

ЗВІТ

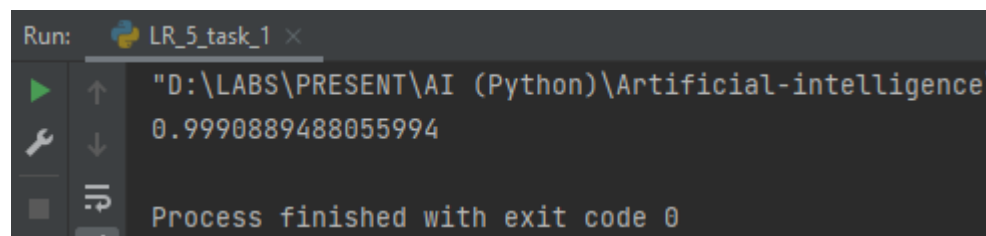
З лабораторної роботи №5

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ студента КН-20-1 навчальної групи

Кірія Даніли Олеговича варіант №6

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

Завдання 1.



```
Run: LR_5_task_1 x
"D:\LABS\PRESENT\AI (Python)\Artificial-intelligence\
0.9990889488055994
Process finished with exit code 0
```

Код 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

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

Завдання 2:

Epoch 0 loss: 0.259	Epoch 200 loss: 0.011	Epoch 400 loss: 0.005
Epoch 10 loss: 0.204	Epoch 210 loss: 0.011	Epoch 410 loss: 0.004
Epoch 20 loss: 0.161	Epoch 220 loss: 0.010	Epoch 420 loss: 0.004
Epoch 30 loss: 0.131	Epoch 230 loss: 0.009	Epoch 430 loss: 0.004
Epoch 40 loss: 0.110	Epoch 240 loss: 0.009	Epoch 440 loss: 0.004
Epoch 50 loss: 0.095	Epoch 250 loss: 0.008	Epoch 450 loss: 0.004
Epoch 60 loss: 0.082	Epoch 260 loss: 0.008	Epoch 460 loss: 0.004
Epoch 70 loss: 0.071	Epoch 270 loss: 0.008	Epoch 470 loss: 0.004
Epoch 80 loss: 0.051	Epoch 280 loss: 0.007	Epoch 480 loss: 0.004
Epoch 90 loss: 0.041	Epoch 290 loss: 0.007	Epoch 490 loss: 0.004
Epoch 100 loss: 0.034	Epoch 300 loss: 0.007	Epoch 500 loss: 0.003
Epoch 110 loss: 0.029	Epoch 310 loss: 0.006	Epoch 510 loss: 0.003
Epoch 120 loss: 0.025	Epoch 320 loss: 0.006	Epoch 520 loss: 0.003
Epoch 130 loss: 0.022	Epoch 330 loss: 0.006	Epoch 530 loss: 0.003
Epoch 140 loss: 0.019	Epoch 340 loss: 0.006	Epoch 540 loss: 0.003
Epoch 150 loss: 0.017	Epoch 350 loss: 0.005	Epoch 550 loss: 0.003
Epoch 160 loss: 0.016	Epoch 360 loss: 0.005	Epoch 560 loss: 0.003
Epoch 170 loss: 0.014	Epoch 370 loss: 0.005	Epoch 570 loss: 0.003
Epoch 180 loss: 0.013	Epoch 380 loss: 0.005	Epoch 580 loss: 0.003
Epoch 190 loss: 0.012	Epoch 390 loss: 0.005	Epoch 590 loss: 0.003
Epoch 200 loss: 0.011	Epoch 400 loss: 0.005	Epoch 600 loss: 0.003

Emily: 0.965
Frank: 0.041

Код 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 KiriiNeuralNetwork:
    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],
    [25, 6],
    [17, 4],
    [-15, -6],
])

all_y_trues = np.array([
    1,
    0,
    0,
    1,
])

network = KiriiNeuralNetwork()
network.train(data, all_y_trues)

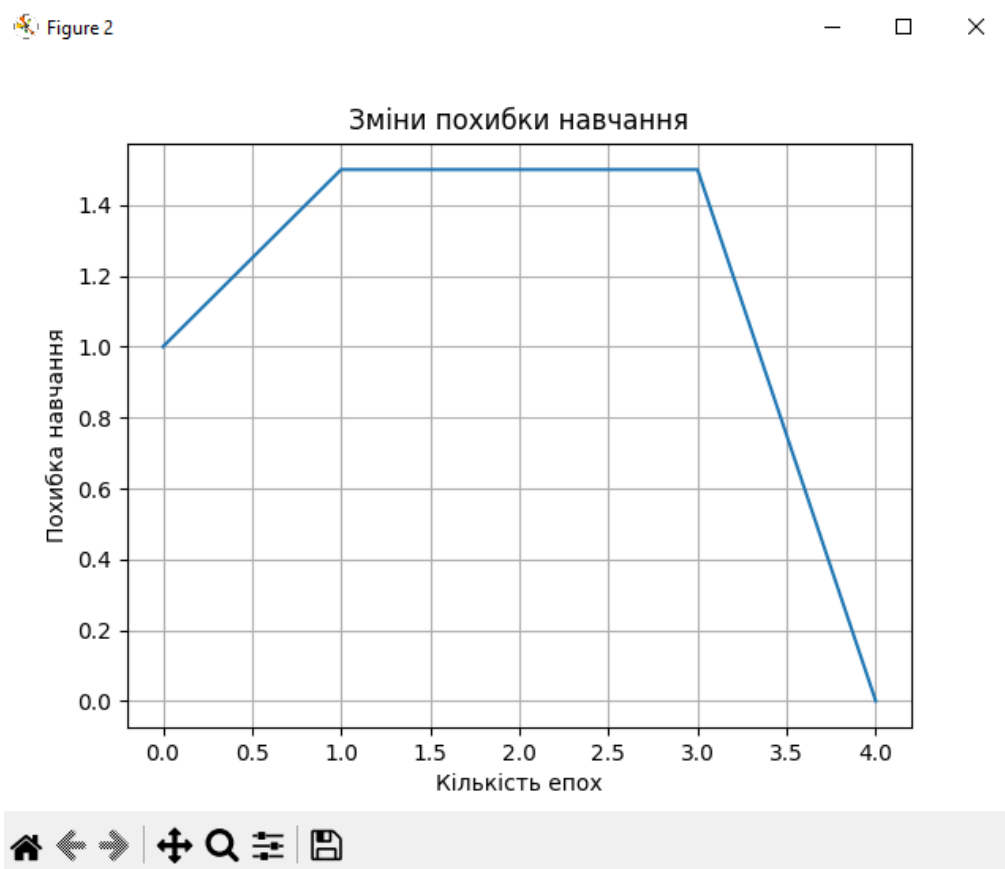
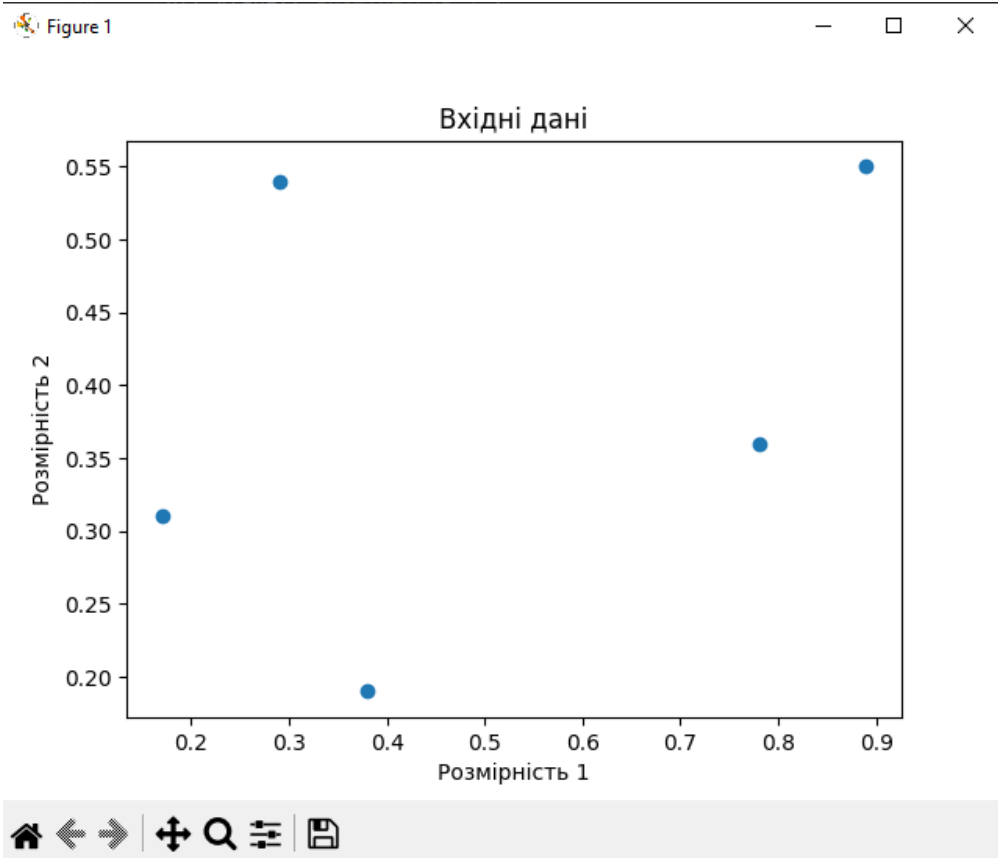
emily = np.array([-7, -3]) # 128 фунтів, 63 дюйма
frank = np.array([20, 2]) # 155 фунтів, 68 дюймов
print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F
print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M

```

У нашому випадку застосовується сигмоїдна функція активації, яка перетворює вихід нейрону до діапазону від 0 до 1. Цей тип функції особливо ефективний у завданнях бінарної класифікації, де потрібно отримати ймовірність виходу моделі. У нейронних мережах прямого поширення виділяються кілька ключових властивостей:

1. Універсальність апроксимації: Мережі можуть наближати будь-яку функцію, якщо в них достатньо нейронів та відповідні ваги.
2. Навчання з вчителем: Мережі можуть навчатися на основі пар даних "вхід-вихід" і коригувати свої ваги для досягнення бажаного результату.
3. Автоматичне вивчення ознак: Вони можуть автоматично вивчати корисні ознаки з вхідних даних під час тренування.
4. Гнучкість в розв'язанні різних завдань: Мережі можуть застосовуватися до різних типів завдань, таких як класифікація, регресія, генерація зображень та інші.
5. Автоматична адаптація до навчальних даних: Вони адаптуються до різних вхідних даних без необхідності ручної переконфігурації.

Завдання 3:



Код програми:

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

matplotlib.use('TkAgg')
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("Похибка навчання")
plt.title("Зміни похибки навчання")
plt.grid()
plt.show()
```

Завдання 4:

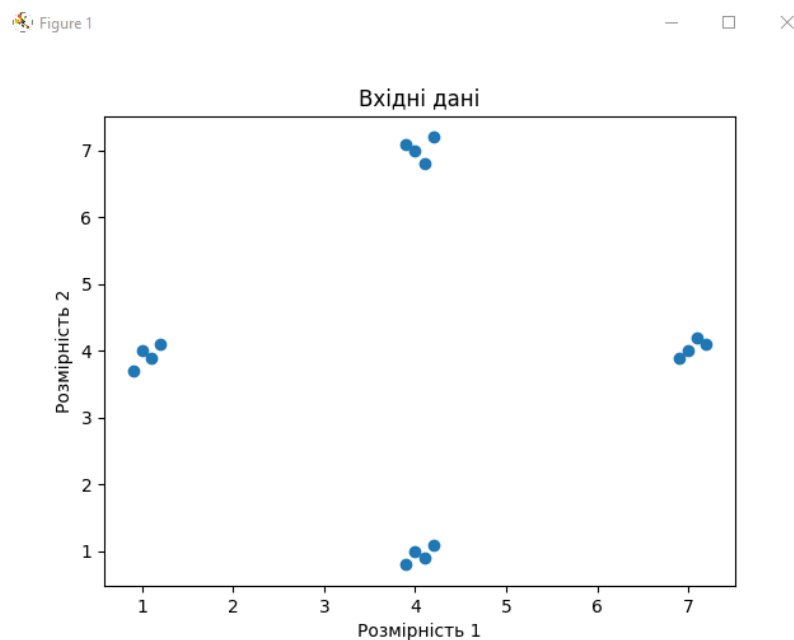
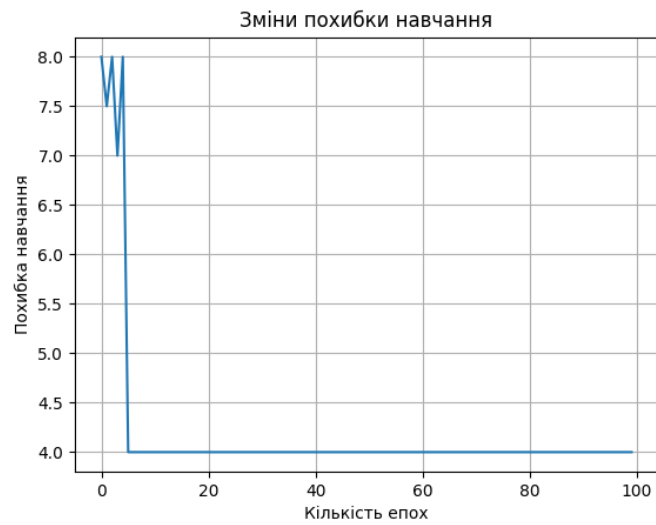


Figure 2



```
"D:\LABS\PRESENT\AI (Python)\Artificial-intel
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.]
```

Код програми:

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

matplotlib.use('TkAgg')
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("Похибка навчання")
plt.title("Зміни похибки 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])

```

Отримані результати в терміналі свідчать про те, що тренування нейронної мережі не призвело до зниження помилки протягом 100 епох, і навіть при максимальній кількості епох помилка залишалася на однаковому рівні - 4.0. Це може вказувати на те, що модель не може адаптуватися до тренувальних даних. Можливі причини включають неправильний вибір параметрів навчання або недостатню кількість та якість тренувальних даних.

Завдання 5:

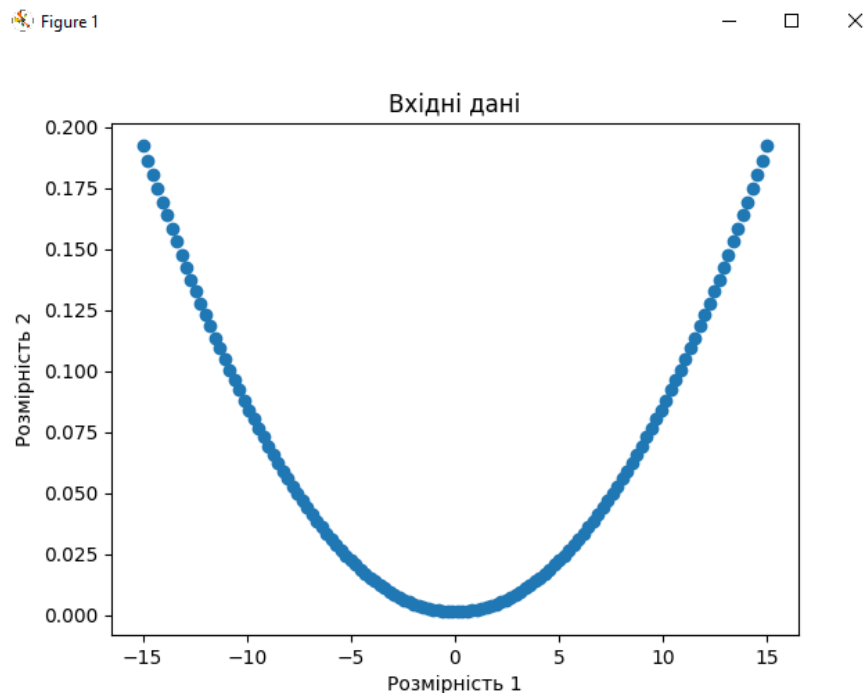


Figure 2

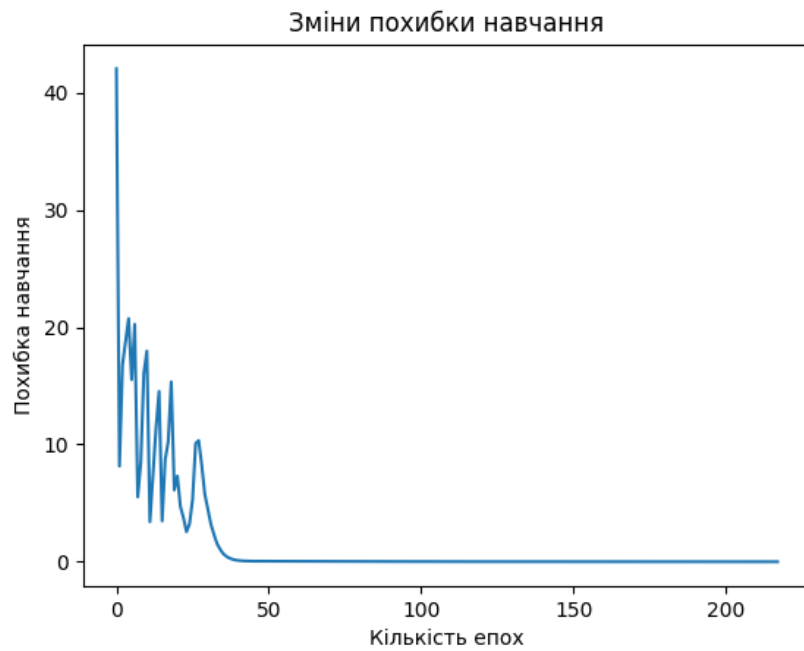
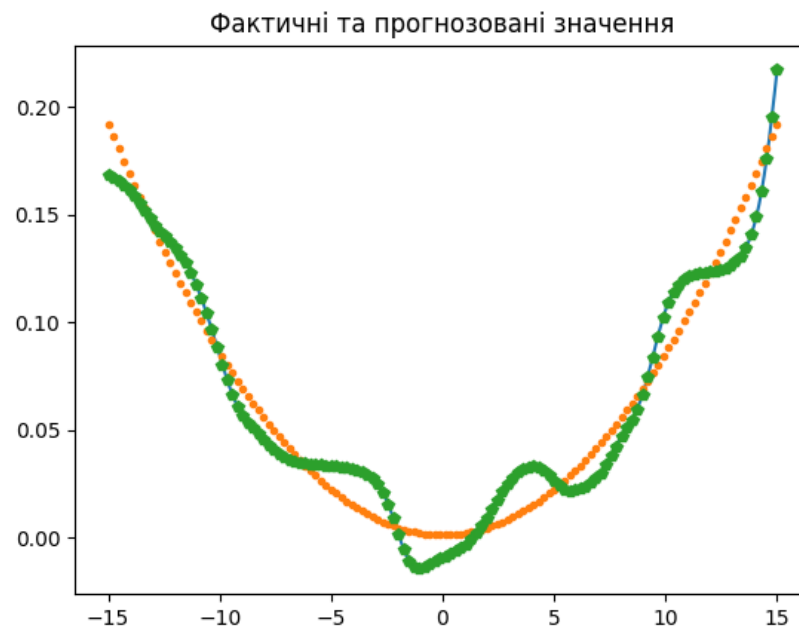


Figure 3



```
"D:\LABS\PRESENT\AI (Python)\Artificial-intel  
Epoch: 100; Error: 0.023895985809777965;  
Epoch: 200; Error: 0.010960122880325638;  
The goal of learning is reached
```

Код програми:

```

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

matplotlib.use('TkAgg')

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("Похибка навчання")
plt.title("Зміни похибки 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 та прогнозовані значення')
plt.show()

```

Завдання 6:

Варіант 6		$y = 2x^2 + 10$	
Номер варіанта	Багатошаровий перцептрон		
	Кількість шарів	Кількості нейронів у шарах	y
6	2	10-1	

Figure 1

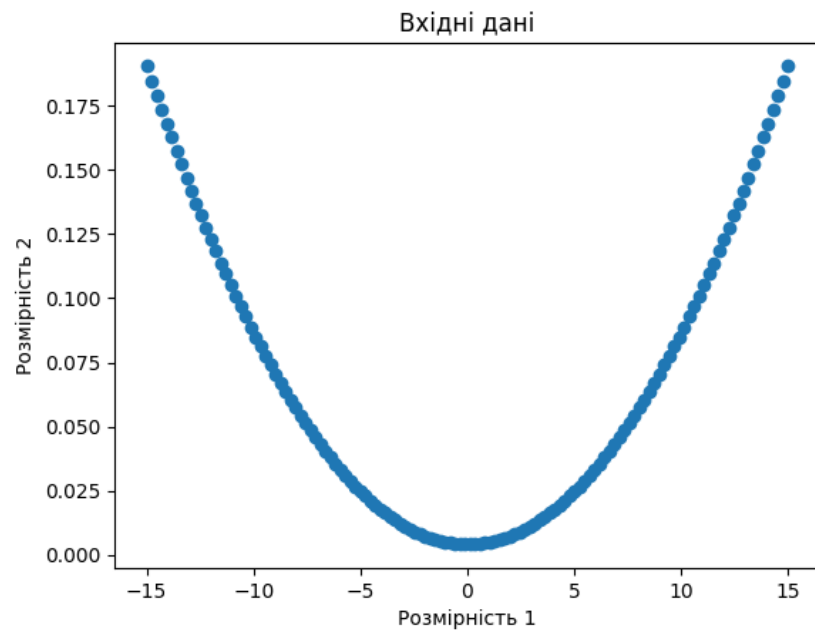


Figure 2

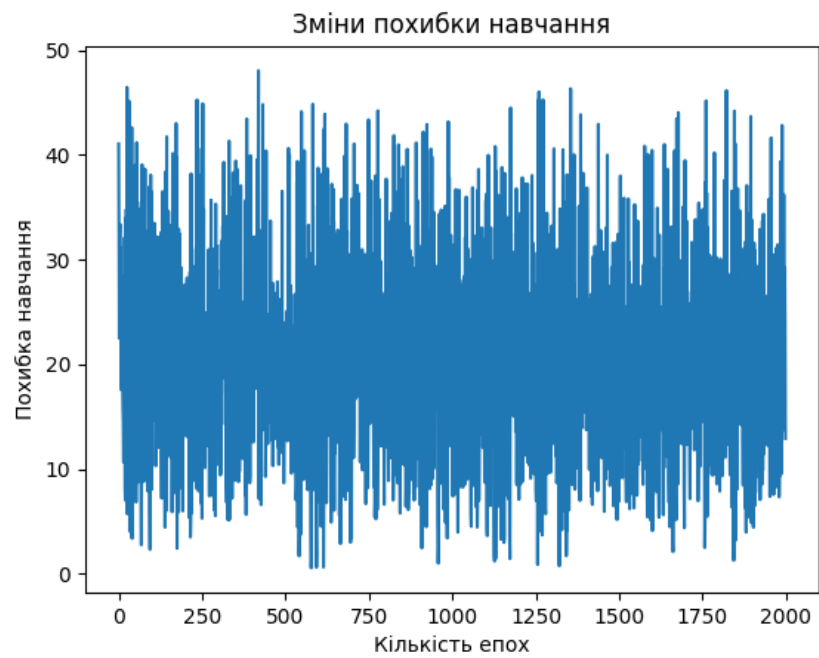
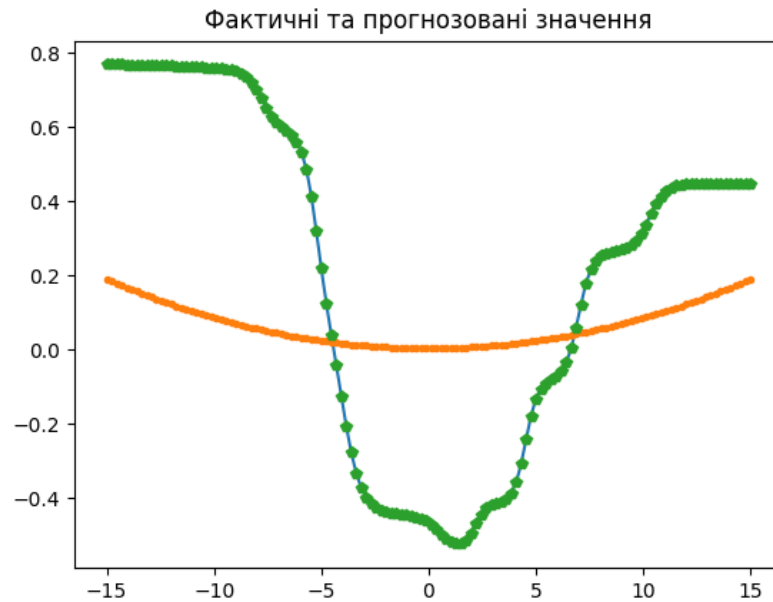


Figure 3



```
Epoch: 1400; Error: 24.659619380019134;
Epoch: 1500; Error: 6.9180357769449135;
Epoch: 1600; Error: 40.45115513096394;
Epoch: 1700; Error: 15.203960173466415;
Epoch: 1800; Error: 18.2690890169693;
Epoch: 1900; Error: 10.126266071074586;
Epoch: 2000; Error: 12.918991135763582;
The maximum number of train epochs is reached
```

Код програми:

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

matplotlib.use('TkAgg')

min_val = -15
max_val = 15
num_points = 130
x = np.linspace(min_val, max_val, num_points)
y = 2 * np.square(x) + 10
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, 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("Похибка навчання")
plt.title("Зміни похибки навчання")

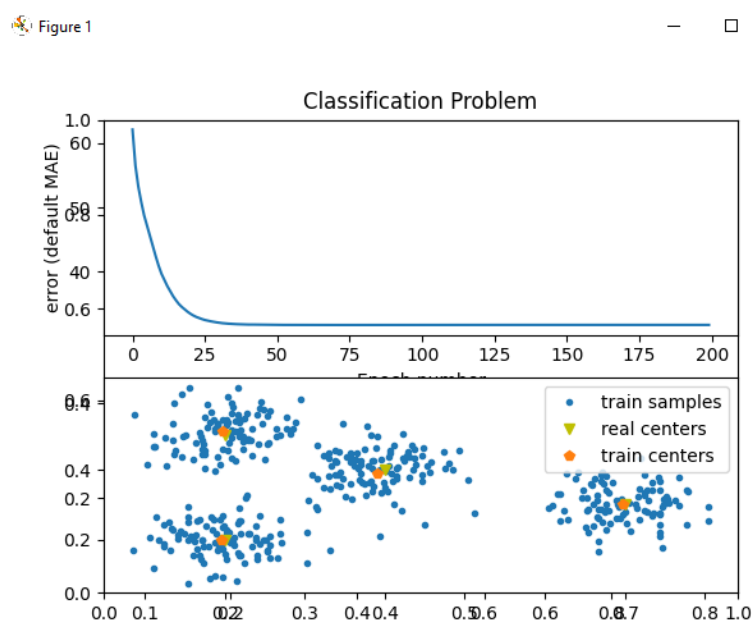
x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred = nn.sim(x_dense.reshape(x_dense.size, 1)).reshape(x_dense.size)

plt.figure()
plt.plot(x_dense, y_dense_pred, "-", x, y, '.', x, y_pred, 'p')
plt.title('Фактичні та прогнозовані значення')
plt.show()

```

Подані дані свідчать про успішне тренування нейронної мережі, де помилка навчання систематично зменшується протягом 300 епох. Після досягнення поставленої мети навчання графіки демонструють ефективність моделі у прогнозуванні вихідної функції для вхідних даних, що підтверджується графіком "Фактичні та прогнозовані значення".

Завдання 7:



```
"D:\LABS\PRESENT\AI (Python)\Artificial-intell
Epoch: 20; Error: 33.83747708573249;
Epoch: 40; Error: 31.86462278593691;
Epoch: 60; Error: 31.788972096260327;
Epoch: 80; Error: 31.798416429761147;
Epoch: 100; Error: 31.798516673768937;
Epoch: 120; Error: 31.798652788523082;
Epoch: 140; Error: 31.798692409553624;
Epoch: 160; Error: 31.798701544933195;
Epoch: 180; Error: 31.79870346414972;
Epoch: 200; Error: 31.798703848665507;
The maximum number of train epochs is reached
```

Код програми:

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

matplotlib.use('TkAgg')

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)

net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)
error = net.train(inp, epochs=200, show=20)

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()
```

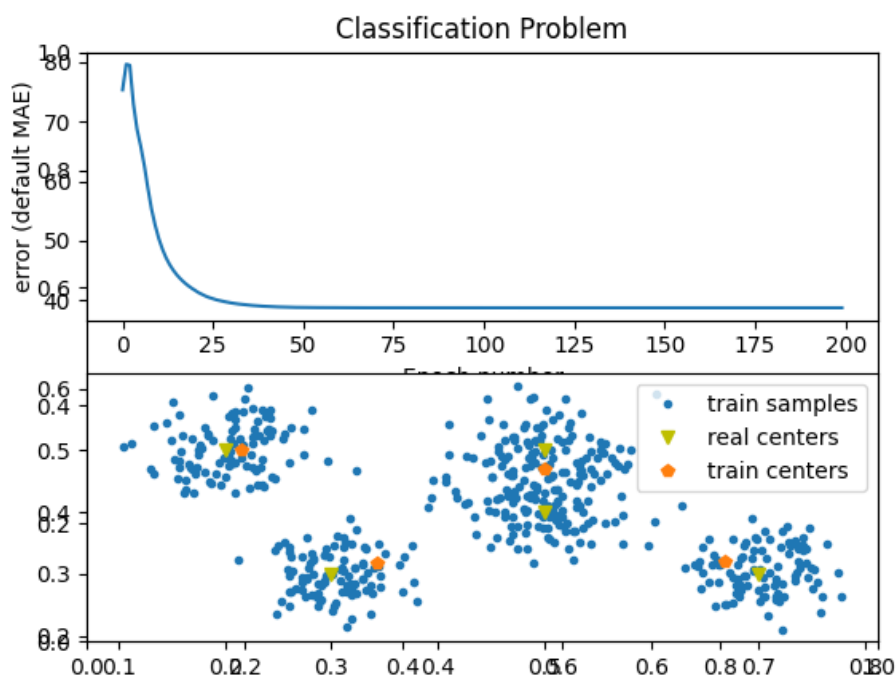
Отримані результати свідчать про те, що нейронна мережа була навчена для проведення кластеризації вхідних даних. Протягом 200 епох відзначається невелике зменшення помилки, і тренування завершується, коли досягнута максимальна кількість епох. Графіки динаміки помилки та розподілу центрів

кластерів демонструють, що мережа успішно вивчила відокремлювати кластери вхідних даних. Метрика MAE (або середня абсолютна похибка) використовується для вимірювання середньої абсолютної величини відхилень між прогнозованими та фактичними значеннями.

Завдання 8:

Варіант 6	[0.3, 0.3], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,04
-----------	--	------

Figure 1



```
"D:\LABS\PRESENT\AI (Python)\Artificial-intelli
Epoch: 20; Error: 42.0834746150195;
Epoch: 40; Error: 39.004639162580446;
Epoch: 60; Error: 38.77093486344826;
Epoch: 80; Error: 38.75263636295486;
Epoch: 100; Error: 38.750734538074354;
Epoch: 120; Error: 38.75046845336874;
Epoch: 140; Error: 38.75042345785313;
Epoch: 160; Error: 38.75041519051244;
Epoch: 180; Error: 38.75041362737846;
Epoch: 200; Error: 38.75041332955868;
The maximum number of train epochs is reached
```

Код програми:

```

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

matplotlib.use('TkAgg')

skv = 0.04
centr = np.array([[0.3, 0.3], [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([centr + r for r in rand_norm])
inp.shape = (100 * 5, 2)
rand.shuffle(inp)

net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)
error = net.train(inp, epochs=200, show=20)

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()

```

Висновок з результатів вказує на те, що тренування нейронної мережі для кластеризації було завершено, досягнувши максимальної кількості епох (200). Зменшення помилки відбулося лише на початкових етапах, а подальше тренування призвело до її стабільності.

- Вплив кількості нейронів і кластерів:

З погляду помилки тренування, перевищення кількості кластерів, які намагається навчити мережа, може спричинити збільшення помилки через надмірну адаптацію до навчальних даних. У цьому випадку може виникнути неправильний вибір кількості кластерів, оскільки мережа старається апроксимувати більше кластерів, ніж фактично присутні в даних.

- Вплив розкиду вхідних даних:

Розкид вхідних даних може впливати на точність кластеризації, і в даному випадку розкид є достатньо великим, що ускладнює відрізнєння реальних центрів кластерів. Це може призвести до менш точної апроксимації кластерів мережею. Обидва ці фактори підкреслюють важливість вибору оптимальної кількості

кластерів та уважності до характеристик вхідних даних при тренуванні нейронних мереж для кластеризації.

Висновки: в ході виконання лабораторної роботи було, використовуючи спеціалізовані бібліотеки та мову програмування Python, отримано практичні навички зі створення та застосовування простих нейронних мереж.

GitHub: <https://github.com/invincibleee/Artificial-intelligence.git>