ЛАБОРАТОРНА РОБОТА №5

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

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

Хід роботи:

GitHub: https://github.com/ipz201svo/AI

Завдання 1: Створити простий нейрон.

```
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))
```

```
0.9990889488055994
```

Рис. 1.1 – Результат виконання програми

Завдання 2: Створити просту нейронну мережу для передбачення статі людини.

```
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)


class SkakovskyiNeuralNetwork:
    def __init__(self):
        weights = np.array([0, 1])
        bias = 0

        # Класс Neuron із попереднього завдання
        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)

        # Входи для o1 є виходами h1 и h2
        out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

        return out_o1


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

```
0.7216325609518421
```

Рис. 2.1 – Результат виконання програми

```python
import numpy as np


def sigmoid(x):
    # Наша функція активації: 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 SkakovskyiNeuralNetwork:
    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 пода-
льшому )
                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))


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 = SkakovskyiNeuralNetwork()
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
```

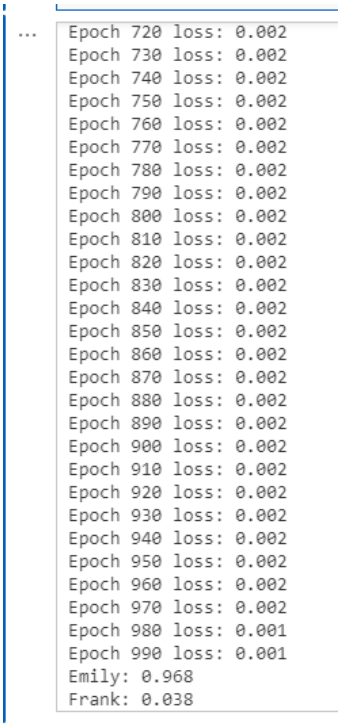

Рис. 2.2 – Результат виконання програми

Функція активації використовується для підключення незв'язаних вхідних даних із виходом, у якого проста та передбачувана форма. Як правило, в якості функції активації найчастіше використовується функція 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.

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

```
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()
```

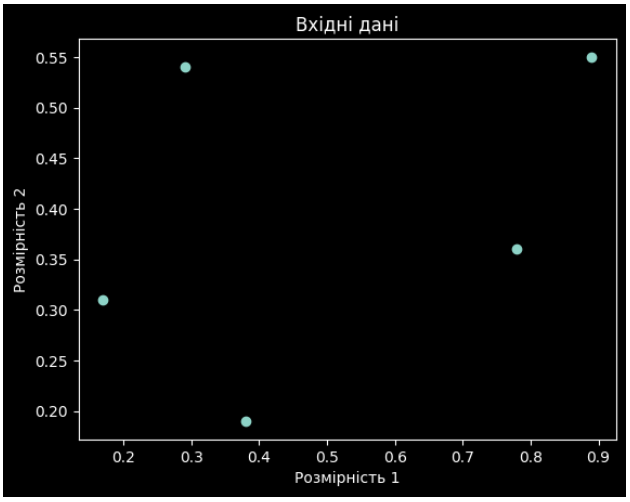

Рис. 3.1 – Результат виконання програми

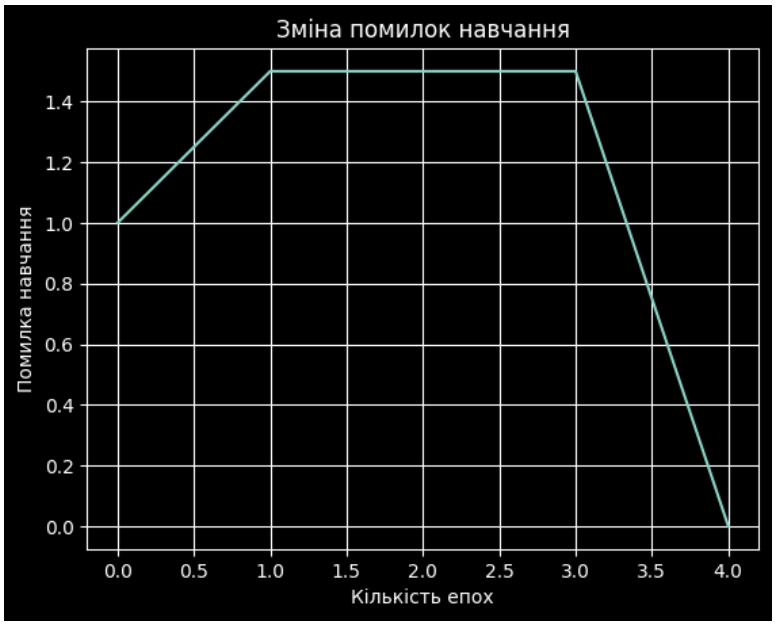

Рис. 3.2 – Результат виконання програми

На другому графіку зображено процес навчання використовуючи метрику помилки. Якщо під час першої епохи відбулося від 1.0 до 1.5 помилок, то вже під час 4 епохи помилки почались зменшуватись. Все через те, що ми навчили перцептрон за допомогою тренувальних даних.

Завдання 4: Побудова одношарової нейронної мережі.

```
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])
```

```
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. 4.1 – Результат виконання програми

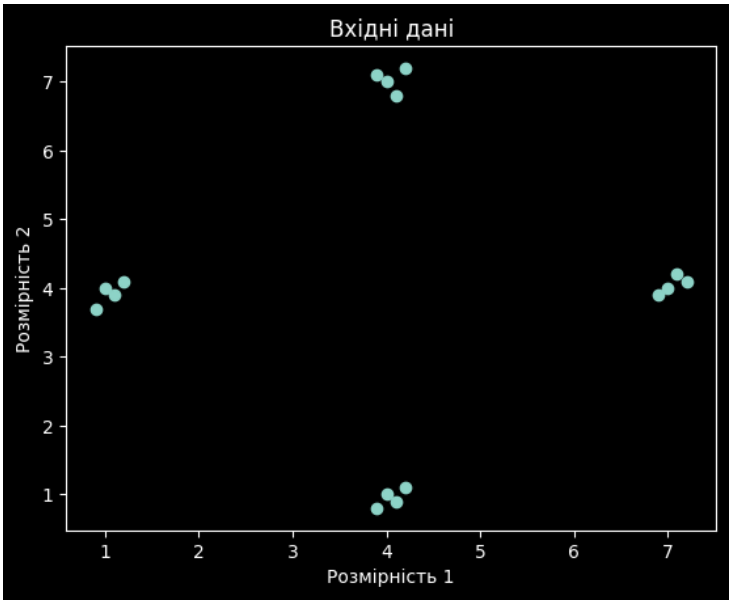

Рис. 4.2 – Результат виконання програми

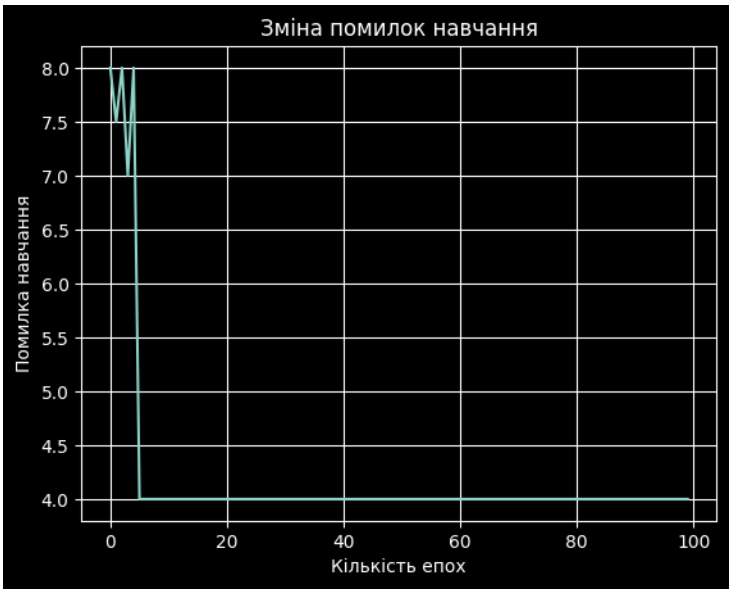

Рис. 4.3 – Результат виконання програми

```
Test results:
[0.4, 4.3] --> [0. 0.]
[4.4, 0.6] --> [1. 0.]
[4.7, 8.1] --> [1. 1.]
```

Рис. 4.4 – Результат виконання програми

На рисунку зображено процес навчання мережі. На 20, 40, 60, 80 та 100 епосі відбулось 4 помилки. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування. Ми вирішили визначити вибіркові тестові точки даних та запустили для них нейронну мережу. Вкінці виведено результат.

Завдання 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()
```

```
Epoch: 100; Error: 0.5608419693708921;
Epoch: 200; Error: 0.02780301382665505;
Epoch: 300; Error: 0.016797133244645;
Epoch: 400; Error: 0.01384671313534364;
Epoch: 500; Error: 0.02194798326284977;
Epoch: 600; Error: 0.03997200063167938;
Epoch: 700; Error: 0.010477565091077883;
The goal of learning is reached
```

Рис. 5.1 – Результат виконання програми

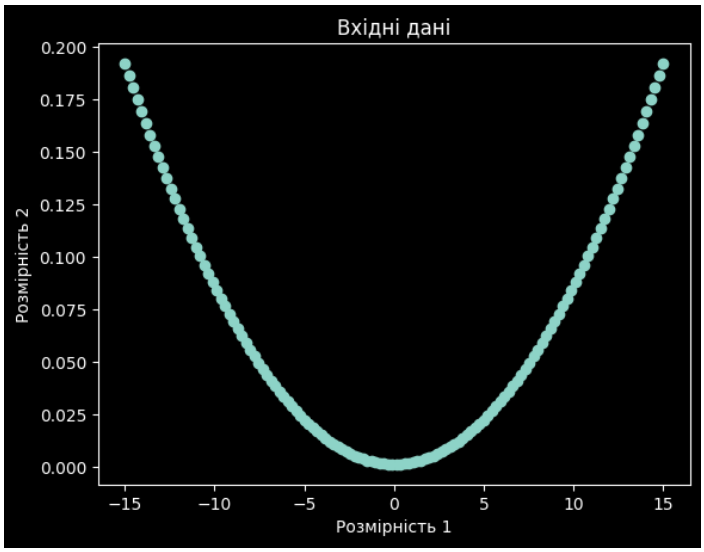

Рис. 5.2 – Результат виконання програми

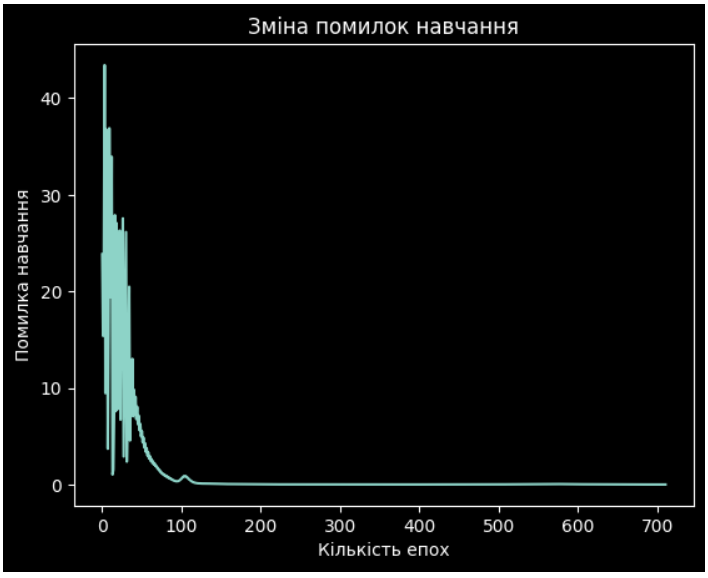

Рис. 5.3 – Результат виконання програми

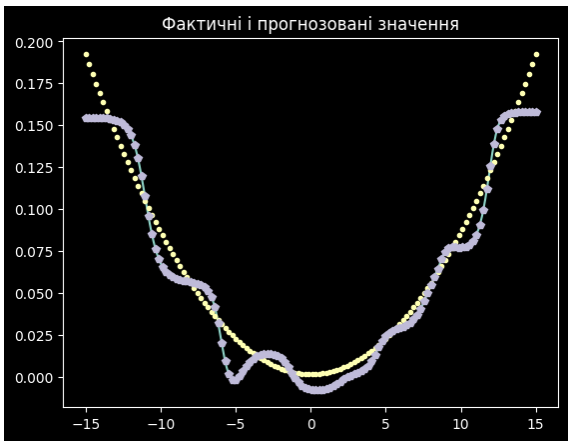

Рис. 5.4 – Результат виконання програми

На рисунку зображено процес навчання мережі. На 100 епосі відбулось 0,56 помилки, на 200 епосі відбулось 0,03 помилки, на 300 епосі відбулось 0,02 помилки, на 400 епосі відбулось 0,01 помилки, на 500 епосі відбулось 0,02 помилки, на 600 епосі відбулось 0,04 помилки, на 700 епосі відбулось 0,01 помилки. Вкінці вивелось повідомлення, що ми досягли цілі навчання

Завдання 6: Побудова багатошарової нейронної мережі для свого варіанту.

Таблиця 6.1

Варіант	Тестові дані
17	y = 5x^2 + 8

Таблиця 6.2

Варіант	Багатошаровий персептрон	
	Кількість шарів	Кількості нейронів у шарах
17	3	7-7-1

```
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 = 5 * np.square(x) + 8
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]], [7, 7, 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()
```

```
Epoch: 100; Error: 8.28074396775963;
Epoch: 200; Error: 8.10539171192785;
Epoch: 300; Error: 8.032875633069342;
Epoch: 400; Error: 7.589633240779492;
Epoch: 500; Error: 0.09748508604439163;
Epoch: 600; Error: 0.057638396090051496;
Epoch: 700; Error: 0.056760217362104536;
Epoch: 800; Error: 0.11906818120211331;
Epoch: 900; Error: 0.11329722608963383;
Epoch: 1000; Error: 0.06919895951012357;
Epoch: 1100; Error: 0.04384948773146877;
Epoch: 1200; Error: 0.030712826181481627;
Epoch: 1300; Error: 0.0298890439110752;
Epoch: 1400; Error: 0.03140678857032342;
Epoch: 1500; Error: 0.02210484645324672;
Epoch: 1600; Error: 0.021218871523170188;
Epoch: 1700; Error: 0.01947952089373889;
Epoch: 1800; Error: 0.017561278083455926;
Epoch: 1900; Error: 0.016578345889514418;
Epoch: 2000; Error: 0.015337548063783072;
The maximum number of train epochs is reached
```

Рис. 6.1 – Результат виконання програми

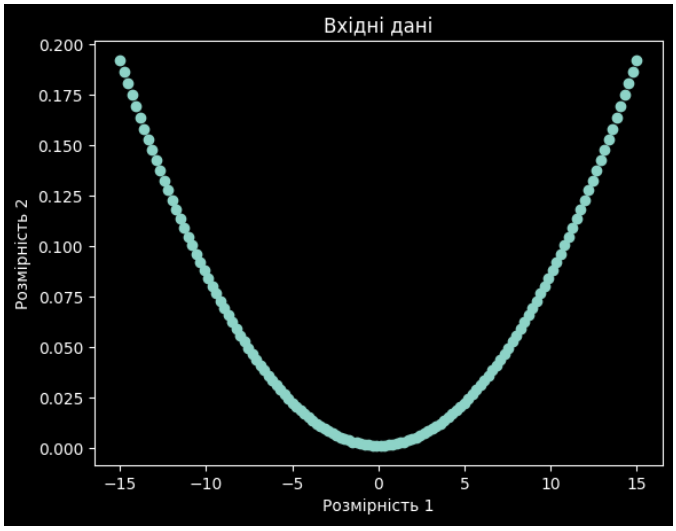

Рис. 6.2 – Результат виконання програми

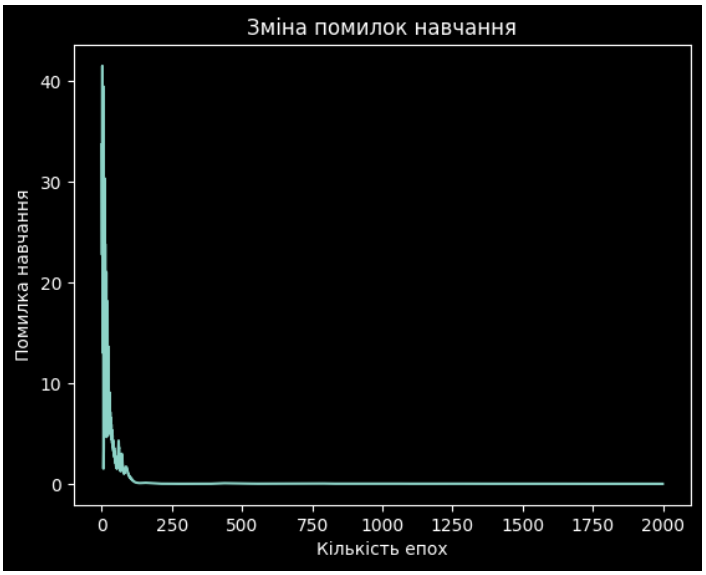

Рис. 6.3 – Результат виконання програми

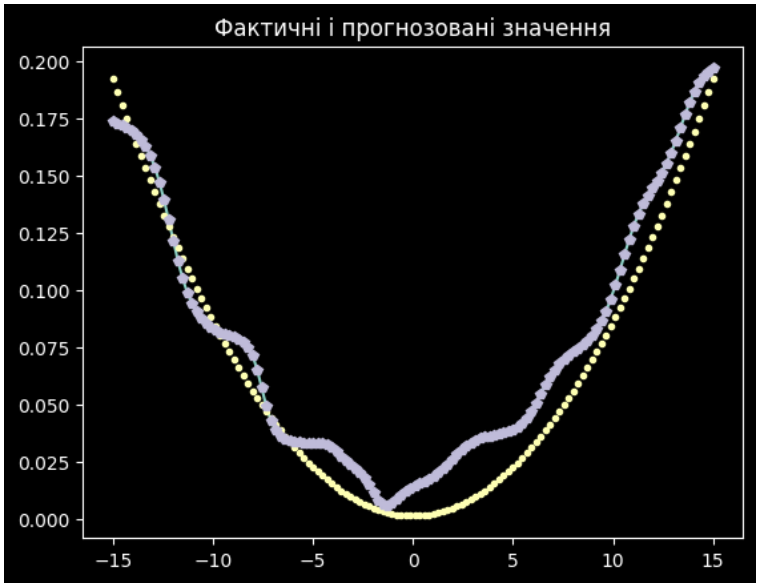

Рис. 6.4 – Результат виконання програми

На рисунку зображено процес навчання мережі. На 100 епосі відбулось 8,3 помилки, на 2000 епосі відбулось 0,015 помилки. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

Завдання 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()
```

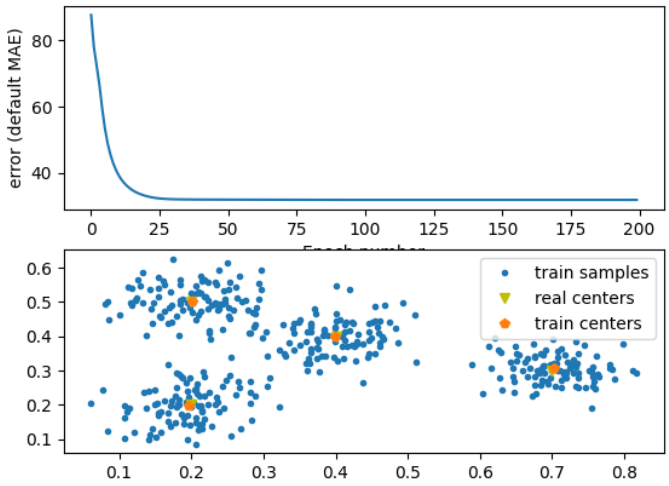

Рис. 7.1 – Результат виконання програми

Помилка MAE - Средня абсолютна помилка (Mean Absolute Error). Середньою абсолютною похибкою називають середнє арифметичне з абсолютних похибок усіх вимірювань.

Завдання 8: Дослідження нейронної мережі на основі карти Кохонена, що само організується.

Таблиця 8.1

Варіант	Центри кластера	skv
17	[0.2, 0.1], [0.5, 0.4], [0.5, 0.3], [0.2, 0.5], [0.5, 0.5]	0,06

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

skv = 0.06
centr = np.array([[0.2, 0.1], [0.5, 0.4], [0.5, 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 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:
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()
```

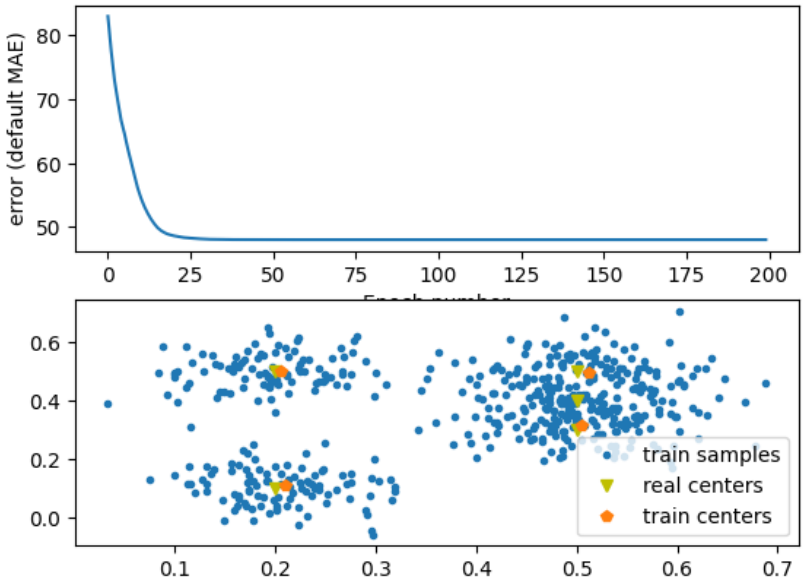

Рис. 8.1 – Результат виконання програми

```
Epoch: 20; Error: 48.77097847797482;
Epoch: 40; Error: 48.056024563885956;
Epoch: 60; Error: 48.043813258054556;
Epoch: 80; Error: 48.04365245967982;
Epoch: 100; Error: 48.0436020730977;
Epoch: 120; Error: 48.04359374832829;
Epoch: 140; Error: 48.04359274845861;
Epoch: 160; Error: 48.0435926387764;
Epoch: 180; Error: 48.04359262713549;
Epoch: 200; Error: 48.04359262591587;
The maximum number of train epochs is reached
```

Рис. 8.2 – Результат виконання програми

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

skv = 0.06
centr = np.array([[0.2, 0.1], [0.5, 0.4], [0.5, 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()
```

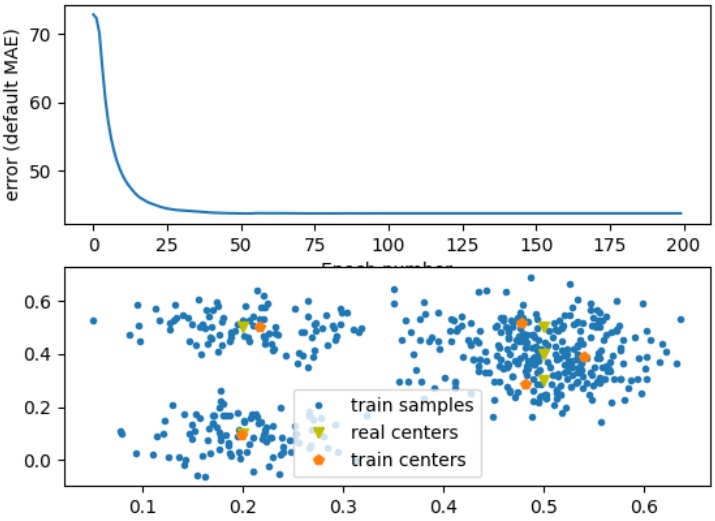

Рис. 8.3 – Результат виконання програми

```
Epoch: 20; Error: 45.28740130432624;
Epoch: 40; Error: 43.89462046699484;
Epoch: 60; Error: 43.79164722773945;
Epoch: 80; Error: 43.76192221806386;
Epoch: 100; Error: 43.77561633840047;
Epoch: 120; Error: 43.77500265038303;
Epoch: 140; Error: 43.77493980501596;
Epoch: 160; Error: 43.774936507580236;
Epoch: 180; Error: 43.77493733378266;
Epoch: 200; Error: 43.77493774268977;
The maximum number of train epochs is reached
```

Рис. 8.4 – 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.

Висновок: в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python навчилися створювати та застосовувати прості нейронні мережі.