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

Лістинг програми:

```
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])  # x
print(n.feedforward(x))
```

Результат виконання програми:

```
0.9990889488055994
```

Рис. 5.1 Результат виконання програми

Завдання 2.2: Створити просту нейронну мережу для передбачення статі людини.

Лістинг програми:

```
import numpy as np
from LR_5_task_1 import Neuron, sigmoid


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


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


class OleksiichukNeuralNetwork:
    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 = 100

        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)

                # Neuron o1
                d_ypred_d_w5 = h1 * derivative_sigmoid(sum_o1)
                d_ypred_d_w6 = h2 * derivative_sigmoid(sum_o1)
                d_ypred_d_b3 = derivative_sigmoid(sum_o1)

                d_ypred_d_h1 = self.w5 * derivative_sigmoid(sum_o1)
                d_ypred_d_h2 = self.w6 * derivative_sigmoid(sum_o1)
```

```
                # Neuron h1
                d_h1_d_w1 = x[0] * derivative_sigmoid(sum_h1)
                d_h1_d_w2 = x[1] * derivative_sigmoid(sum_h1)
                d_h1_d_b1 = derivative_sigmoid(sum_h1)

                # Neuron h2
                d_h2_d_w3 = x[0] * derivative_sigmoid(sum_h2)
                d_h2_d_w4 = x[1] * derivative_sigmoid(sum_h2)
                d_h2_d_b2 = derivative_sigmoid(sum_h2)

                # Update weights and biases
                # Neuron 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

                # Neuron 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

                # Neuron 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

            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))


if __name__ == "__main__":
    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 = OleksiichukNeuralNetwork()
    network.train(data, all_y_trues)

    emily = np.array([-7, -3])  # 128 pounds, 63 inches
    frank = np.array([20, 2])  # 155 pounds, 68 inches
    print("Emily: %.3f" % network.feedforward(emily))  # +-0.96 - F
    print("Frank: %.3f" % network.feedforward(frank))  # +-0.039 - M
```

Результат виконання програми:

```
0.9990889488055994
Epoch 0 loss: 0.094
Epoch 10 loss: 0.044
Epoch 20 loss: 0.031
Epoch 30 loss: 0.025
Epoch 40 loss: 0.022
Epoch 50 loss: 0.019
Epoch 60 loss: 0.017
Epoch 70 loss: 0.015
Epoch 80 loss: 0.014
Epoch 90 loss: 0.013
Emily: 0.924
Frank: 0.111
```

Рис. 5.2 Результат виконання програми

Завдання 2.3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

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 даних на точки даних та мітки
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('Прогрес помилки навчання')
```

```
plt.grid()
plt.show()
```

Результат виконання програми:

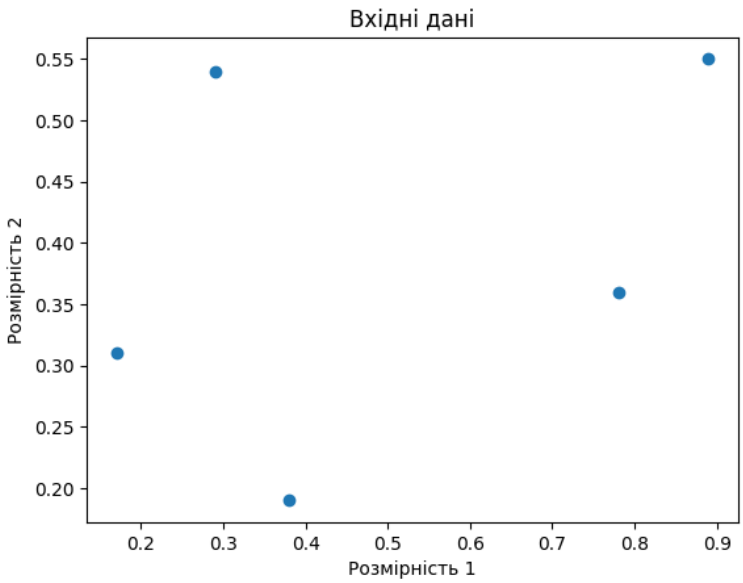

Рис. 5.3 Результат виконання програми

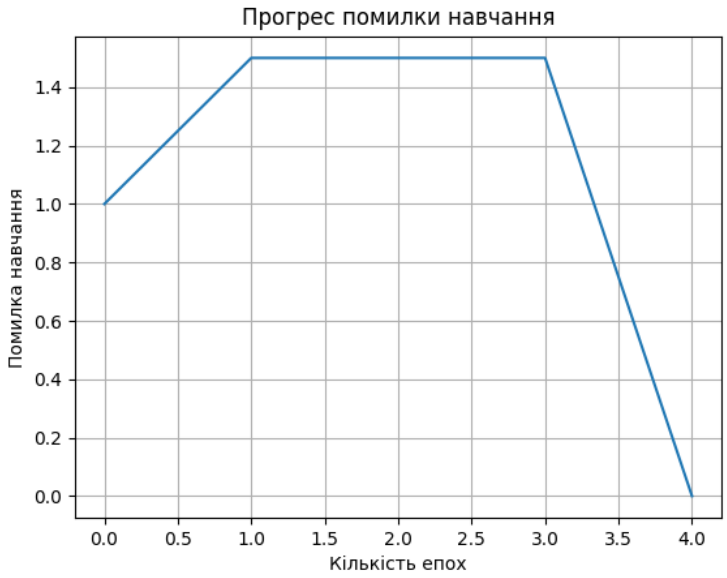

Рис. 5.4 Результат виконання програми

Завдання 2.4. Побудова одношарової нейронної мережі

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:]

# 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)

# Навчання нейронної мережі
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('Прогрес помилки навчання')
plt.grid()
plt.show()

# Виконання класифікатора на тестових точках даних
print('\nРезультати тесту:')
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])
```

Результат виконання програми:

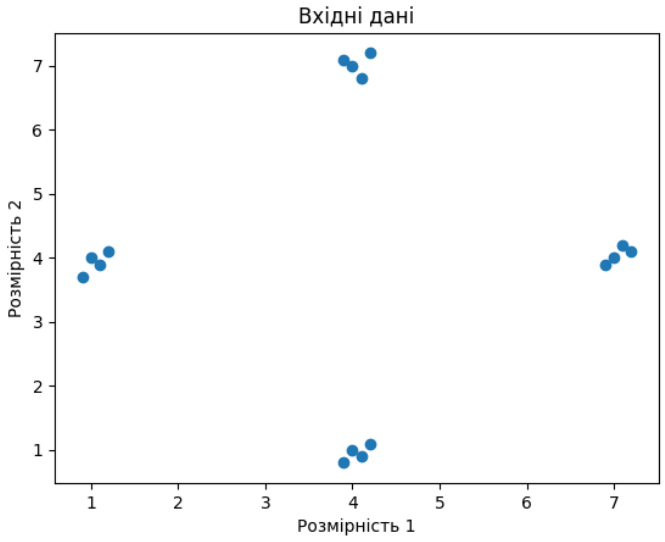

Рис. 5.5 Результат виконання програми

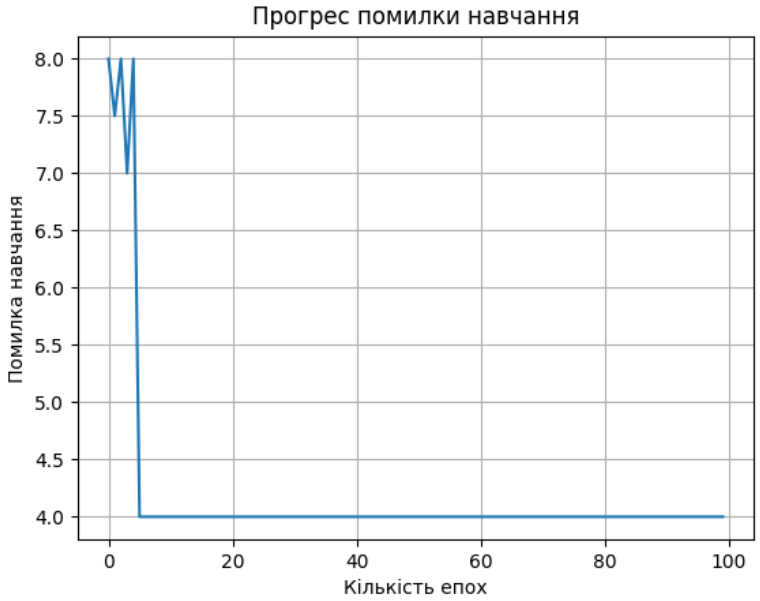

Рис. 5.6 Результат виконання програми

```
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

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

Рис. 5.7 Результат виконання програми

Завдання 2.5. Побудова багатошарової нейронної мережі

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('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('Прогрес 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('Дійсні і передбачені значення')
plt.show()
```

Результат виконання програми:

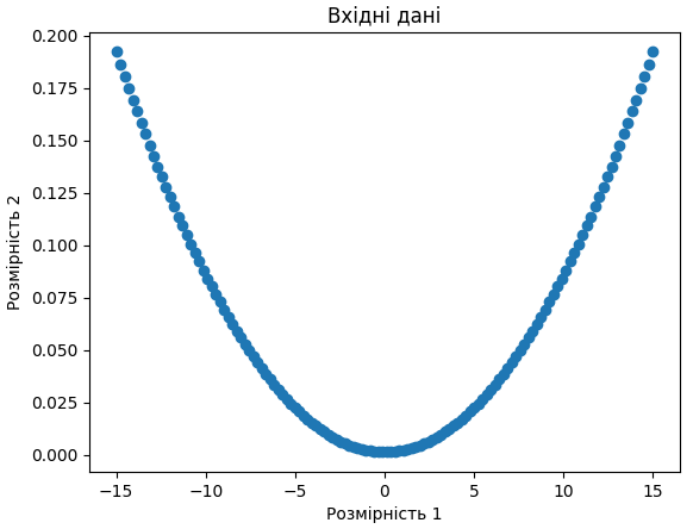

Рис. 5.8 Результат виконання програми

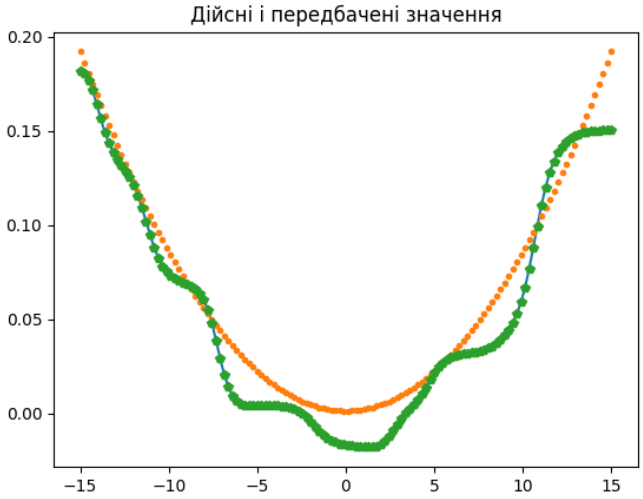

Рис. 5.9 Результат виконання програми

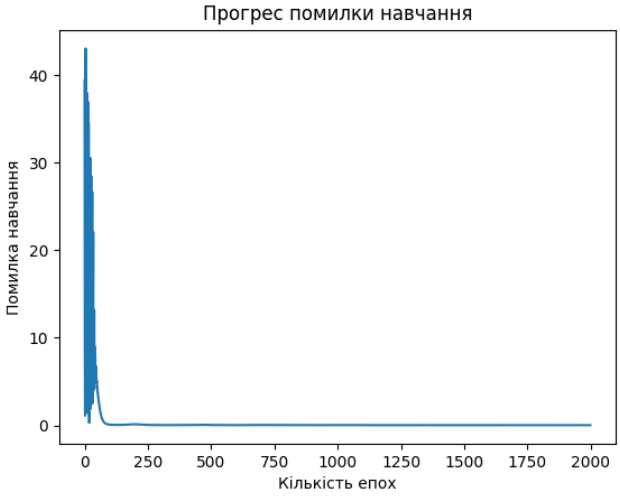

Рис. 5.10 Результат виконання програми

```
Epoch: 100; Error: 0.07731425597387277;
Epoch: 200; Error: 0.12325240703089034;
Epoch: 300; Error: 0.024493801000407357;
Epoch: 400; Error: 0.03190374416291492;
Epoch: 500; Error: 0.04044845069948201;
Epoch: 600; Error: 0.020835180601432565;
Epoch: 700; Error: 0.03524153184280601;
Epoch: 800; Error: 0.02605792192980762;
Epoch: 900; Error: 0.019369004647355203;
Epoch: 1000; Error: 0.029238807705214434;
Epoch: 1100; Error: 0.021103330953722327;
Epoch: 1200; Error: 0.017339920126350678;
Epoch: 1300; Error: 0.022969055236416555;
Epoch: 1400; Error: 0.018757416613808787;
Epoch: 1500; Error: 0.015055655018978514;
Epoch: 1600; Error: 0.017712242059445736;
Epoch: 1700; Error: 0.017359839904746748;
Epoch: 1800; Error: 0.013530407129116831;
Epoch: 1900; Error: 0.013538852517671512;
Epoch: 2000; Error: 0.015088465994234876;
The maximum number of train epochs is reached
```

Рис. 5.11 Результат виконання програми

Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

Табл. 5.1

№ варіанта	Тестові дані
Варіант 5	$y = 2x^2+9$

Табл. 5.2

Номер варіанта	Багатошаровий персептрон	
	Кількість шарів	Кількості нейронів у шарах
5	3	2-2-1

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) + 9
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 трьома прихованими
# шарами. Перший прихований шар складається із трьох нейронів.
# Другий прихований шар складається з чотирьох нейронів.
# Третій прихований шар складається з одного нейрона.
# Вихідний шар складається з одного нейрона
nn = nl.net.newff([[min_val, max_val]], [3, 2, 2, 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('Прогрес 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('Дійсні і передбачені значення')
plt.show()
```

Результат виконання програми:

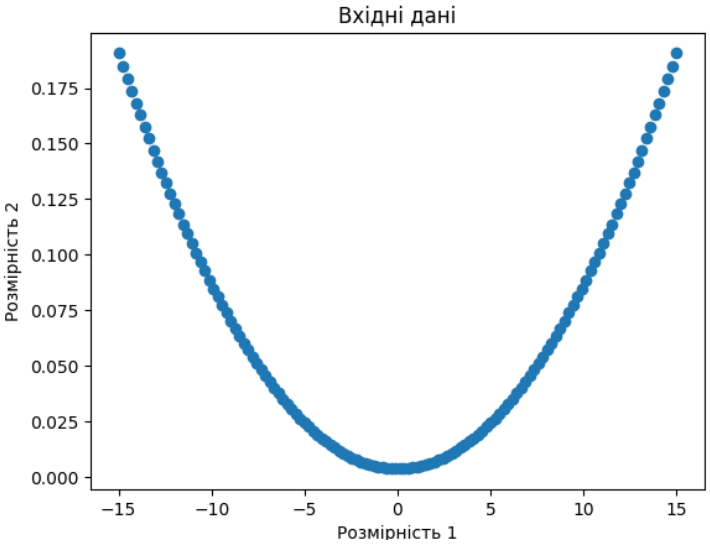

Рис. 5.12 Результат виконання програми

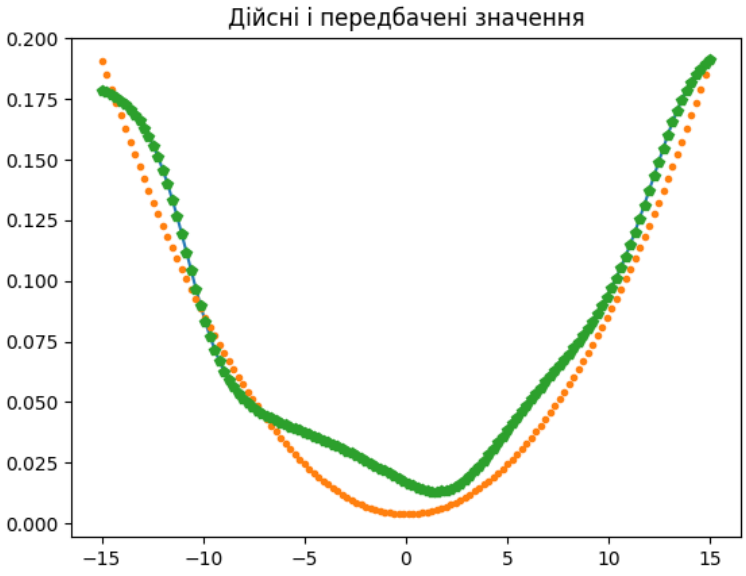

Рис. 5.13 Результат виконання програми

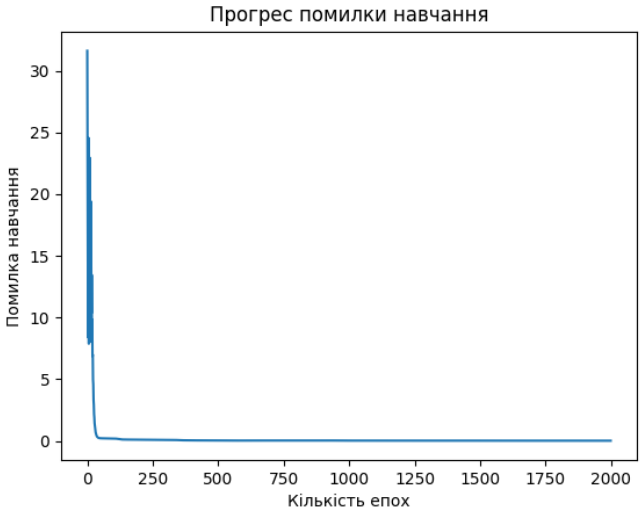

Рис. 5.14 Результат виконання програми

```
Epoch: 100; Error: 0.19525995432766957;
Epoch: 200; Error: 0.09854040980084011;
Epoch: 300; Error: 0.08798664681366675;
Epoch: 400; Error: 0.03833820023649181;
Epoch: 500; Error: 0.023486571211226732;
Epoch: 600; Error: 0.024436325385697295;
Epoch: 700; Error: 0.021365524887707647;
Epoch: 800; Error: 0.015174523589767135;
Epoch: 900; Error: 0.025632596496968543;
Epoch: 1000; Error: 0.019152114469030323;
Epoch: 1100; Error: 0.017382474241524483;
Epoch: 1200; Error: 0.01980380201344554;
Epoch: 1300; Error: 0.015381919843741716;
Epoch: 1400; Error: 0.014752696364995502;
Epoch: 1500; Error: 0.014934570135182231;
Epoch: 1600; Error: 0.013123446139156312;
Epoch: 1700; Error: 0.012352348853869162;
Epoch: 1800; Error: 0.012369130730259883;
Epoch: 1900; Error: 0.011731524587786306;
Epoch: 2000; Error: 0.011036326834860583;
The maximum number of train epochs is reached
```

Рис. 5.15 Результат виконання програми

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

Лістинг програми:

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


skv = 0.05
center = 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([center + 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:
fig, axs = pl.subplots(2)
fig.suptitle('Classification Problem')
axs.flat[1].set(xlabel='Epoch number', ylabel='error (default MAE)')
axs[1].plot(error)
w = net.layers[0].np['w']
axs[0].plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0],
w[:, 1], 'p')
axs[0].legend(['train samples', 'real centers', 'train centers'], loc='upper
left')
pl.show()
```

Результат виконання програми:

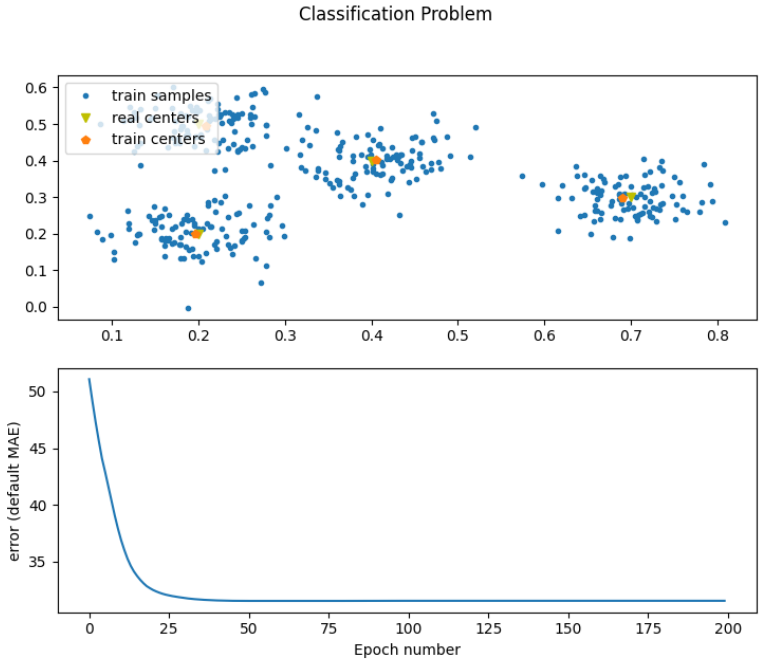

Рис. 5.16 Результат виконання програми

```
Epoch: 20; Error: 32.65352717669228;
Epoch: 40; Error: 31.581117835029346;
Epoch: 60; Error: 31.52669544211257;
Epoch: 80; Error: 31.528866029604686;
Epoch: 100; Error: 31.537880406528718;
Epoch: 120; Error: 31.539208515982065;
Epoch: 140; Error: 31.53938823175293;
Epoch: 160; Error: 31.539412503151606;
Epoch: 180; Error: 31.539415773246006;
Epoch: 200; Error: 31.539416212502275;
The maximum number of train epochs is reached
```

Рис. 5.17 Результат виконання програми

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

Табл. 5.3

№ варіанту	Центри кластера	skv
Варіант 5	[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,03

Лістинг програми:

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


skv = 0.03
center = np.array([[0.2, 0.1], [0.5, 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([center + 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:
fig, axs = pl.subplots(2)
fig.suptitle('Classification Problem')
axs.flat[1].set(xlabel='Epoch number', ylabel='error (default MAE)')
axs[1].plot(error)
w = net.layers[0].np['w']
axs[0].plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0],
w[:, 1], 'p')
axs[0].legend(['train samples', 'real centers', 'train centers'],loc='upper left')
pl.show()
```

Результат виконання програми:

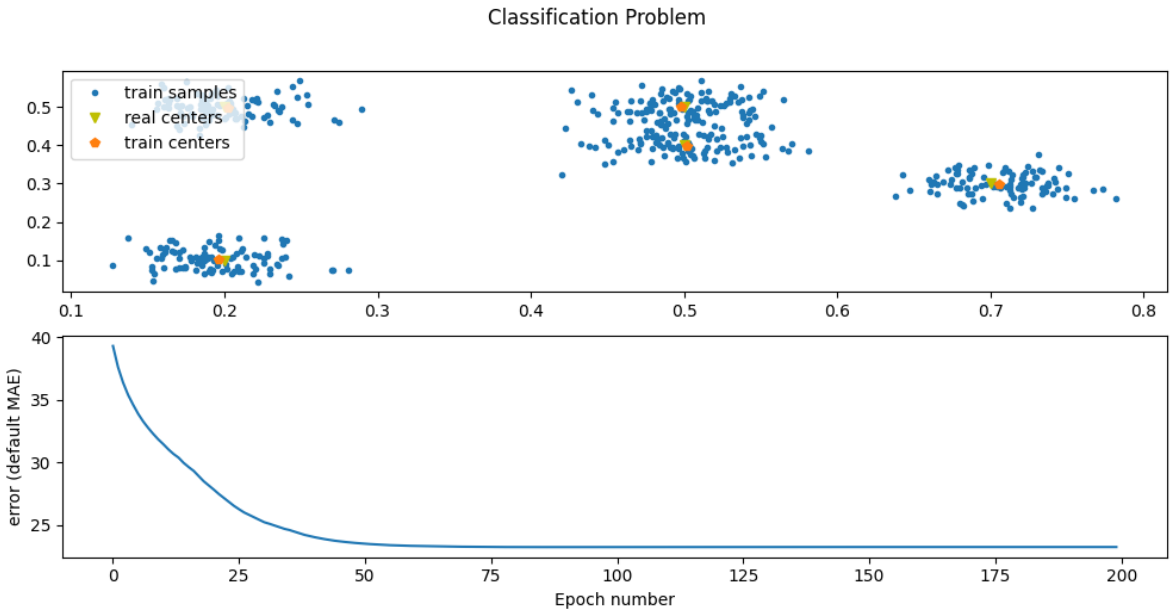

Рис. 5.18 Результат виконання програми

```
Epoch: 20; Error: 26.155134451655293;
Epoch: 40; Error: 24.50903539654272;
Epoch: 60; Error: 24.427671375424865;
Epoch: 80; Error: 24.43417997116672;
Epoch: 100; Error: 24.438797487929577;
Epoch: 120; Error: 24.44033454437416;
Epoch: 140; Error: 24.440801645926754;
Epoch: 160; Error: 24.440938665807664;
Epoch: 180; Error: 24.44097882076752;
Epoch: 200; Error: 24.4409905769574;
The maximum number of train epochs is reached
```

Рис. 5.19 Результат виконання програми

Посилання на GitHub: https://github.com/BogdanStelmah/Basics-of-AI_labs

Висновок: На даній лабораторній роботі ми використовуючи спеціалізовані бібліотеки та мову програмування Python навчилися створювати та застосовувати прості нейронні мережі.