Лабораторна робота 5

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

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

Хід роботи

Завдання 2.1. Створити простий нейрон

Лістинг програми:

```
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):
        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))
```

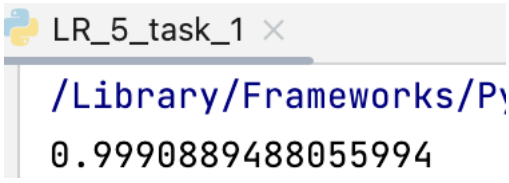

Рис. 5.1 Результат виконання програми

Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини

					ДУ «Житомирська політехніка».22.121.14.000 – Лр5		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Сірач А.С.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів ZZ
Перевір.		Філіпов В.О.					
Керівник						ФІКТ Гр. ІПЗ-19-3[2]	
Н. контр.							
Зав. каф.							

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])  # w1 = 0, w2 = 1
bias = 4  # b = 4
n = Neuron(weights, bias)

x = np.array([2, 3])  # x1 = 2, x2 = 3

class SirachNeuralNetwork:

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

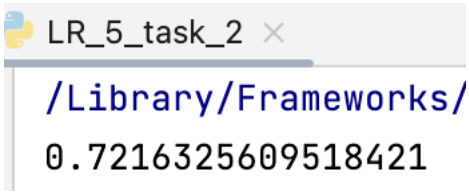

Рис. 5.2 Результат виконання програми

Лістинг програми:

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def deriv_sigmoid(x):
    fx = sigmoid(x)
    return fx * (1 - fx)


def mse_loss(y_true, y_pred):
    return ((y_true - y_pred) ** 2).mean()


class SirachNeuralNetwork:

    def __init__(self):
        self.w1 = np.random.normal()
        self.w2 = np.random.normal()
        self.w3 = np.random.normal()
        self.w4 = np.random.normal()
        self.w5 = np.random.normal()
        self.w6 = np.random.normal()
        self.b1 = np.random.normal()
        self.b2 = np.random.normal()
        self.b3 = np.random.normal()

    def feedforward(self, x):
        h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
        h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
        o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
        return o1

    def train(self, data, all_y_trues):
        learn_rate = 0.1
        epochs = 1000

        for epoch in range(epochs):
            for x, y_true in zip(data, all_y_trues):
                sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
                h1 = sigmoid(sum_h1)

                sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2
                h2 = sigmoid(sum_h2)

                sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
                o1 = sigmoid(sum_o1)
                y_pred = o1

                d_L_d_ypred = -2 * (y_true - y_pred)

                d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
                d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
                d_ypred_d_b3 = deriv_sigmoid(sum_o1)

                d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
                d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

                d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
                d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
```

```
                d_h1_d_b1 = deriv_sigmoid(sum_h1)

                d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
                d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
                d_h2_d_b2 = deriv_sigmoid(sum_h2)

                self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
                self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
                self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1

                self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3
                self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4
                self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2

                self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
                self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
                self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3

            if epoch % 10 == 0:
                y_preds = np.apply_along_axis(self.feedforward, 1, data)
                loss = mse_loss(all_y_trues, y_preds)
                print("Epoch %d loss: %.3f" % (epoch, loss))


data = np.array([
    [-2, -1],  # Alice
    [25, 6],  # Bob
    [17, 4],  # Charlie
    [-15, -6],  # Diana
])

all_y_trues = np.array([
    1,  # Alice
    0,  # Bob
    0,  # Charlie
    1,  # Diana
])

network = SirachNeuralNetwork()
network.train(data, all_y_trues)
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 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
Epoch 940 loss: 0.003
Epoch 950 loss: 0.003
Epoch 960 loss: 0.002
Epoch 970 loss: 0.002
Epoch 980 loss: 0.002
Epoch 990 loss: 0.002
Emily: 0.949
Frank: 0.040
```

Рис. 5.3 Результат виконання завдання

Висновок: 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 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

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()
```

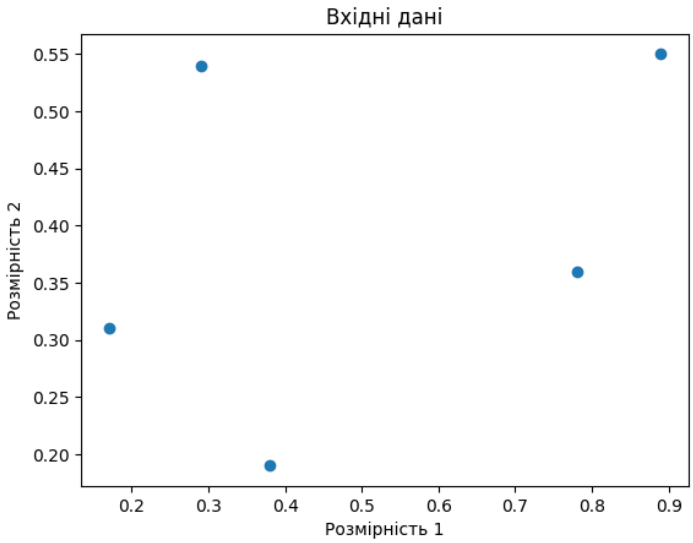

Рис. 5.4 Графік вхідних даних

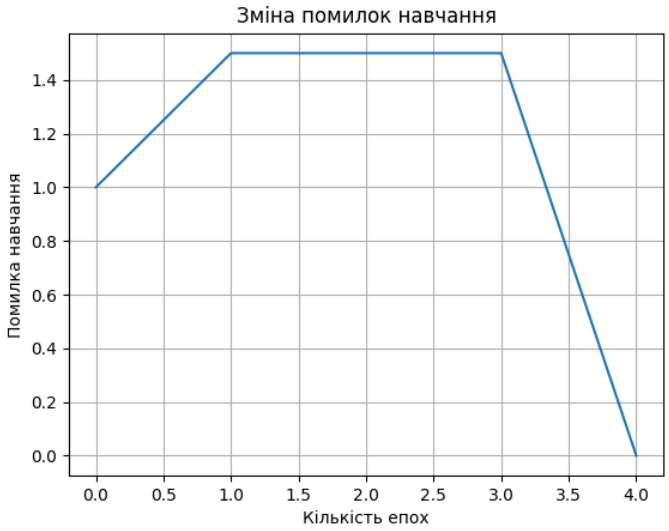

Рис. 5.5 Графік процесу навчання

Висновок: 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 зменшувтась. Все через те, що ми навчили перцептрон за допомогою тренувальних даних.

Завдання 2.4. Побудова одношарової нейронної мережі

Лістинг програми:

```
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])
```

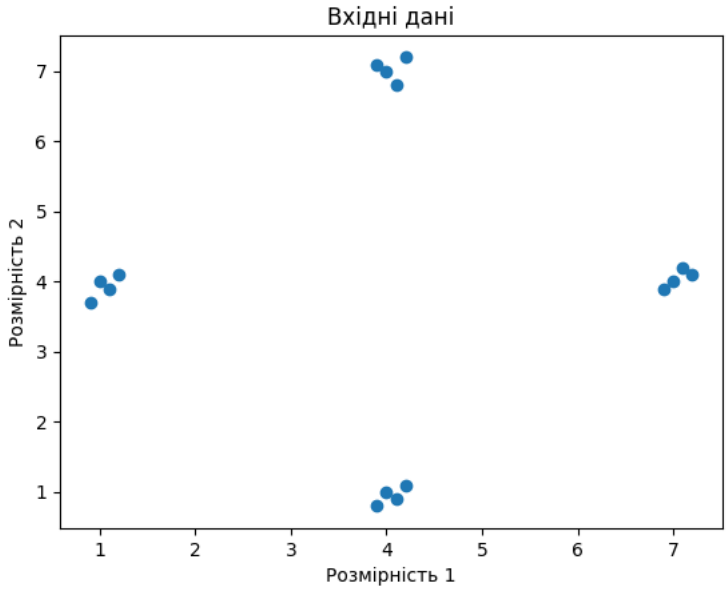

Рис. 5.6 Графік вхідних даних

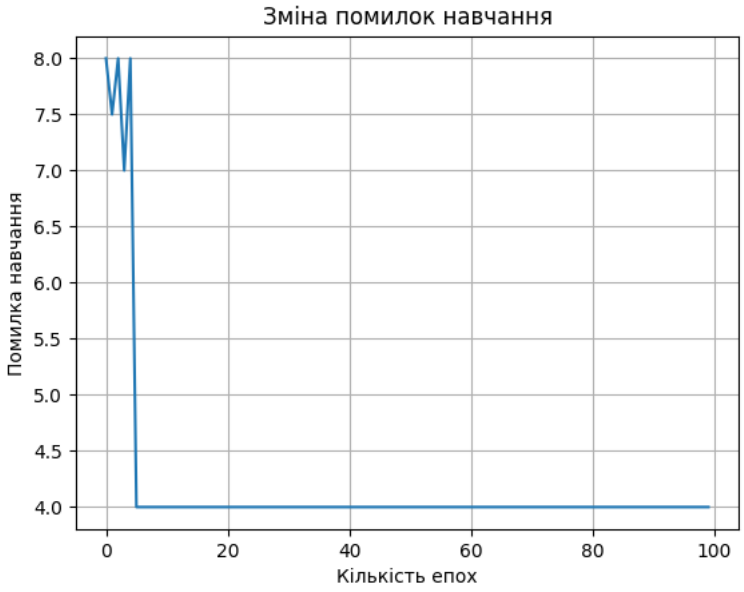

Рис. 5.7 Графік просування процесу навчання

```
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
/Users/annasirach/Desktop/comp/Універ/Штучний
  plt.figure()
/Users/annasirach/Desktop/comp/Універ/Штучний
  plt.show()

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

Рис. 5.8 Результат виконання програми

Висновок: На рис. 20 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 2.5. Побудова багатошарової нейронної мережі

Лістинг програми:

```
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()
```

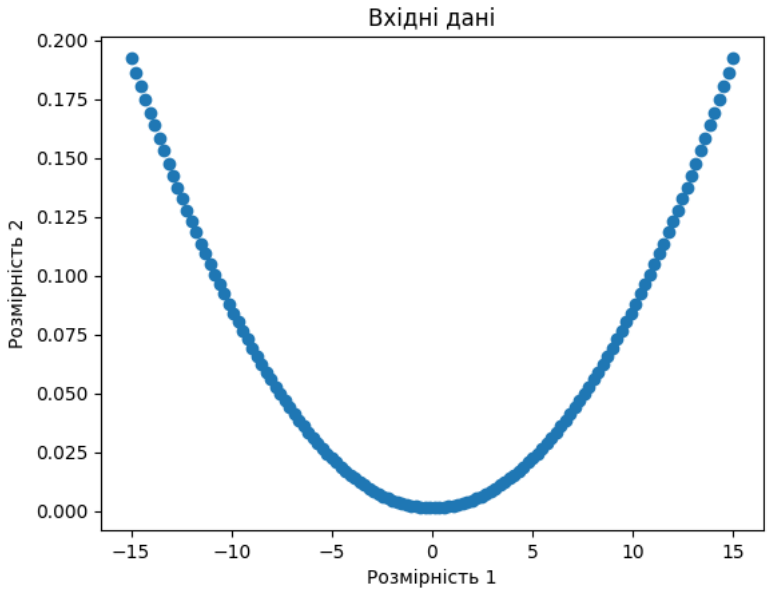

Рис. 5.9 Результат виконання програми

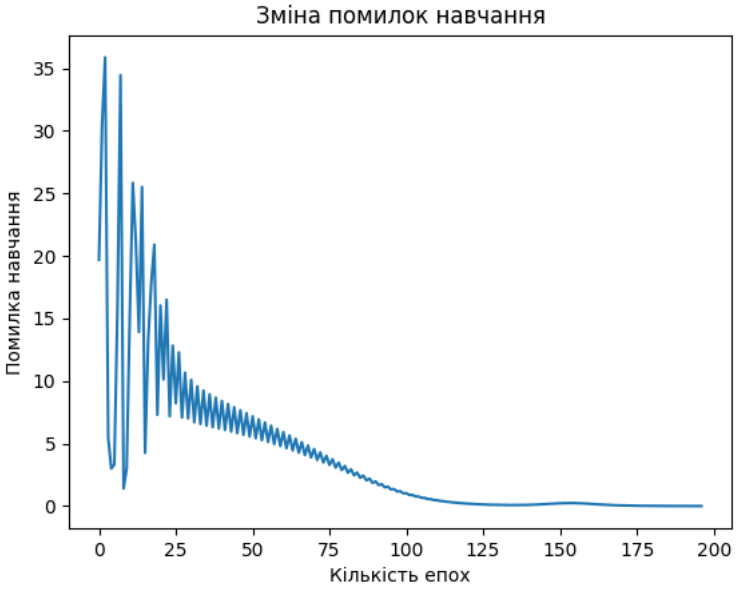

Рис. 5.10 Результат виконання програми

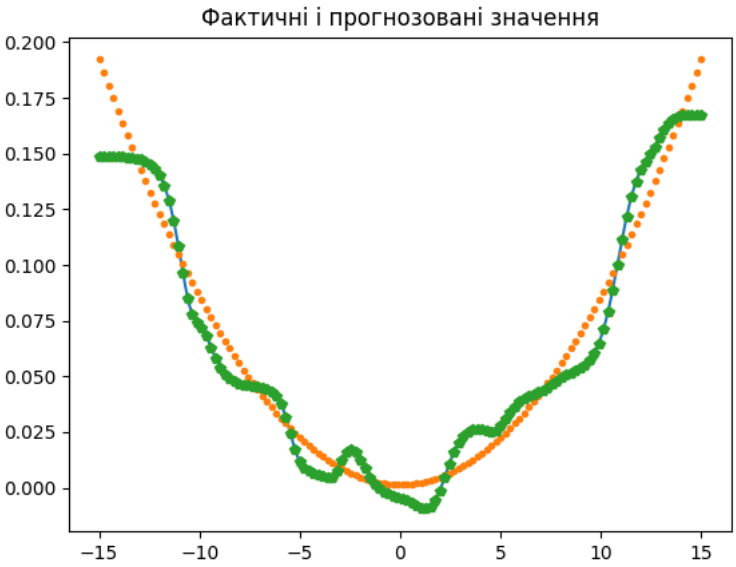

Рис. 5.11 Результат виконання програми

```
Epoch: 100; Error: 0.11505020498354428;
Epoch: 200; Error: 0.029924836110759703;
Epoch: 300; Error: 0.024118043312683683;
Epoch: 400; Error: 0.01967671069851365;
Epoch: 500; Error: 0.015500666839536725;
Epoch: 600; Error: 0.013059370917738688;
Epoch: 700; Error: 0.020197815295033218;
The goal of learning is reached
```

Рис. 5.12 Результат виконання програми

Висновок: на рис. 5.10 зображено процес навчання мережі. На 100 епосі відбулось 1,15 помилки, на 200 епосі відбулось 0,29 помилки, на 300 епосі відбулось 0,02 помилки, на 400 епосі відбулось 0,01 помилки. Вкінці вивелось повідомлення, що ми досягли цілі навчання.

Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

Таблиця 1

№ варіанта	Тестові дані
Варіант 14	$y = 5x^2+5$

Таблиця 2

Номер варіанта	Багатошаровий персептрон	
	Кількість шарів	Кількості нейронів у шарах
14	3	6-3-1

Лістинг програми:

```
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 * x * 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]], [6, 3, 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()
```

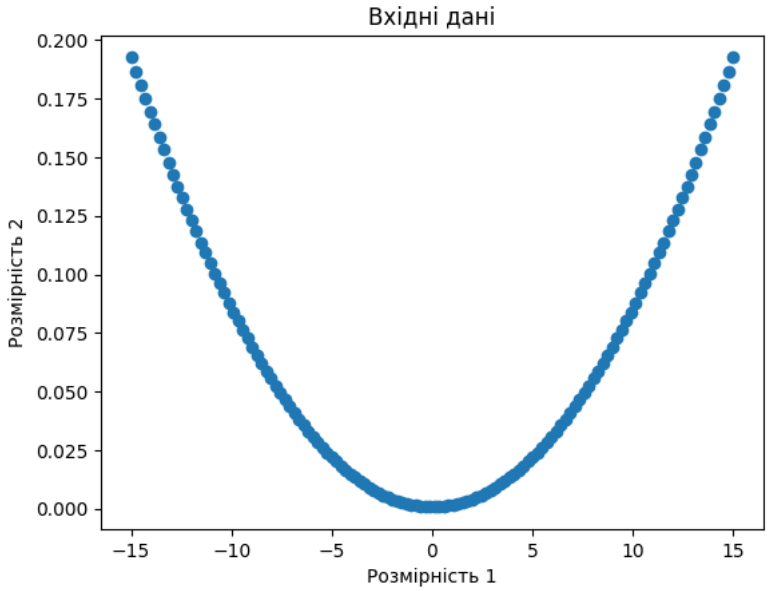

Рис. 5.13 Результат виконання програми

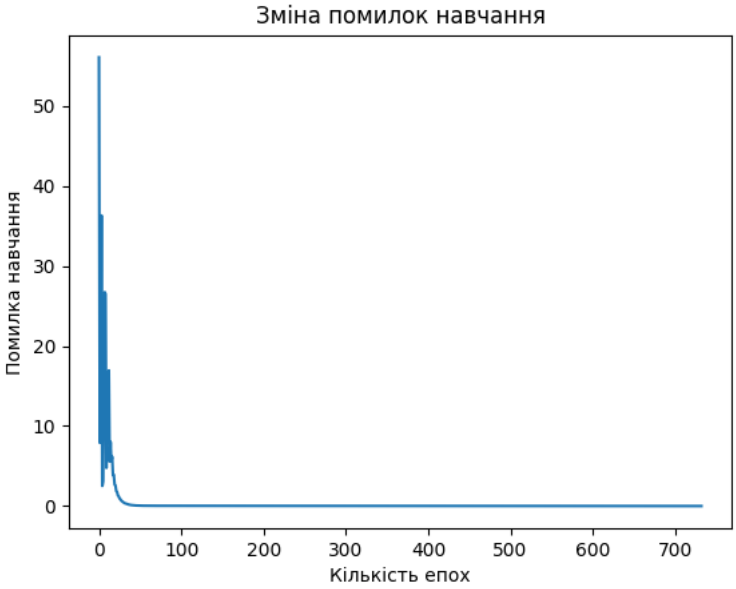

Рис. 5.14 Результат виконання програми

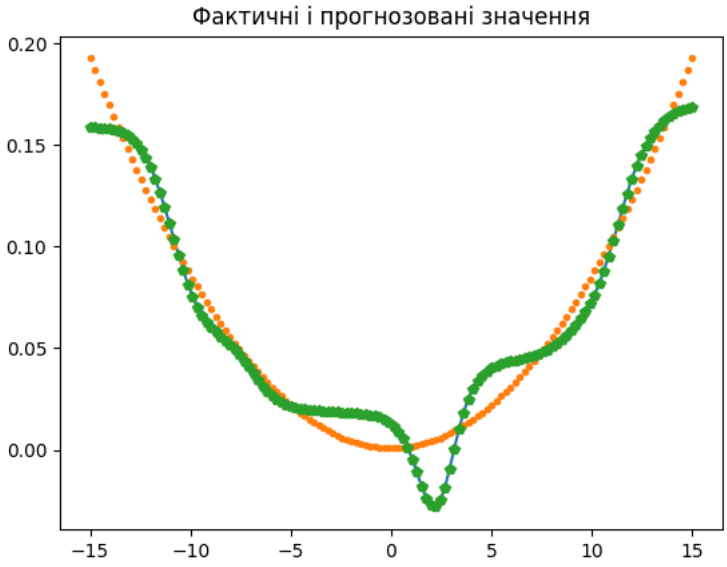

Рис. 5.15 Результат виконання програми

```
Epoch: 100; Error: 0.037373791902495 73;
Epoch: 200; Error: 0.030603183836056702;
Epoch: 300; Error: 0.02511445036372962;
Epoch: 400; Error: 0.020369466324069156;
Epoch: 500; Error: 0.016287634796700687;
Epoch: 600; Error: 0.01300330248301242;
Epoch: 700; Error: 0.01060488937853446;
The goal of learning is reached
```

Рис. 5.16 Результат виконання програми

На рис. 5.14 зображено процес навчання мережі. На 100 епосі відбулось 0,037 помилки, на 200 епосі відбулось 0,03 помилки, на 300 епосі відбулось 0,02 помилки. Вже на 700 епосі відбулось 0,01 помилки. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

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

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()
```

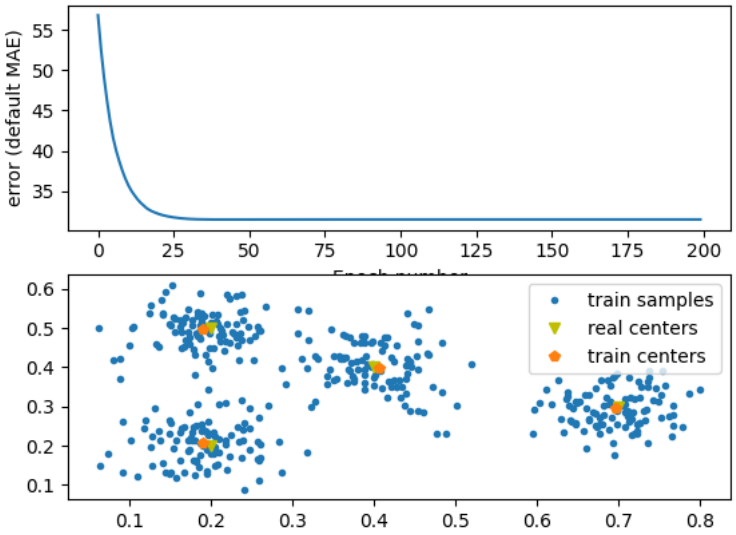

Рис. 5.17 Результат виконання програми

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

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

Таблиця 3

№ варіанту	Центри кластера	skv
Варіант 14	[0.1, 0.2], [0.3, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]	0,05

Лістинг програми з 4 нейронами:

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

skv = 0.05
centr = np.array([[0.1, 0.2], [0.3, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]])
rand_norm = skv * rand.randn(100, 5, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 5, 2)
rand.shuffle(inp)
```

```
# 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()
```

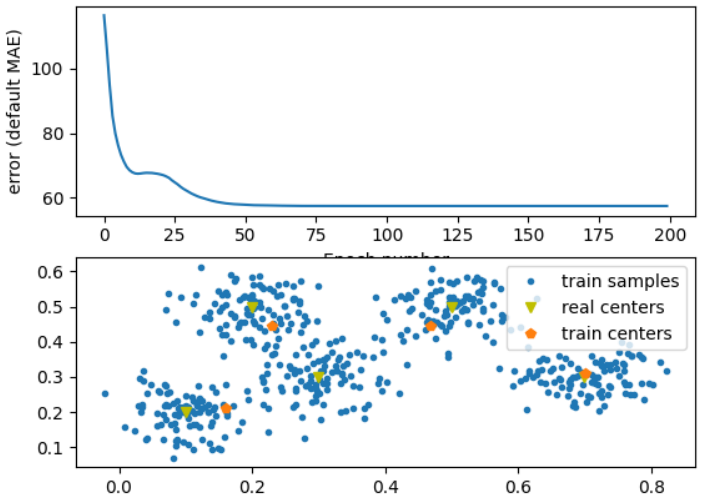

Рис. 5.18 Результат виконання програми

```
Epoch: 20; Error: 67.39341139108593;
Epoch: 40; Error: 58.87910938436842;
Epoch: 60; Error: 57.60050112766683;
Epoch: 80; Error: 57.440898576371495;
Epoch: 100; Error: 57.434215713551836;
Epoch: 120; Error: 57.43332592551165;
Epoch: 140; Error: 57.43319928990693;
Epoch: 160; Error: 57.43318151242063;
Epoch: 180; Error: 57.433179025836786;
Epoch: 200; Error: 57.43317867740812;
The maximum number of train epochs is reached
```

Рис. 5.19 Результат виконання програми

Лістинг програми з 5 нейронами:

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

skv = 0.05
centr = np.array([[0.1, 0.2], [0.3, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]])
rand_norm = skv * rand.randn(100, 5, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 5, 2)
rand.shuffle(inp)

# 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()
```

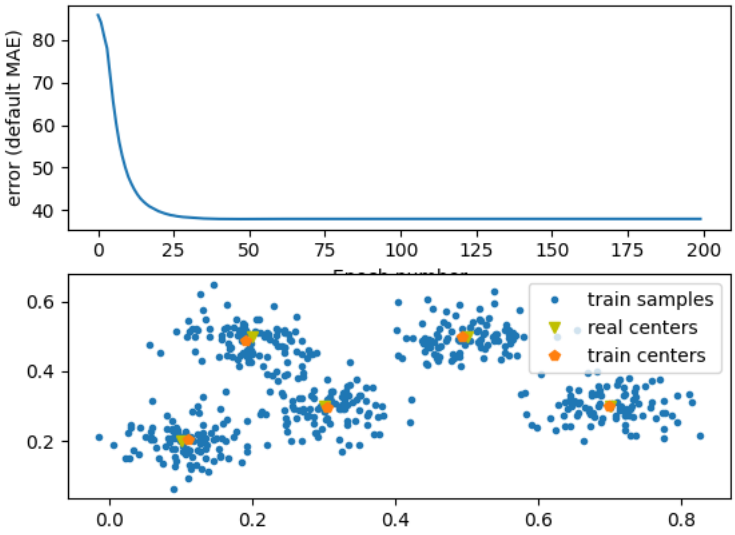

Рис. 5.20 Результат виконання програми

```
Epoch: 20; Error: 40.14969932395324;
Epoch: 40; Error: 37.99573853581643;
Epoch: 60; Error: 37.98403789097898;
Epoch: 80; Error: 37.991975472477435;
Epoch: 100; Error: 37.994813646592114;
Epoch: 120; Error: 37.99527055028848;
Epoch: 140; Error: 37.995344676723164;
Epoch: 160; Error: 37.99535679440077;
Epoch: 180; Error: 37.99535878986135;
Epoch: 200; Error: 37.99535912075953;
The maximum number of train epochs is reached
```

Рис. 5.21 Результат виконання програми

Висновок: при 4 нейронах: На 20 епосі відбулось 67,39 помилки, на 40 епосі відбулось 58,87 помилки, на 60 епосі відбулось 57,6 помилки і так далі, а на 200 епосі відбулось 57,43 помилки. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

При 5 нейронах: На 20 епосі відбулось 40, 14 помилки, на 40 епосі відбулось 37,99 помилки, на 60 епосі відбулось 37, 98 помилки і так далі, а на 200 епосі відбулось 37, 99 помилки. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

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, тому на рис. 18 набагато гірші результати, ніж на рис. 20

Висновок: під час виконання лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python було вивчено як створювати та застосовувати прості нейронні мережі**.**

Посилання на GitHub: https://github.com/annasirach/AI_IPZ193_Sirach