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 1. Створити простий нейрон

Лістинг коду файлу Task_1.py:

```
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)


if __name__ == "__main__":
    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
```

```
0.9990889488055994
```

Рис.5.1 – Результат роботи нейрона

Завдання 2. Створити просту нейронну мережу для передбачення статі людини

Лістинг коду файлу Task_2.py:

```
import numpy as np
from Task_1 import Neuron, sigmoid


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

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


class OleksiichukNeuralNetwork:
    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)

                # Neuron o1
                d_ypred_d_w5 = h1 * derivative_sigmoid(sum_o1)
                d_ypred_d_w6 = h2 * derivative_sigmoid(sum_o1)
                d_ypred_d_b3 = derivative_sigmoid(sum_o1)

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

                # Neuron h1
                d_h1_d_w1 = x[0] * derivative_sigmoid(sum_h1)
                d_h1_d_w2 = x[1] * derivative_sigmoid(sum_h1)
                d_h1_d_b1 = derivative_sigmoid(sum_h1)

                # Neuron h2
                d_h2_d_w3 = x[0] * derivative_sigmoid(sum_h2)
                d_h2_d_w4 = x[1] * derivative_sigmoid(sum_h2)
                d_h2_d_b2 = derivative_sigmoid(sum_h2)

                # Update weights and biases
                # Neuron 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

                # Neuron 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
```

```
                # Neuron 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__":
    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 = OleksiichukNeuralNetwork()
    network.train(data, all_y_trues)

    emily = np.array([-7, -3])  # 128 pounds, 63 inches
    frank = np.array([20, 2])  # 155 pounds, 68 inches
    print("Emily: %.3f" % network.feedforward(emily))  # +-0.96 - F
    print("Frank: %.3f" % network.feedforward(frank))  # +-0.039 - M
```

```
Epoch 950 loss: 0.002
Epoch 960 loss: 0.002
Epoch 970 loss: 0.002
Epoch 980 loss: 0.001
Epoch 990 loss: 0.001
Emily: 0.966
Frank: 0.038
```

Рис.5.2 – Результат навчання нейронної мережі

Функція активації необхідна для підключення незв'язаних вхідних даних із виходом з простою та передбачуваною формою. Нейронні мережі прямого поширення дозволяють, використовуючи функції активації, передбачати відповідь (класифікувати).

Завдання 3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

Лістинг коду файлу Task_3.py:

```
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('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')
plt.show()

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('Number of epochs')
plt.ylabel('Training error')
plt.title('Training error progress')
plt.grid()
plt.show()
```

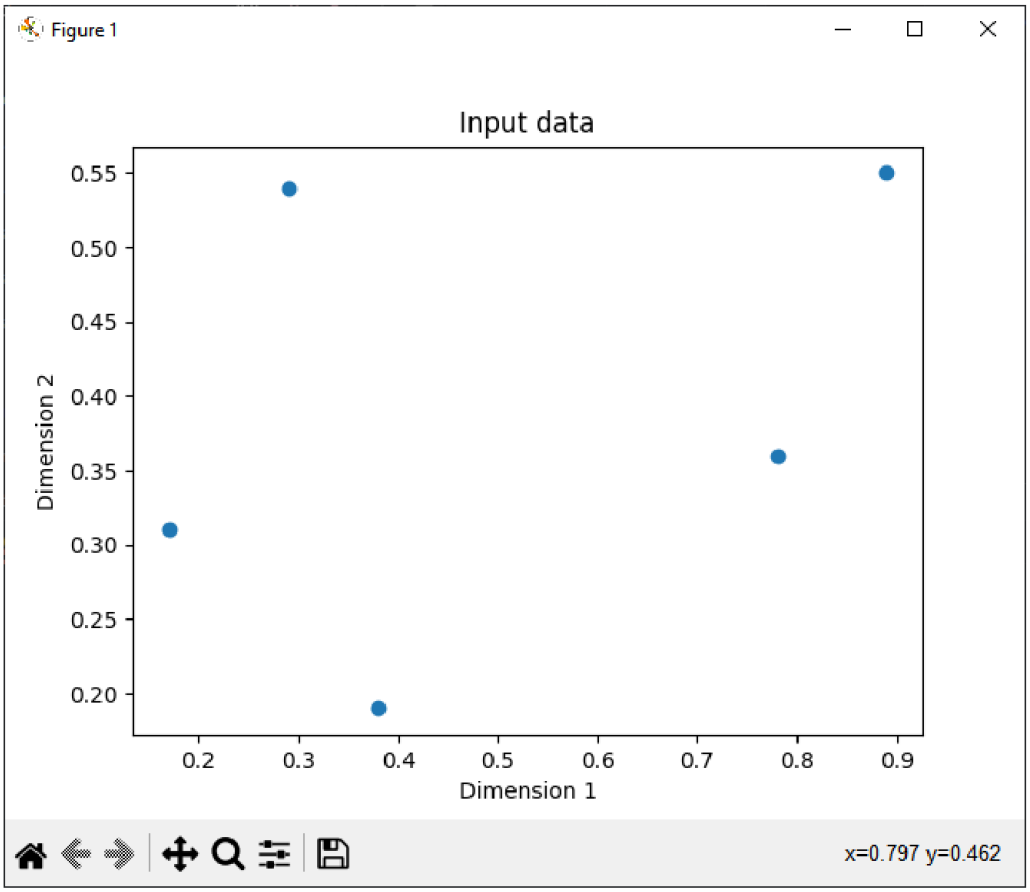

Рис.5.3 – Вхідні дані до перцептрону

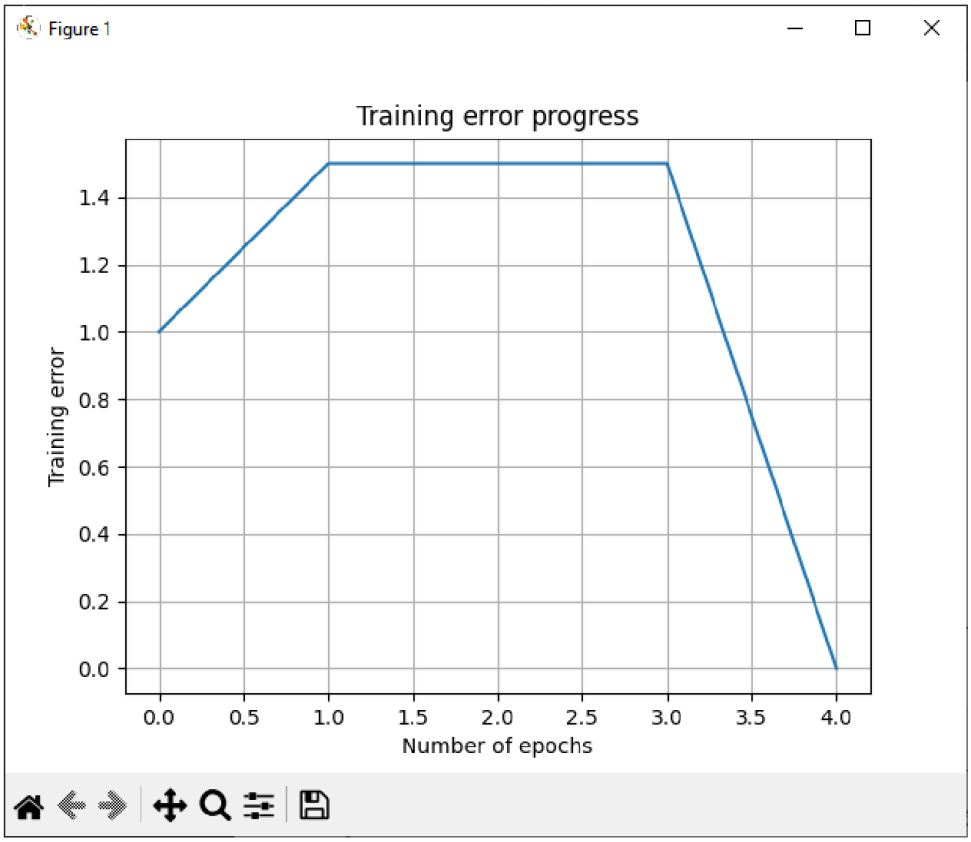

Рис.5.4 – Навчання перцептрону

Завдання 4. Побудова одношарової нейронної мережі

Лістинг коду файлу Task_4.py:

```
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('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')
plt.show()

dim1 = [data[:, 0].min(), data[:, 0].max()]
dim2 = [data[:, 1].min(), data[:, 1].max()]
num output = labels.shape[1]

nn = nl.net.newff([dim1, dim2], [3, num output])
error_progress = nn.train(data, labels, epochs=1000, show=100, goal=0.02)

plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Training error progress')
plt.grid()
plt.show()
```

```
print('Test 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])
```

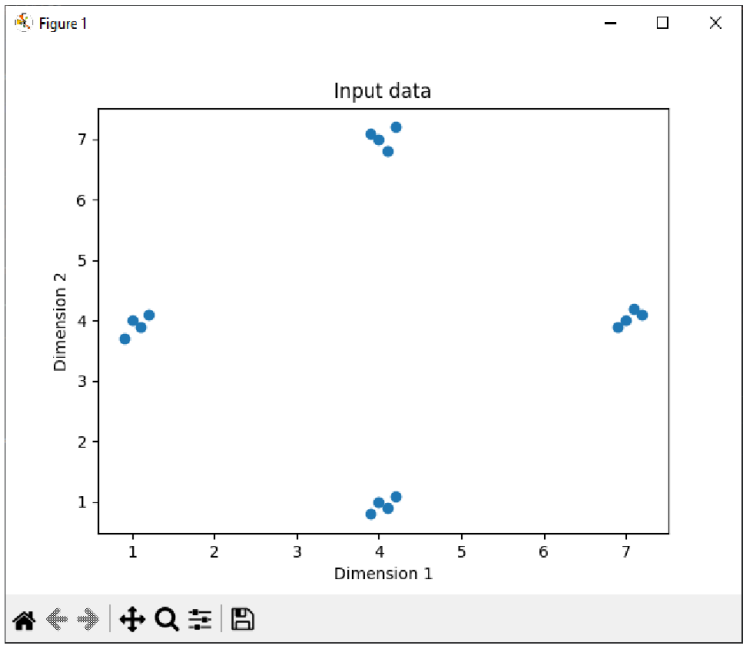

Рис.5.5 – Вхідні дані до нейронної мережі

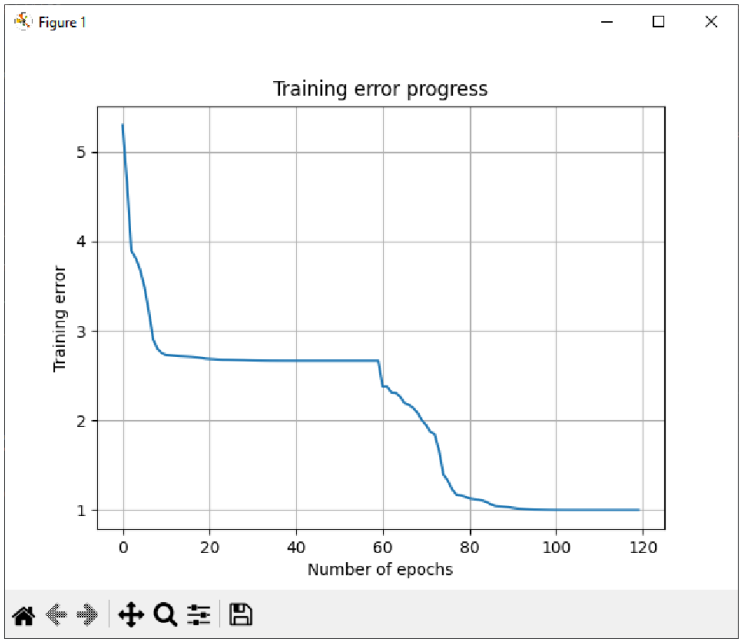

Рис.5.6 – Навчання мережі

```
Epoch: 100; Error: 1.001134734737751;
Test results:
[0.4, 4.3] --> [-3.47916952e-07 -1.40064911e-08]
[4.4, 0.6] --> [ 9.99999987e-01 -1.47756474e-10]
[4.7, 8.1] --> [0.49999997 0.99997127]
```

Рис.5.7 – Тестові результати

Завдання 5. Побудова багатошарової нейронної мережі

Лістинг коду файлу Task_5.py:

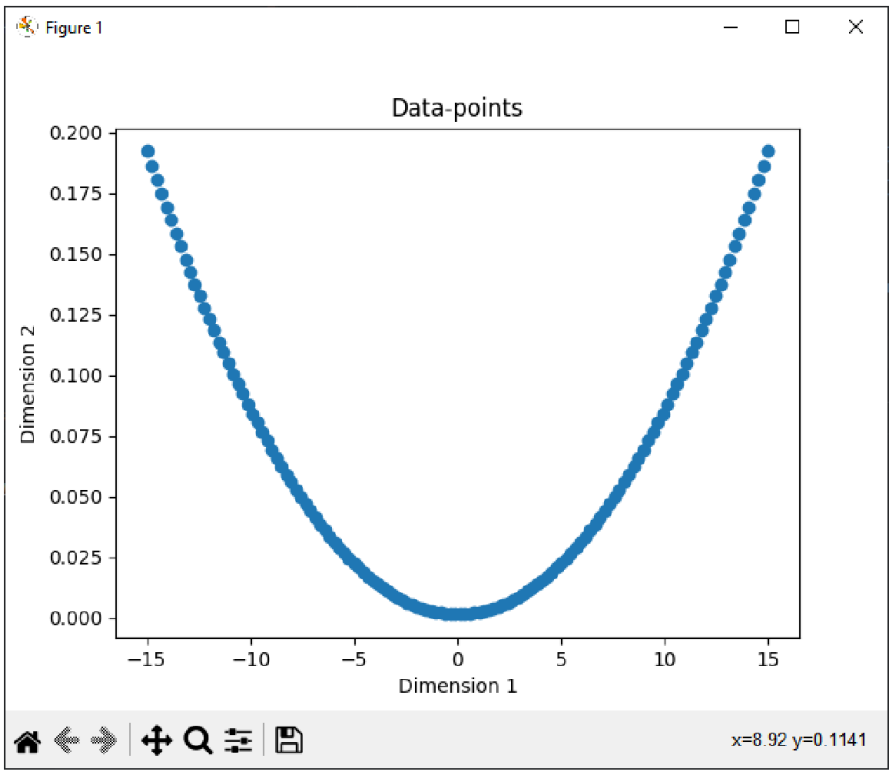

Рис.5.8 – Дані рівняння $3x^2+5$

```
Epoch: 100; Error: 0.0651405908758953;
Epoch: 200; Error: 0.24253124855257335;
Epoch: 300; Error: 0.04089600582050827;
Epoch: 400; Error: 0.05295252146824097;
Epoch: 500; Error: 0.03692260087324199;
Epoch: 600; Error: 0.020979509300989553;
The goal of learning is reached
```

Рис.5.9 – Звітність про навчання по епохам

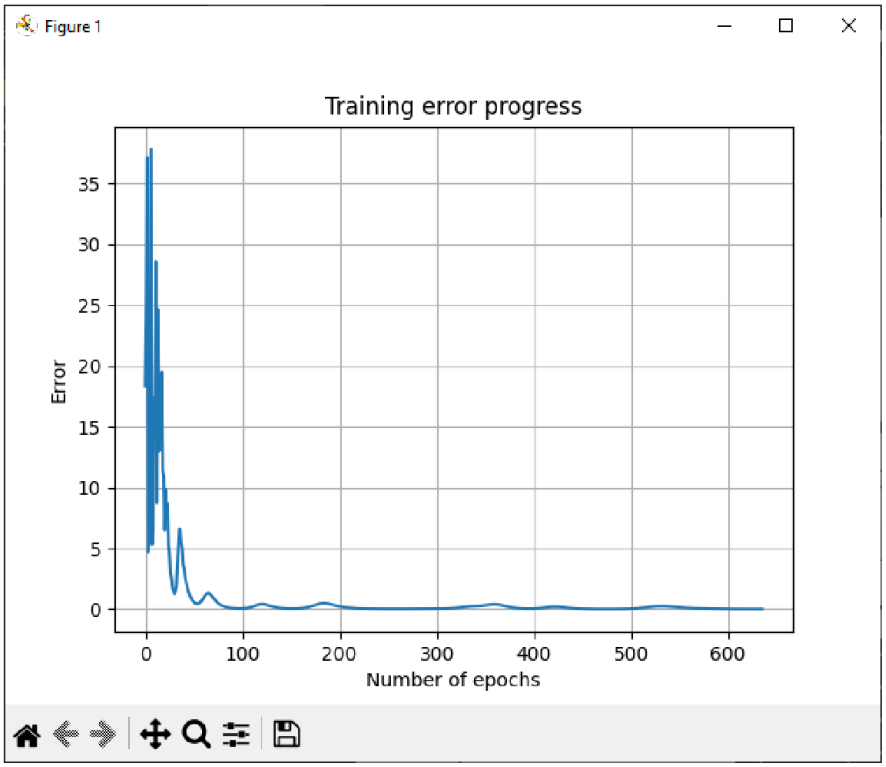

Рис.5.10 – Графік навчання мережі

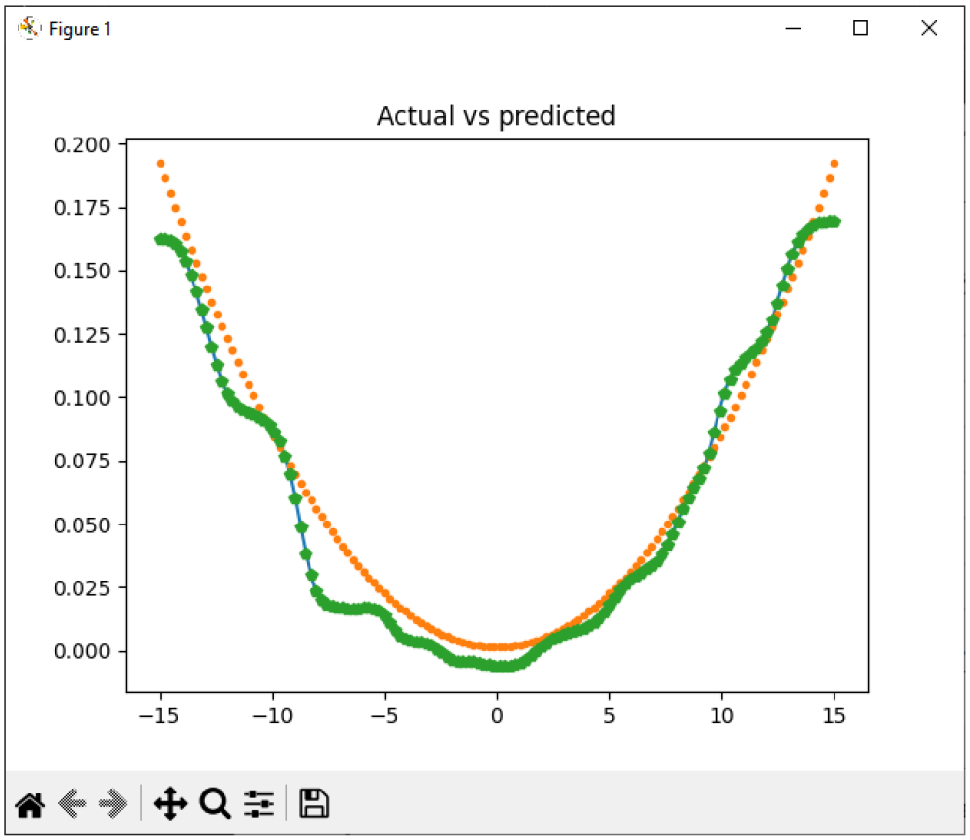

Рис.5.11 – Графік-порівняння істинних та отриманих даних

Завдання 6. Побудова багатошарової нейронної мережі для свого варіанту

Варіант 10, дані: $y = 5x^2+1$, кількість шарів: 2, кількість нейронів: 4-1

Лістинг коду файлу Task_6.py:

```
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) + 1
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('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Data-points')
plt.show()

nn = nl.net.newff([[min_val, max_val]], [4, 1])
nn.trainf = nl.train.train_gd
error_progress = nn.train(data, labels, epochs=3000, show=100, goal=0.01)

output = nn.sim(data)
y pred = output.reshape(num points)

plt.figure()
plt.plot(error progress)
plt.xlabel('Number of epochs')
plt.ylabel('Error')
plt.title('Training error progress')
plt.grid()
plt.show()

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('Actual vs predicted')
plt.show()
```

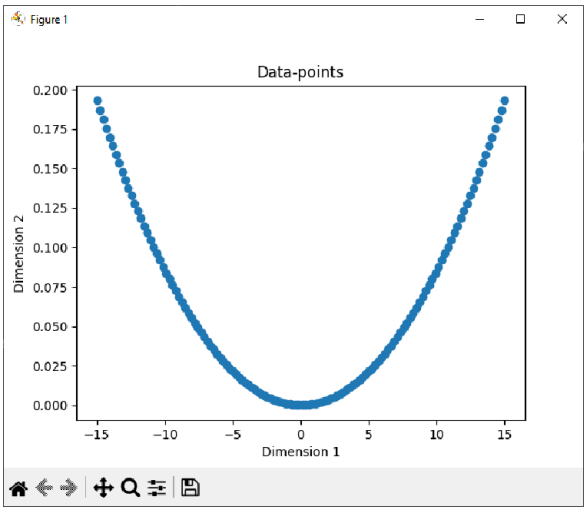

Рис.5.12 – Графік вхідних даних по варіанту

```
Epoch: 100; Error: 4.795546553908089;
Epoch: 200; Error: 7.853865244622360 5;
Epoch: 300; Error: 6.576102435373703;
Epoch: 400; Error: 5.586859097459907;
Epoch: 500; Error: 6.6989063425565725;
Epoch: 600; Error: 3.8843394721550246;
```

Рис.5.13 – Звітність навчання по епохам (вивід 100х епох)

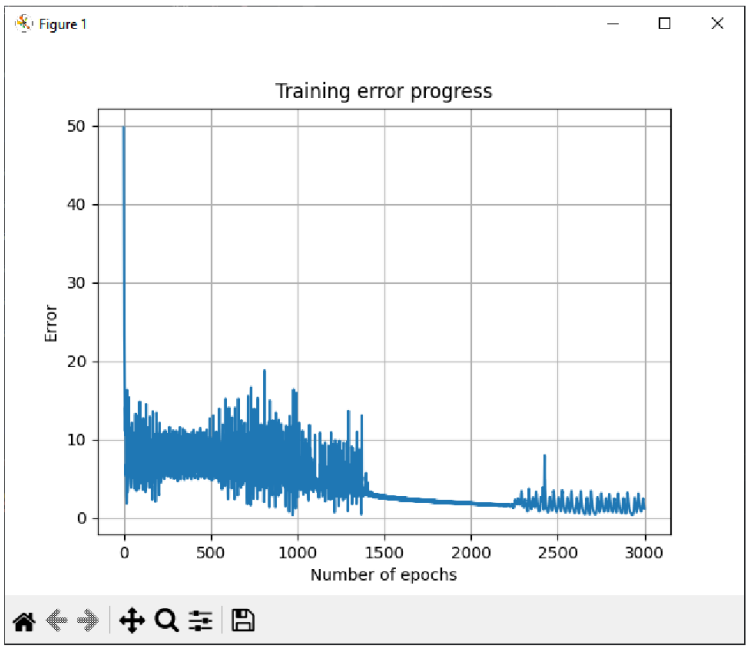

Рис.5.14 – Прогрес помилковості при навчанні

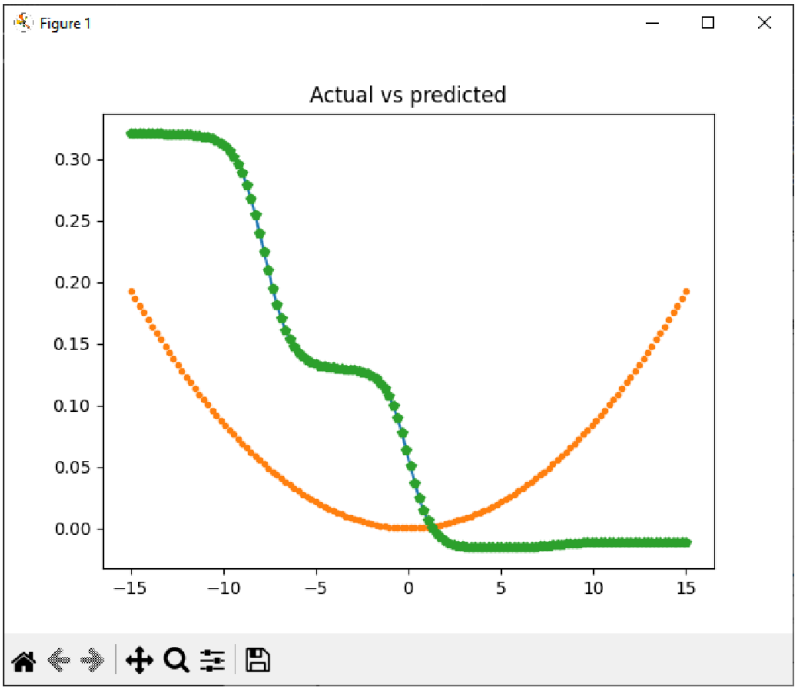

Рис.5.15 – Графік-порівняння дійсних та передбачених даних

У результаті навчання точність нейронної мережі є досить низькою, що може бути пов’язано з кількістю шарів або нейронів у шарах.

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

Лістинг коду файлу Task_7.py:

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

skv = .05
center = np.array([[.2, .2], [.4, .4], [.7, .3], [.2, .5]])
random_norm = skv * rand.randn(100, 4, 2)
inp = np.array([center + r for r in random norm])
inp = inp.reshape(100 * 4, 2)
rand.shuffle(inp)

net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)
error = net.train(inp, epochs=200, show=20)

pl.title('Classification problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('error (default SSE)')
w = net.layers[0].np['w']

pl.subplot(212)
pl.plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0], w[:, 1], 'p')
pl.legend(['train samples', 'centers', 'train centers'])
pl.show()
```

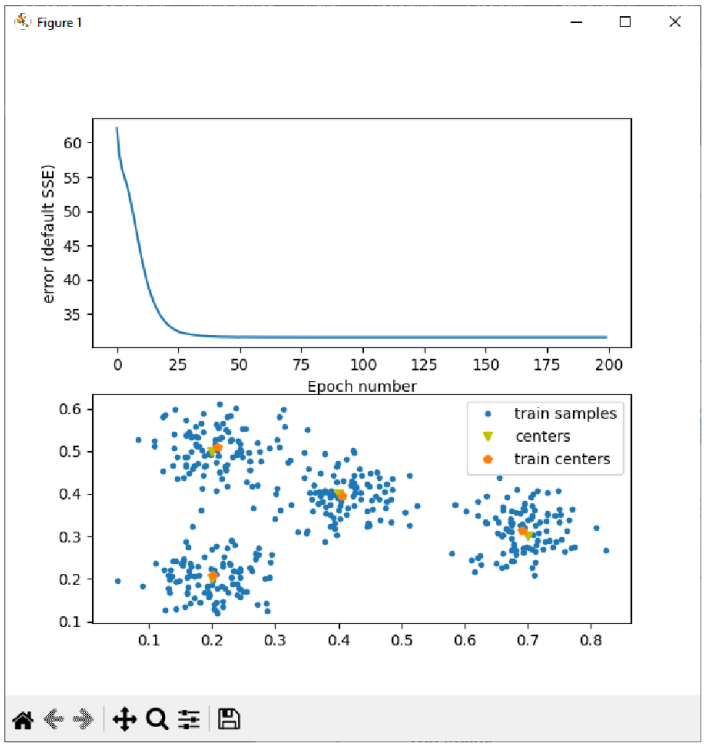

Рис.5.18 – Графік помилковості по епохам та класифікація центрів

```
Epoch: 20; Error: 32.989431468402245;
Epoch: 40; Error: 30.58234932715783;
Epoch: 60; Error: 30.576163749569844;
Epoch: 80; Error: 30.58228409888426;
Epoch: 100; Error: 30.58344432535441;
Epoch: 120; Error: 30.583625194573955;
Epoch: 140; Error: 30.583653538126306;
```

Рис.5.17 – Звітність навчання

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

Варіант 10, центри: [0.2, 0.2], [0.3, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7], skv = 0.04.

Лістинг коду файлу Task_8.py:

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

skv = .04
center = np.array([[0.2, 0.2], [0.3, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]])
random_norm = skv * rand.randn(100, 5, 2)
inp = np.array([center + r for r in random_norm])
inp = inp.reshape(100 * 5, 2)
rand.shuffle(inp)

net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)
error = net.train(inp, epochs=200, show=20)

pl.title('Classification problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('error (default SSE)')
w = net.layers[0].np['w']

pl.subplot(212)
pl.plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0], w[:, 1], 'p')
pl.legend(['train samples', 'centers', 'train centers'])
pl.show()
```

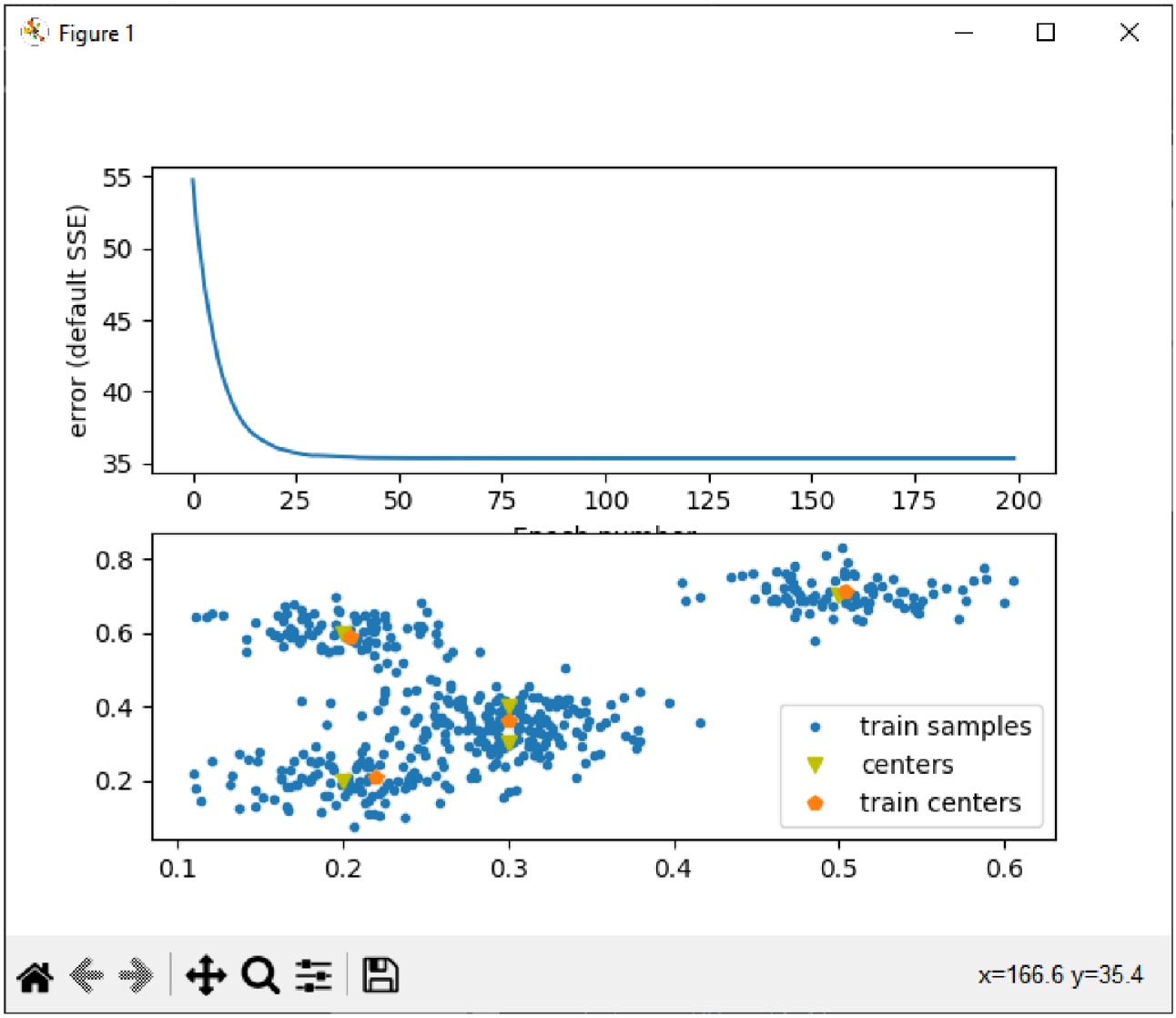

Рис.5.18 – Графік навчання та класифікації за 4х нейронів

```
Epoch: 20; Error: 38.27698501954072;
Epoch: 40; Error: 37.274441230031044;
Epoch: 60; Error: 37.16365121194071;
Epoch: 80; Error: 37.15426507679648;
Epoch: 100; Error: 37.151876789587355;
```

Рис.5.19 – Звітність за 4х нейронів

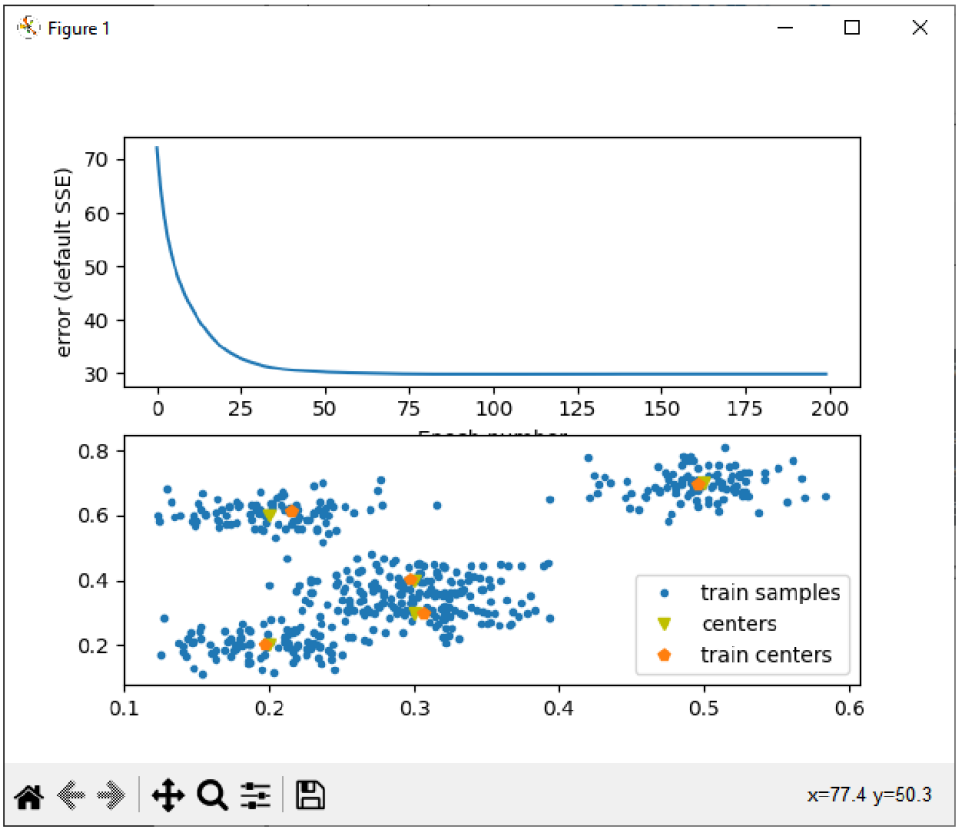

Рис.5.22 – Графік навчання та класифікація за 5ти нейронів

```
Epoch: 20; Error: 35.08560754503296;
Epoch: 40; Error: 30.7315222279;
Epoch: 60; Error: 30.091050729343742;
Epoch: 80; Error: 29.867167502067154;
Epoch: 100; Error: 29.851217376615438;
```

Рис.5.21 – Звітність за 5ти нейронів

Висновок: під час виконання завдань лабораторної роботи було отримано навички зі створення та застосовування простих нейронних мереж використовуючи спеціалізовані бібліотеки та мову програмування Python.

Посилання на репозиторій: https://github.com/nikitoss888/AI_LR5