані, справжні та центри, навчені мережею. Результати досить непогані (центри розташовані недалеко один від одного).

Завдання 8

Варіант 4	[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]	0,03

Лістинг програми:

```
import numpy as np
import neurolab as nl
import numpy.random as rand

# Задання центрів кластерів
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5,
0.7]])

# Стандартне відхилення для генерації даних
skv = 0.03

# Генерація даних навколо центрів кластерів
rand_norm = skv * rand.randn(100, centr.shape[0], 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * centr.shape[0], 2)
rand.shuffle(inp)

# Створення нейронної мережі з 2 входами та 4 нейронами
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)

# Навчання мережі за допомогою методу CWTA
error = net.train(inp, epochs=200, show=20)

# Побудова графіків
import pylab as pl
pl.title('Classification Problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Номер епохи')
pl.ylabel('Помилка (за замовчуванням MAE)')
w = net.layers[0].np['w']
pl.subplot(212)
pl.plot(inp[:, 0], inp[:, 1], '.', \
        centr[:, 0], centr[:, 1], 'yv', \
        w[:, 0], w[:, 1], 'p')
pl.legend(['Навчальні приклади', 'Справжні центри', 'Центри мережі'])
pl.show()
```

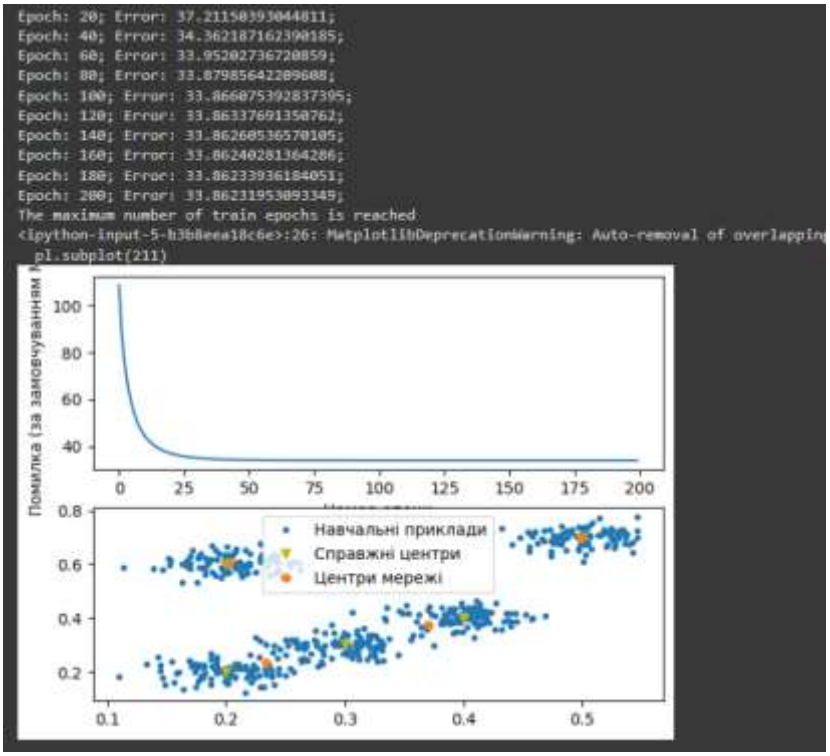

Рис.20. Результат створення нейронної мережі Кохонена з 2 входами та 4 нейронами.

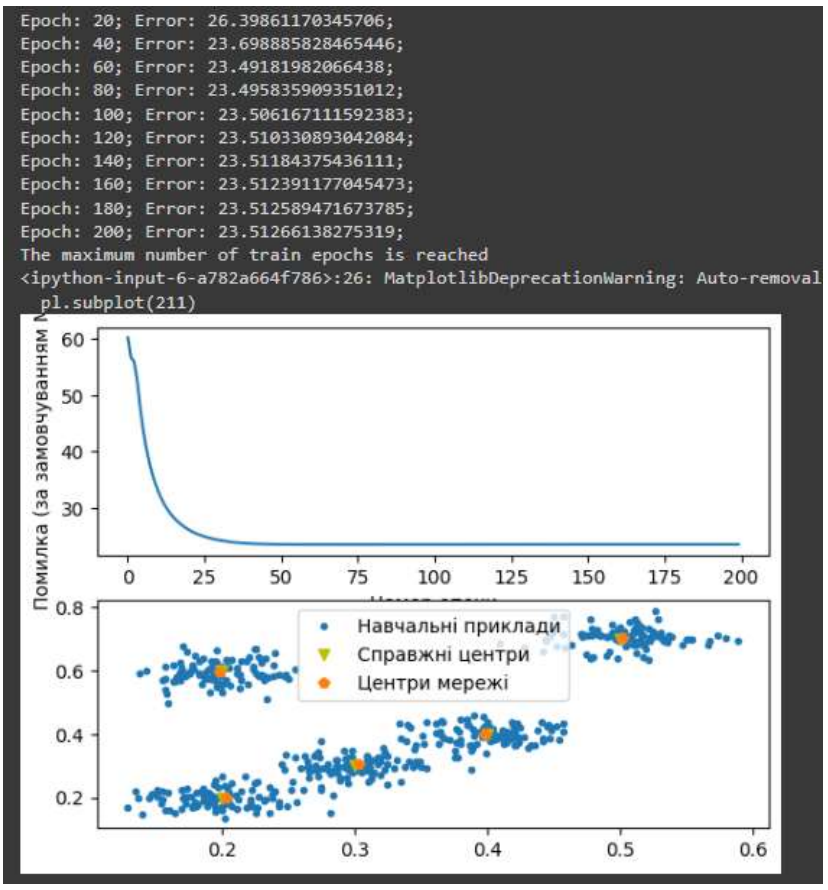

Рис.21. Результат створення нейронної мережі Кохонена з 2 входами та 5 нейронами.

Висновки: Якщо порівняти два набори результатів, можна побачити, що помилка MAE зменшилася при використанні 5 нейронів у порівнянні з 4.

На першому графіку видно, що прогнозований центр розташований між двома реальними - це відображає недостатню точність кластеризації, через невірний вибір кількості нейронів. У той же час при використанні 5 нейронів центри кластерів майже співпадають з реальними.

Тобто більше нейронів може дозволити моделі краще апроксимувати вхідні дані і підвищити її точність, але, водночас, може призвести до перенавчання.

Тож перед тим як обрати к-ть нейронів необхідно провести аналіз даних. Це видно по результатам попереднього завдання. Ми використали 4 нейрони та отримали бажаний результат, в той час як з 5 центрами – неточність даних.