Лабораторна робота 5

Тема: 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

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):
        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

Завдання 2. Створити просту нейронну мережу для передбачення статі людини

Лістинг програми

```
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 VoitkoNeuralNetwork:

    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 = VoitkoNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x))  # 0.7216325609518421
```

```
0.7216325609518421
```

Рис. 2 Результат виконання програми

Лістинг програми

```
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 VoitkoNeuralNetwork:

    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 = VoitkoNeuralNetwork()
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
```

```
Epoch 0 loss: 0.272
Epoch 10 loss: 0.212
Epoch 20 loss: 0.163
Epoch 30 loss: 0.093
Epoch 40 loss: 0.066
Epoch 50 loss: 0.051
Epoch 60 loss: 0.041
Epoch 70 loss: 0.034
Epoch 80 loss: 0.029
Epoch 90 loss: 0.025
Epoch 100 loss: 0.022
Epoch 110 loss: 0.019
Epoch 120 loss: 0.017
Epoch 130 loss: 0.016
Epoch 140 loss: 0.014
Epoch 150 loss: 0.013
Epoch 160 loss: 0.012
Epoch 170 loss: 0.011
Epoch 180 loss: 0.011
Epoch 190 loss: 0.010
Epoch 200 loss: 0.009
Epoch 210 loss: 0.009
Epoch 220 loss: 0.008
Epoch 230 loss: 0.008
Epoch 240 loss: 0.008
Epoch 250 loss: 0.007
Epoch 260 loss: 0.007
Epoch 270 loss: 0.007
Epoch 280 loss: 0.006
Epoch 290 loss: 0.006
Epoch 300 loss: 0.006
Epoch 310 loss: 0.006
Epoch 320 loss: 0.005
Epoch 330 loss: 0.005
Epoch 340 loss: 0.005
Epoch 350 loss: 0.005
Epoch 360 loss: 0.005
Epoch 370 loss: 0.005
Epoch 380 loss: 0.004
Epoch 390 loss: 0.004
Epoch 400 loss: 0.004
Epoch 410 loss: 0.004
Epoch 420 loss: 0.004
Epoch 430 loss: 0.004
Epoch 440 loss: 0.004
Epoch 450 loss: 0.004
Epoch 460 loss: 0.004
Epoch 470 loss: 0.003
Epoch 480 loss: 0.003
Epoch 490 loss: 0.003
Epoch 500 loss: 0.003
Epoch 510 loss: 0.003
Epoch 520 loss: 0.003
Epoch 530 loss: 0.003
Epoch 540 loss: 0.003
Epoch 550 loss: 0.003
```

Рис. 3 Результат виконання програми

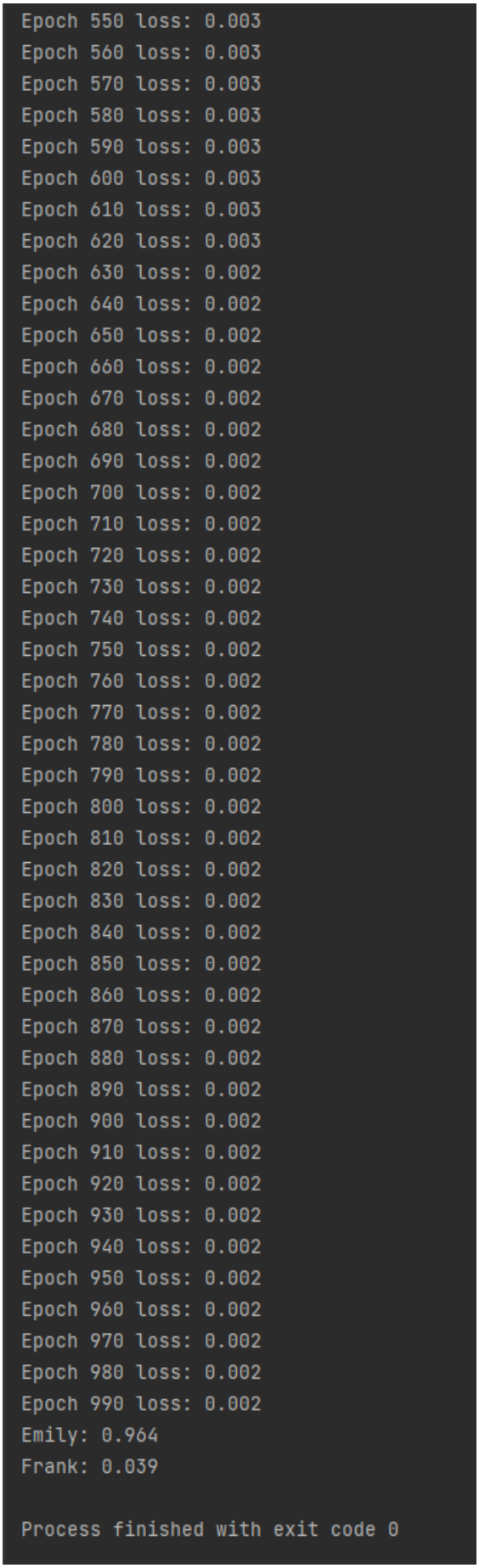

Рис. 4 Результат виконання програми

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

Лістинг програми

```
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()
```

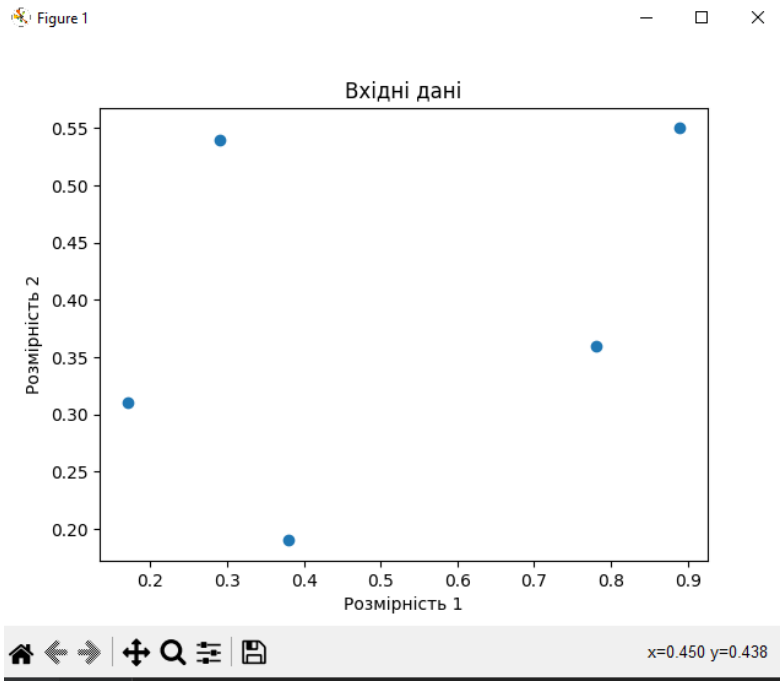

Рис. 5 Результат виконання програми

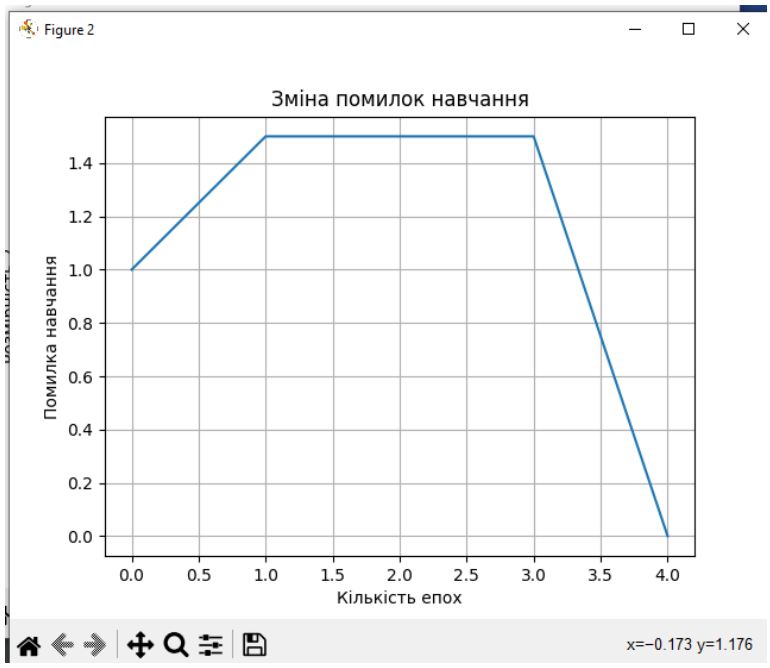

Рис. 6 Результат виконання програми

Висновок

На другому графіку я відобразив процес навчання, використовуючи метрику помилки. Під час першої епохи відбулося від 1.0 до 1.5 помилок, під час наступних двох епох відбулось 1.5 помилок. Потім під час 4 епохи помилки почались зменшувтась, тому що ми навчили перцептрон за допомогою тренувальних даних.

Завдання 2.4. Побудова одношарової нейронної мережі

Лістинг програми

```
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])
```

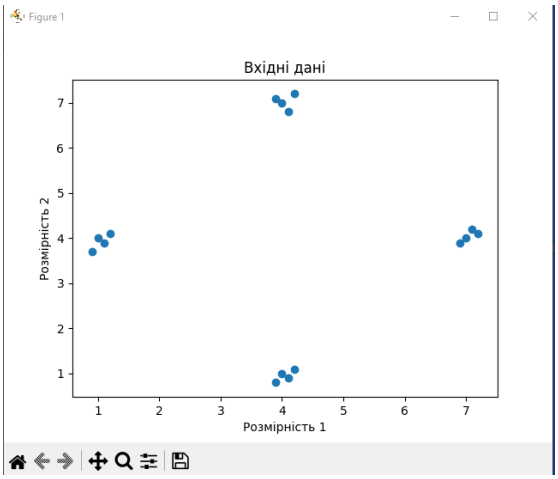

Рис. 7 Графік вхідних даних

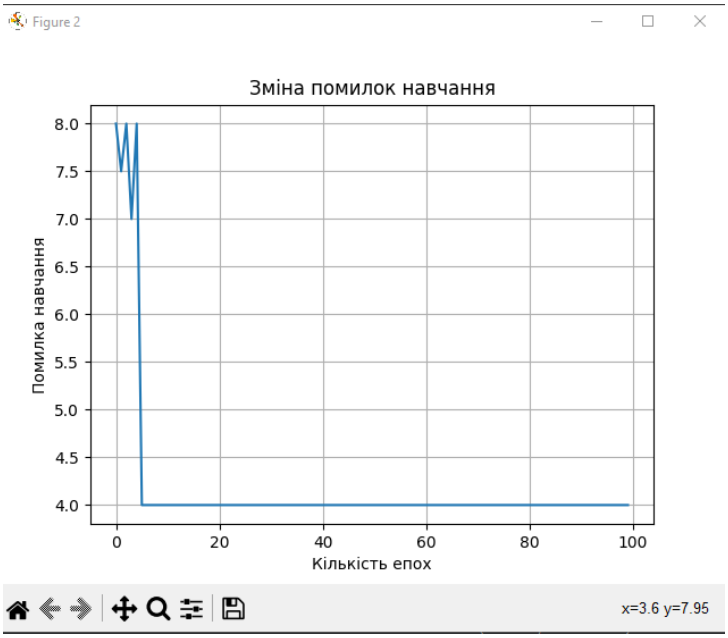

Рис. 8 Графік вхідних даних

```
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
```

Рис. 9 Результат виконання програми

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

```
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()
```

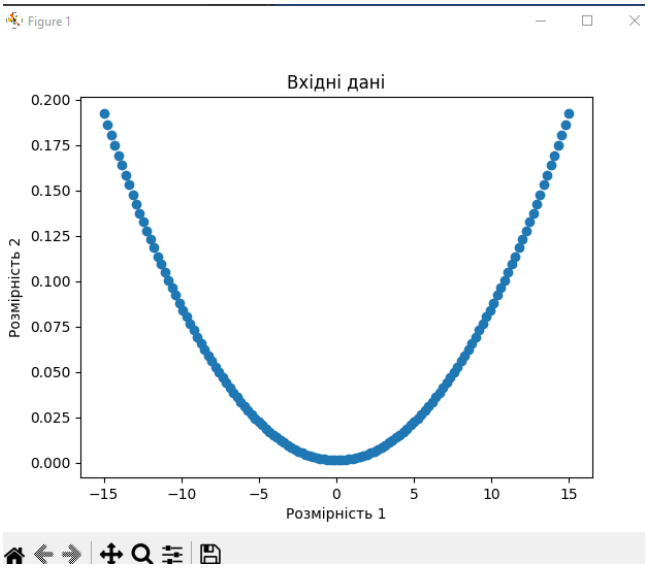

Рис. 10 Результат виконання програми

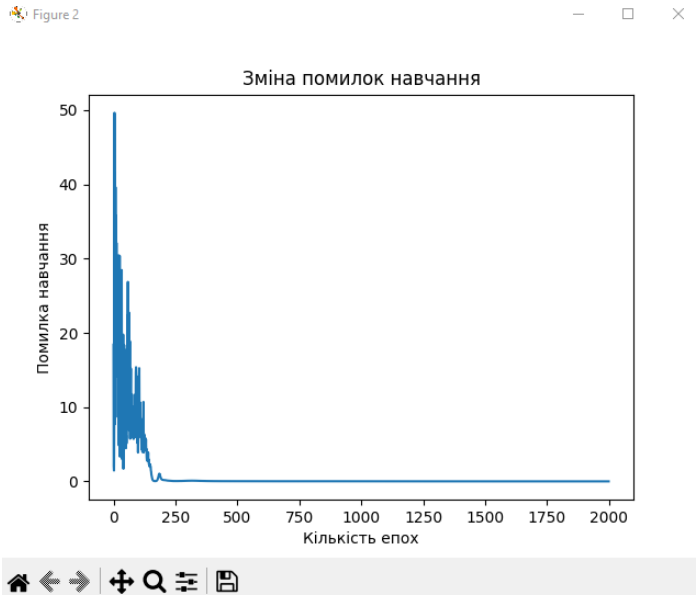

Рис. 11 Результат виконання програми

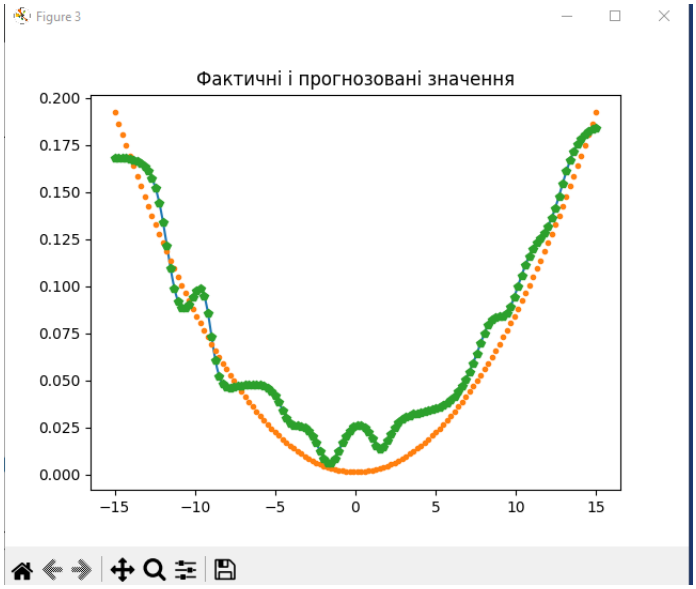

Рис. 12 Результат виконання програми

```
C:\Users\Admin\PycharmProjects\labka5\venv\Scripts\python.exe C:\Users\Admin\PycharmProjects\labka5\LR_5_task_6.py
Epoch: 100; Error: 3.8971502113067884;
Epoch: 200; Error: 0.20659635218589764;
Epoch: 300; Error: 0.09394051853970468;
Epoch: 400; Error: 0.04750492778482725;
Epoch: 500; Error: 0.03598989003940468;
Epoch: 600; Error: 0.03135003406058164;
Epoch: 700; Error: 0.02804570858272175;
Epoch: 800; Error: 0.028305764193339432;
Epoch: 900; Error: 0.02771944513094505;
Epoch: 1000; Error: 0.025043919737972747;
Epoch: 1100; Error: 0.022373510315360175;
Epoch: 1200; Error: 0.019886571924697283;
Epoch: 1300; Error: 0.017816685222463526;
Epoch: 1400; Error: 0.0163172013114228;
Epoch: 1500; Error: 0.015212590425481627;
Epoch: 1600; Error: 0.014349822661857086;
Epoch: 1700; Error: 0.013673668767267295;
Epoch: 1800; Error: 0.013140543892706188;
Epoch: 1900; Error: 0.012700816690986383;
Epoch: 2000; Error: 0.012325086835247445;
The maximum number of train epochs is reached
```

Рис. 13 Результат виконання програми

Висновок

На рис. 13 зображено процес навчання мережі. Відносно кожної епосі відбувались помилки. На 100 3.87 помилки. На 2000 0.01. Потім вивелось повідомлення, що ми досягли цілі навчання.

Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

Варіант 2	$y = 2x^2+6$

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

```
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 = 2 * x * x + 6
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]], [2, 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()
```

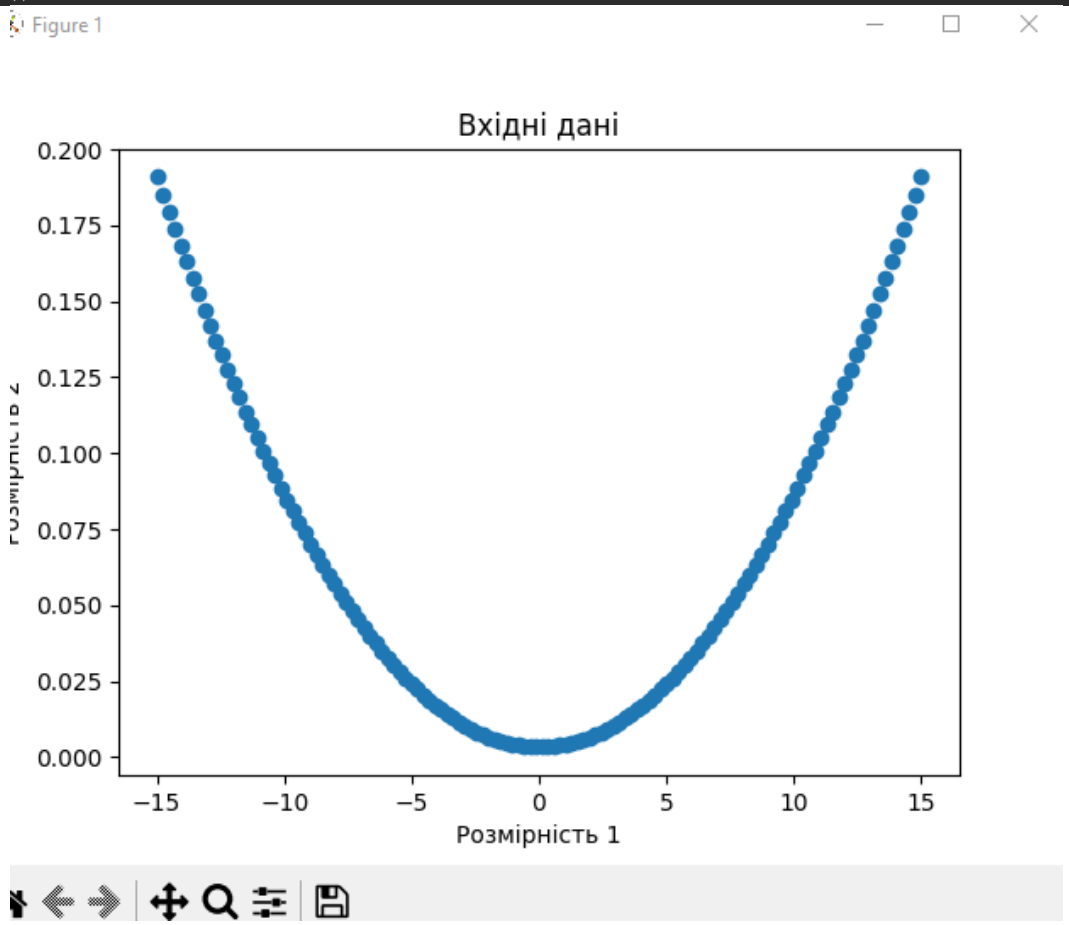

Рис. 14 Результат виконання програми

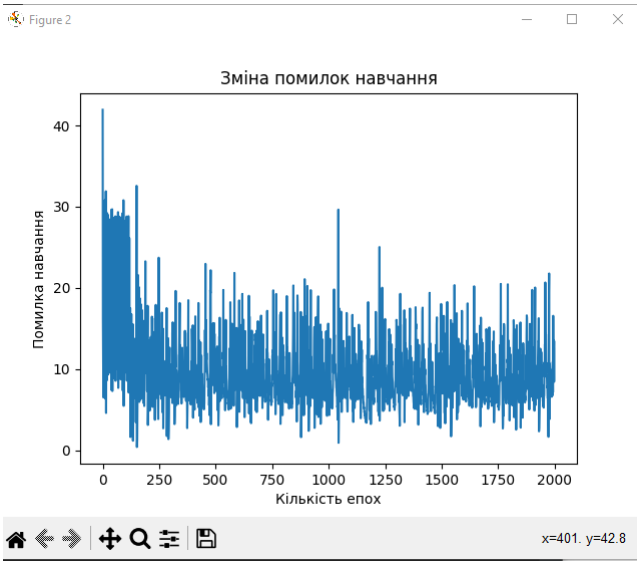

Рис. 15 Результат виконання програми

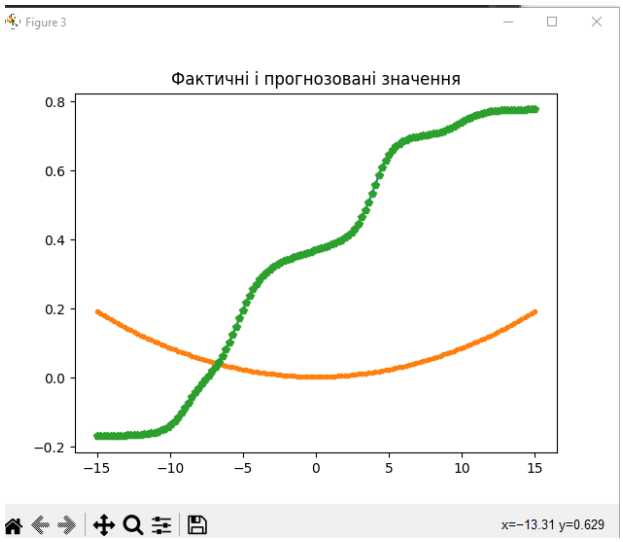

Рис. 16 Результат виконання програми

```
Epoch: 100; Error: 13.458320123952326;
Epoch: 200; Error: 17.80867288745851;
Epoch: 300; Error: 7.433569857986825;
Epoch: 400; Error: 15.40960944571329;
Epoch: 500; Error: 11.317734053148037;
Epoch: 600; Error: 10.603515381348357;
Epoch: 700; Error: 13.191694198393776;
Epoch: 800; Error: 12.968243318133414;
Epoch: 900; Error: 8.940315633287401;
Epoch: 1000; Error: 5.216232905957388;
Epoch: 1100; Error: 7.232253380853884;
Epoch: 1200; Error: 5.712953011278708;
Epoch: 1300; Error: 11.670225025515498;
Epoch: 1400; Error: 9.73595638975371;
Epoch: 1500; Error: 7.040613744361959;
Epoch: 1600; Error: 12.314251532708436;
Epoch: 1700; Error: 14.474346679071562;
Epoch: 1800; Error: 7.587203693530311;
Epoch: 1900; Error: 11.271178093709597;
Epoch: 2000; Error: 13.368675068553793;
The maximum number of train epochs is reached
```

Рис. 17 Результат виконання програми

На рис. 17 зображено процес навчання мережі. На 100 епосі відбулось 13.45 помилки, на 200 епосі відбулось 17.80 помилки, і так далі, на 2000 епосі відбулось 13.36 помилки,. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

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()
```

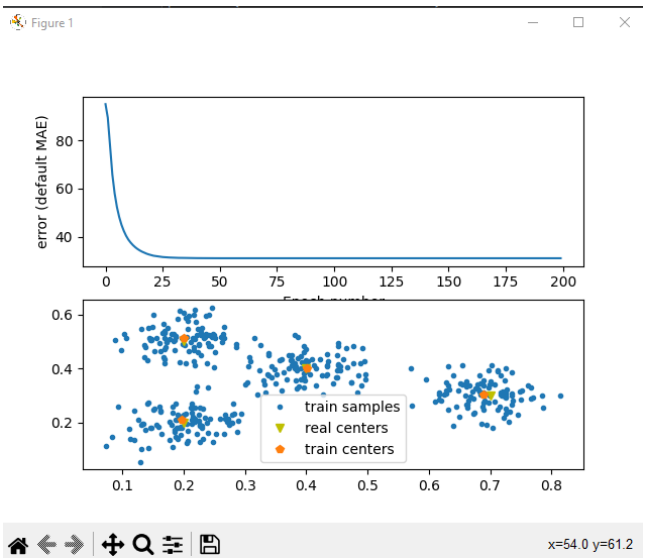

Рис. 18 Результат виконання програми

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 само організується

Таблиця

№ 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

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

skv = 0.03
centr = np.array([[0.1, 0.2], [0.4, 0.3], [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:
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()
```

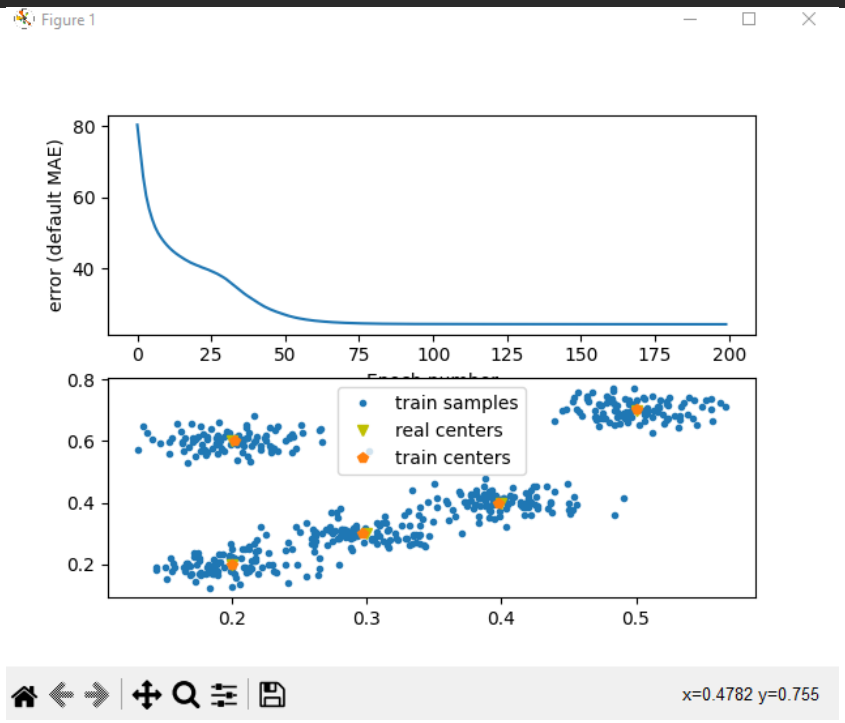

Рис. 19 Результат виконання програми

```
Epoch: 20; Error: 41.283959246160755;
Epoch: 40; Error: 31.329237021092233;
Epoch: 60; Error: 25.36329247625688;
Epoch: 80; Error: 24.463384392016515;
Epoch: 100; Error: 24.335845989924763;
Epoch: 120; Error: 24.312458596609424;
Epoch: 140; Error: 24.305425103271443;
Epoch: 160; Error: 24.2881992850575;
Epoch: 180; Error: 24.273043961974373;
Epoch: 200; Error: 24.26844318851236;
The maximum number of train epochs is reached
```

Рис. 20 Результат виконання програми

На рис. 20 зображено процес навчання мережі. На 20 епосі відбулось 41.28 помилки, помилки і так далі, на 200 епосі відбулось 24.26 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)
```

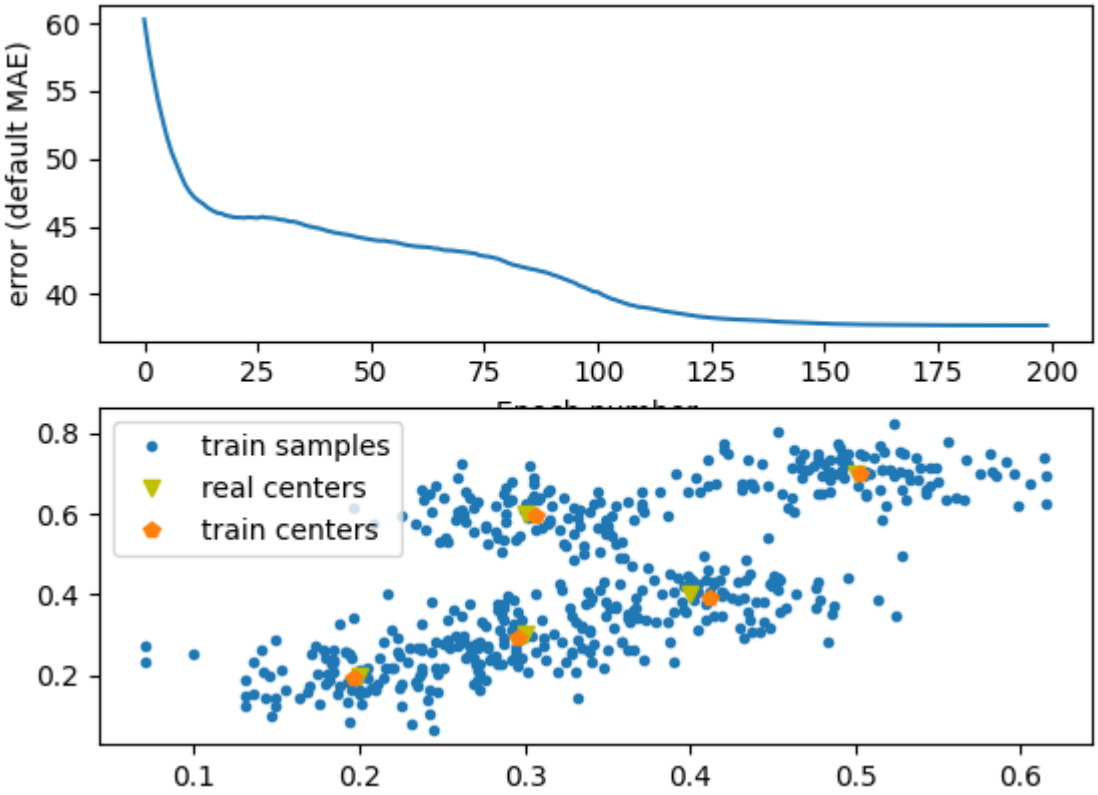

Рис. 21 Результат виконання програми

```
Epoch: 20; Error: 45.73217653445968;
Epoch: 40; Error: 44.82499284201299;
Epoch: 60; Error: 43.570459583975094;
Epoch: 80; Error: 42.48647638310686;
Epoch: 100; Error: 40.207231361042254;
Epoch: 120; Error: 38.53016325498826;
Epoch: 140; Error: 37.98861556660097;
Epoch: 160; Error: 37.76699808236715;
Epoch: 180; Error: 37.71504318937701;
Epoch: 200; Error: 37.70617628328979;
The maximum number of train epochs is reached
```

Рис. 22 Результат виконання програми

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. 33 набагато гірші результати, ніж на рис. 36

Висновок: під час виконання лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовувати прості нейронні мережі**.**

Репозиторій: https://github.com/BOYYYKO/ai/tree/main/lr5