Лабораторна робота 5

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

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

Завдання 2.1 Створити простий нейрон

```python
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))
```

Рис 5.1 Код файлу LR_5_task1.py

```
PS C:\ztu\штучний інтелект\lab5> python .\LR_5_task_1.py
0.9990889488055994
```

Рис 5.2 Результат файлу LR_5_task1.py

Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини

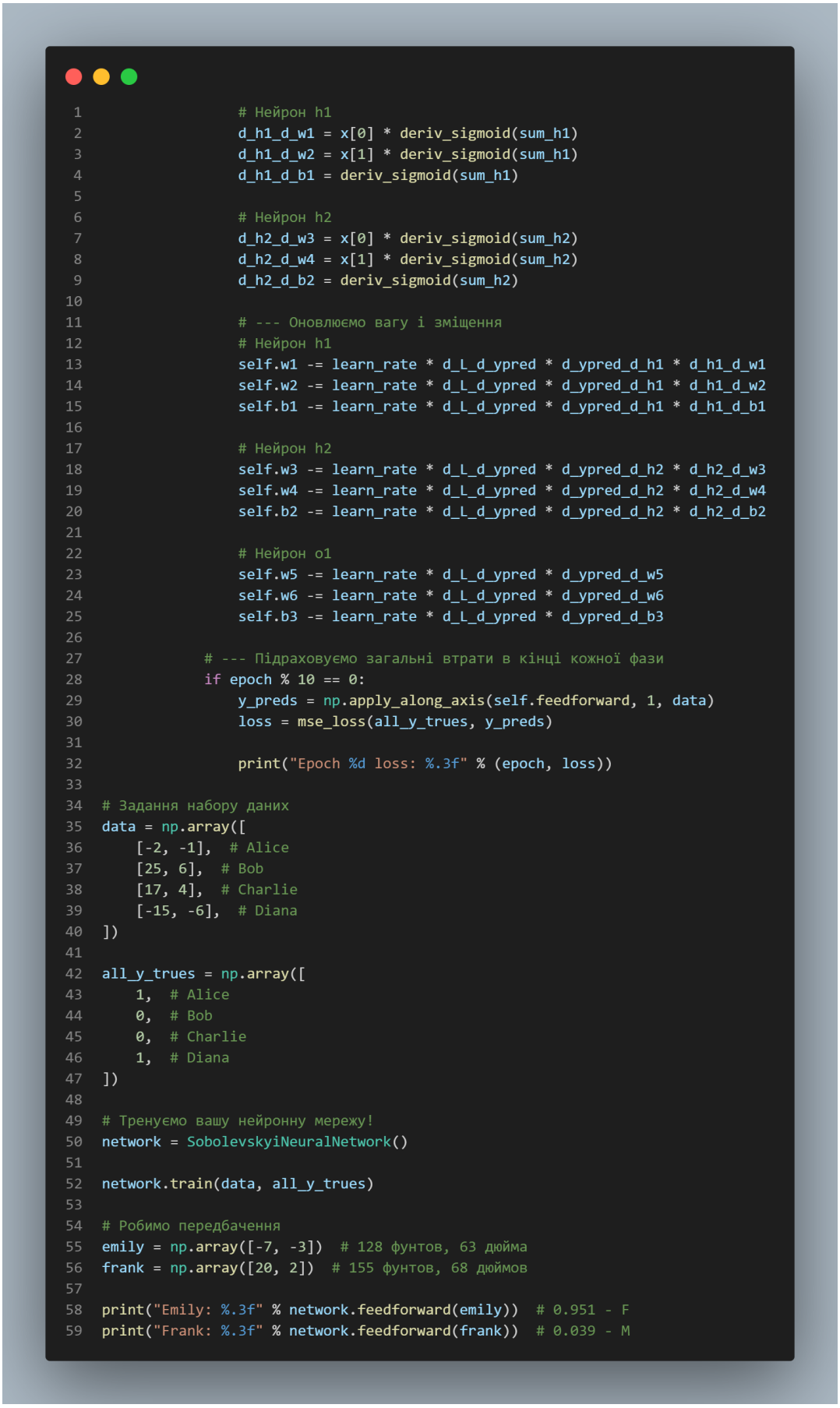
```python
import numpy as np

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

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

    def feedforward(self, inputs):
        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

class SobolevskyiNeuralNetwork:
    def __init__(self):
        weights = np.array([0, 1])
        bias = 0

        # Класс Neuron із попереднього завдання
        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)

        # Входи для o1 є виходами h1 и h2
        out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

        return out_o1

network = SobolevskyiNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x))
```

Рис 5.3 Код файлу LR_5_task_2_1.py

```
PS C:\ztu\штучний інтелект\lab5> python .\LR_5_task_2_1.py
0.7216325609518421
```

Рис 5.4 Результат файлу LR_5_task_2_1.py

```python
import numpy as np

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

def deriv_sigmoid(x):
    # Похідна від sigmoid: f'(x) = f(x) * (1 - f(x))
    fx = sigmoid(x)

    return fx * (1 - fx)
def mse_loss(y_true, y_pred):
    # y_true и y_pred є масивами numpy з одинаковою довжиною
    return ((y_true - y_pred) ** 2).mean()

class SobolevskyiNeuralNetwork:
    def __init__(self):
        #Вага
        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()

        #Зміщення
        self.b1 = np.random.normal()
        self.b2 = np.random.normal()
        self.b3 = np.random.normal()

    def feedforward(self, x):
        # x є масивом numpy з двома елементами
        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):
                # --- Виконуємо зворотній зв'язок (ці значання нам потрібні в подальшому )
                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

                # --- Підрахунок часткових похідних
                # --- Найменування: d_L_d_w1 означає "частково L / частково w1"
                d_L_d_ypred = -2 * (y_true - y_pred)

                # Нейрон o1
                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)

```



```
                # Нейрон h1
                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)

                # Нейрон h2
                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)

                # --- Оновлюємо вагу і зміщення
                # Нейрон h1
                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

                # Нейрон h2
                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

                # Нейрон o1
                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

            # --- Підраховуємо загальні втрати в кінці кожної фази
            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,  # Alice
    0,  # Bob
    0,  # Charlie
    1,  # Diana
])

# Тренуємо вашу нейронну мережу!
network = SobolevskyiNeuralNetwork()

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
```

Рис 5.5 Код файлу LR_5_task_2_2.py

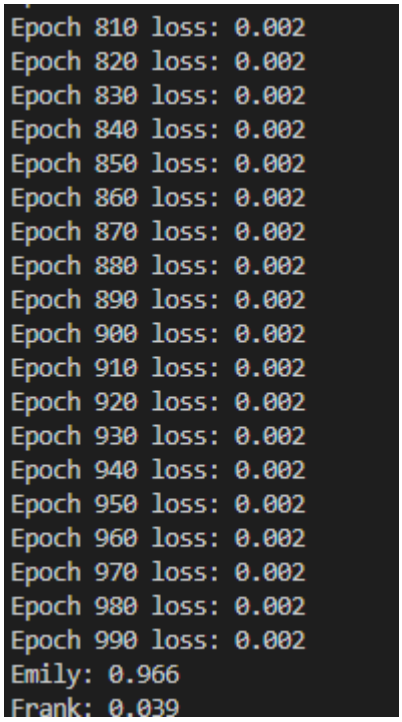


Рис 5.6 Результат файлу LR_5_task_2_2.py

Висновок: Функція активації, або передавальна функція штучного нейрона — залежність вихідного сигналу штучного нейрона від вхідного. Більшість видів нейронних мереж для функції активації використовують сигмоїди.

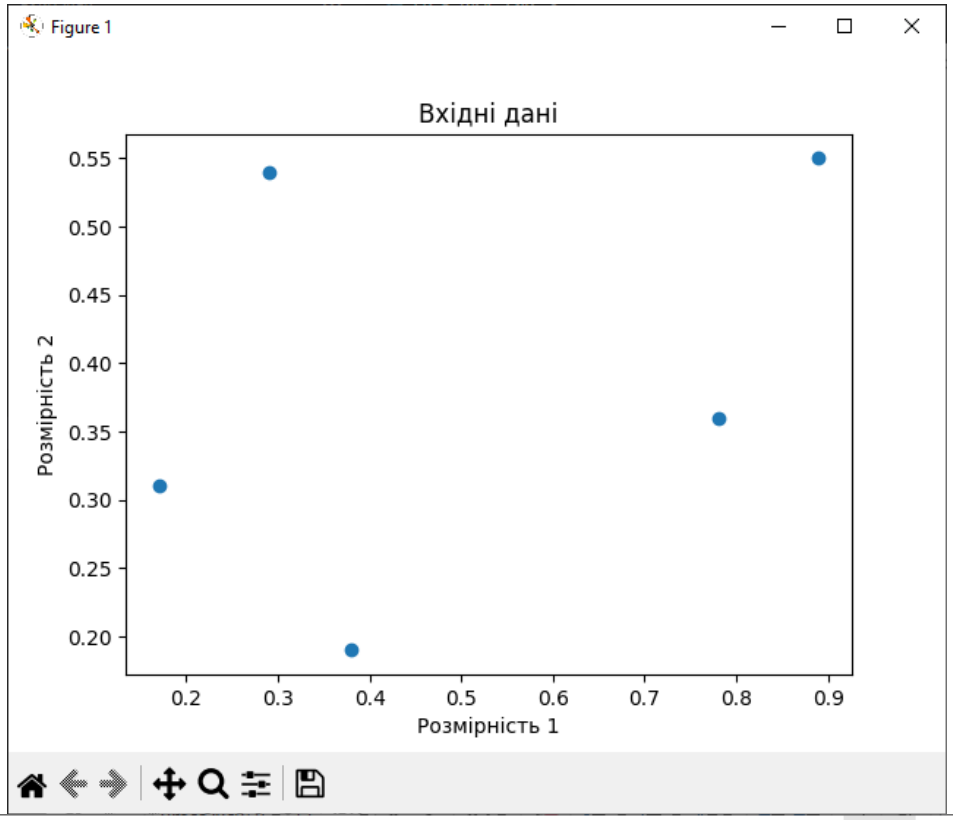
Можливості нейроннних мереж прямого поширення полягають в тому, що сигнали поширюються в одному напрямку, починаючи від вхідного шару нейронів, через приховані шари до вихідного шару і на вихідних нейронах отримується результат опрацювання сигналу. В мережах такого виду немає зворотніх зв’язків.

Нейронні мережі прямого поширення знаходять своє застосування в задачах комп'ютерного бачення та розпізнаванні мовлення, де класифікація цільових класів ускладнюється. Такі типи нейронних мереж добре справляються із зашумленими даними.

Завдання 2.3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

```python
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('Розмірність 1')
plt.ylabel('Розмірність 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 помилок навчання')
plt.grid()
plt.show()
```

Рис 5.7 Код файлу LR_5_task_3.py



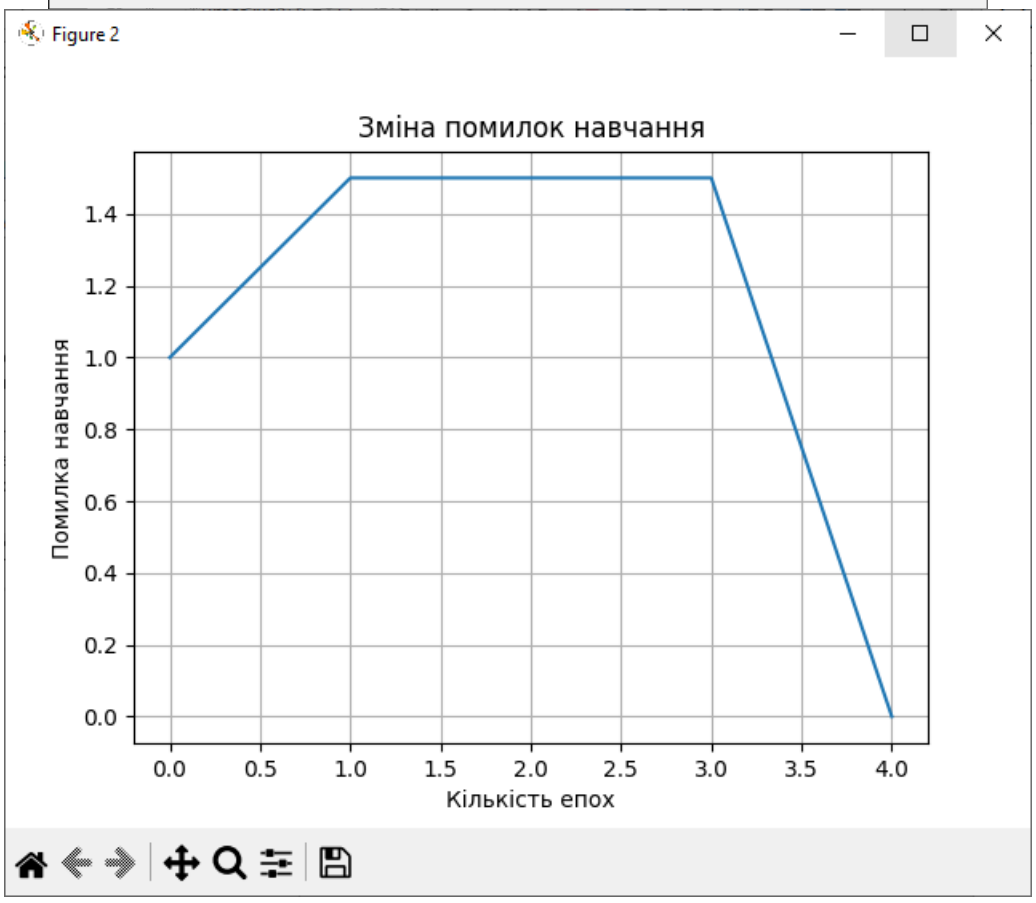


Рис 5.8 Результат файлу LR_5_task_3.py

Завдання 2.4. Побудова одношарової нейронної мережі

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

text = np.loadtxt('data_simple_nn.txt')
data = text[:, 0:2]
labels = text[:, 2:]

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

dim1_min, dim1_max = data[:, 0].min(), data[:, 0].max()
dim2_min, dim2_max = data[:, 1].min(), data[:, 1].max()
num_output = labels.shape[1]
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)
error_progress = nn.train(data, labels, epochs = 100, show = 20, lr = 0.03)

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

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])
```

Рис 5.9 Код файлу LR_5_task_4.py

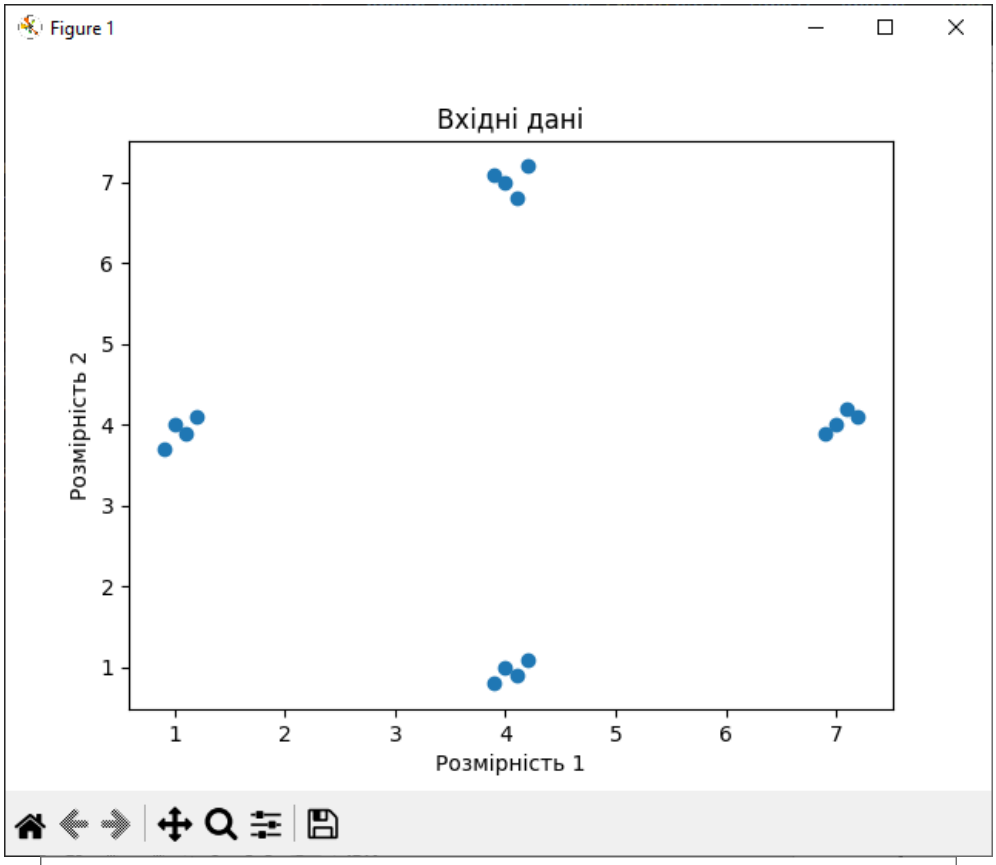


Figure 1
Вхідні дані
Розмірність 2
Розмірність 1
1
2
3
4
5
6
7

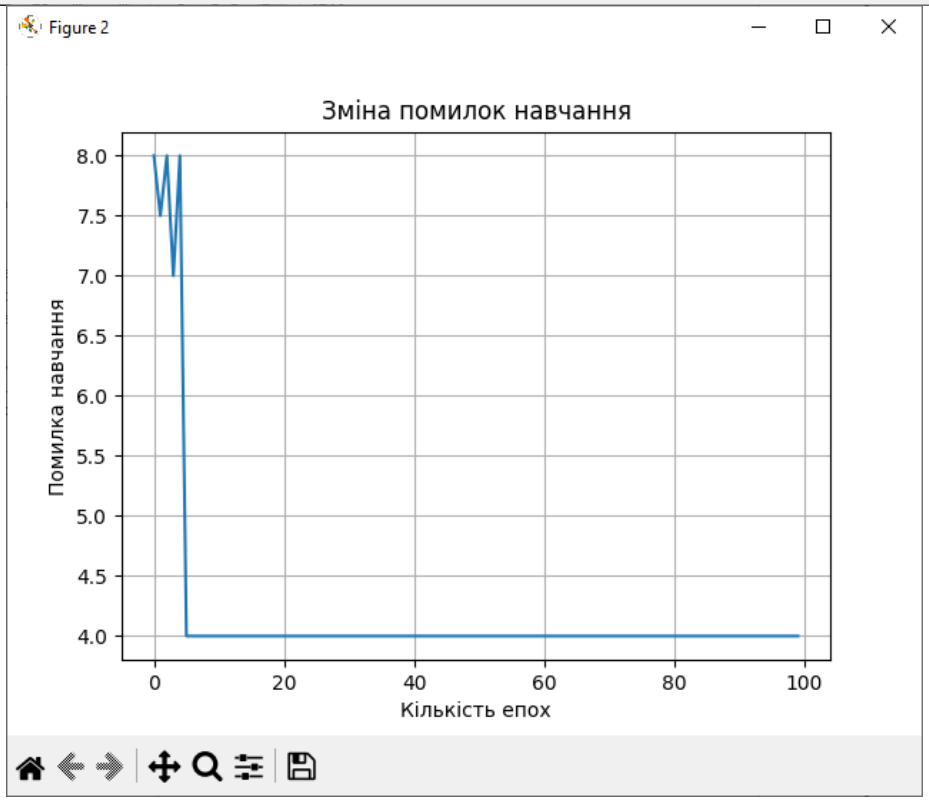


Figure 2
Зміна помилок навчання
Помилка навчання
Кількість епох
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
0
20
40
60
80
100

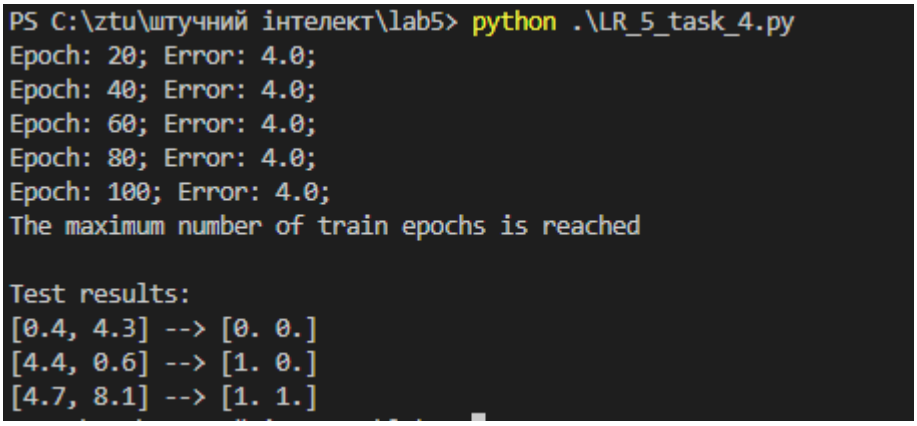
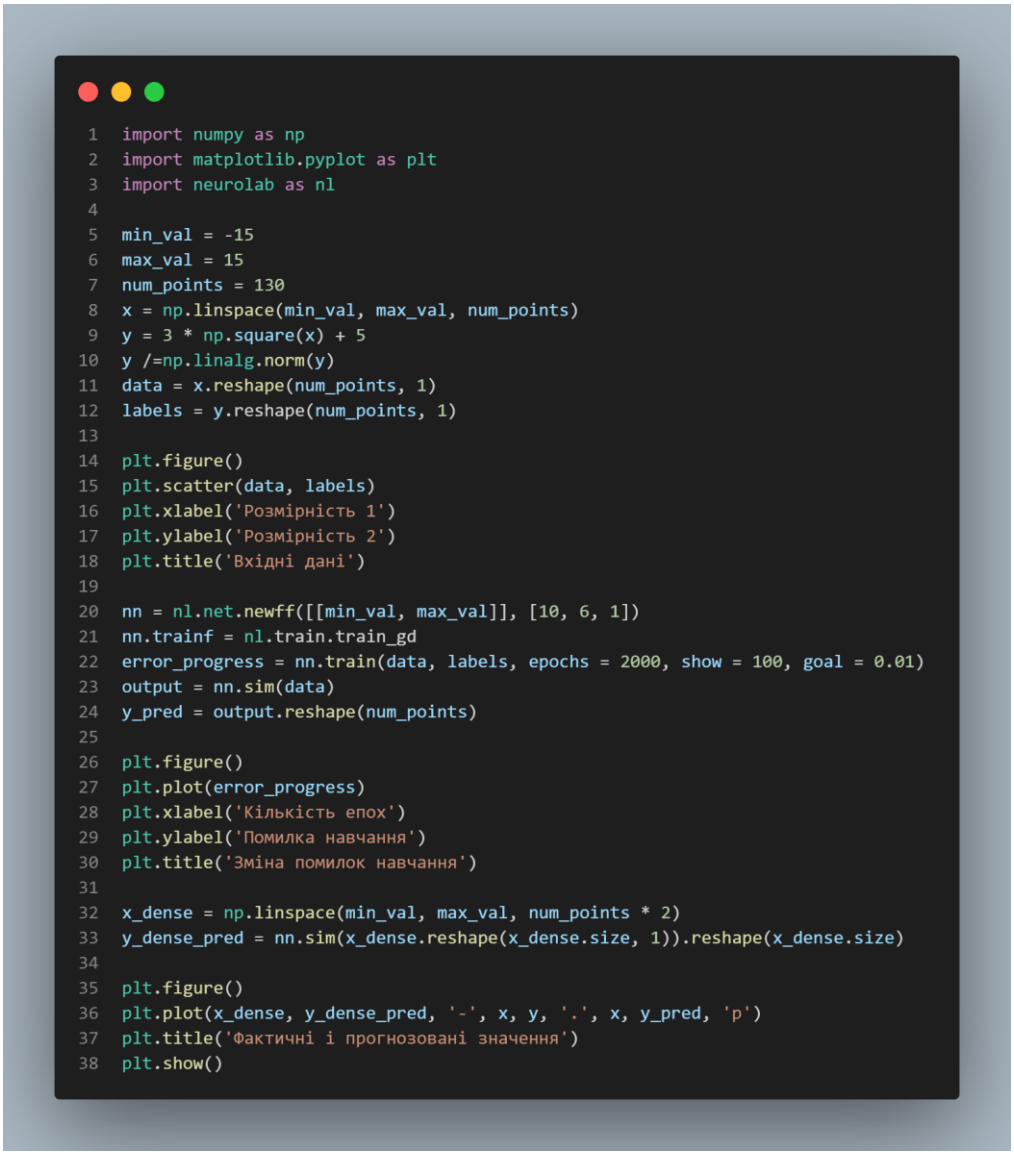


Рис 5.10 Результат файлу LR_5_task_4.py

Висновок: На рис. 20 зображено процес навчання мережі. На 20 епосі відбулось 4 помилки, аналогічно на 40, 60, 80 та 100. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування. Ми вирішили визначити вибіркові тестові точки даних та запустили для них нейронну мережу. І це його результат.

Завдання 2.5. Побудова багатошарової нейронної мережі



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

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)
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)

plt.figure()
plt.scatter(data, labels)
plt.xlabel('Розмірність 1')
plt.ylabel('Розмірність 2')
plt.title('Вхідні дані')

nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])
nn.trainf = nl.train.train_gd
error_progress = nn.train(data, labels, epochs = 2000, show = 100, goal = 0.01)
output = nn.sim(data)
y_pred = output.reshape(num_points)

plt.figure()
plt.plot(error_progress)
plt.xlabel('Кількість епох')
plt.ylabel('Помилка навчання')
plt.title('Зміна помилок навчання')

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('Фактичні і прогнозовані значення')
plt.show()
```

Рис 5.11 Код файлу LR_5_task_5.py

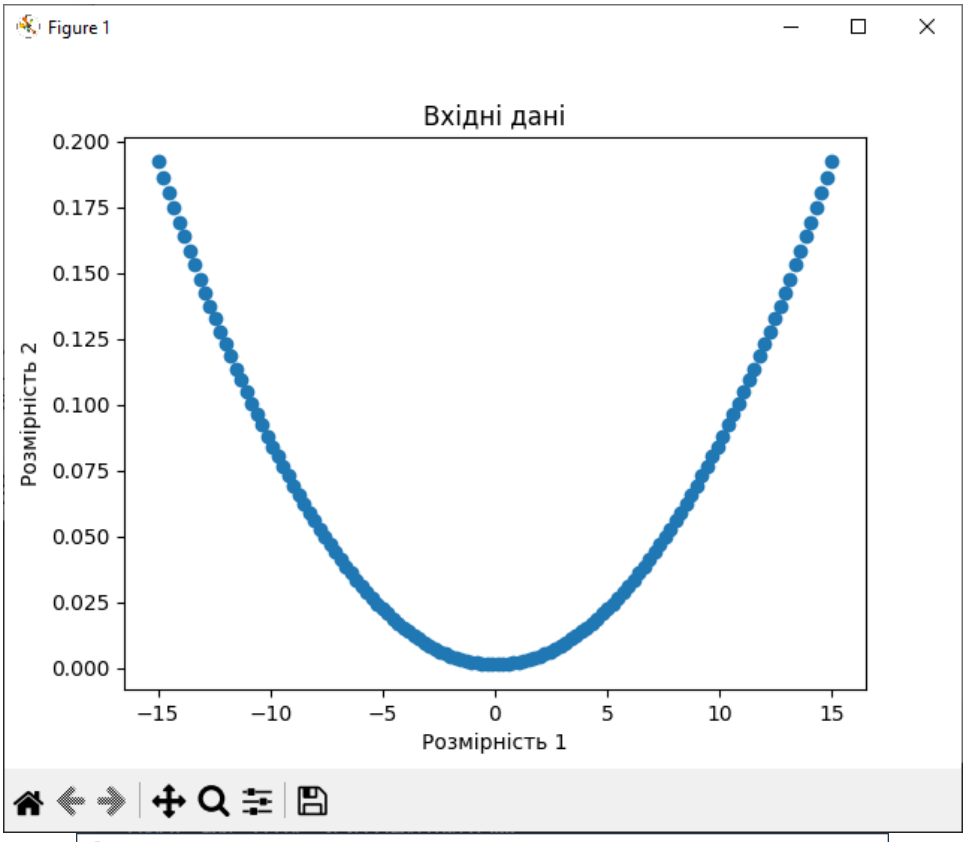


Figure 1
Вхідні дані
Розмірність 2
Розмірність 1
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−15
−10
−5
0
5
10
15

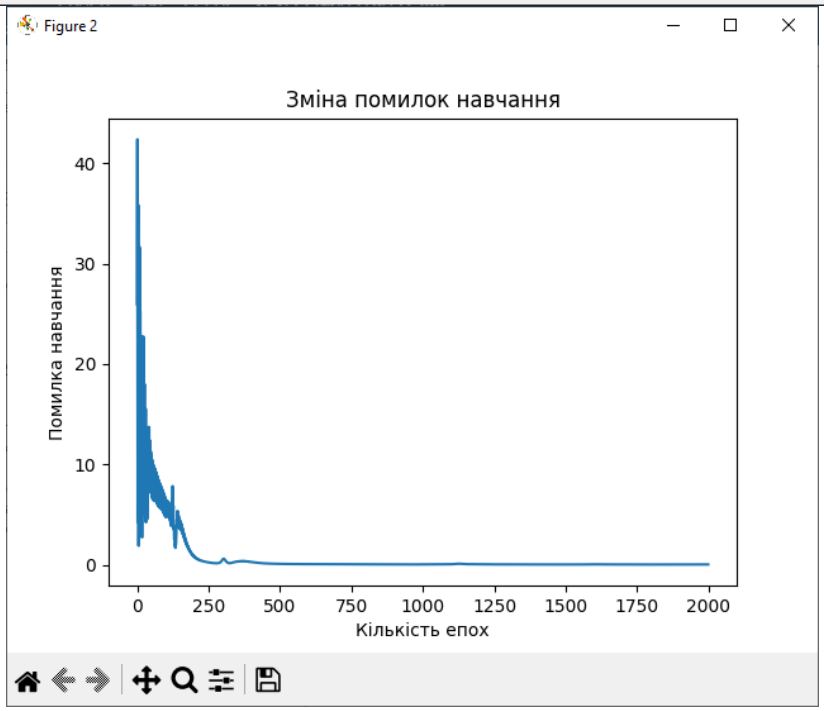


Figure 2
Зміна помилок навчання
Помилка навчання
Кількість епох
40
30
20
10
0
0
250
500
750
1000
1250
1500
1750
2000

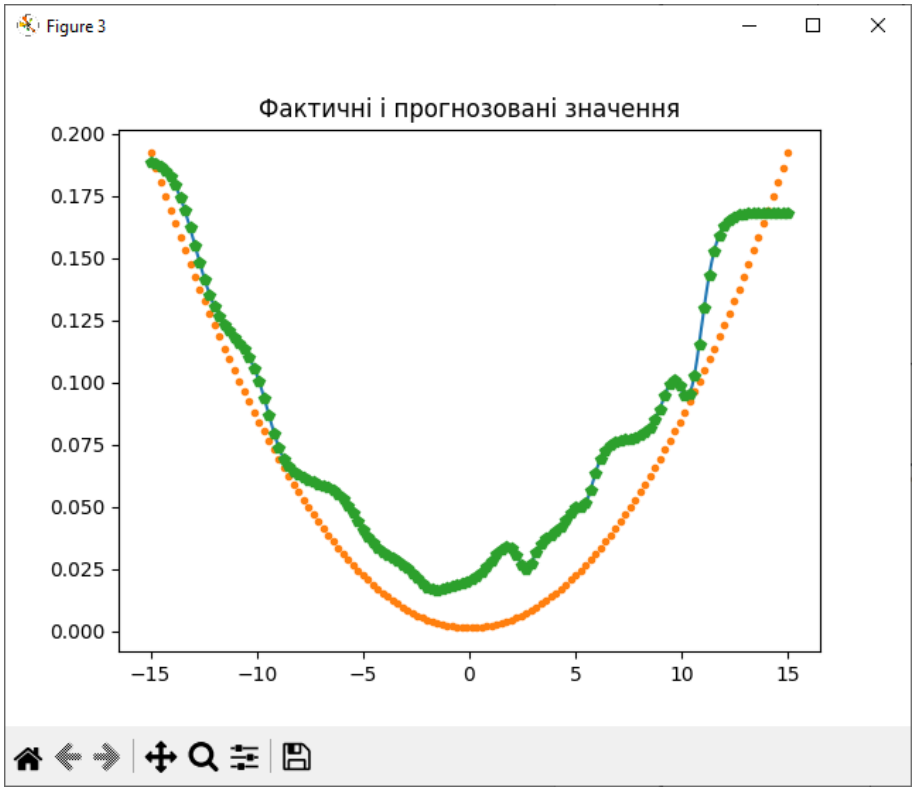
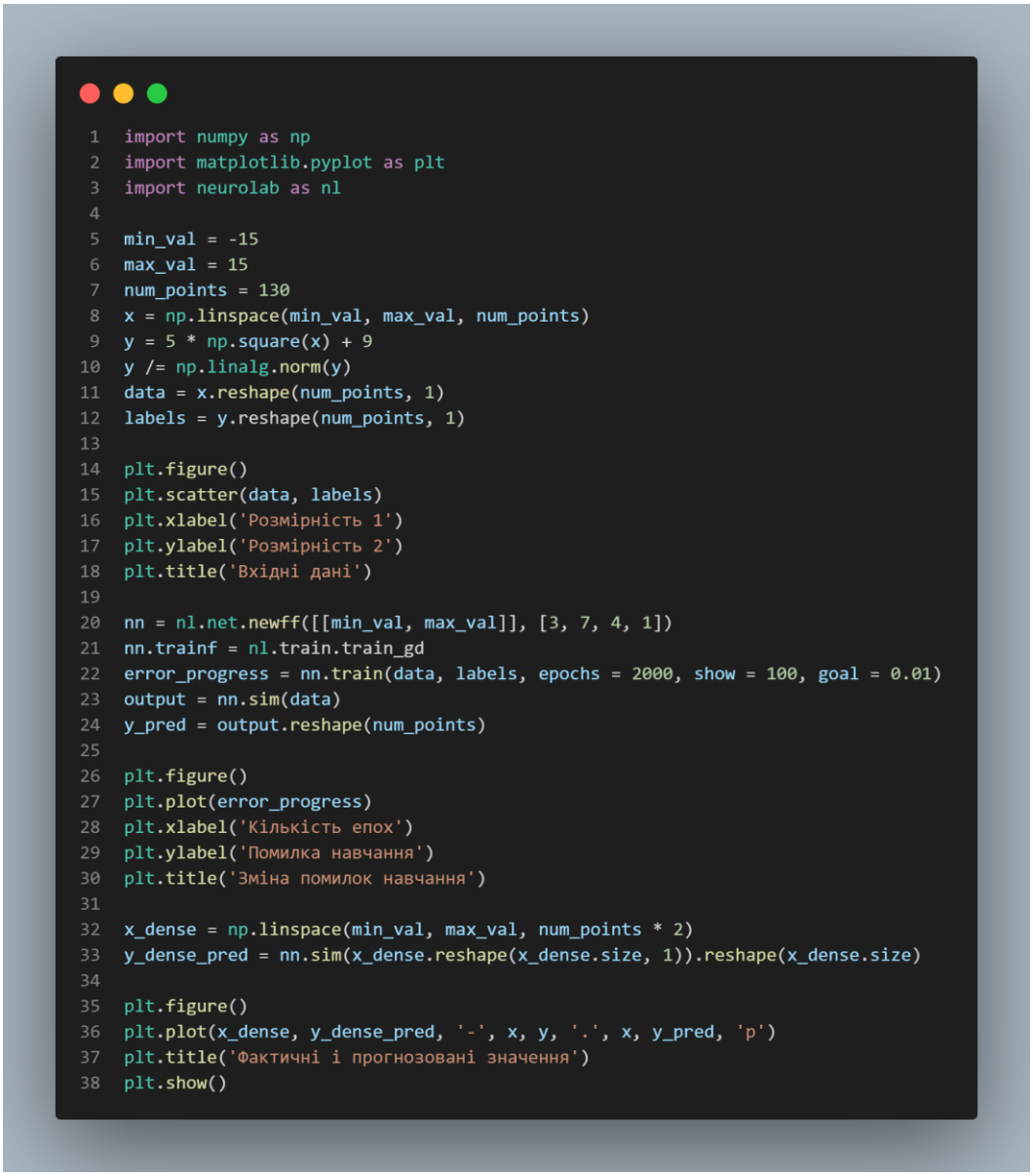


Рис 5.12 Результат файлу LR_5_task_5.py

Висновок: На рис. 12 зображено процес навчання мережі. Відносно кожної епосі відбувались помилки. На 100 0.83 помилки. На 1200 0.06. Потім вивелось повідомлення, що ми досягли цілі навчання.

Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

Варіант 18		$y = 5x^2+9$
18	3	7-4-1



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

min_val = -15
max_val = 15
num_points = 130
x = np.linspace(min_val, max_val, num_points)
y = 5 * np.square(x) + 9
y /= np.linalg.norm(y)
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)

plt.figure()
plt.scatter(data, labels)
plt.xlabel('Розмірність 1')
plt.ylabel('Розмірність 2')
plt.title('Вхідні дані')

nn = nl.net.newff([[min_val, max_val]], [3, 7, 4, 1])
nn.trainf = nl.train.train_gd
error_progress = nn.train(data, labels, epochs = 2000, show = 100, goal = 0.01)
output = nn.sim(data)
y_pred = output.reshape(num_points)

plt.figure()
plt.plot(error_progress)
plt.xlabel('Кількість епох')
plt.ylabel('Помилка навчання')
plt.title('Зміна помилок навчання')

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('Фактичні і прогнозовані значення')
plt.show()
```

Рис 5.13 Код файлу LR_5_task_6.py

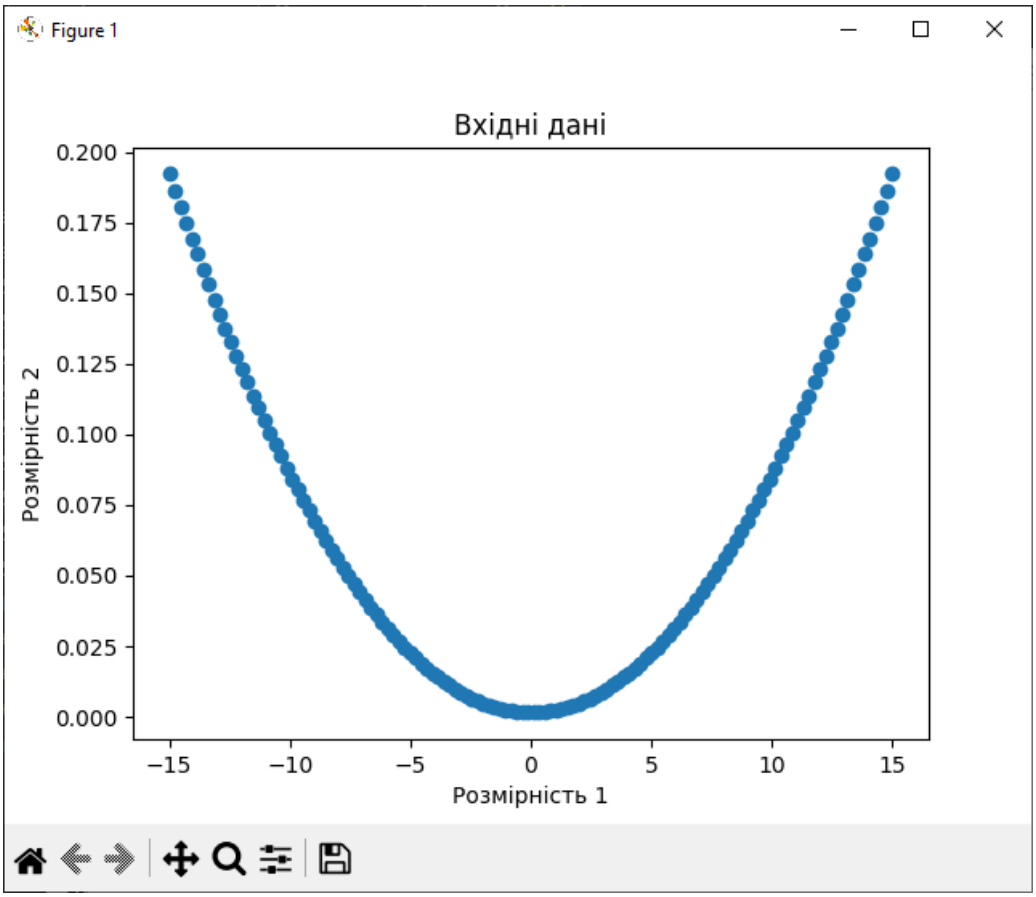
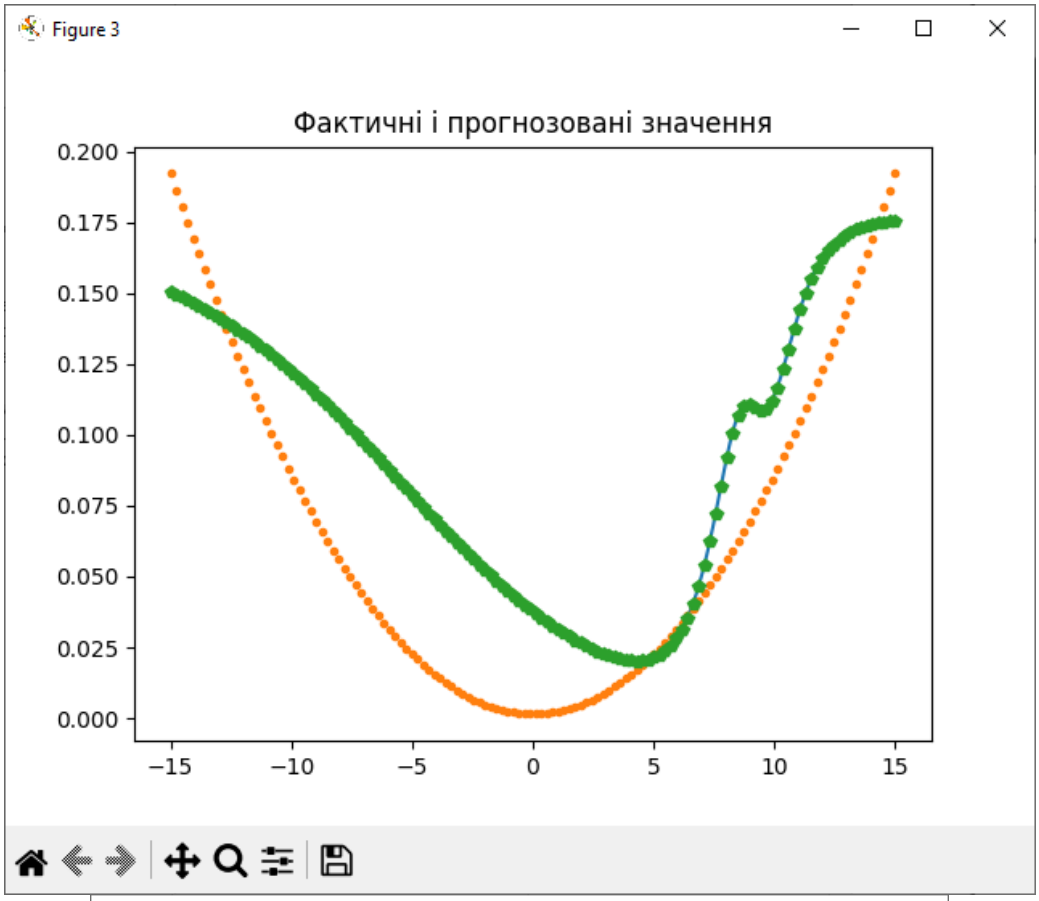


Figure 1
Вхідні дані
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Розмірність 2
−15
−10
−5
0
5
10
15
Розмірність 1



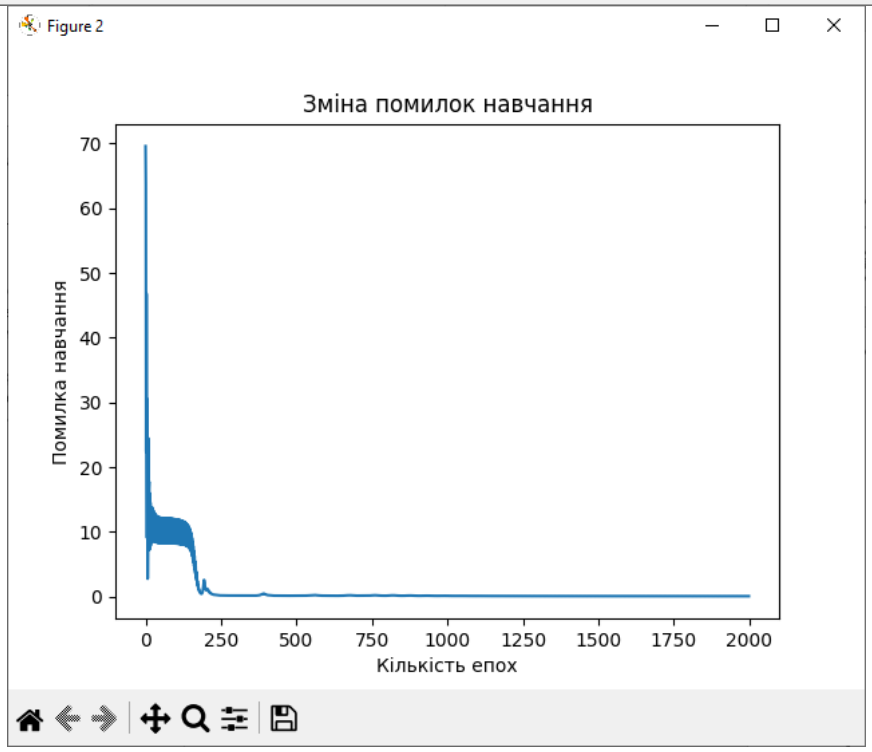


Рис 5.14 Результат файлу LR_5_task_6.py

Завдання 2.7. Побудова нейронної мережі на основі карти Кохонена, що самоорганізується

```python
import numpy as np
import neurolab as nl
import numpy.random as rand
import pylab as pl

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 = 100)

# Plot results:
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()
```

Рис 5.15 Код файлу LR_5_task_7.py

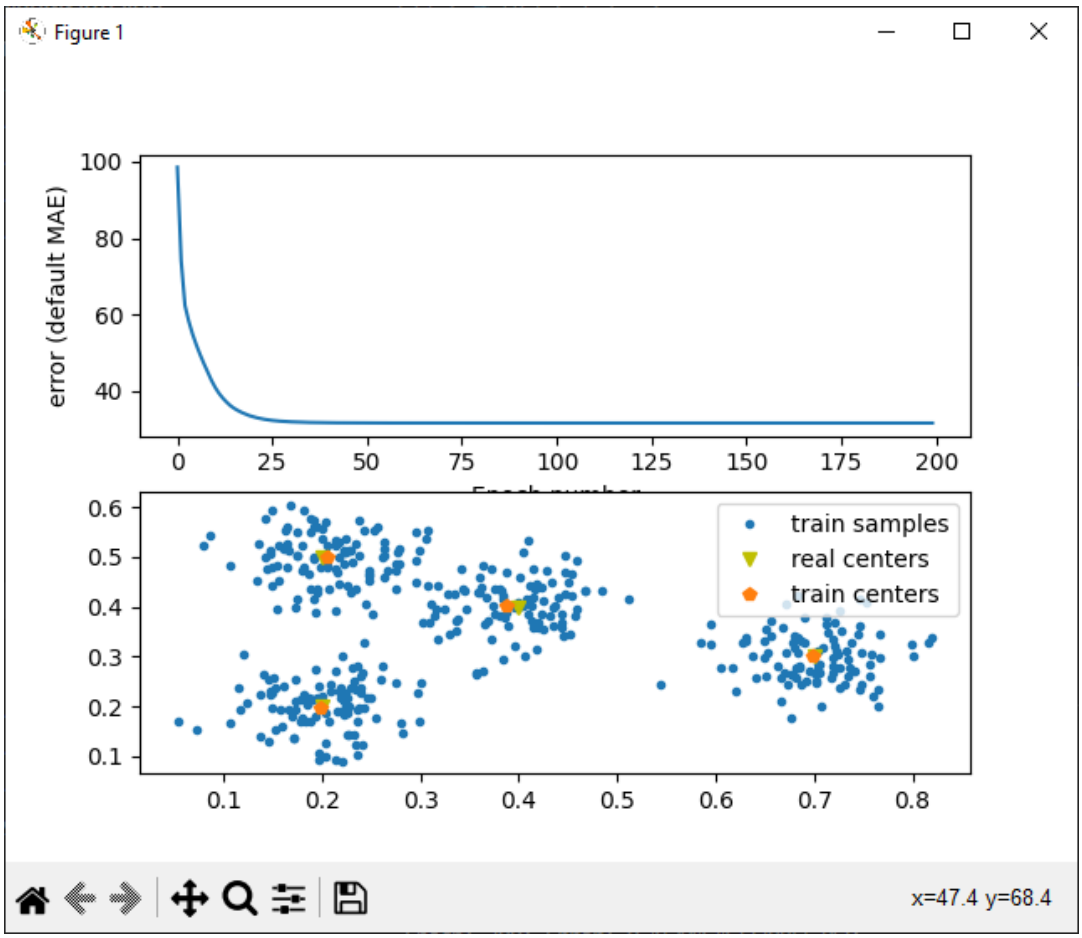


Рис 5.16 Результат файлу LR_5_task_7.py

Помилка MAE - Средня абсолютна помилка (Mean Absolute Error). Середньою абсолютною похибкою називають середнє арифметичне з абсолютних похибок усіх вимірювань.

Завдання 2.8. Дослідження нейронної мережі на основі карти Кохонена, що самоорганізується

Варіант 18	[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,06

```python
import numpy as np
import neurolab as nl
import numpy.random as rand
import pylab as pl

skv = 0.06
centr = np.array([[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]])
rand_norm = skv * rand.randn(100, 5, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 5, 2)

rand.shuffle(inp)

# Create net with 2 inputs and 5 neurons
net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 5)

# train with rule: Conscience Winner Take All algoritm (CWTA)
error = net.train(inp, epochs=200,  show=20)

# Plot results:
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()
```

Рис 5.17 Код файлу LR_5_task_8.py

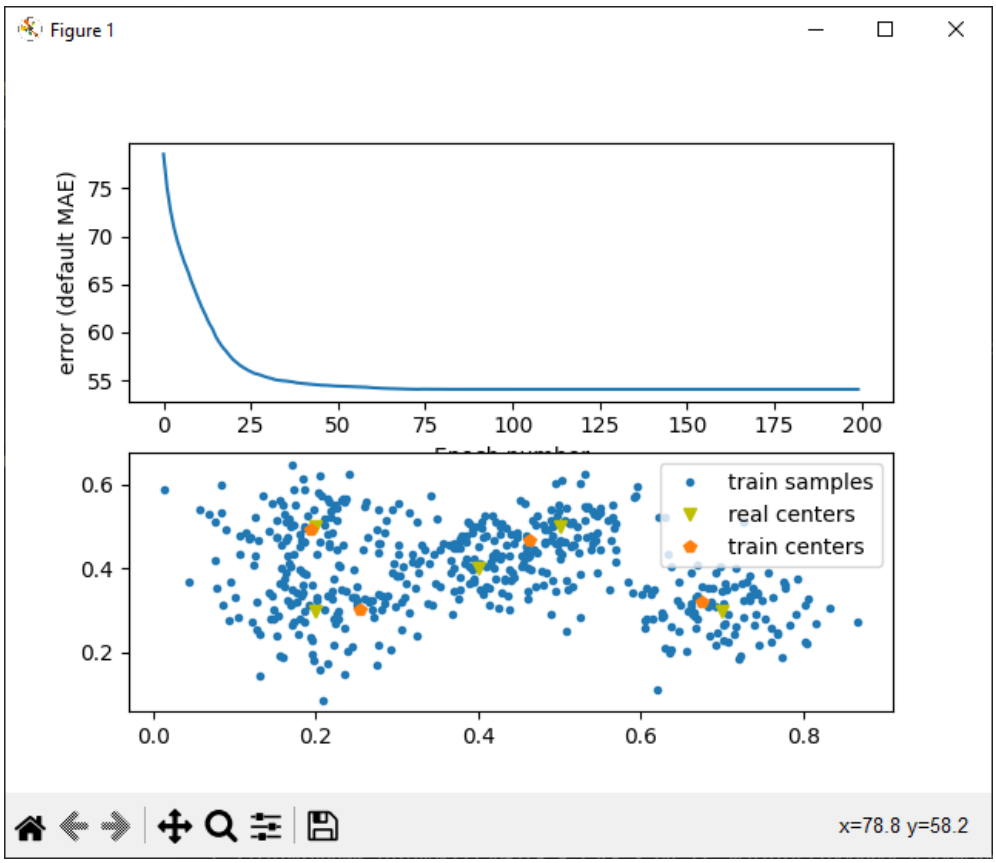


Рис 5.18 Результат файлу LR_5_task_8.py з чотирьма нейронами

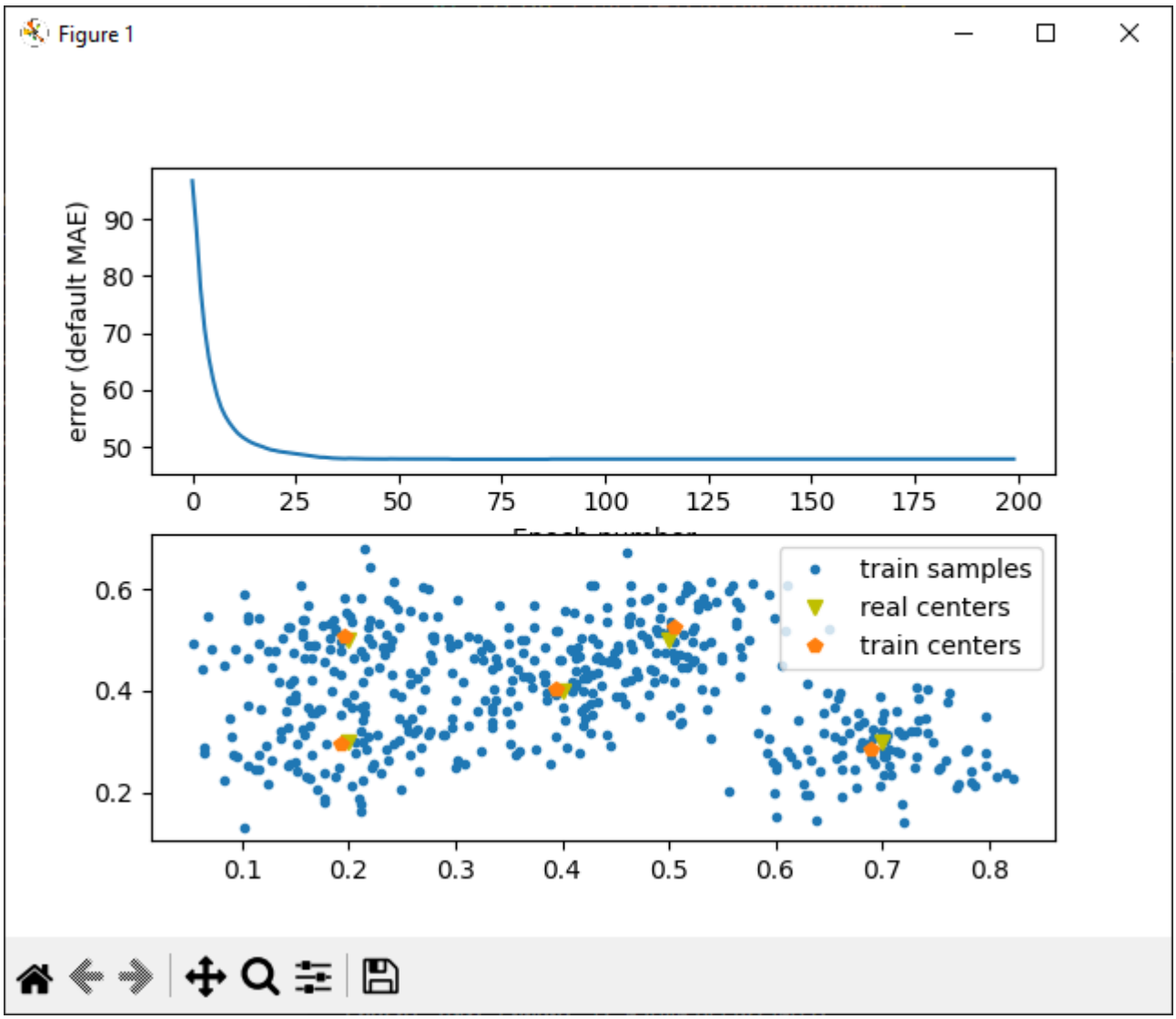
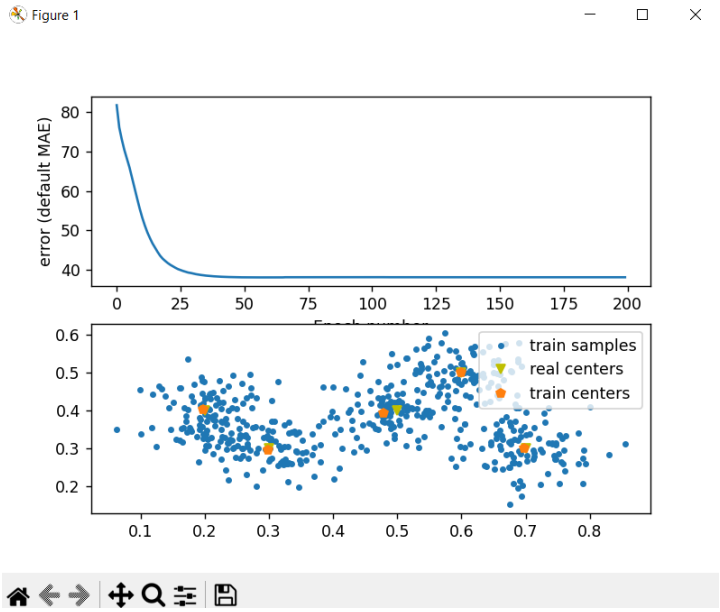


Рис 5.19 Результат файлу LR_5_task_8.py з 5-ма нейронами

На рис. 18 зображено процес навчання мережі. На 20 епосі відбулось 44.36 помилки, помилки і так далі, на 200 епосі відбулось 39.01 помилки,. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

```
# Create net with 2 inputs and 5 neurons
net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 5)
```




```
Epoch: 20; Error: 42.250786824072875;
Epoch: 40; Error: 38.31151846004284;
Epoch: 60; Error: 38.07749387644577;
Epoch: 80; Error: 38.125049360270694;
Epoch: 100; Error: 38.13050556364453;
Epoch: 120; Error: 38.11429249868449;
Epoch: 140; Error: 38.11198400241311;
Epoch: 160; Error: 38.11159032438987;
Epoch: 180; Error: 38.11152262192267;
Epoch: 200; Error: 38.111510231115986;
The maximum number of train epochs is reached
```

Рис 5.19 Результат файлу LR_5_task_8.py

Якщо порівнювати нейронну мережу Кохонена з 4 нейронами та 5 нейронами, можна зробити такі висновки. При 4 нейронах Помилка MAE повільніше зменшується, ніж з 5 нейронами, також з 5 нейронами ця помилка нижча. З 5 нейронами обоє центрів збігаються майже в одні точці. Число нейронів в шарі Кохонена має відповідати числу класів вхідних сигналів. Тобто в нашому випадку нам давалось 5 вхідних сигналів, значить у нас має бути 5 нейронів, а не 4.Отже, невірний вибір кількості нейронів числу кластерів впливає на величину помилки ускладнюючи навчання мережі і швидкості, тому на рис. 18 набагато гірші результати, ніж на рис. 19.

Висновок: на лабораторній роботі я навчився використовувати спеціалізовані бібліотеки та мову програмування Python, навчився створювати та застосовувати прості нейронні мережі**.**