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. Створити простий нейрон

Код скрипту LR_5_task_1.py:

```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):
        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))
```

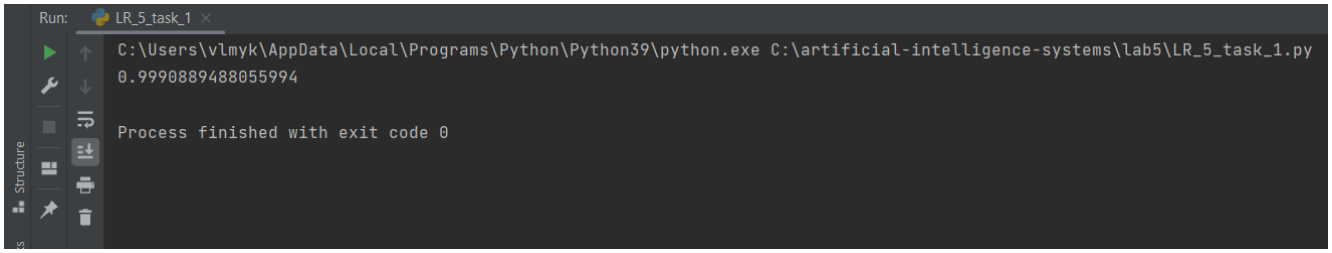

Рис. 1. Результат роботи скрипта LR_5_task_1.py

Завдання 2. Створити просту нейронну мережу для передбачення статі людини

Код скрипту LR_5_task_2.py:

```
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 MykoliukNeuralNetwork:

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

```
C:\Users\vlmyk\AppData\Local\Programs\Python\Python39\python.exe C:\artificial-intelligence-systems\lab5\LR_5_task_2.py
0.7216325609518421

Process finished with exit code 0
```

Рис. 2. Результат роботи скрипту LR_5_task_2.py

Код скрипту LR_5_task_2_1.py

```
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 MykoliukNeuralNetwork:

    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))


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
])

network = MykoliukNeuralNetwork()
network.train(data, all_y_trues)
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
```

```
C:\Users\vlmyk\AppData\Local\Programs\Python\Python39\python.exe C:\artificial-intelligence-systems\lab5\LR_5_task_2_1.py
Epoch 0 loss: 0.141
Epoch 10 loss: 0.109
Epoch 20 loss: 0.087
Epoch 30 loss: 0.071
Epoch 40 loss: 0.059
Epoch 50 loss: 0.050
Epoch 60 loss: 0.044
Epoch 70 loss: 0.038
Epoch 80 loss: 0.034
Epoch 90 loss: 0.030
Epoch 100 loss: 0.027
Epoch 110 loss: 0.025
Epoch 120 loss: 0.023
Epoch 130 loss: 0.021
Epoch 140 loss: 0.019
Epoch 150 loss: 0.018
Epoch 160 loss: 0.017
Epoch 170 loss: 0.016
Epoch 180 loss: 0.015
Epoch 190 loss: 0.014
Epoch 200 loss: 0.013
Epoch 210 loss: 0.013
Epoch 220 loss: 0.012
Epoch 230 loss: 0.011
Epoch 240 loss: 0.011
Epoch 250 loss: 0.010
Epoch 260 loss: 0.010
Epoch 270 loss: 0.010
Epoch 280 loss: 0.009
```

```
Epoch 290 loss: 0.009
Epoch 300 loss: 0.009
Epoch 310 loss: 0.008
Epoch 320 loss: 0.008
Epoch 330 loss: 0.008
Epoch 340 loss: 0.007
Epoch 350 loss: 0.007
Epoch 360 loss: 0.007
Epoch 370 loss: 0.007
Epoch 380 loss: 0.007
Epoch 390 loss: 0.006
Epoch 400 loss: 0.006
Epoch 410 loss: 0.006
Epoch 420 loss: 0.006
Epoch 430 loss: 0.006
Epoch 440 loss: 0.006
Epoch 450 loss: 0.005
Epoch 460 loss: 0.005
Epoch 470 loss: 0.005
Epoch 480 loss: 0.005
Epoch 490 loss: 0.005
Epoch 500 loss: 0.005
Epoch 510 loss: 0.005
Epoch 520 loss: 0.005
Epoch 530 loss: 0.005
Epoch 540 loss: 0.004
Epoch 550 loss: 0.004
Epoch 560 loss: 0.004
Epoch 570 loss: 0.004
```

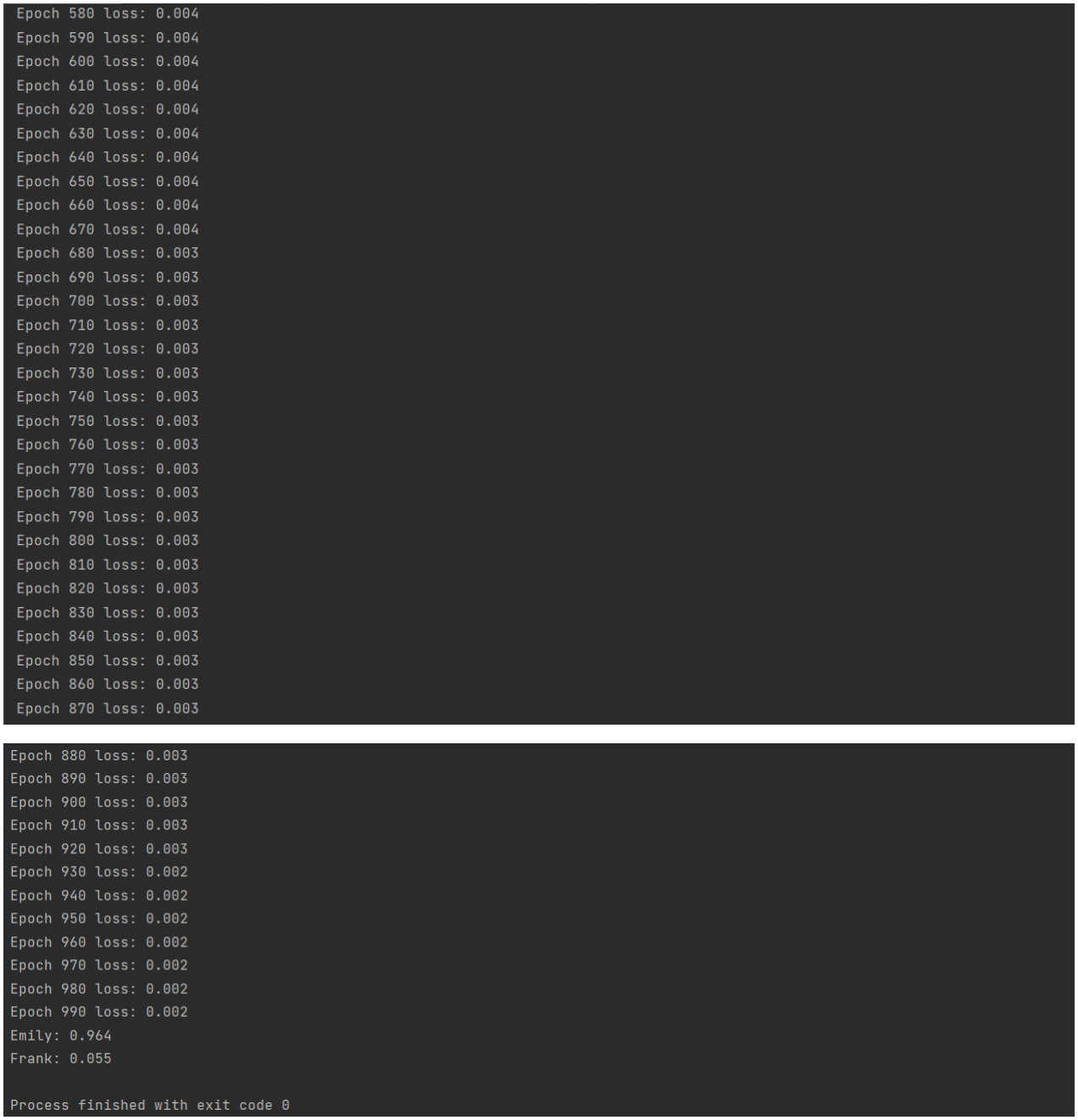

Рис. 3-6. Результат виконання скрипта LR_5_task_2_1.py

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 3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

Код скрипту LR_5_task_3.py:

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

text = np.loadtxt('data_perceptron.txt')
data = text[:, :2]
labels = text[:, 2].reshape((text.shape[0], 1))
plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Розмірність 1')
plt.ylabel('Розмірність 2')
plt.title('Вхідні дані')
dim1_min, dim1_max, dim2_min, dim2_max = 0, 1, 0, 1
num_output = labels.shape[1]
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
perceptron = nl.net.newp([dim1, dim2], num_output)
error_progress = perceptron.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()
```

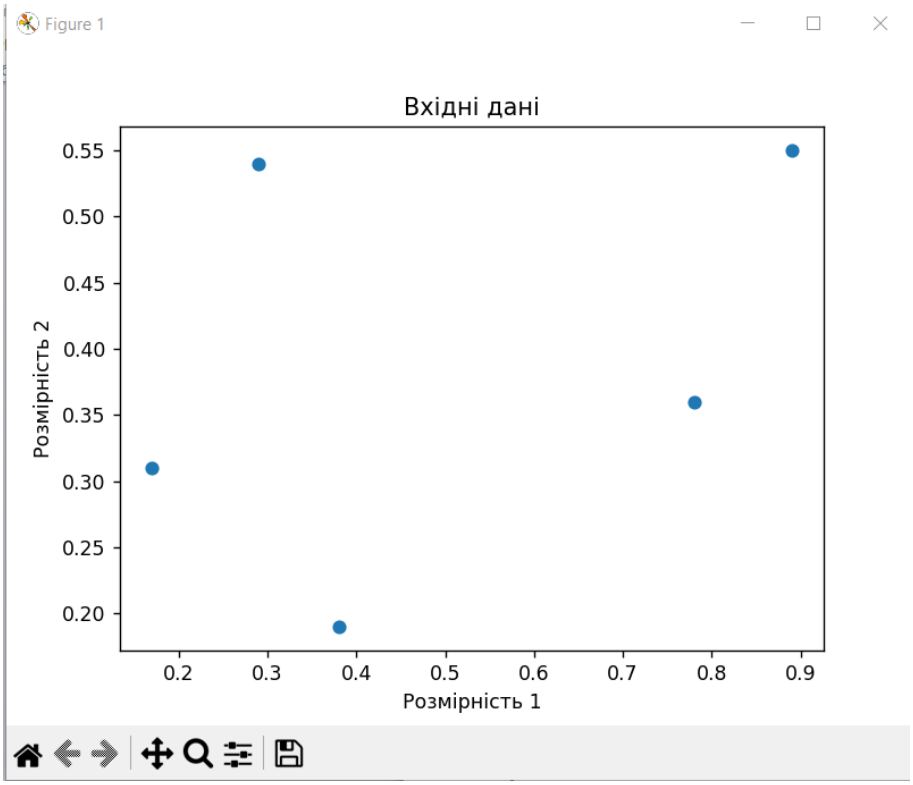

Рис. 7. Результат виконання скрипту LR_5_task_3.py

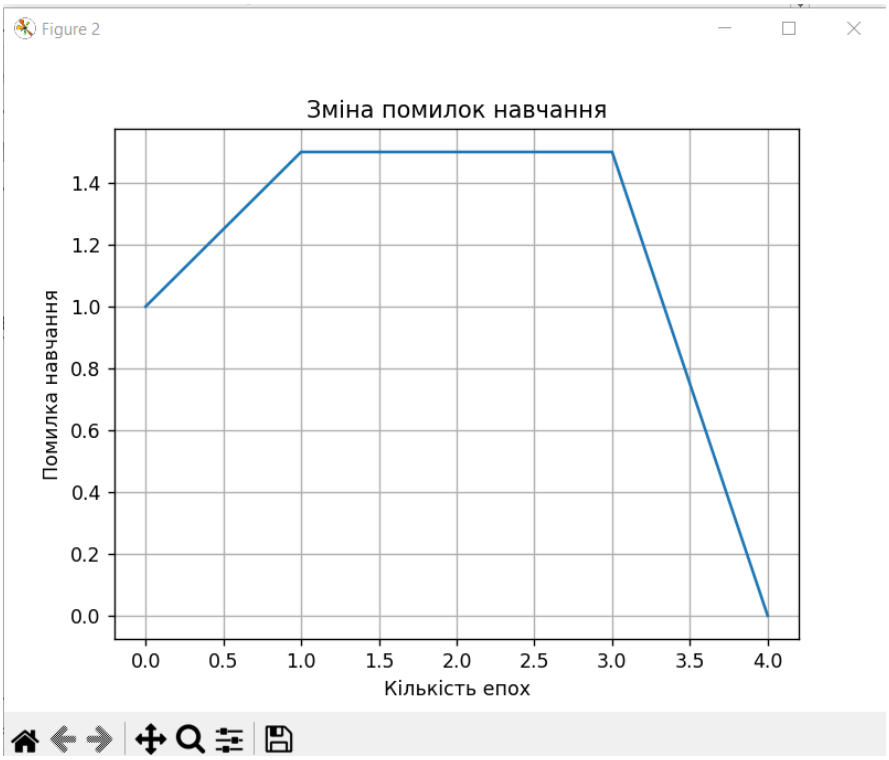

Рис. 8. Результат виконання скрипту LR_5_task_3.py

Завдання 4. Побудова одношарової нейронної мережі

Код скрипту LR_5_task_4.py:

```
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])
```

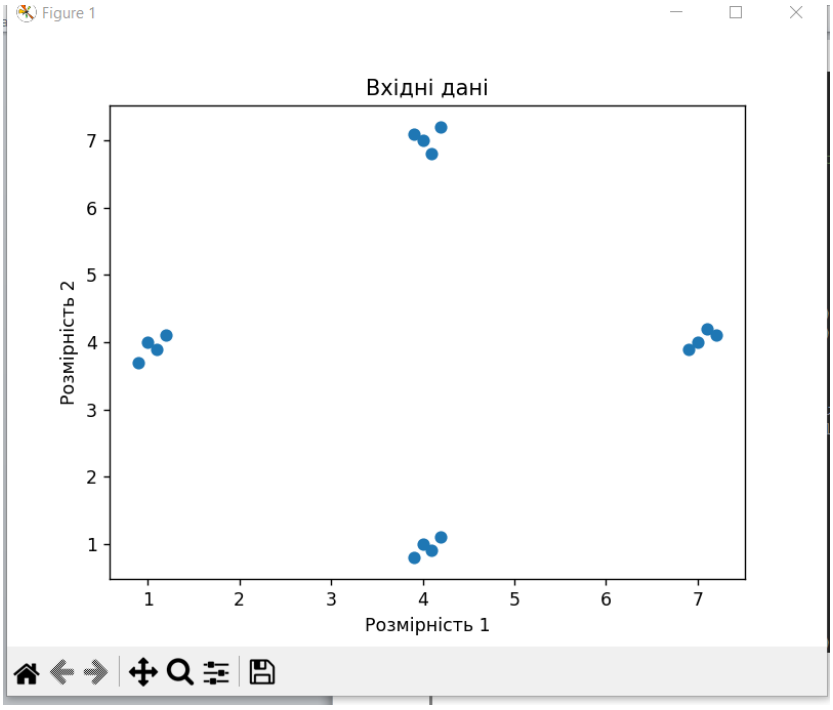

Рис. 9. Результат виконання скрипта LR_5_task_4.py

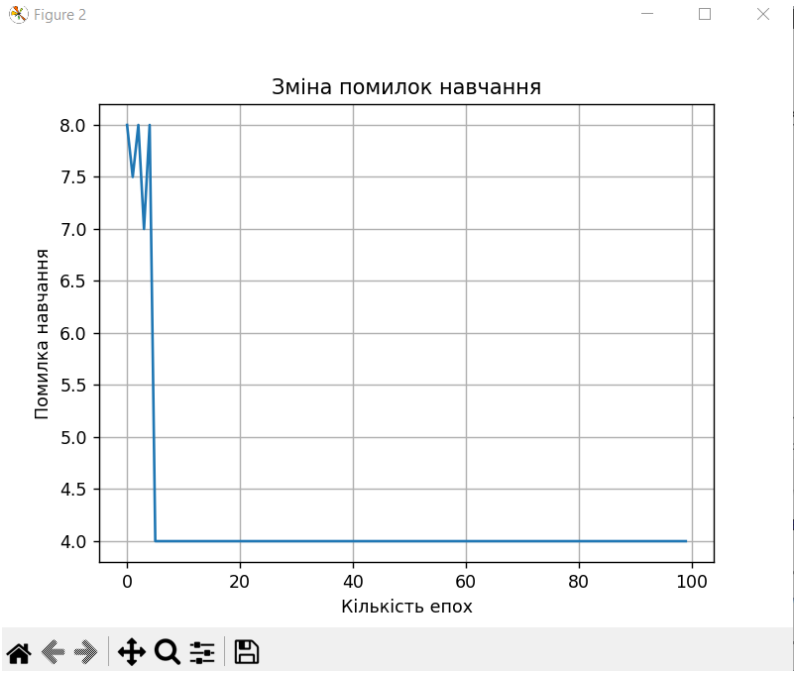

Рис. 10. Результат виконання скрипта LR_5_task_4.py

```
C:\Users\vlmyk\AppData\Local\Programs\Python\Python39\python.exe C:\artificial-intelligence-systems\lab5\LR_5_task_4.py
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.]

Process finished with exit code 0
```

Рис. 11. Результат виконання скрипта LR_5_task_4.py

Висновок: На рисунку 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 5. Побудова багатошарової нейронної мережі

Код скрипту LR_5_task_5.py:

```
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()
```

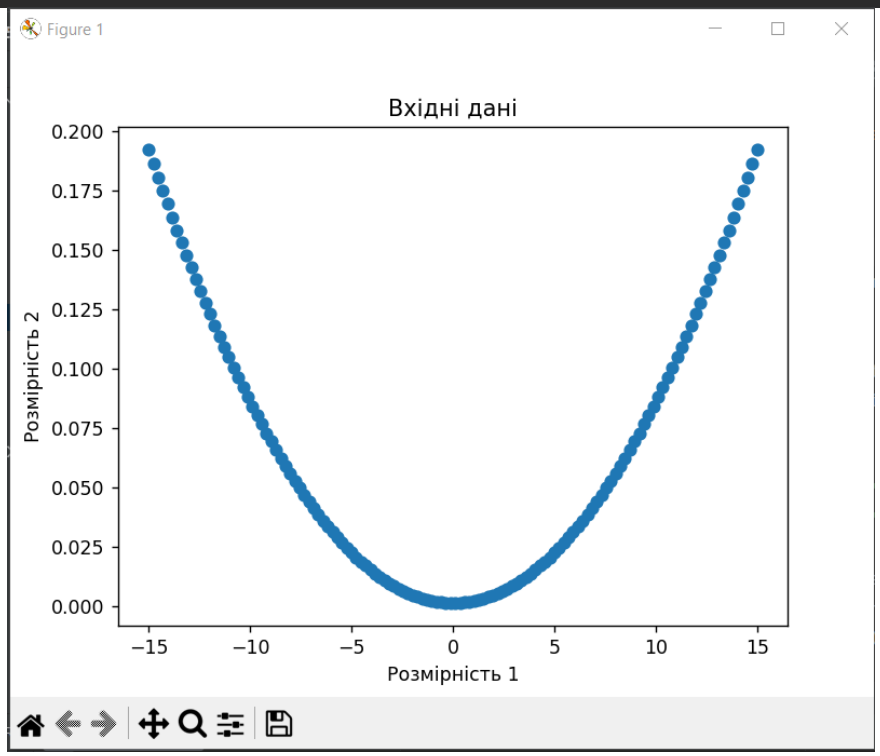

Рис. 12. Результат виконання скрипта LR_5_task_5.py

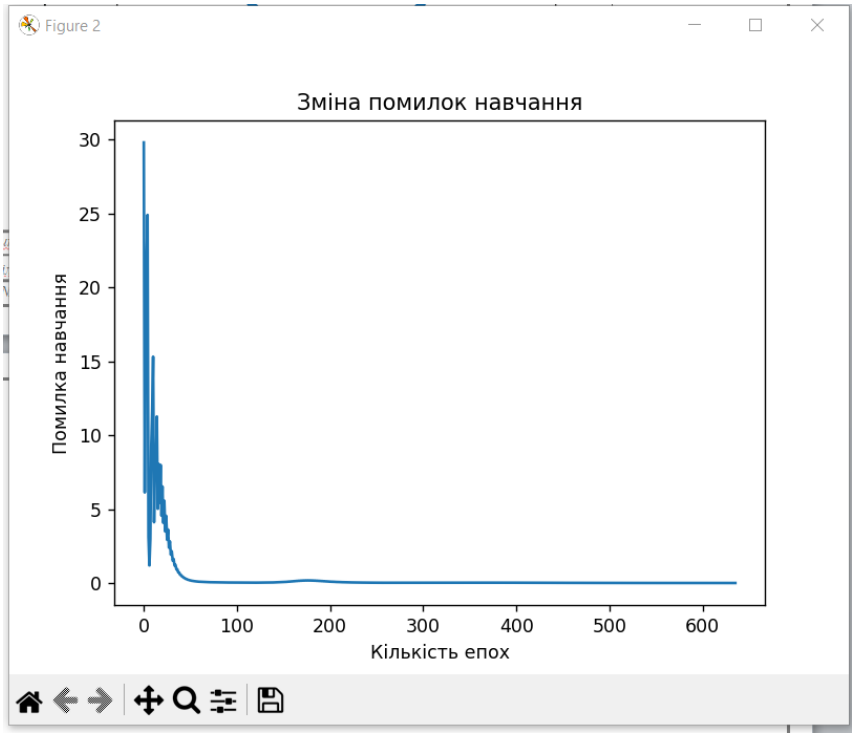

Рис. 13. Результат виконання скрипта LR_5_task_5.py

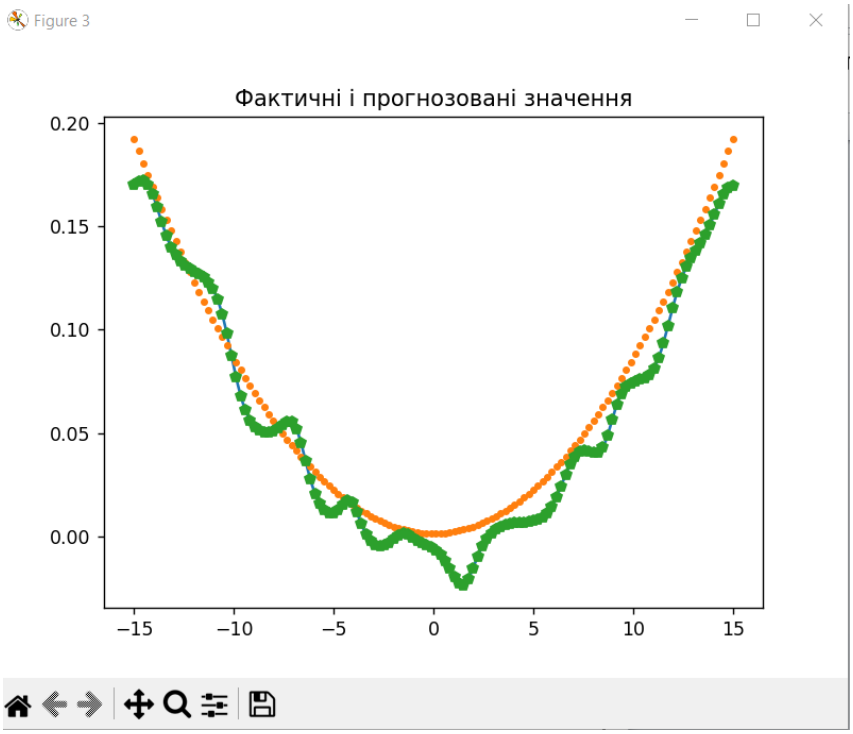

Рис. 14. Результат виконання скрипта LR_5_task_5.py

```
C:\Users\vlmyk\AppData\Local\Programs\Python\Python39\python.exe C:\artificial-intelligence-systems\lab5\LR_5_task_5.py
Epoch: 100; Error: 0.03846723673541512;
Epoch: 200; Error: 0.1053344610500729;
Epoch: 300; Error: 0.024888471612652327;
Epoch: 400; Error: 0.030033852672106873;
Epoch: 500; Error: 0.01506208528458021;
Epoch: 600; Error: 0.010383489158194436;
The goal of learning is reached

Process finished with exit code 0
```

Рис. 15. Результат виконання скрипта LR_5_task_5.py

Висновок: На рис. 15 зображено процес навчання мережі. Відносно кожної епосі відбувались помилки. На 100 0.03 помилки. На 600 0.01. Потім вивелось повідомлення, що ми досягли цілі навчання.

Завдання 6. Побудова багатошарової нейронної мережі для свого варіанту

№ варіанта	Тестові дані
Варіант 1	$y = 2x^2+5$
Варіант 2	$y = 2x^2+6$
Варіант 3	$y = 2x^2+7$
Варіант 4	$y = 2x^2+8$
Варіант 5	$y = 2x^2+9$
Варіант 6	$y = 2x^2+10$
Варіант 7	$y = 3x^2+7$
Варіант 8	$y = 3x^2+8$
Варіант 9	$y = 3x^2+9$
Варіант 10	$y = 5x^2+1$
Варіант 11	$y = 5x^2+2$
Варіант 12	$y = 5x^2+3$
Варіант 13	$y = 5x^2+4$
Варіант 14	$y = 5x^2+5$
Варіант 15	$y = 5x^2+6$

Рис. 16. Варіант тестових даних

Номер варіанта	Багатошаровий персептрон	
	Кількість шарів	Кількості нейронів у шарах
1	2	3-1
2	2	2-1
3	3	3-3-1
4	2	5-1
5	3	2-2-1
6	2	10-1
7	2	5-1
8	3	5-5-1
9	3	3-5-1
10	2	4-1

Рис. 17. Параметри багатошарової мережі за варіантом

Код скрипту LR_5_task_6.py:

```
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 * x * x + 1
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]], [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()
```

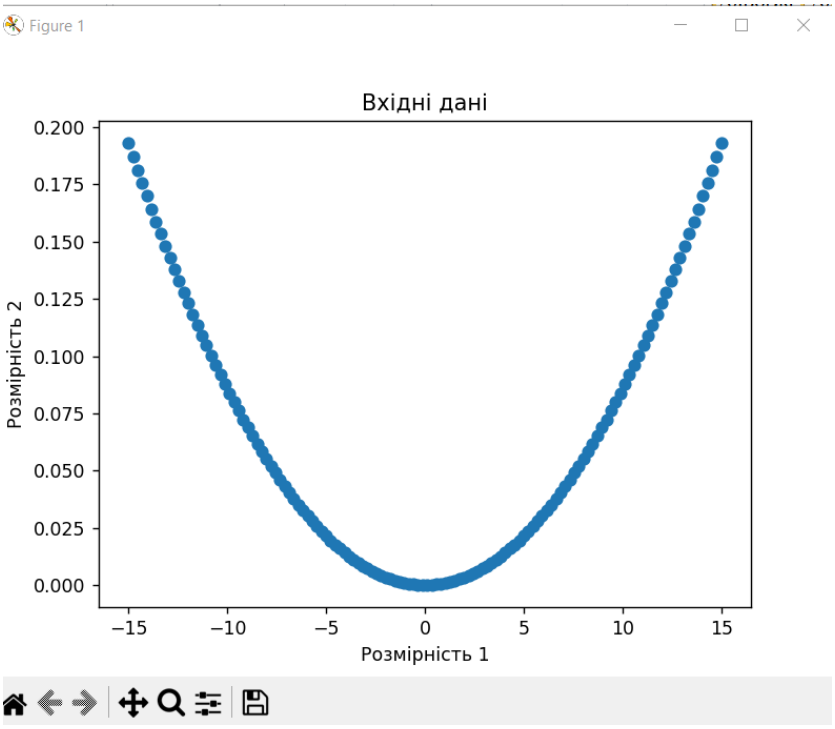

Рис. 18. Результат виконання скрипта LR_5_task_6.py

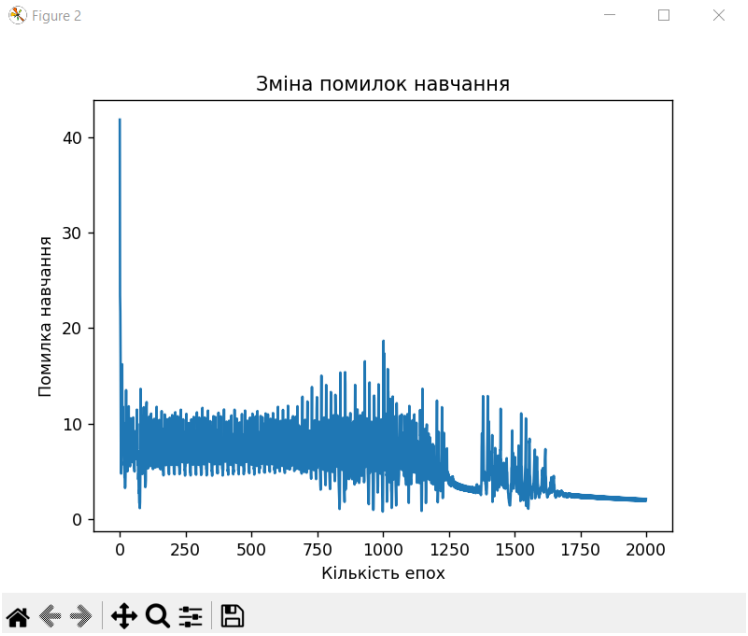

Рис. 19. Результат виконання скрипта LR_5_task_6.py

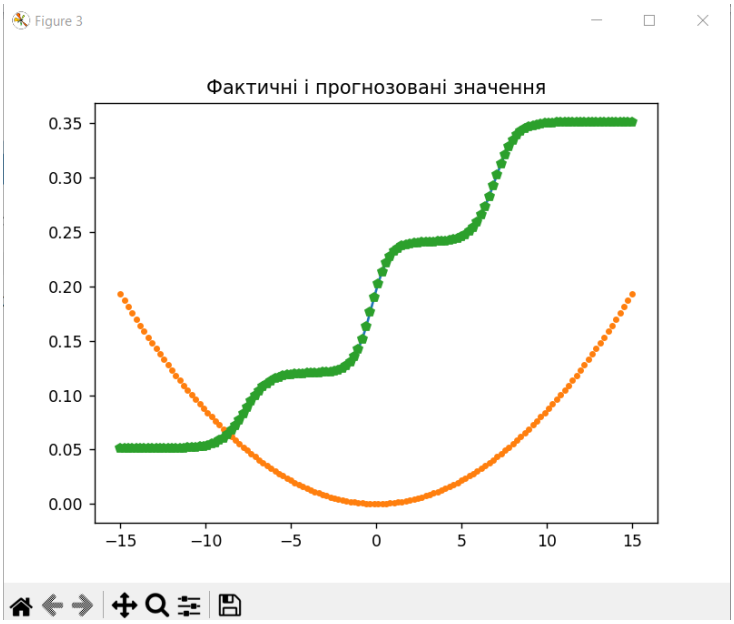

Рис. 20. Результат виконання скрипта LR_5_task_6.py

```
C:\Users\vlmyk\AppData\Local\Programs\Python\Python39\python.exe C:\artificial-intelligence-systems\lab5\LR_5_task_6.py
Epoch: 100; Error: 4.090388271400568;
Epoch: 200; Error: 10.892022333097374;
Epoch: 300; Error: 5.67191402339862;
Epoch: 400; Error: 6.7191592957110835;
Epoch: 500; Error: 6.532917229220255;
Epoch: 600; Error: 8.535901301524115;
Epoch: 700; Error: 5.26387464554297;
Epoch: 800; Error: 3.1363227244712704;
Epoch: 900; Error: 4.236850608085568;
Epoch: 1000; Error: 0.7698815083956132;
Epoch: 1100; Error: 4.357068067604641;
Epoch: 1200; Error: 3.4925038798776202;
Epoch: 1300; Error: 3.5673437375446424;
Epoch: 1400; Error: 5.403933593645712;
Epoch: 1500; Error: 5.741926276637583;
Epoch: 1600; Error: 3.292504659523393;
Epoch: 1700; Error: 2.6225332680672655;
Epoch: 1800; Error: 2.436157074702584;
Epoch: 1900; Error: 2.2519921176992;
Epoch: 2000; Error: 2.09699480658224;
The maximum number of train epochs is reached

Process finished with exit code 0
```

Рис. 20. Результат виконання скрипта LR_5_task_6.py

Висновок: На рис. 20 зображено процес навчання мережі. На 100 епосі відбулось 4.09 помилки, на 200 епосі відбулось 10.89 помилки, і так далі, на 2000 епосі відбулось 2.09 помилки,. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

Завдання 7. Побудова нейронної мережі на основі карти Кохонена, що самоорганізується

Код скрипту LR_5_task_7.py:

```python
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=100)

# 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()
```

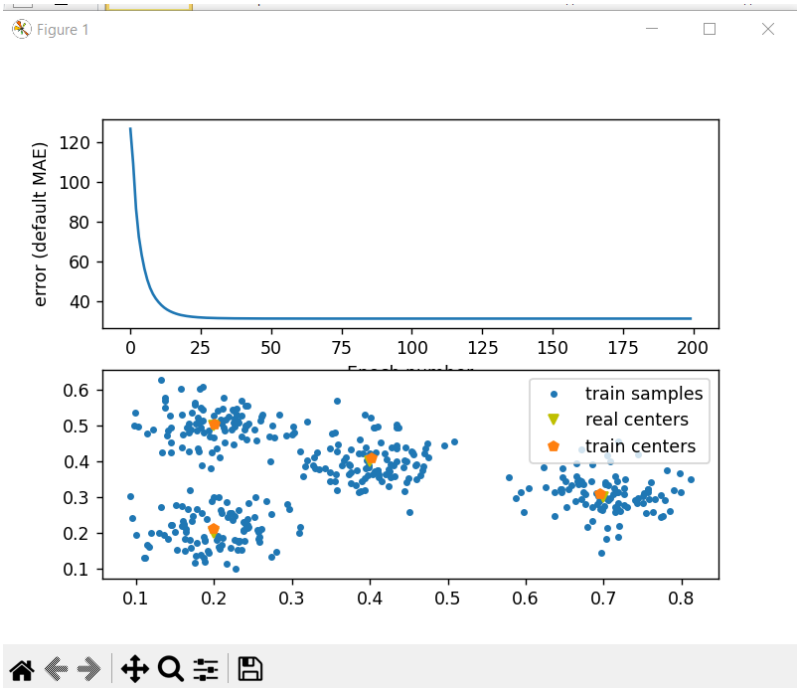

Рис. 21. Результат виконання скрипта LR_5_task_7.py

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

Завдання 8. Дослідження нейронної мережі на основі карти Кохонена, що само зорганізується

Проведіть дослідження по аналогії з попереднім завданням. Використовуючи готовий код внесіть зміни у вхідні данні згідно вашого варіанту у таблиці 3

Таблиця 3

№ варіанту	Центри кластера	skv
Варіант 1	[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,03
Варіант 2	[0.1, 0.2], [0.4, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,03
Варіант 3	[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.4, 0.5]	0,03
Варіант 4	[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]	0,03
Варіант 5	[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,03
Варіант 6	[0.3, 0.3], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,04
Варіант 7	[0.2, 0.1], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.3, 0.5]	0,04
Варіант 8	[0.1, 0.2], [0.4, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.3]	0,04
Варіант 9	[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.1, 0.5], [0.4, 0.5]	0,04
Варіант 10	[0.2, 0.2], [0.3, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]	0,04

Код скрипту LR_5_task_8.py:

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

skv = 0.04
centr = np.array([[0.2, 0.2], [0.3, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]])
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 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:
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()
```

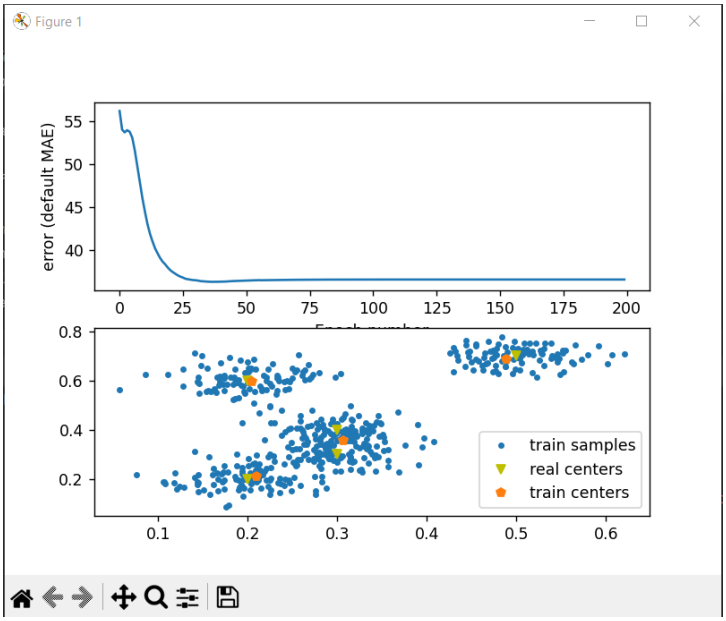

Рис. 22. Результат виконання скрипта LR_5_task_8.py

```
Epoch: 100; Error: 36.56259469595095;
Epoch: 200; Error: 36.56495704851727;
The maximum number of train epochs is reached
```

Рис. 23. Результат виконання скрипта LR_5_task_8.py

На рис. 23 зображено процес навчання мережі. На 100 епосі відбулось 36.562 помилки, на 200 епосі відбулось 36.564 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)
```

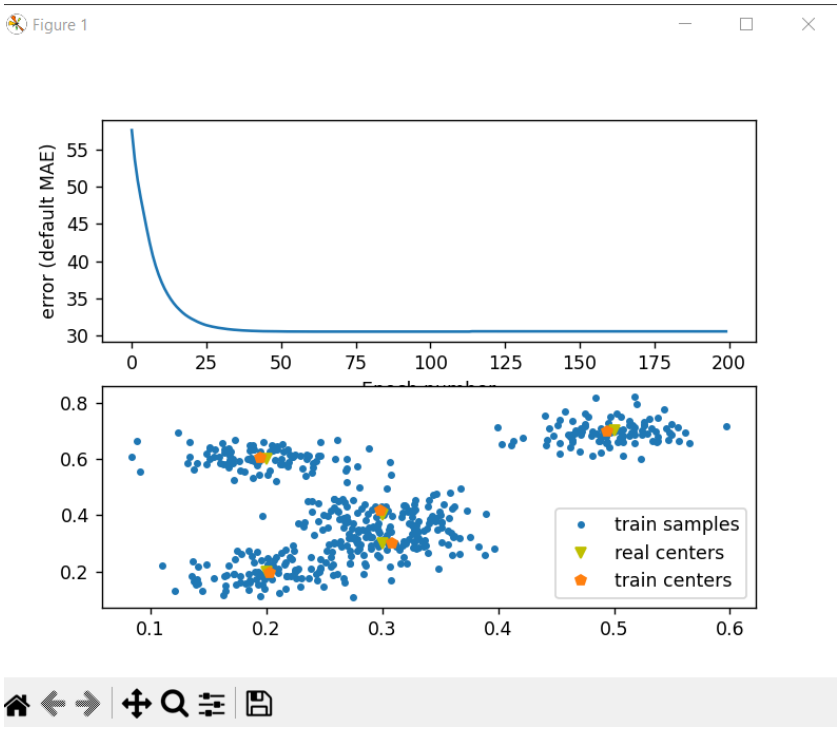

Рис. 24. Результат виконання скрипта LR_5_task_8.py

```
C:\Users\vlmyk\AppData\Local\Programs\Python\Python39\python.exe C:\artificial-intelligence-systems\lab5\LR_5_task_8.py
Epoch: 20; Error: 32.516515735639985;
Epoch: 40; Error: 30.671494158344796;
Epoch: 60; Error: 30.552795558878056;
Epoch: 80; Error: 30.562461991133272;
Epoch: 100; Error: 30.546710320475086;
Epoch: 120; Error: 30.59266452134321;
Epoch: 140; Error: 30.5873228293311;
Epoch: 160; Error: 30.582879937148505;
Epoch: 180; Error: 30.58188082493991;
Epoch: 200; Error: 30.58165485461435;
The maximum number of train epochs is reached
```

Рис. 25. Результат виконання скрипта LR_5_task_8.py

Висновок: 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. 23 гірші результати, ніж на рис. 35.

Висновок: під час виконання лабораторної роботи було використано спеціалізовані бібліотеки мови програмування Python, а також створено та застосовано прості нейронні мережі

GitHub:
https://github.com/VladyslavMyk/artificial-intelligence-systems.git