ЛАБОРАТОРНА РОБОТА № 5

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

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

Завдання 1:

```python
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])  # x1 = 2, x2 = 3
print(n.feedforward(x))
```

```
"C:\Users\ivanp\Desktop\Python A.I\Lab 5\venv\S
 A.I\Lab 5\LR_5_task_1.py"
0.9990889488055994
```

Рис.5.1 – Результат роботи нейрона

					ДУ «Житомирська політехніка». 23.121.8.000 – Лр5		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Палій І.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		Голенко М.Ю.					
Керівник						ФІКТ Гр. ІПЗ-20-2	
Н. контр.							
Зав. каф.							

Завдання 2:

```
import numpy as np


def sigmoid(x):
    # Функція активації sigmoid:: f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))


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 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)


class PaliiNeuralNetwork:

    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):
        """
        - data is a (n x 2) numpy array, n = # of samples in the dataset.
        - all_y_trues is a numpy array with n elements.
            Elements in all_y_trues correspond to those in data.
        """
        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

            # --- Підраховуємо загальні втрати в кінці кожної фази
            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 = PaliiNeuralNetwork()
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 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: 0.002
Emily: 0.963
Frank: 0.041
```

Рис.5.2 – Результат навчання нейронної мережі

Функція активації необхідна для підключення непов'язаних вхідних даних з виводом з простою формою. Нейронні мережі прямого поширення дозволяють передбачати відповідь, використовуючи функції активації.

Завдання 3:

```python
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')

# 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()
```

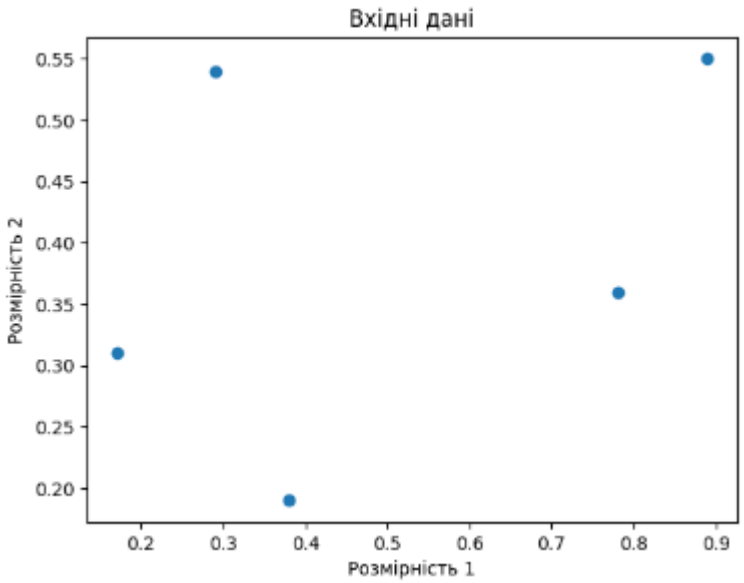

Рис.5.3 – Вхідні дані до перцептрону

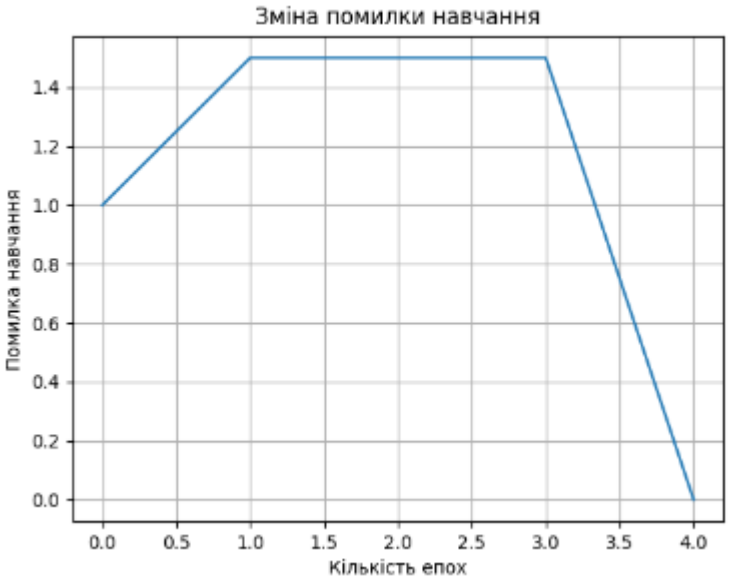

Рис.5.4 – Навчання перцептрону

Завдання 4:

```python
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)

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])
```

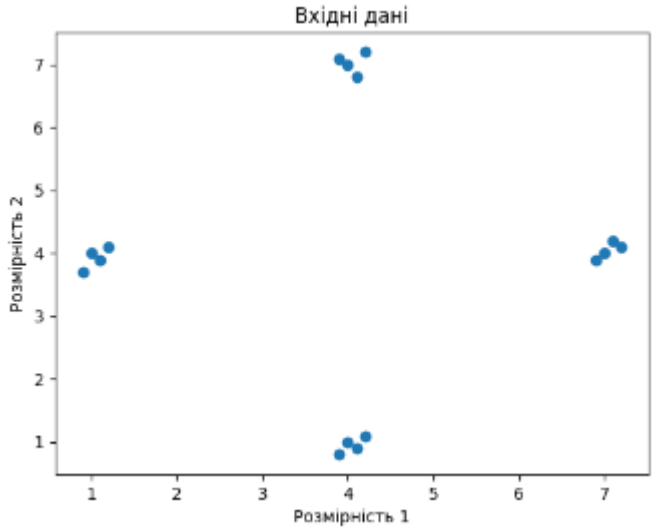

Рис.5.5 – Вхідні дані до нейронної мережі

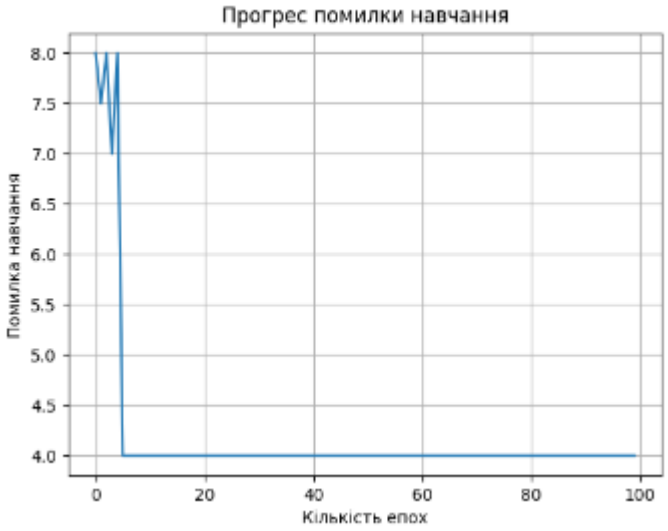

Рис.5.6 – Навчання мережі

```
"C:\Users\ivanp\Desktop\Python A.I\Lab 5\venv\Scr
 A.I\Lab 5\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.5.7 – Тестові результати

Завдання 5:

```
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()
```

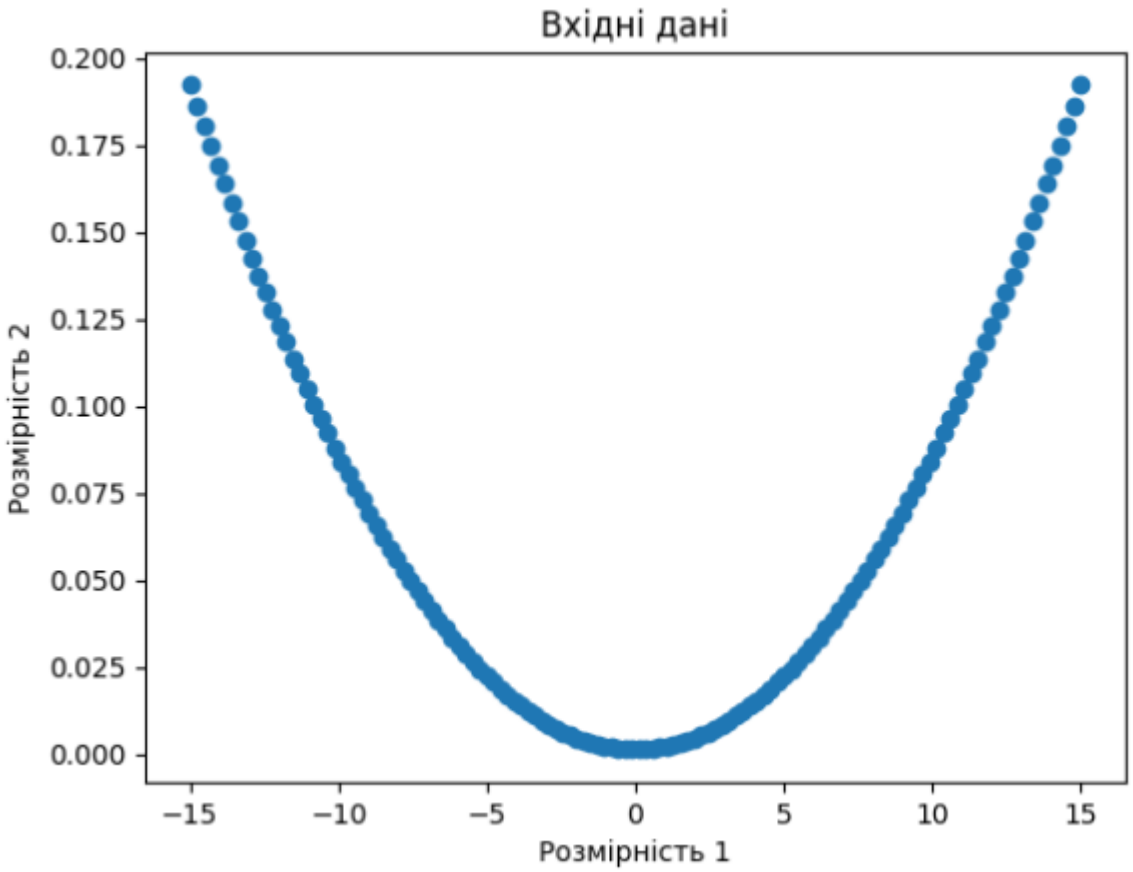

Рис.5.8 – Дані рівняння $3x^2+5$

```
"C:\Users\ivanp\Desktop\Python A.I\Lab 5\venv\
Epoch: 100; Error: 0.013528078703804 82;
Epoch: 200; Error: 0.010573439205489239;
The goal of learning is reached
```

Рис.5.9 – Звітність про навчання по епохам

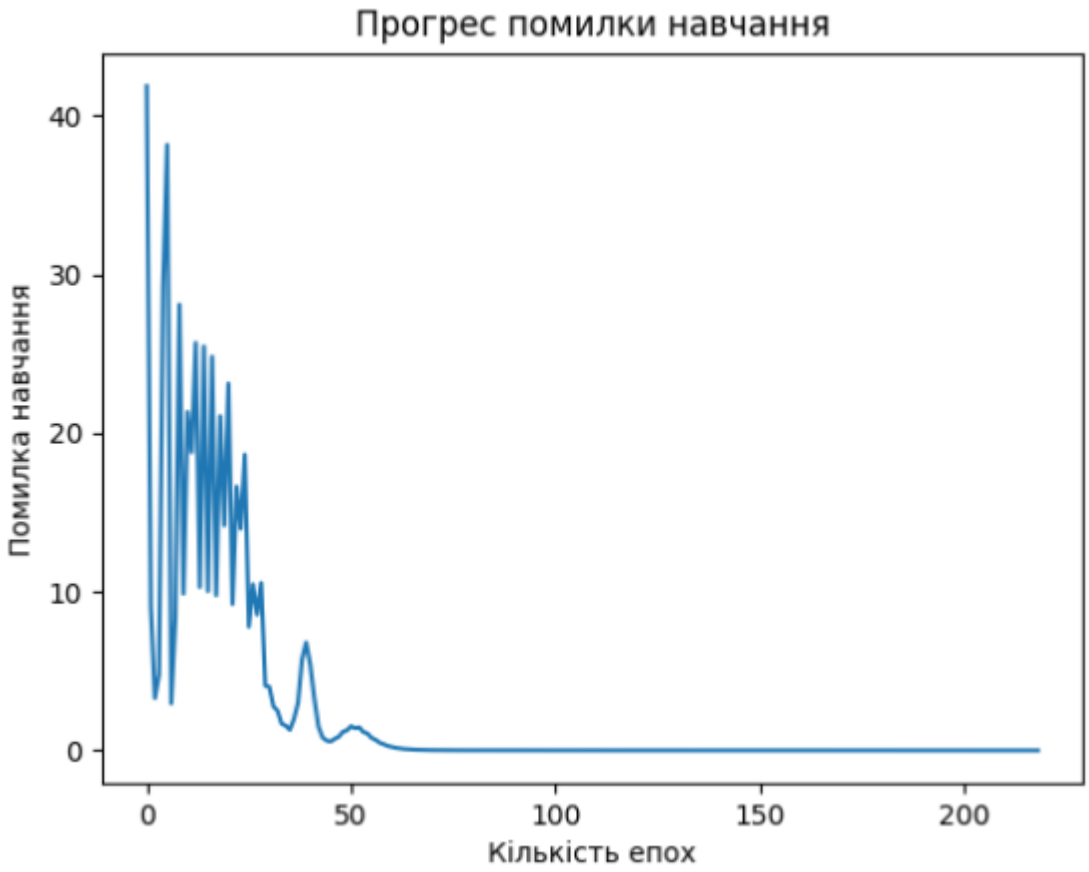

Рис.5.10 – Графік навчання мережі

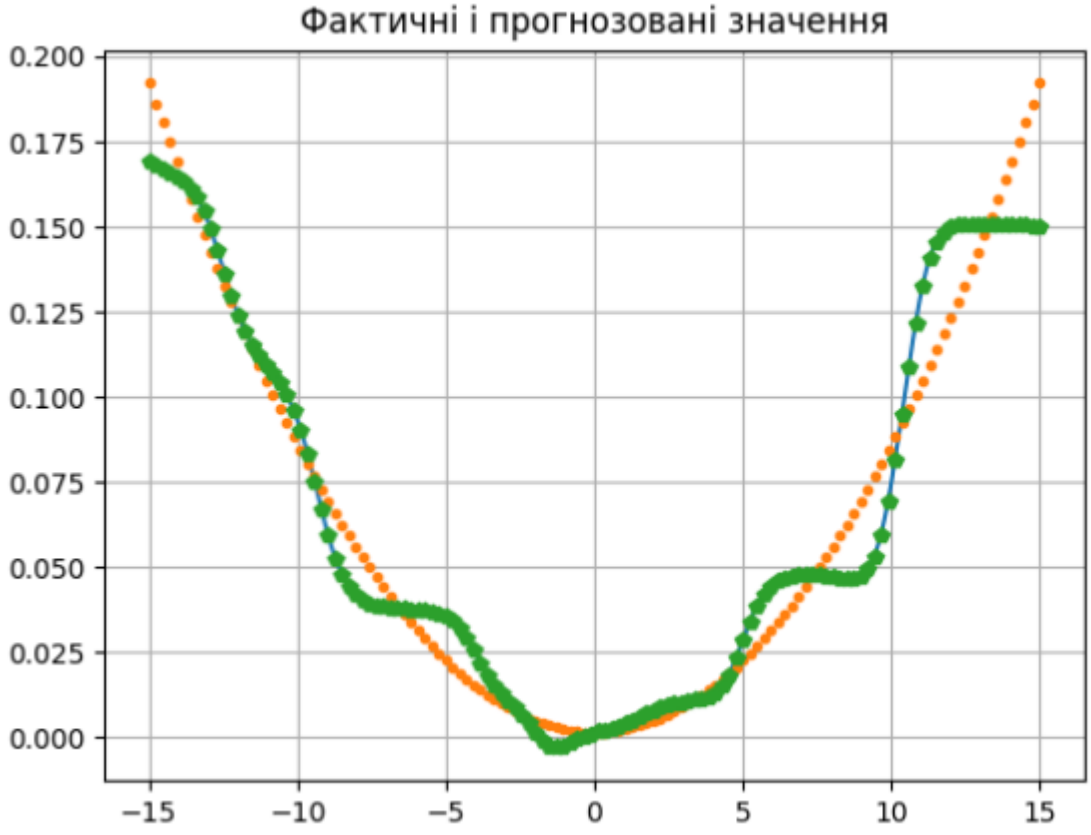

Рис.5.11 – Графік-порівняння істинних та отриманих даних

Завдання 6:

```
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 = 6 * np.square(x) + 2 * x + 2  # Варіант 22
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]], [2, 1])

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

# Тренування нейронної мережі
error_progress = nn.train(data, labels, epochs=10000, 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()
```

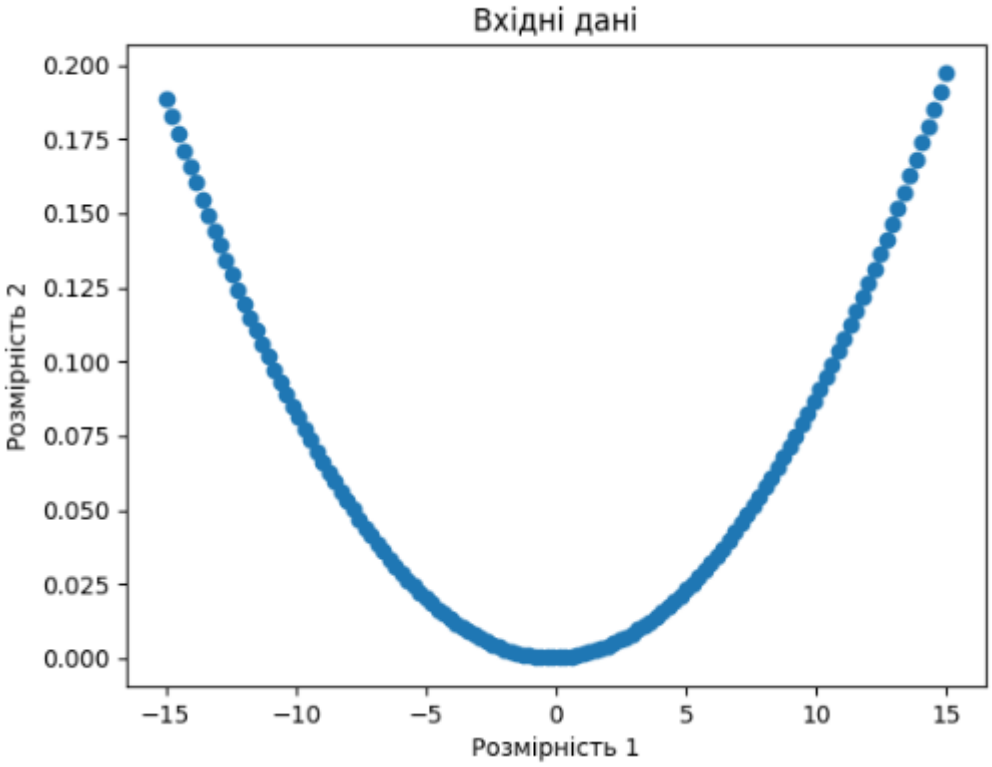

Рис.5.12 – Графік вхідних даних по варіанту

```
Epoch: 4000; Error: 0.2405646892058746;
Epoch: 5000; Error: 0.23788951944658004;
Epoch: 6000; Error: 0.23499759568663692;
Epoch: 7000; Error: 0.23187329117808014;
Epoch: 8000; Error: 0.2285010162493103;
Epoch: 9000; Error: 0.22486563162951176;
Epoch: 10000; Error: 0.22095297421380167;
The maximum number of train epochs is reached
```

Рис.5.13 – Звітність навчання по епохам

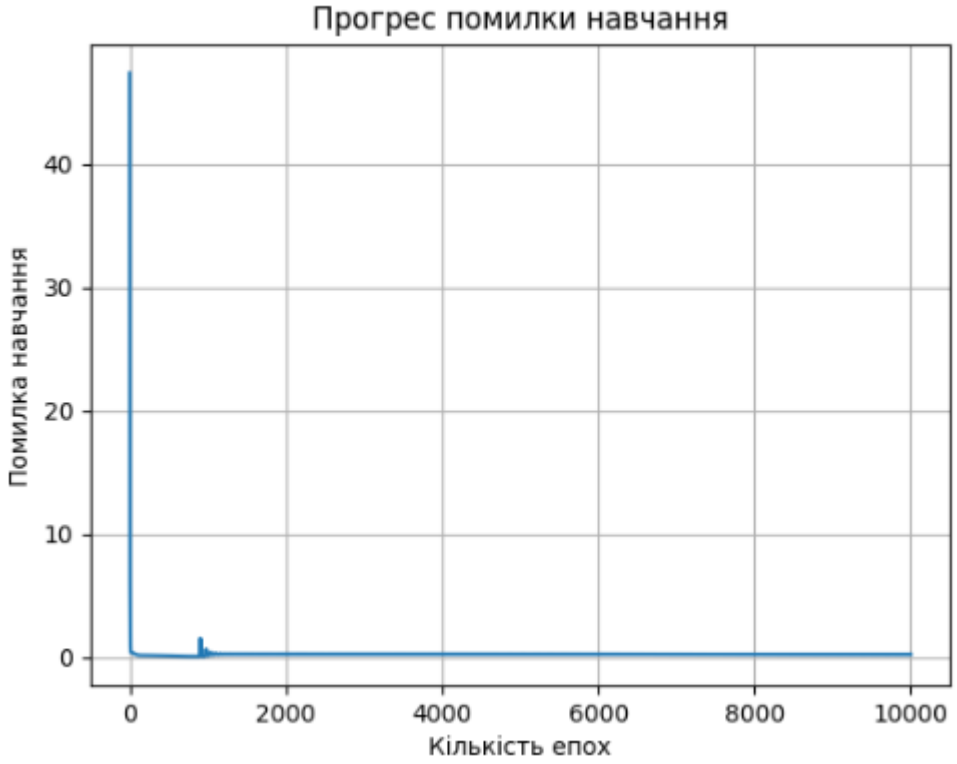

Рис.5.14 – Прогрес помилковості при навчанні

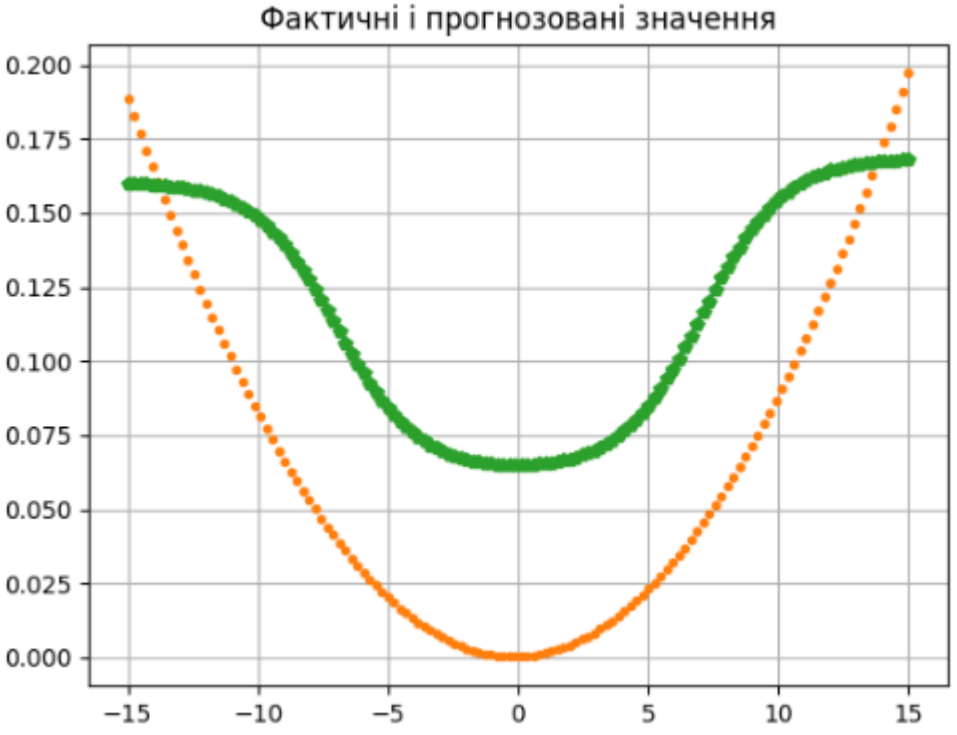

Рис.5.15 – Графік-порівняння дійсних та передбачених даних

У результаті навчання точність нейронної мережі є низькою, що може бути пов’язано з кількістю шарів або нейронів у шарах, точність збільшується в залежності від кількості епох.

Завдання 7:

```
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()
```

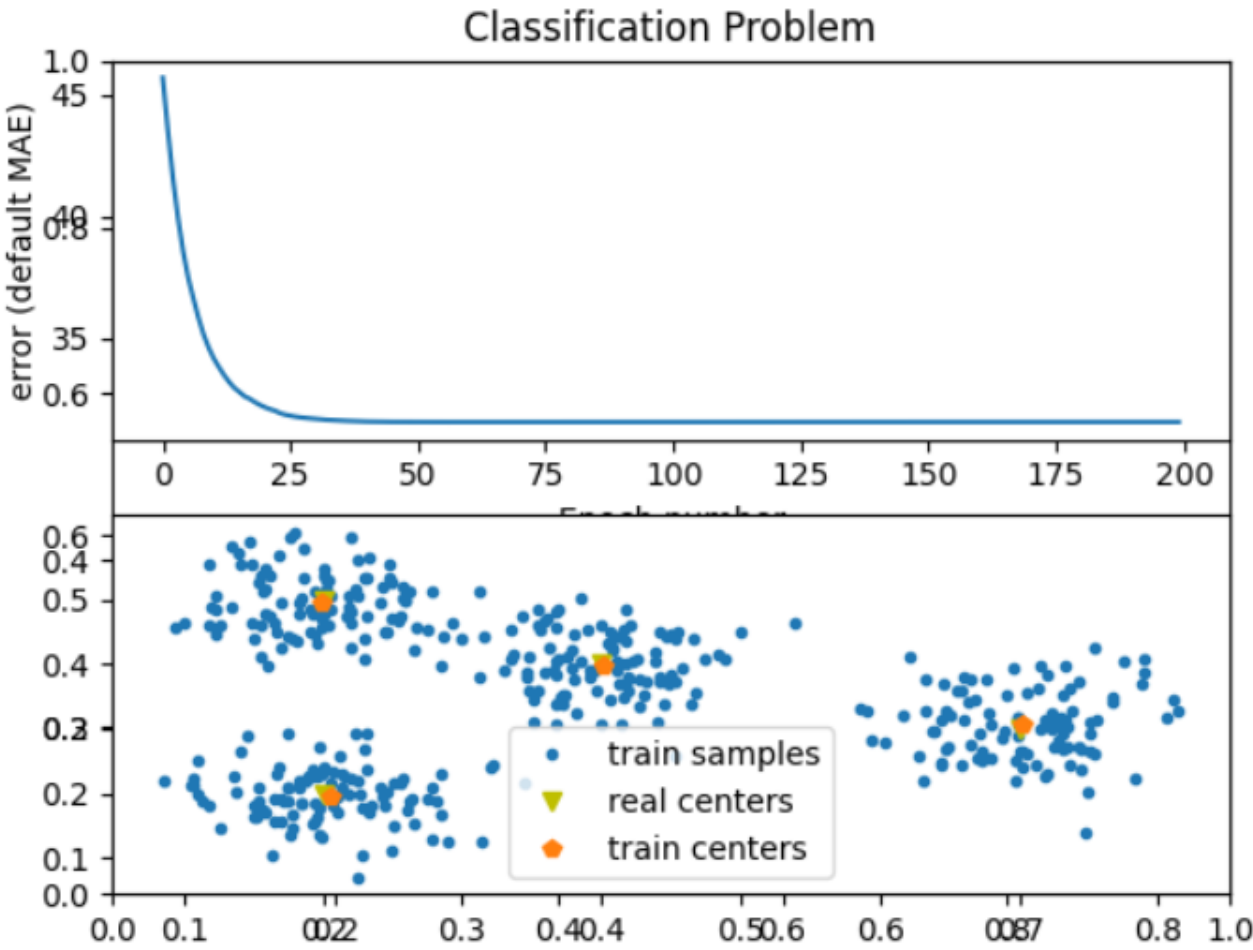

Рис.5.18 – Графік помилковості по епохам та класифікація центрів

```
"C:\Users\ivanp\Desktop\Python A.I\Lab 5\venv\Sc
Epoch: 20; Error: 32.248018026520285;
Epoch: 40; Error: 31.56667498256955;
Epoch: 60; Error: 31.541747957818068;
Epoch: 80; Error: 31.54364440764784;
Epoch: 100; Error: 31.544189211037818;
Epoch: 120; Error: 31.5442824856937;
Epoch: 140; Error: 31.544297402493665;
Epoch: 160; Error: 31.544299790282864;
Epoch: 180; Error: 31.54430017286004;
Epoch: 200; Error: 31.544300234211892;
The maximum number of train epochs is reached
```

Рис.5.17 – Звітність навчання

Завдання 8:

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

skv = 0.07
centr = np.array([[0.5, 0.2], [0.4, 0.4], [0.3, 0.3], [0.3, 0.6], [0.5, 0.7]])
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]], 5)
# 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()
```

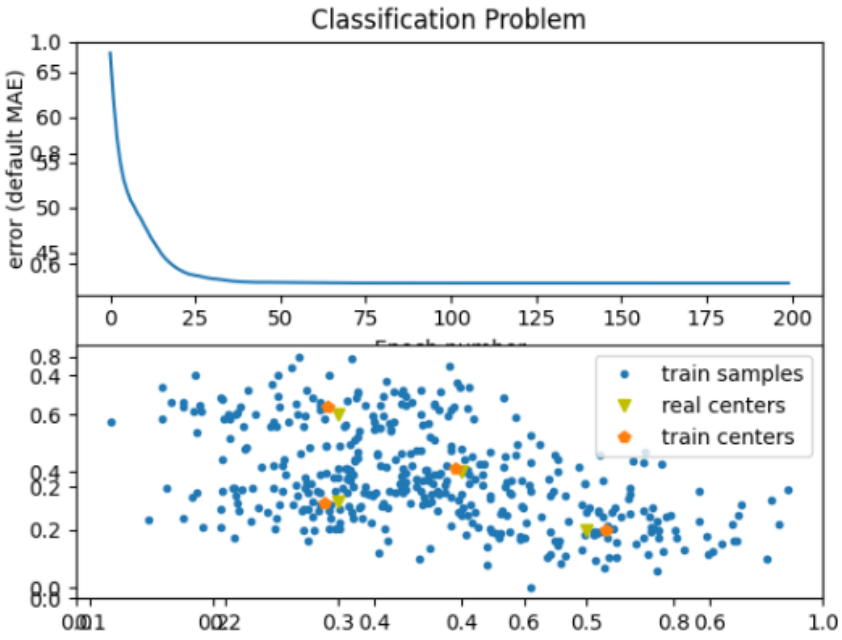

Рис.5.18 – Графік навчання та класифікації за 4х нейронів

```
Epoch: 100; Error: 61.68577241956744;
Epoch: 120; Error: 61.685769758260946;
Epoch: 140; Error: 61.685769690390046;
Epoch: 160; Error: 61.685769688279166;
Epoch: 180; Error: 61.68576968819278;
Epoch: 200; Error: 61.685769688188564;
The maximum number of train epochs is reached
```

Рис.5.19 – Звітність за 4х нейронів

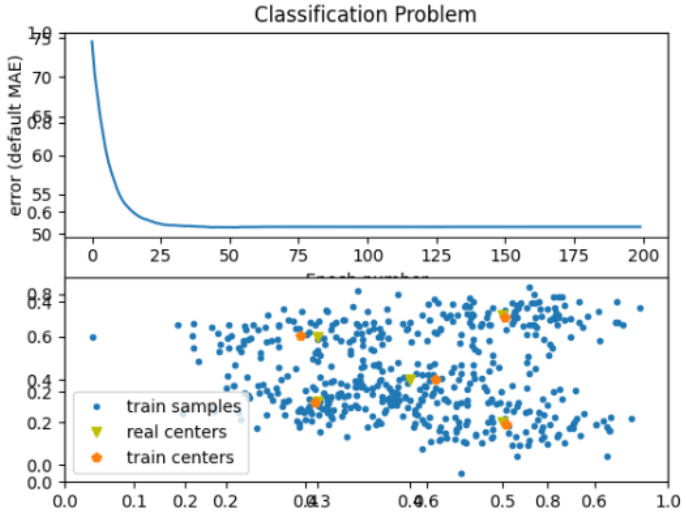

Рис.5.20 – Графік навчання та класифікації за 5х нейронів

```
Epoch: 100; Error: 50.83863067465501;
Epoch: 120; Error: 50.839878139105494;
Epoch: 140; Error: 50.85412927910191;
Epoch: 160; Error: 50.85501339531463;
Epoch: 180; Error: 50.855110877442186;
Epoch: 200; Error: 50.8551216185638;
The maximum number of train epochs is reached
```

Рис.5.21 – Звітність за 5х нейронів

Висновок: в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python було отримано навички створення та застосовування простих нейронних мереж

Посилання на репозиторій GitHub: https://github.com/IvanPaliy/A.I.-Lab-5-IPZ-Palii.git