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 2.1. Створити простий нейрон

```
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))
```

```
0.9990889488055994
```

Рис.1. Результат виконання LR_5_task_1.py

Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини

```
import numpy as np

sigmoid = lambda x: 1 / (1 + np.exp(-x))
deriv_sigmoid = lambda x: sigmoid(x) * (1 - sigmoid(x))
mse_loss = lambda y_true, y_pred: ((y_true - y_pred) ** 2).mean()
```

```
class KochubeiNeuralNetwork:
    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
])

# Тренуємо вашу нейронну мережу!
network = KochubeiNeuralNetwork()
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
```

```
Epoch 0 loss: 0.080     Epoch 200 loss: 0.013    Epoch 400 loss: 0.006    Epoch 600 loss: 0.004    Epoch 800 loss: 0.003
Epoch 10 loss: 0.045    Epoch 210 loss: 0.012    Epoch 410 loss: 0.006    Epoch 610 loss: 0.004    Epoch 810 loss: 0.003
Epoch 20 loss: 0.038    Epoch 220 loss: 0.012    Epoch 420 loss: 0.005    Epoch 620 loss: 0.004    Epoch 820 loss: 0.003
Epoch 30 loss: 0.034    Epoch 230 loss: 0.011    Epoch 430 loss: 0.005    Epoch 630 loss: 0.004    Epoch 830 loss: 0.003
Epoch 40 loss: 0.031    Epoch 240 loss: 0.011    Epoch 440 loss: 0.005    Epoch 640 loss: 0.003    Epoch 840 loss: 0.003
Epoch 50 loss: 0.029    Epoch 250 loss: 0.010    Epoch 450 loss: 0.005    Epoch 650 loss: 0.003    Epoch 850 loss: 0.002
Epoch 60 loss: 0.027    Epoch 260 loss: 0.009    Epoch 460 loss: 0.005    Epoch 660 loss: 0.003    Epoch 860 loss: 0.002
Epoch 70 loss: 0.025    Epoch 270 loss: 0.009    Epoch 470 loss: 0.005    Epoch 670 loss: 0.003    Epoch 870 loss: 0.002
Epoch 80 loss: 0.024    Epoch 280 loss: 0.008    Epoch 480 loss: 0.005    Epoch 680 loss: 0.003    Epoch 880 loss: 0.002
Epoch 90 loss: 0.022    Epoch 290 loss: 0.008    Epoch 490 loss: 0.005    Epoch 690 loss: 0.003    Epoch 890 loss: 0.002
Epoch 100 loss: 0.021   Epoch 300 loss: 0.008    Epoch 500 loss: 0.005    Epoch 700 loss: 0.003    Epoch 900 loss: 0.002
Epoch 110 loss: 0.020   Epoch 310 loss: 0.007    Epoch 510 loss: 0.004    Epoch 710 loss: 0.003    Epoch 910 loss: 0.002
Epoch 120 loss: 0.019   Epoch 320 loss: 0.007    Epoch 520 loss: 0.004    Epoch 720 loss: 0.003    Epoch 920 loss: 0.002
Epoch 130 loss: 0.018   Epoch 330 loss: 0.007    Epoch 530 loss: 0.004    Epoch 730 loss: 0.003    Epoch 930 loss: 0.002
Epoch 140 loss: 0.017   Epoch 340 loss: 0.007    Epoch 540 loss: 0.004    Epoch 740 loss: 0.003    Epoch 940 loss: 0.002
Epoch 150 loss: 0.016   Epoch 350 loss: 0.007    Epoch 550 loss: 0.004    Epoch 750 loss: 0.003    Epoch 950 loss: 0.002
Epoch 160 loss: 0.015   Epoch 360 loss: 0.006    Epoch 560 loss: 0.004    Epoch 760 loss: 0.003    Epoch 960 loss: 0.002
Epoch 170 loss: 0.015   Epoch 370 loss: 0.006    Epoch 570 loss: 0.004    Epoch 770 loss: 0.003    Epoch 970 loss: 0.002
Epoch 180 loss: 0.014   Epoch 380 loss: 0.006    Epoch 580 loss: 0.004    Epoch 780 loss: 0.003    Epoch 980 loss: 0.002
Epoch 190 loss: 0.013   Epoch 390 loss: 0.006    Epoch 590 loss: 0.004    Epoch 790 loss: 0.003    Epoch 990 loss: 0.002
```

Рис.2. Результат виконання LR_5_task_2.py

```
Emily: 0.975
Frank: 0.047
```

Рис.3. Результат виконання LR_5_task_2.py

Завдання 2.3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

```
iimport 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()
```

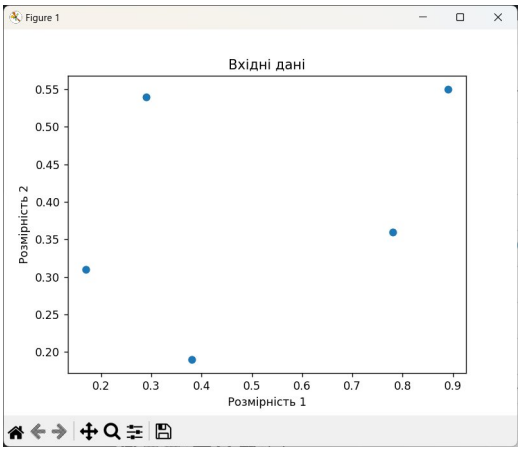

Рис.4. Результат виконання LR_5_task_3.py

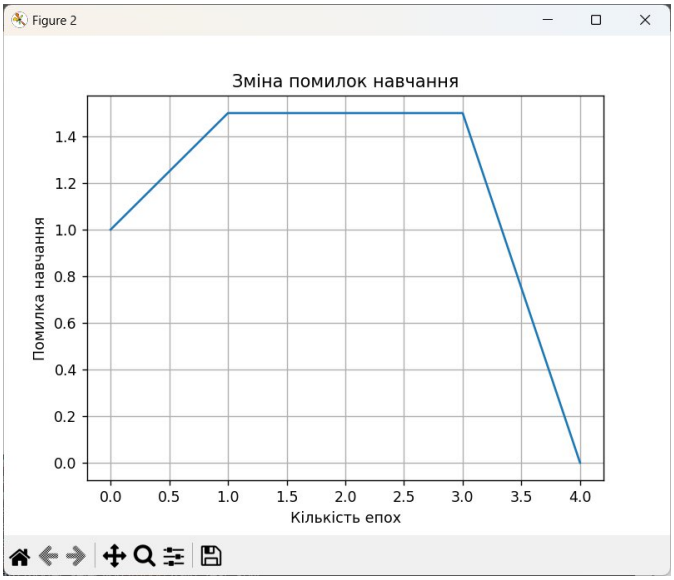

Рис.5. Результат виконання LR_5_task_3.py

Завдання 2.4. Побудова одношарової нейронної мережі

```
iimport 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:]

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

# 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
num_output = labels.shape[1]

# Визначення одношарової нейронної мережі
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)

# Побудова графіка просування 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()

# 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])
```

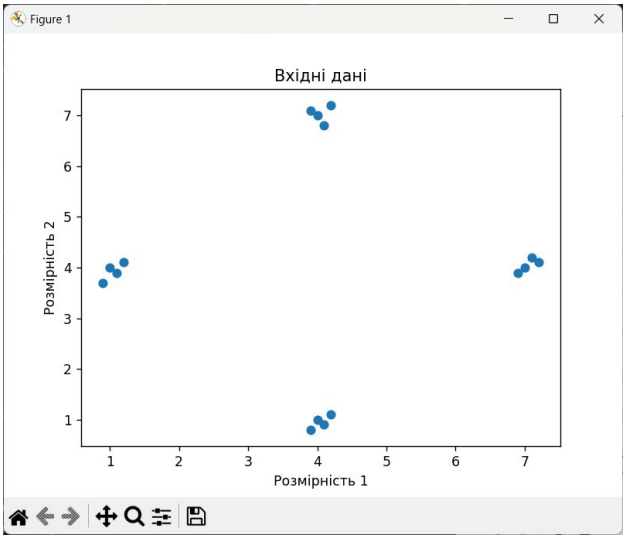

Рис.6. Результат виконання LR_5_task_4.py

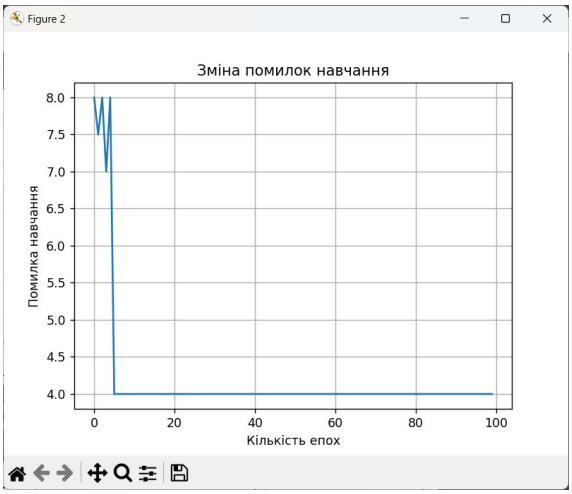

Рис.7. Результат виконання LR_5_task_4.py

```
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.]
```

Рис.8. Результат виконання LR_5_task_4.py

Завдання 2.5. Побудова багатошарової нейронної мережі

```
iimport 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('Розмірність 1')
plt.ylabel('Розмірність 2')
plt.title('Вхідні дані')

# 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
output = nn.sim(data)
y_pred = output.reshape(num_points)

# Побудова графіка помилки навчання
plt.figure()
plt.plot(error_progress)
plt.xlabel('Кількість епох')
plt.ylabel('Помилка навчання')
plt.title('Зміна помилок 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('Фактичні і прогнозовані значення')
plt.show()
```

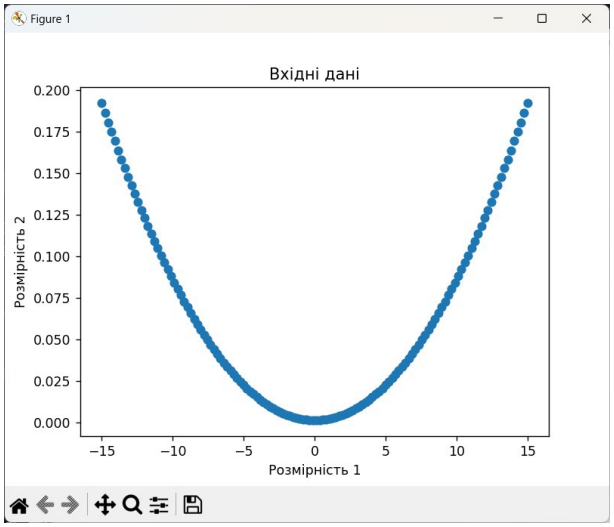

Рис.9. Результат виконання LR_5_task_5.py

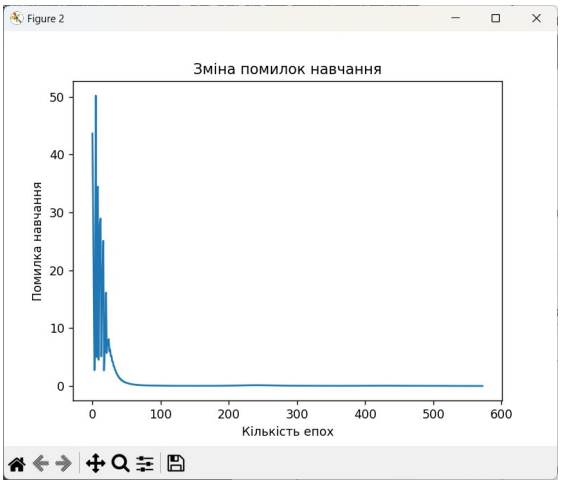

Рис.10. Результат виконання LR_5_task_5.py

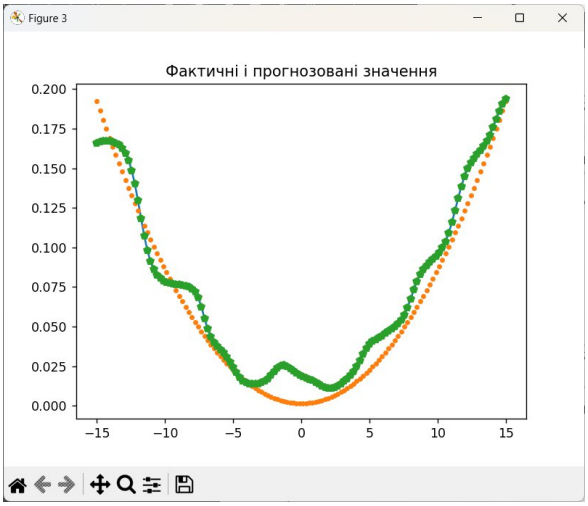

Рис.11. Результат виконання LR_5_task_5.py

```
Epoch: 100; Error: 0.05297958025890666;
Epoch: 200; Error: 0.054424685862353335;
Epoch: 300; Error: 0.04316877460965424;
Epoch: 400; Error: 0.04713378412703095;
Epoch: 500; Error: 0.021194791042959873;
The goal of learning is reached
```

Рис.12. Результат виконання LR_5_task_5.py

Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

```
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 = 5 * x * x + 4
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('Розмірність 1')
plt.ylabel('Розмірність 2')
plt.title('Вхідні дані')

# 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
output = nn.sim(data)
y_pred = output.reshape(num_points)

# Побудова графіка помилки навчання
plt.figure()
plt.plot(error_progress)
plt.xlabel('Кількість епох')
plt.ylabel('Помилка навчання')
plt.title('Зміна помилок 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('Фактичні і прогнозовані значення')
plt.show()
```

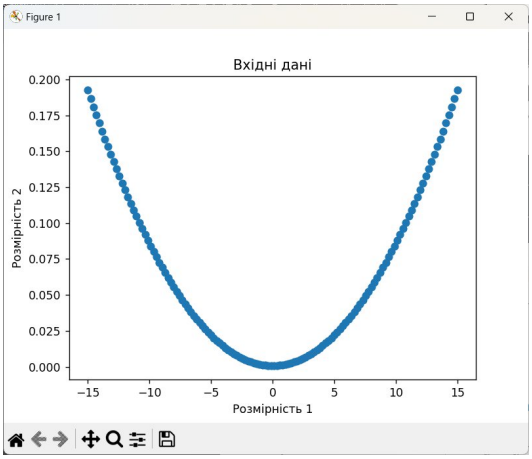

Рис.13. Результат виконання LR_5_task_6.py

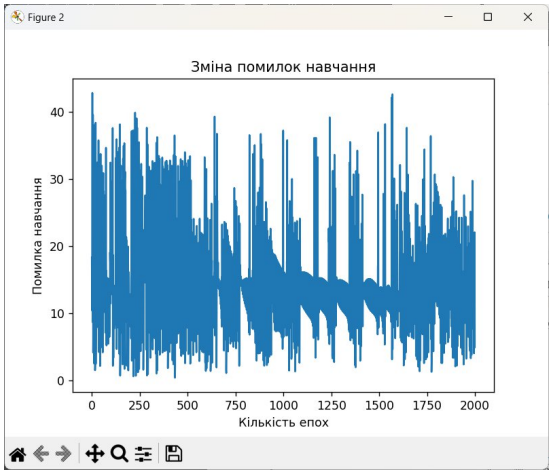

Рис.14. Результат виконання LR_5_task_6.py

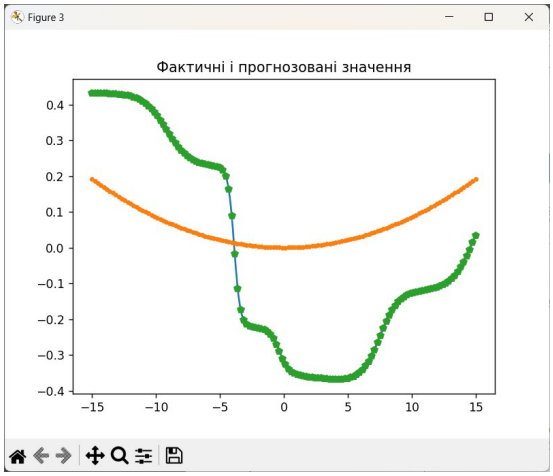

Рис.15. Результат виконання LR_5_task_6.py

```
Epoch: 100; Error: 23.01631531839916;
Epoch: 200; Error: 9.88408015825278;
Epoch: 300; Error: 11.249336723364245;
Epoch: 400; Error: 22.865658564294172;
Epoch: 500; Error: 9.222305993390092;
Epoch: 600; Error: 1.7688267816822494;
Epoch: 700; Error: 14.166590840648555;
Epoch: 800; Error: 15.110322622905935;
Epoch: 900; Error: 21.99558172564417;
Epoch: 1000; Error: 37.23747391149833;
Epoch: 1100; Error: 15.297486476045297;
Epoch: 1200; Error: 11.015510083308097;
Epoch: 1300; Error: 14.956217094760534;
Epoch: 1400; Error: 17.476974642168244;
Epoch: 1500; Error: 11.2930155043864;
Epoch: 1600; Error: 13.31143368290245;
Epoch: 1700; Error: 9.371162293766439;
Epoch: 1800; Error: 11.774804324493275;
Epoch: 1900; Error: 6.503608668784356;
Epoch: 2000; Error: 4.919883764652418;
The maximum number of train epochs is reached
```

Рис.16. Результат виконання LR_5_task_6.py

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

```
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()
```

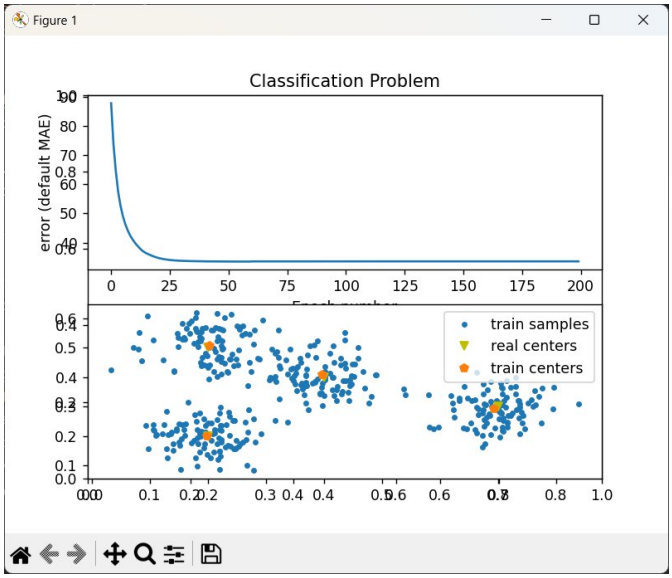

Рис.17. Результати виконання LR_5_task_7.py

```
Epoch: 100; Error: 33.5704982519297O6;
Epoch: 200; Error: 33.57123295449014;
The maximum number of train epochs is reached
```

Рис.18. Результати виконання LR_4_task_7.py

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

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

skv = 0.05
centr = np.array([[0.2, 0.2], [0.3, 0.4], [0.6, 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=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()
```

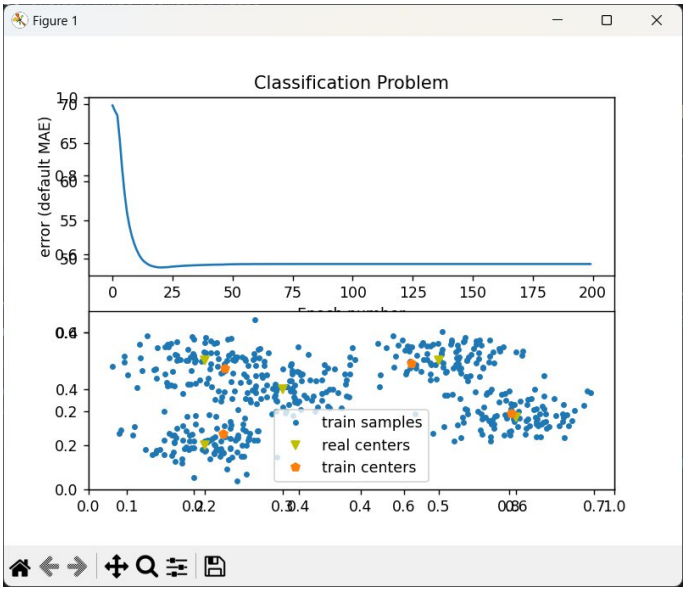

Рис.19. Результати виконання LR_5_task_8.py для 4 нейронів

```
Epoch: 100; Error: 49.29880766897483;
Epoch: 200; Error: 49.298881212188704;
The maximum number of train epochs is reached
```

Рис.20. Результати виконання LR_5_task_8.py для 4 нейронів

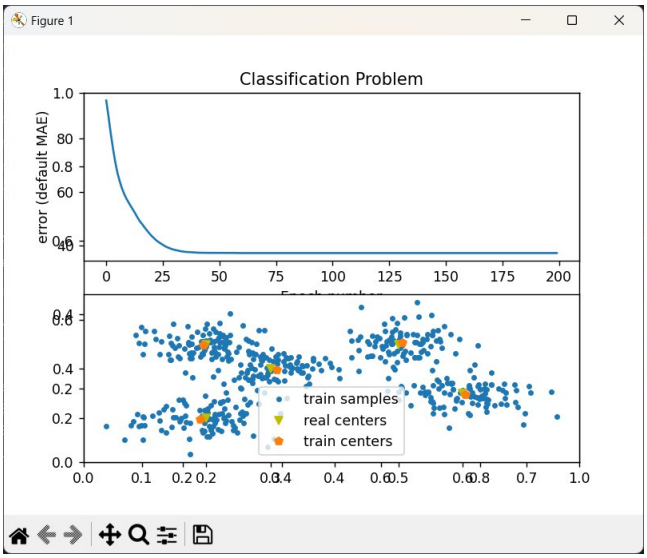

Рис.21. Результати виконання LR_5_task_8.py для 5 нейронів

```
Epoch: 100; Error: 37.127004409846734;
Epoch: 200; Error: 37.12823008549995;
The maximum number of train epochs is reached
```

Рис.22. Результати виконання LR_5_task_8.py для 5 нейронів

Репозиторій: https://github.com/Kochubei-Kostiantyn/AI_labs

Висновки: в ході виконання лабораторної роботи ми використовуючи спеціалізовані бібліотеки та мову програмування Python навчилися створювати та застосовувати прості нейронні мережі.