ЛАБОРАТОРНА РОБОТА № 5

Тема: **«РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ»**

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

Хід роботи

Посилання на програмнй код на Github: https://github.com/NightSDay/AI-2023/tree/main/AI-subject/L-5_Neuron

Завдання 1. **Створити простий нейрон**

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

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

Результат виконання програми:

```
C:\__My\__7-SEM\AI\AI_Labs_Git\ipz-20-2_bubenko\lab5>python LR_5_task_1.py
0.9990889488055994

C:\__My\__7-SEM\AI\AI_Labs_Git\ipz-20-2_bubenko\lab5>
```

Рис. 1. Результат виконання програми

Завдання 2. Створити просту нейронну мережу для передбачення статі людини

Реалізуємо пряме розповсюдження (feedforward) ваг по відношенню до нейронної мережі.

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

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


class BubenkoNeuralNetwork:

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

Результат виконання програми:

```
E:\>cd E:\AI-subject\L-5_Neuron

E:\AI-subject\L-5_Neuron>python LR_5_task_2_1.py
0.9990889488055994
0.7216325609518421

E:\AI-subject\L-5_Neuron>
```

Рис. 2.1. Результат виконання програми

Тепер створимо нейронну мережу для передбачення статі.

Лістинг програми:

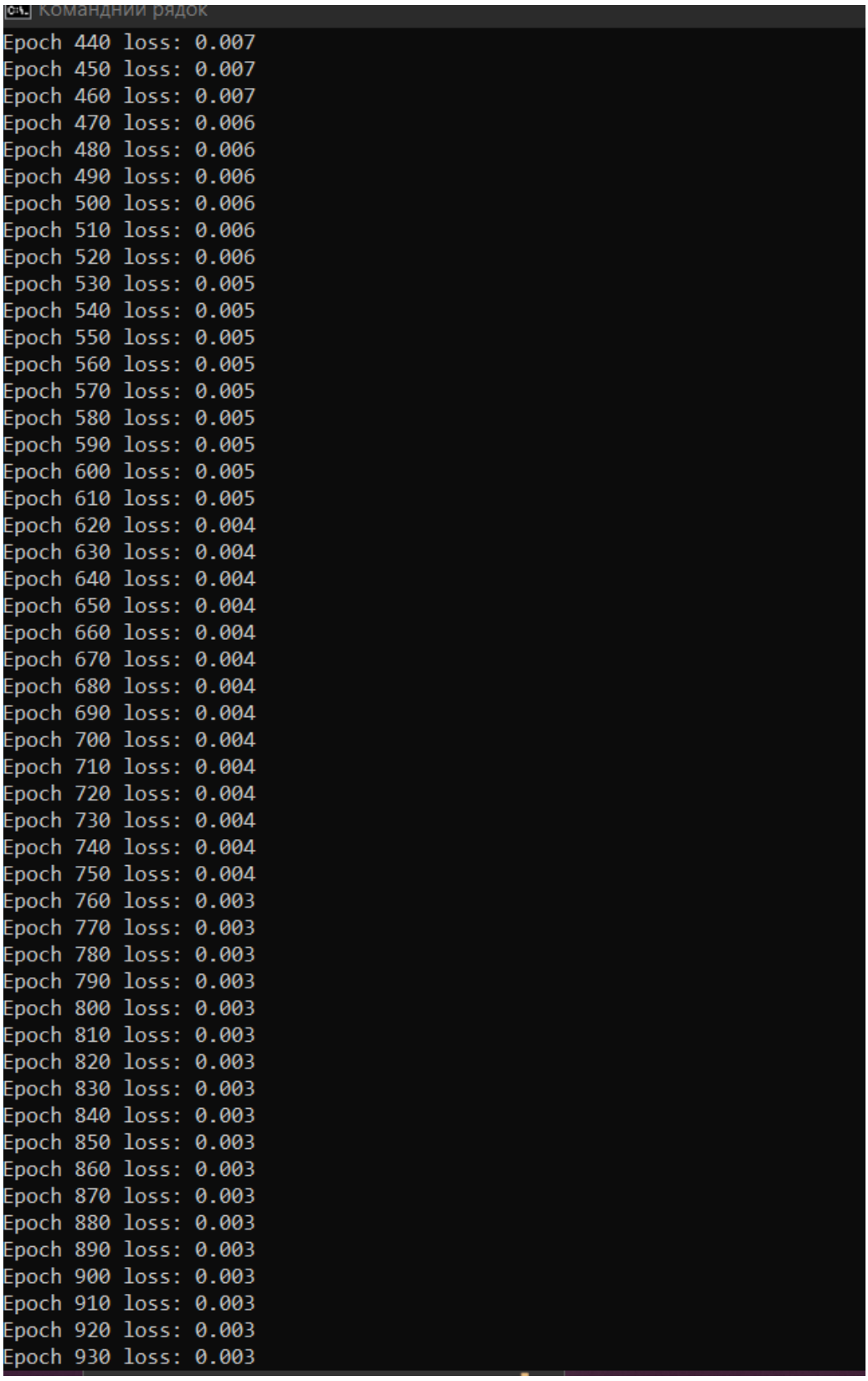
```
import numpy as np

from LR_5_task_1 import 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 BubenkoNeuralNetwork:
    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))


if __name__ == "__main__":
    # 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 = BubenkoNeuralNetwork()
    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:

```
0.7216325609518421

E:\AI-subject\L-5_Neuron>python LR_5_task_2_2.py
0.9990889488055994
Epoch 0 loss: 0.660
Epoch 10 loss: 0.603
Epoch 20 loss: 0.552
Epoch 30 loss: 0.499
Epoch 40 loss: 0.413
Epoch 50 loss: 0.274
Epoch 60 loss: 0.188
Epoch 70 loss: 0.143
Epoch 80 loss: 0.113
Epoch 90 loss: 0.091
Epoch 100 loss: 0.075
Epoch 110 loss: 0.062
Epoch 120 loss: 0.053
Epoch 130 loss: 0.046
Epoch 140 loss: 0.040
Epoch 150 loss: 0.036
Epoch 160 loss: 0.032
Epoch 170 loss: 0.029
Epoch 180 loss: 0.026
Epoch 190 loss: 0.024
Epoch 200 loss: 0.022
Epoch 210 loss: 0.020
Epoch 220 loss: 0.019
Epoch 230 loss: 0.018
Epoch 240 loss: 0.016
Epoch 250 loss: 0.015
Epoch 260 loss: 0.015
Epoch 270 loss: 0.014
Epoch 280 loss: 0.013
Epoch 290 loss: 0.012
Epoch 300 loss: 0.012
Epoch 310 loss: 0.011
Epoch 320 loss: 0.011
Epoch 330 loss: 0.010
Epoch 340 loss: 0.010
Epoch 350 loss: 0.009
Epoch 360 loss: 0.009
Epoch 370 loss: 0.009
Epoch 380 loss: 0.008
Epoch 390 loss: 0.008
Epoch 400 loss: 0.008
Epoch 410 loss: 0.008
Epoch 420 loss: 0.007
Epoch 430 loss: 0.007
Epoch 440 loss: 0.007
Epoch 450 loss: 0.007
```


Командний рядок
Epoch 440 loss: 0.007
Epoch 450 loss: 0.007
Epoch 460 loss: 0.007
Epoch 470 loss: 0.006
Epoch 480 loss: 0.006
Epoch 490 loss: 0.006
Epoch 500 loss: 0.006
Epoch 510 loss: 0.006
Epoch 520 loss: 0.006
Epoch 530 loss: 0.005
Epoch 540 loss: 0.005
Epoch 550 loss: 0.005
Epoch 560 loss: 0.005
Epoch 570 loss: 0.005
Epoch 580 loss: 0.005
Epoch 590 loss: 0.005
Epoch 600 loss: 0.005
Epoch 610 loss: 0.005
Epoch 620 loss: 0.004
Epoch 630 loss: 0.004
Epoch 640 loss: 0.004
Epoch 650 loss: 0.004
Epoch 660 loss: 0.004
Epoch 670 loss: 0.004
Epoch 680 loss: 0.004
Epoch 690 loss: 0.004
Epoch 700 loss: 0.004
Epoch 710 loss: 0.004
Epoch 720 loss: 0.004
Epoch 730 loss: 0.004
Epoch 740 loss: 0.004
Epoch 750 loss: 0.004
Epoch 760 loss: 0.003
Epoch 770 loss: 0.003
Epoch 780 loss: 0.003
Epoch 790 loss: 0.003
Epoch 800 loss: 0.003
Epoch 810 loss: 0.003
Epoch 820 loss: 0.003
Epoch 830 loss: 0.003
Epoch 840 loss: 0.003
Epoch 850 loss: 0.003
Epoch 860 loss: 0.003
Epoch 870 loss: 0.003
Epoch 880 loss: 0.003
Epoch 890 loss: 0.003
Epoch 900 loss: 0.003
Epoch 910 loss: 0.003
Epoch 920 loss: 0.003
Epoch 930 loss: 0.003

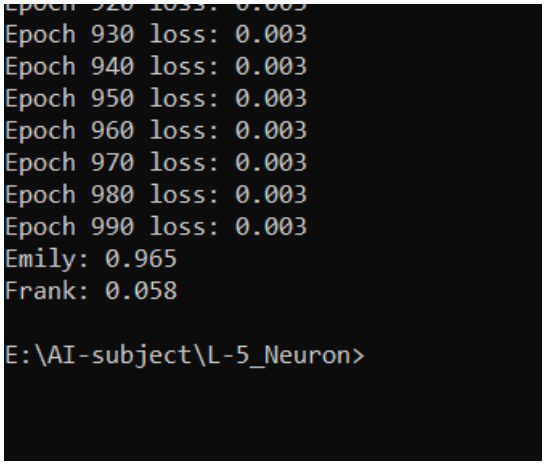


Рис. 2.2. – 2.4. Результат виконання програми

Висновки щодо створення простої нейронної мережі

Функція активації у нейронних мережах необхідна для внесення нелінійності та утворення складних зв'язків між вхідними та вихідними даними. Зазвичай використовується сигмоїдна функція. Її завданням є трансформувати ваговану суму вхідних даних та зсув у вихідне значення, що знаходиться у діапазоні від 0 до 1. Ця нелінійність дозволяє нейронам ефективно моделювати складні зв'язки між вхідними та вихідними даними.

Наша нейронна мережа прямого поширення успішно вирішує завдання класифікації, де об'єкти потрібно розподілити на класи 1 чи 0, використовуючи вхідні дані. Вона навчилася коригувати свої ваги та зсуви для точних прогнозів, що наближаються до правильних відповідей. Нейронні мережі прямого поширення широко застосовуються у сферах комп'ютерного бачення та розпізнавання мовлення, де складна класифікація цільових класів стає складним завданням. Ці типи мереж продемонстрували ефективність, навіть у випадках з шумними даними, що підкреслює їх потужність та адаптивність у аналізі складних наборів інформації.

Завдання 3. **Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab**

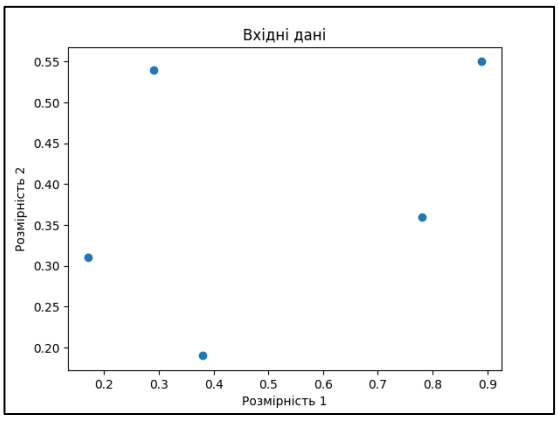
Розробіть класифікатор на основі перцептрону з використанням бібліотеки NeuroLab для файлу даних data_perceptron.txt.

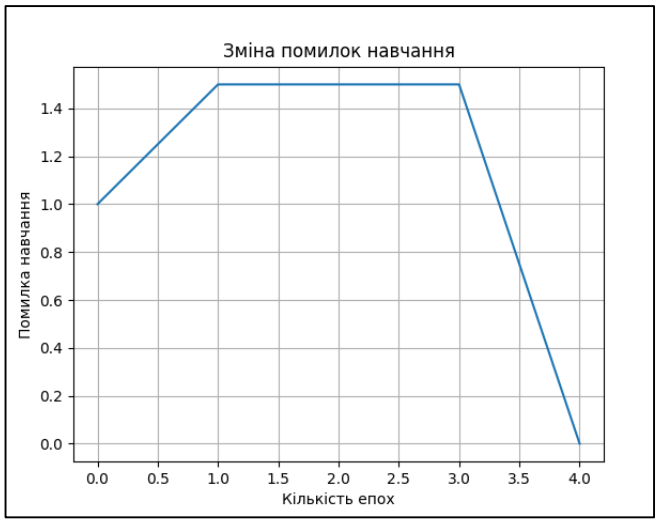
Лістнг програми:

```
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')

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

Результат виконання програми:






```
The goal of learning is reached
```

Рис. 3.1 - 3.3. Результати виконання програми

Висновки щодо рисунку 3.2.

На другому графіку представлений процес навчання, який оцінюється за

допомогою метрики помилок. На осі X кількість епох, а на осі Y – значення помилки навчання на кожній епосі. Під час першої епохи помилки коливалися від 1.0 до 1.5. Протягом двох наступних епох вони залишалися на рівні близько 1.5. Однак у четвертій епохі ми спостерігали поступове зменшення помилок. Це сталося через те, що ми успішно навчили перцептрон, використовуючи тренувальні дані.

Завдання 4. Побудова одношарової нейронної мережі.

Створіть одношарову нейронну мережу, що складається з незалежних нейронів, для вхідного файлу data_simple_nn.txt.

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)
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('Зміна помилок 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:

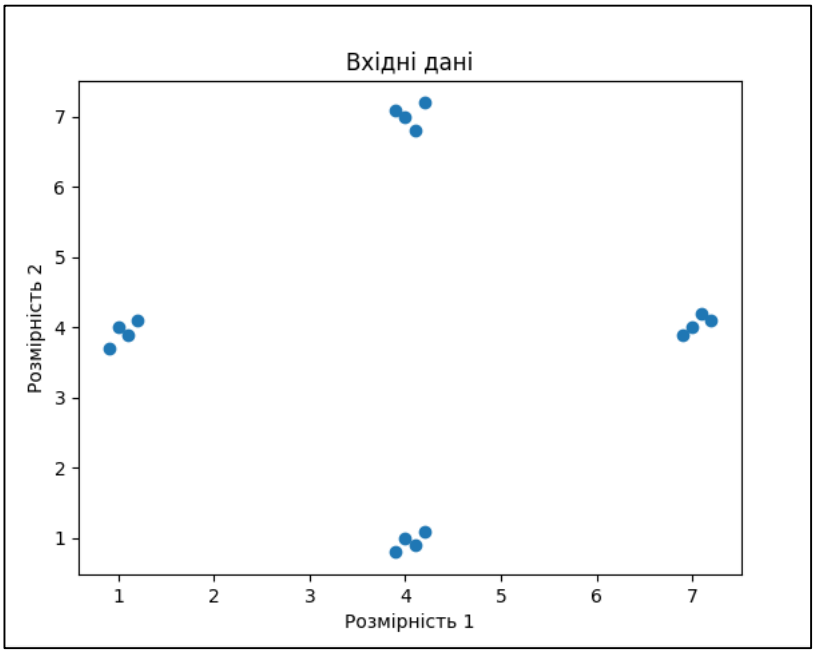


Рис. 4.1. Графік вхідних даних

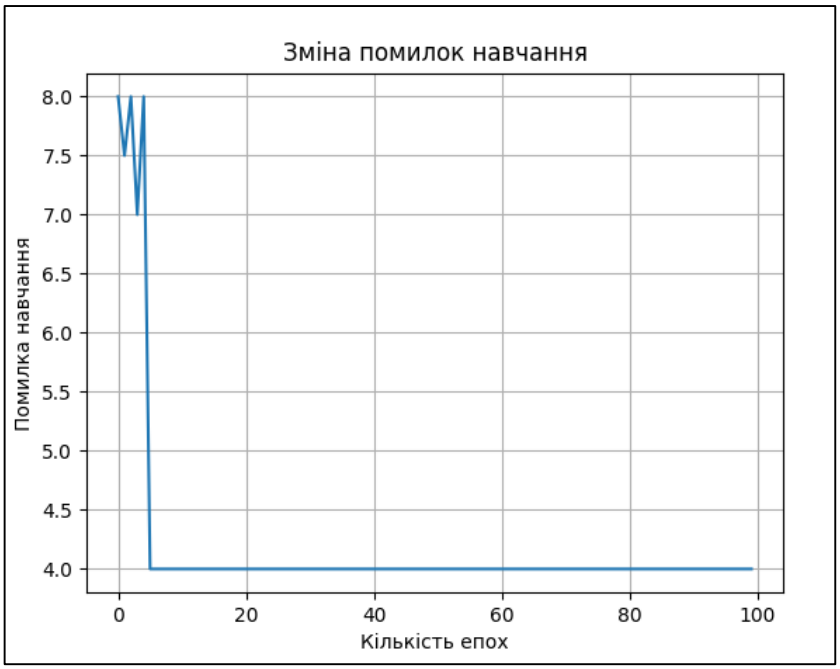


Рис. 4.2. Графік зміни помилок

```
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. 4.3. Результат виконаня програми в консолі

Висновки щодо побудови одношарової нейронної мережі

На графіку 4.1. представлені вхідні дані у двох розмірностях.

У графіку 4.2. показана зміна помилок під час навчання моделі на різних епохах. Помітно, що помилка навчання залишається на рівні 4.0 протягом 20, 40, 60, 80 та 100 епох, що свідчить про досягнення мінімуму втрат нейромережею для навчального набору даних.

Графік 4.3. відображає процес навчання моделі, де вона досягла помилки 4.0. Тестування моделі проводилося на трьох точках, результати представлені у вигляді векторів. Наприклад, точка [4.7, 8.1] була класифікована моделлю як [1. 1.].

Завдання 5. **Побудова багатошарової нейронної мережі**

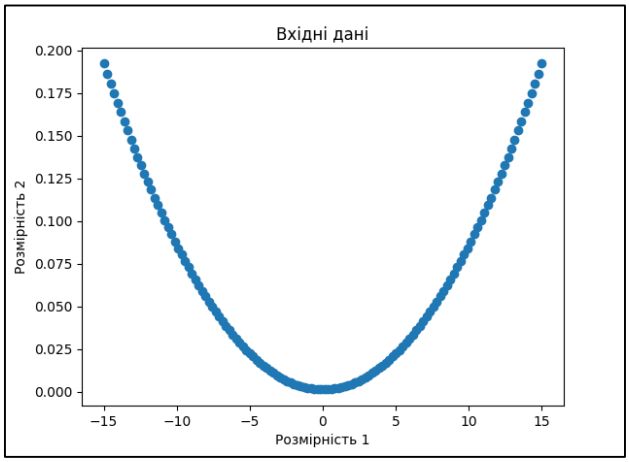
Побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних $y = 3x^2 + 5$

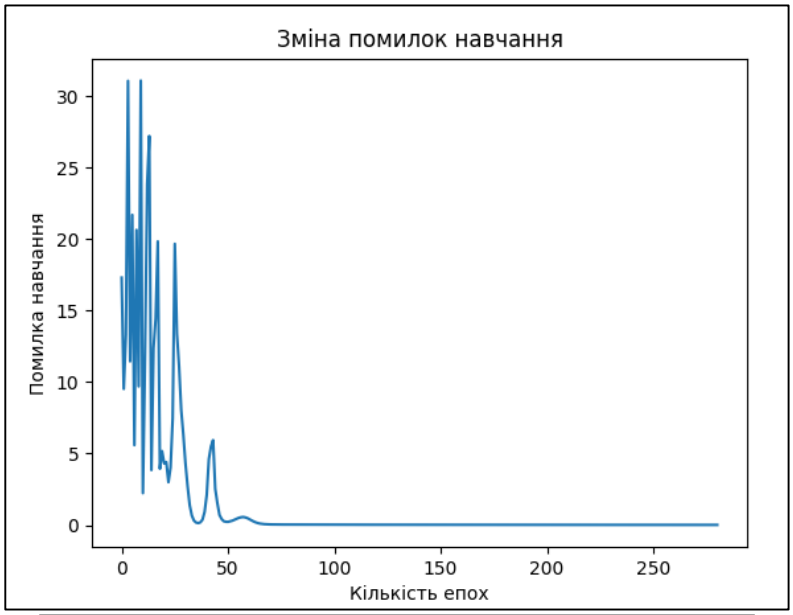
Лістинг програми:

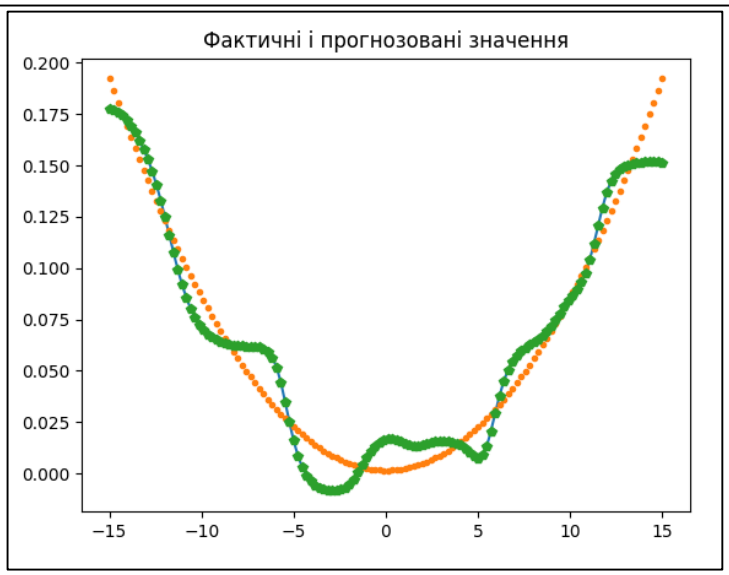
```python
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# 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('Зміна помилок 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('Фактичні і прогнозовані значення')
plt.show()
```

Результат виконання програми:








```
Epoch: 100; Error: 0.025288191838434 85;
Epoch: 200; Error: 0.013232148678455 42;
The goal of learning is reached
```

Рис. 5.1. – 5.4. Результат виконання програми

Висновки щодо побудови багатошарової нейронної мережі

На рисунку 5.4. зображено процес навчання багатошарової нейронної мережі.

Як бачимо, помилка зменшувалась і досягла значення 0,0132. Цілі навчання були досягнуті після 200 епох, коли помилка досягла значення менше за 0.01. Після цього навчання завершилось.

Завдання 6. **Побудова багатошарової нейронної мережі для свого варіанту**

По аналогії з попереднім завданням, побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних вашого варіанту.

№ варіанта	Тестові дані
Варіант 1	$y = 2x^2+5$
Варіант 2	$y = 2x^2+6$
Варіант 3	$y = 2x^2+7$
Варіант 4	$y = 2x^2+8$
Варіант 5	$y = 2x^2+9$

Номер варіанта	Багатошаровий персептрон	
	Кількість шарів	Кількості нейронів у шарах
1	2	3-1
2	2	2-1
3	3	3-3-1
4	2	5-1
5	3	2-2-1

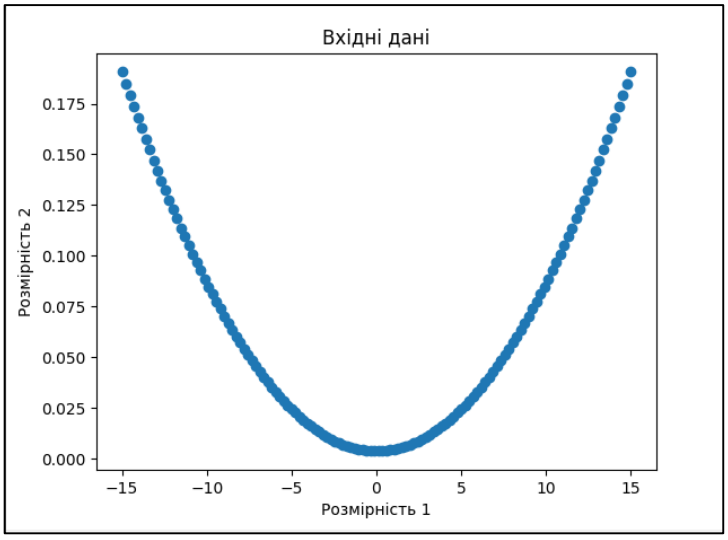
Рис. 6.1. - 6.2. Тестові дані та параметри багатошарової мережі

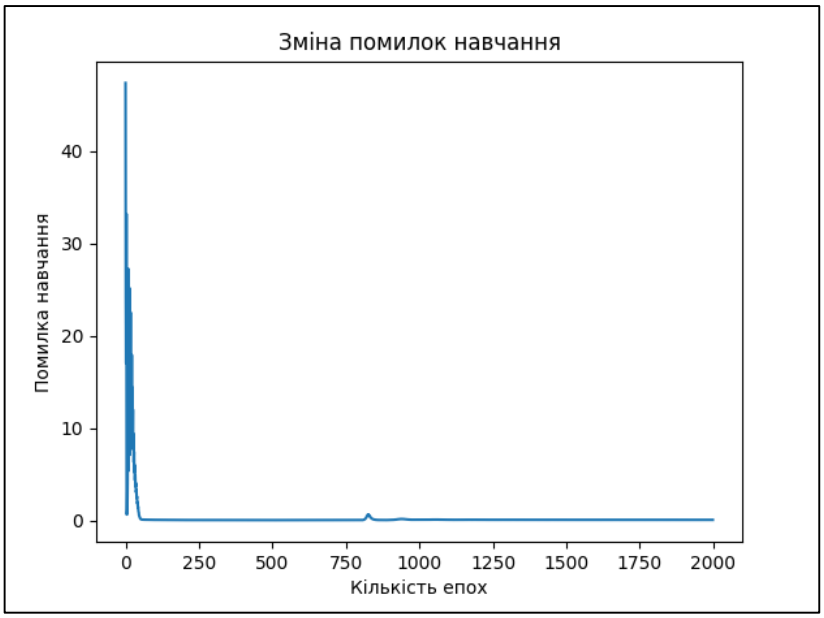
Лістинг програми:

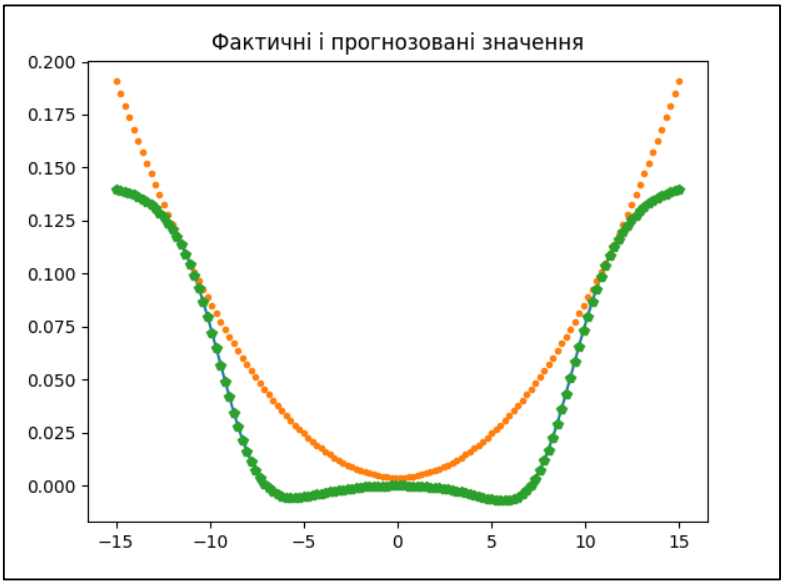
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Генерація тренувальних даних
min_val = -15
max_val = 15
num_points = 130
x = np.linspace(min_val, max_val, num_points)
y = 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
nn = nl.net.newff([[min_val, max_val]], [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('Зміна помилок 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('Фактичні і прогнозовані значення')
plt.show()
```

Результат виконання програми:








```
Epoch: 100; Error: 0.04561480296046684;
Epoch: 200; Error: 0.0267921184344944;
Epoch: 300; Error: 0.021083605578252965;
Epoch: 400; Error: 0.018479085781831397;
Epoch: 500; Error: 0.016956518234810647;
Epoch: 600; Error: 0.0159488365681O309;
Epoch: 700; Error: 0.01522554901083425;
Epoch: 800; Error: 0.01624445550060011;
Epoch: 900; Error: 0.030875125872952318;
Epoch: 1000; Error: 0.04614036943004873;
Epoch: 1100; Error: 0.055129137501130944;
Epoch: 1200; Error: 0.05494410390322945;
Epoch: 1300; Error: 0.05079855251878643;
Epoch: 1400; Error: 0.04876779456132502;
Epoch: 1500; Error: 0.047830160232501875;
Epoch: 1600; Error: 0.0469694004524286;
Epoch: 1700; Error: 0.04620775655798119;
Epoch: 1800; Error: 0.04556023409294961;
Epoch: 1900; Error: 0.044972818702821174;
Epoch: 2000; Error: 0.04442631928603576;
The maximum number of train epochs is reached
```

Рис. 6.1 – 6.4. Результат виконання програми

Висновки щодо побудови багатошарової нейронної мережі для свого варіанту

На діаграмі 6.4. відображено процес навчання багатошарової нейронної мережі. Можна помітити, що помилка періодично збільшується та зменшується, але так і не досягає цільового значення 0.01 навіть після 2000 епох навчання. У тестуванні після 20 000 епох також не було досягнуто цільового значення помилки. Це може свідчити про те, що побудована нейронна мережа не здатна зменшити помилку, і навчання залишається стагнантним після перших 800 епох.

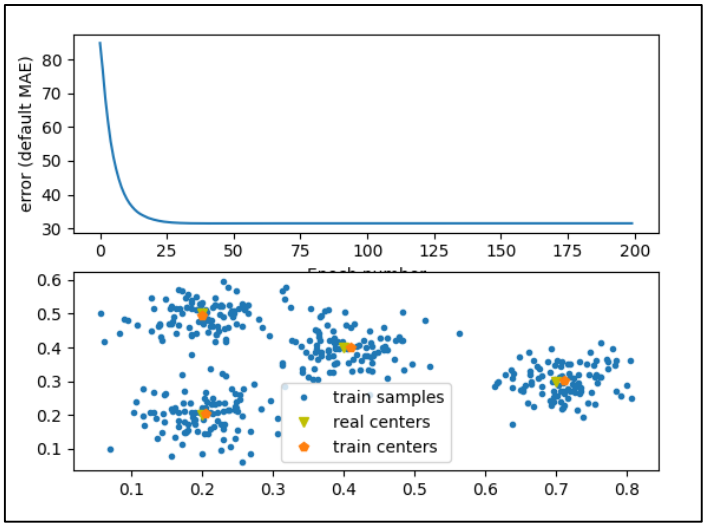
Завдання 7. **Побудова нейронної мережі на основі карти Кохонена, що самоорганізується**

Лістинг програми:

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

```
import pylab as pl

# Генерація даних
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)

# Створення мережі з 2 вхідними нейронами і 4 нейронами в одному шарі
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)

# Тренування мережі
error = net.train(inp, epochs=200, show=100)

# Побудова графіків
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:




```
Epoch: 100; Error: 30.605860005397947;
Epoch: 200; Error: 30.605866658839844;
The maximum number of train epochs is reached
```

Рис. 7.1. – 7.2. Результат виконання програми

Висновки щодо побудови нейронної мережі на основі карти Кохонена,

що самоорганізується

MAE (Mean Absolute Error) — це середня абсолютна похибка. Середньою абсолютною похибкою називають середнє абсолютне значення різниці між фактичними та прогнозованими значеннями. Ця метрика використовується для вимірювання точності в задачах регресії.

Як бачимо з рисунку 7.2 після 200 епох навчання помилка не змінюється та залишається на рівні 30,60586. Аналогічні результат для 2000 епох. Тому можна зробити висновок, що нейронна мережа досягла свого максимуму навчання і покращення результатів не є можливим.

Завдання 8. **Дослідження нейронної мережі на основі карти Кохонена, що самоорганізується.**

Проведіть дослідження по аналогії з попереднім завданням.

Таблиця 3

№ 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
Варіант 3	[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.4, 0.5]	0,03
Варіант 4	[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]	0,03
Варіант 5	[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,03

Рис. 8.1. Параметри за варіантом

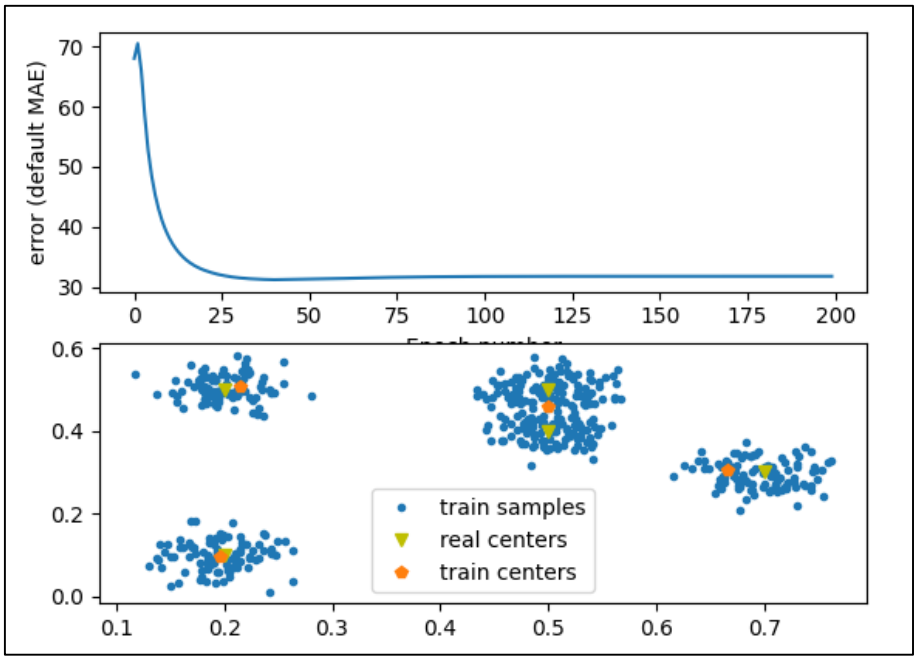
Створіть нейронну мережу Кохонена з 2 входами та 4 нейронами

Лістинг програми:

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

# Генерація даних з новими параметрами
skv = 0.03
centr = np.array([[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]])

# Додано нові центри кластерів
rand_norm = skv * rand.randn(100, 4, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 5, 2)
rand.shuffle(inp)
```

```
# Створення мережі з 2 вхідними нейронами і 4 нейронами в одному шарі
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)

# Тренування мережі
error = net.train(inp, epochs=200, show=100)

# Побудова графіків
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:




```
Epoch: 20; Error: 36.97875277836742;
Epoch: 40; Error: 32.00812795159361;
Epoch: 60; Error: 31.92709249954373;
Epoch: 80; Error: 32.01708642045644;
Epoch: 100; Error: 32.038907510816394;
Epoch: 120; Error: 32.04397790817703;
Epoch: 140; Error: 32.04512362807044;
Epoch: 160; Error: 32.04538388157215;
Epoch: 180; Error: 32.04544440343068;
Epoch: 200; Error: 32.04546362464938;
The maximum number of train epochs is reached
```

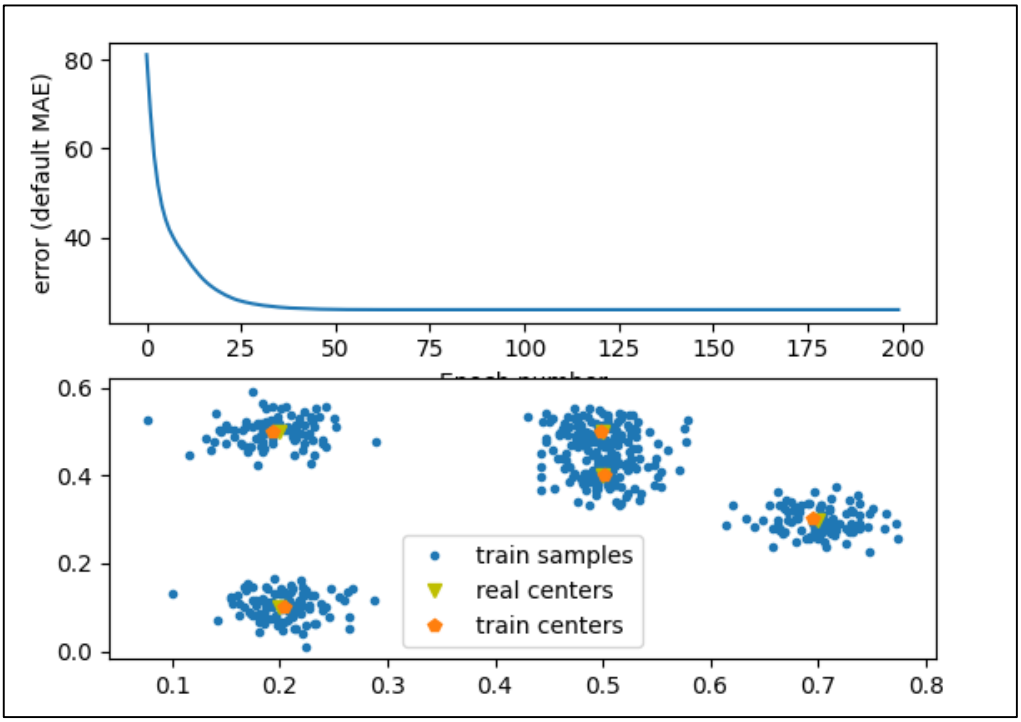
Рис. 8.1 – 8.2. Результат виконання програми

Створіть нейронну мережу Кохонена з 2 входами та 4 нейронами

Лістинг програми:

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

# Генерація даних з новими параметрами
skv = 0.03
centr = np.array([[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]])

# Додано нові центри кластерів
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)

# Створення мережі з 2 вхідними нейронами і 5 нейронами в одному шарі
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 5)

# Тренування мережі
error = net.train(inp, epochs=200, show=100)

# Побудова графіків
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:




```
Epoch: 20; Error: 28.163911067274505;
Epoch: 40; Error: 24.697023734455087;
Epoch: 60; Error: 24.204690813281573;
Epoch: 80; Error: 24.139170889679882;
Epoch: 100; Error: 24.13321205738736;
Epoch: 120; Error: 24.13251935056232;
Epoch: 140; Error: 24.13242407392687;
Epoch: 160; Error: 24.132428258366282;
Epoch: 180; Error: 24.13243806343884;
Epoch: 200; Error: 24.132444173590763;
The maximum number of train epochs is reached
```

Рис. 8.3. – 8.4. Результат виконання програми

Висновки щодо отриманих результатів

Основний висновок полягає у тому, що мережа із п'ятьма нейронами в шарі Кохонена показує меншу похибку, порівняно з мережею із чотирма нейронами: 24.13 проти 32.05. Неправильний вибір кількості нейронів у мережі Кохонена може призвести до неточної кластеризації даних. Якщо нейронів замало, мережа може недостатньо точно відобразити структуру даних, збільшуючи помилку. З іншого боку, якщо нейронів забагато, це може призвести до надмірного подрібнення кластерів та збільшення внутрішньокластерної варіабельності. Оптимальний результат досягається, коли кількість нейронів відповідає кількості кластерів у даних. Зменшення розкиду вхідних даних покращує точність кластеризації. У восьмому завданні з меншим значенням skv = 0.03, мережа ефективніше відобра-

жає розташування кластерів. Отже, розкид вхідних даних суттєво впливає на точність кластеризації.

Висновок: у ході виконання лабораторної роботи я, використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовувати прості нейронні мережі.