ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

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

Хід роботи:

Посилання на репозиторій: https://github.com/AlexanderSydorchuk/AI-Lab5

Завдання 2.1. Створити простий нейрон лісів.

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):
        # 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])
bias = 4
n = Neuron(weights, bias)

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

Результат виконання програми:

					ДУ «Житомирська політехніка».19.121.25.000 - Лр1		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Сидорчук О.С.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 5
Перевір.		Голенко М.Ю					
Керівник						ФІКТ Гр. ІПЗ-20-2[2]	
Н. контр.							
Зав. каф.							

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\
0.9990889488055994

Process finished with exit code 0
```

Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини

Лістинг програми:

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


def deriv_sigmoid(x):
    # Похідна від sigmoid: f'(x) = f(x) * (1 - f(x))
    fx = sigmoid(x)
    return fx * (1 - fx)


def mse_loss(y_true, y_pred):
    # y_true и y_pred є масивами numpy з одинаковою довжиною
    return ((y_true - y_pred) ** 2).mean()


class SydorchukNeuralNetwork:
    def __init__(self):
        # Вага
        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()

        # Зміщення
        self.b1 = np.random.normal()
        self.b2 = np.random.normal()
        self.b3 = np.random.normal()

    def feedforward(self, x):
        # x є масивом numpy з двома елементами
        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  # кількість циклів у всьому наборі даних

        for epoch in range(epochs):
            for x, y_true in zip(data, all_y_trues):
                # --- Виконуємо зворотній зв'язок (ці значання нам потрібні в по-
```

```
дальшому )
                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

                # --- Підрахунок часткових похідних
                # --- Найменування: d_L_d_w1 означає "частково L / частково w1"
                d_L_d_ypred = -2 * (y_true - y_pred)

                # Нейрон o1
                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)

                # Нейрон h1
                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)

                # Нейрон h2
                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)

                # Оновлюємо вагу і зміщення
                # Нейрон 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

                # Нейрон 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

                # Нейрон 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))


# Задання набору даних
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 = SydorchukNeuralNetwork()
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 дюймів
print("Emily: %.3f" % network.feedforward(emily))  # +-0.966 - F
print("Frank: %.3f" % network.feedforward(frank))  # +-0.038 - M
```

Результат виконання програми:

```
Epoch 900 loss: 0.002
Epoch 910 loss: 0.002
Epoch 920 loss: 0.002
Epoch 930 loss: 0.002
Epoch 940 loss: 0.002
Epoch 950 loss: 0.002
Epoch 960 loss: 0.002
Epoch 970 loss: 0.002
Epoch 980 loss: 0.002
Epoch 990 loss: 0.001
Emily: 0.967
Frank: 0.039

Process finished with exit code 0
```

Висновок: Функція активації необхідна для підключення непов’язаних вхідних даних з виводом з простою формою. Нейронні мережі прямого поширення можуть передбачати відповідь, використовуючи попередньо згадані функції активації.

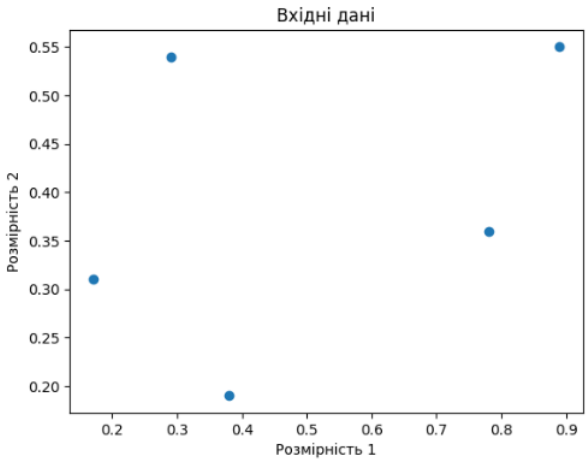
Завдання 2.3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

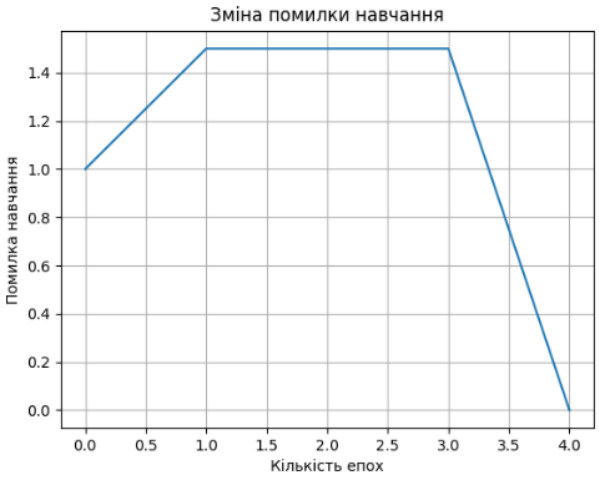
Лістинг програми:

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

# Завантаження вхідних даних
text = np.loadtxt('data_perceptron.txt')
# Поділ 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')
plt.show()

# 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 помилки навчання')
plt.grid()
plt.show()
```

Результат виконання програми:

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\
The goal of learning is reached

Process finished with exit code 0
```





На рисунках зображені вхідні дані до перцептрону та графік навчання перцептрону. Як видно з графіка, кількість помилок спочатку росла, а потім різко впала.

Завдання 2.4. Побудова одношарової нейронної мережі

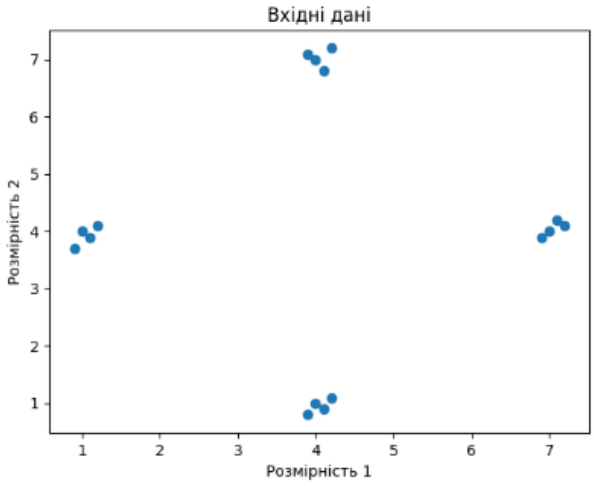
Лістинг програми:

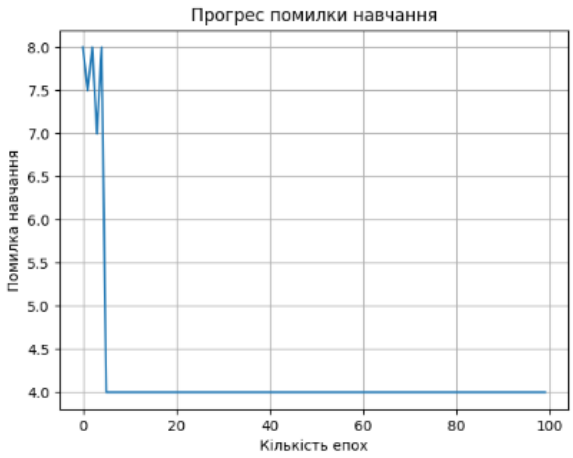
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# 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')
plt.show()

# 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('Прогрес помилки 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])
```

Результат виконання програми:

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\Lab5\venv\Sc
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
```





На рисунках зображені вхідні дані до перцептрону та графік навчання перцептрону. Як видно з графіка, було набагато більше помилок на початку навчання, проте згодом їх кількість стабілізувалася на рівні 4 за епоху.

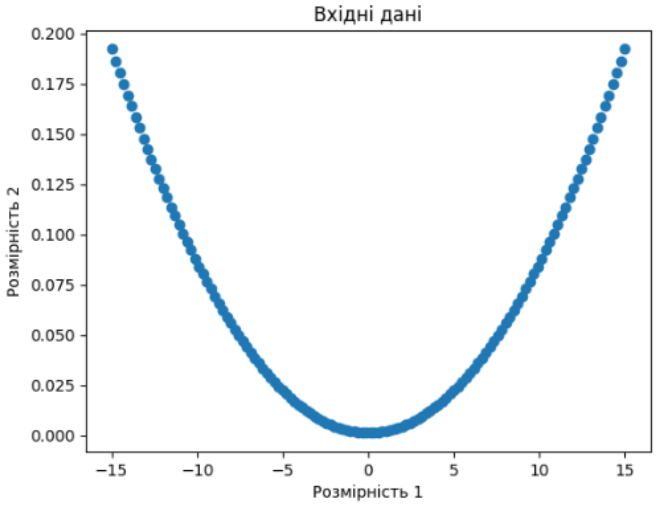
Завдання 2.5.

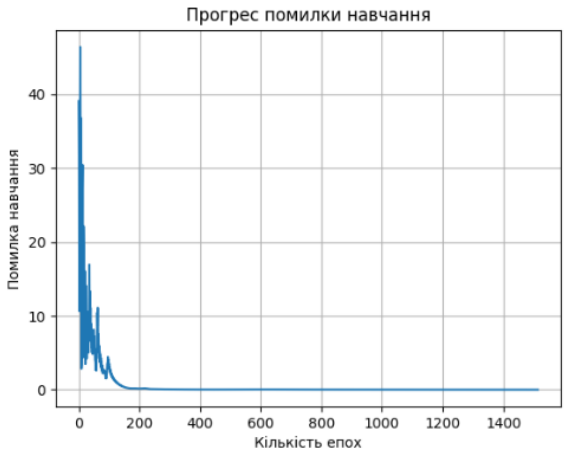
Лістинг програми:

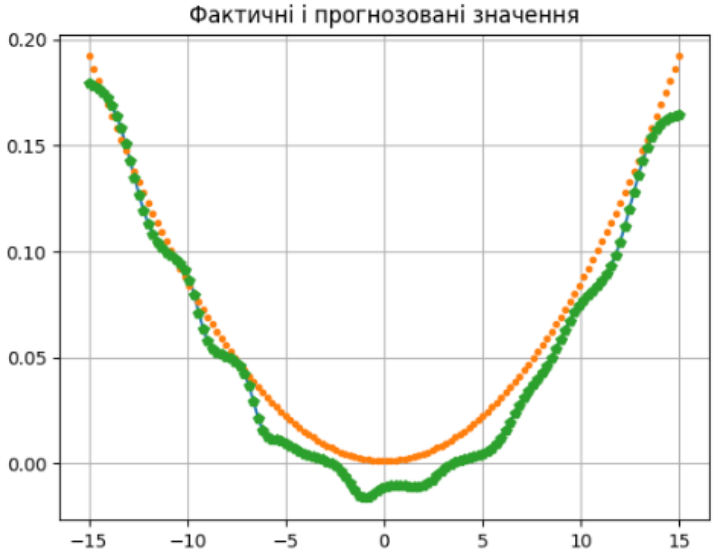
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# 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')

nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])

# Задання 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')
plt.grid()
plt.show()
```

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

Результат виконання програми:








```
Epoch: 600; Error: 0.045225805883883495;
Epoch: 700; Error: 0.03809760128018867;
Epoch: 800; Error: 0.029222248071033213;
Epoch: 900; Error: 0.025851985948475213;
Epoch: 1000; Error: 0.022824825294333943;
Epoch: 1100; Error: 0.019037997968621663;
Epoch: 1200; Error: 0.015746245025165802;
Epoch: 1300; Error: 0.013347420337683607;
Epoch: 1400; Error: 0.01160029326085767;
Epoch: 1500; Error: 0.01023710910367336;
The goal of learning is reached

Process finished with exit code 0
```

На початку була велика кількість помилок, але починаючи з 200 епохи, помилки знизилися майже до нуля

Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

Варіант 24	$y = 2x^2+2x+4$

Номер варіанта	Багатошаровий персептрон	
	Кількість шарів	Кількості нейронів у шарах
24	2	5-1

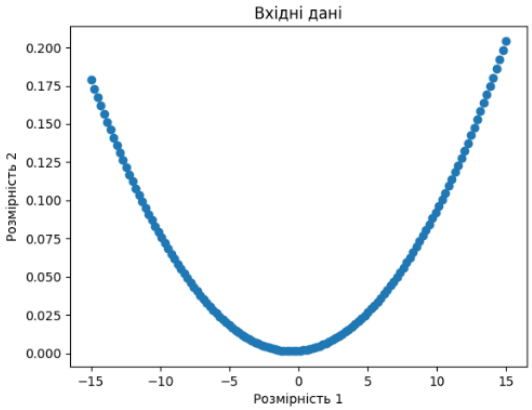
Лістинг програми:

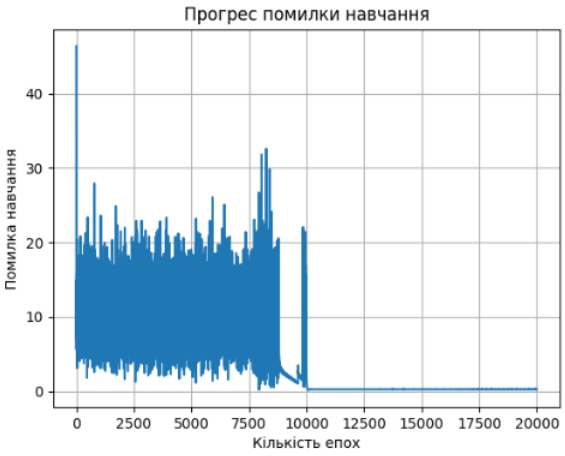
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# 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) + 2 * 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')

nn = nl.net.newff([[min_val, max_val]], [5, 1])

# Задання градієнтного спуску як навчального алгоритму
nn.trainf = nl.train.train_gd

# Тренування нейронної мережі
error_progress = nn.train(data, labels, epochs=20000, show=1000, 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')
plt.grid()
plt.show()
```

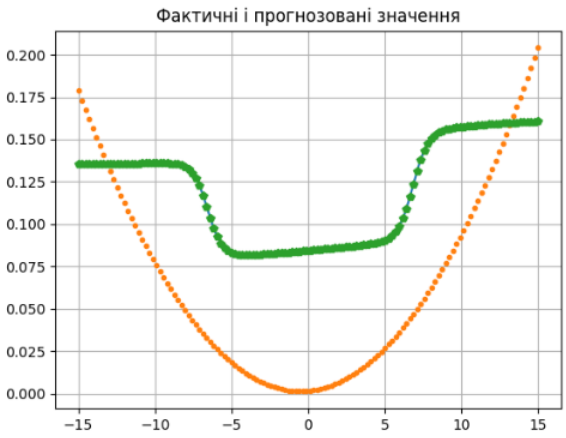
```
# Побудова графіка результатів
x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred = nn.sim(x_dense.reshape(x_dense.size, 1)).reshape(x_dense.size)

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

Результат виконання програми:

```
Epoch: 14000; Error: 0.2716838377260744;
Epoch: 15000; Error: 0.2867483483847824;
Epoch: 16000; Error: 0.2793064264510342;
Epoch: 17000; Error: 0.2897260455856103;
Epoch: 18000; Error: 0.2951085343423723;
Epoch: 19000; Error: 0.2629434157763172;
Epoch: 20000; Error: 0.2693628189223788;
The maximum number of train epochs is reached
```







Точність тестування занадто низька, на мою думку, потрібно більше епох та відповідна кількість шарів із нейронами, а також підвищити вимоги до цільового значення.

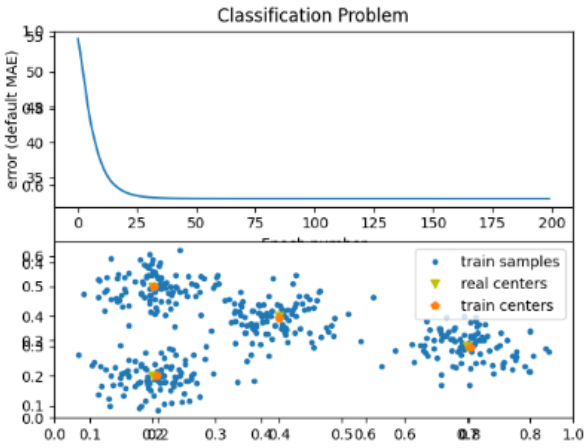
Завдання 2.7. Побудова нейронної мережі на основі карти Кохонена, що самоорганізується

Лістинг програми:

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

skv = 0.05
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=20)

# Plot results:
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()
```

Результат виконання програми:

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\Lab5\venv\Sc
Epoch: 20; Error: 33.118789460985454;
Epoch: 40; Error: 32.074875688028904;
Epoch: 60; Error: 32.027141001828824;
Epoch: 80; Error: 32.02293205578408;
Epoch: 100; Error: 32.022412439009535;
Epoch: 120; Error: 32.022334463484654;
Epoch: 140; Error: 32.022322802193464;
Epoch: 160; Error: 32.02232106335591;
Epoch: 180; Error: 32.022320804710695;
Epoch: 200; Error: 32.02232076631726;
The maximum number of train epochs is reached
```



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

Варіант 24	[0.2, 0.1], [0.3, 0.3], [0.7, 0.3], [0.2, 0.5], [0.6, 0.5]	0,07

Лістинг програми:

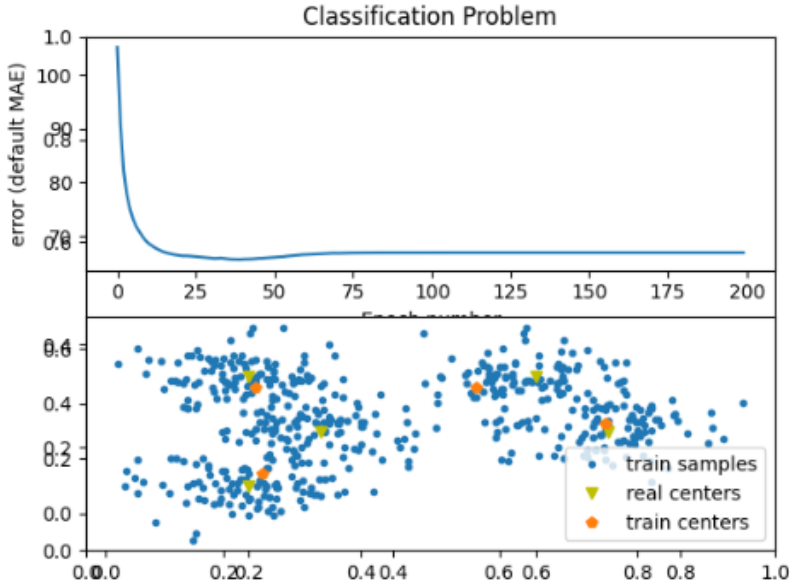
```
import numpy as np
import neurolab as nl
import numpy.random as rand
import pylab as pl

skv = 0.07
centr = np.array([
    [0.2, 0.1],
    [0.3, 0.3],
    [0.7, 0.3],
    [0.2, 0.5],
    [0.6, 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 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:
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()
```

Результат виконання програми:

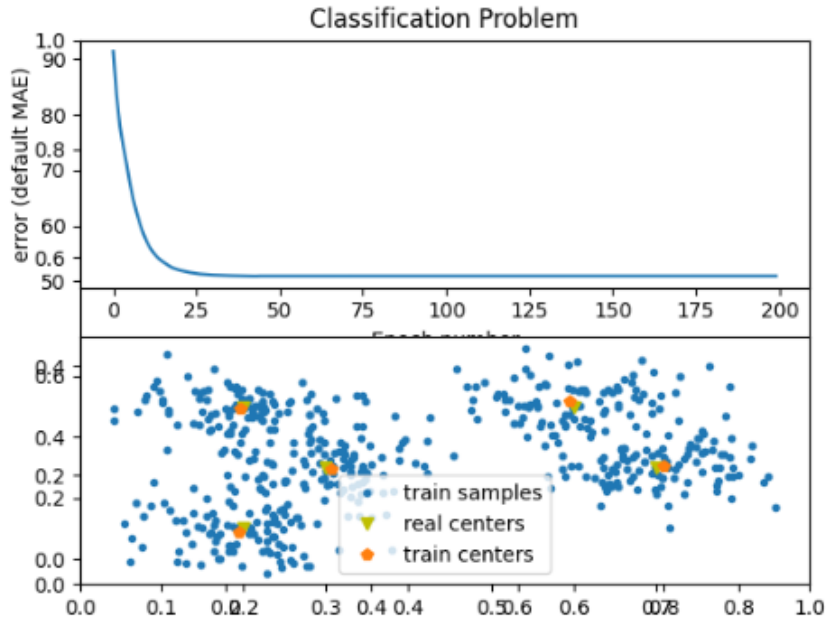
Нейронна мережа Кохонена з 2 входами та 4 нейронами

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\Lab5\venv\Sc
Epoch: 20; Error: 66.43074477545042;
Epoch: 40; Error: 65.60012102514712;
Epoch: 60; Error: 66.51316448854774;
Epoch: 80; Error: 66.86676895689803;
Epoch: 100; Error: 66.88538202011405;
Epoch: 120; Error: 66.88589553698576;
Epoch: 140; Error: 66.88591587234532;
Epoch: 160; Error: 66.88591683219539;
Epoch: 180; Error: 66.88591688046105;
Epoch: 200; Error: 66.88591688294592;
The maximum number of train epochs is reached
```



Нейронна мережа Кохонена з 2 входами та 5 нейронами

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\Lab5\venv\
Epoch: 20; Error: 52.17275959082076;
Epoch: 40; Error: 50.93966017441599;
Epoch: 60; Error: 50.97911287599409;
Epoch: 80; Error: 50.95361610776229;
Epoch: 100; Error: 50.954879329506426;
Epoch: 120; Error: 50.95500271839686;
Epoch: 140; Error: 50.955014961684654;
Epoch: 160; Error: 50.95501619036092;
Epoch: 180; Error: 50.95501631459905;
Epoch: 200; Error: 50.95501632721807;
The maximum number of train epochs is reached
```



Більша кількість нейронів для заданого числа кластерів значно знизила кількість помилок. Для досягнення кращих результатів потрібно більше і краще тренуватись.

Висновки: в ході лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python, було навчено створювати та застосовувати прості нейронні мережі.