Лабораторна робота №6

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

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

Завдання на лабораторну роботу

Завдання 2.1. Створити простий нейрон

Лістинг програми

```python
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])
    bias = 4
    n = Neuron(weights, bias)
    x = np.array([2, 3])
    print(n.feedforward(x))
```

Результат виконання

```
0.9990889488055994
PS E:\labs\WIKI\lab6>
```

Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини

Лістинг програми

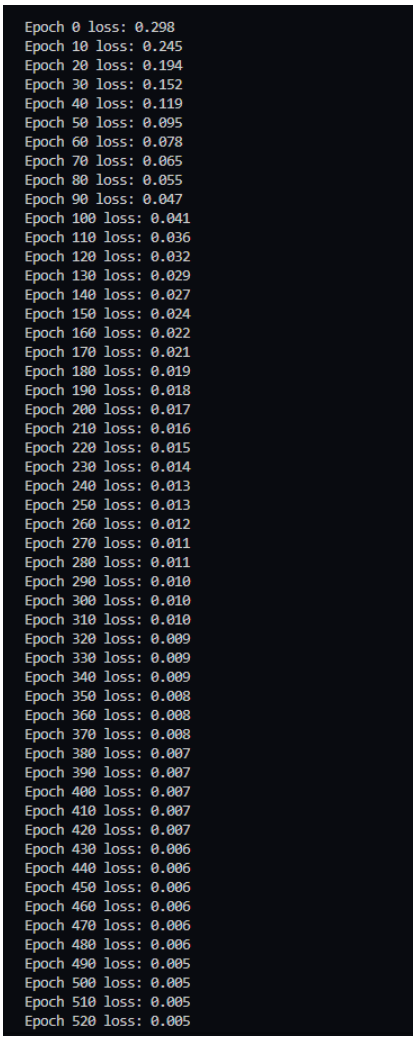
```python
import numpy as np
from lab6_task1 import Neuron, sigmoid

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 MischenchukNeuralNetwork:
    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
                # --- Підрахунок часткових похідних
                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__":
    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 = MischenchukNeuralNetwork()
    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

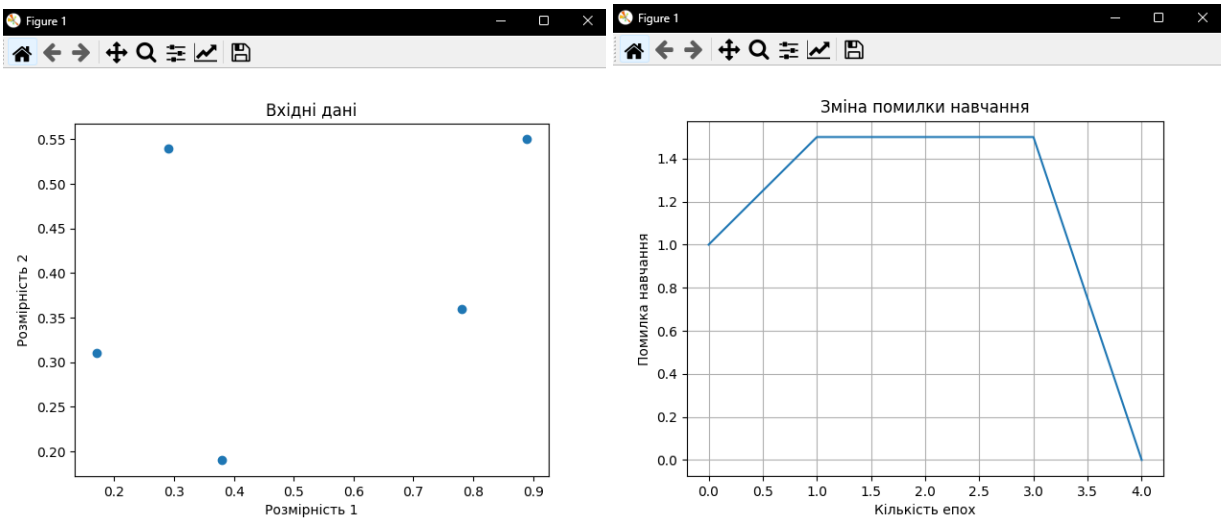


Завдання 2.3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

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')
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('Кількість епох')
plt.ylabel('Помилка навчання')
plt.title('Зміна помилки навчання')
plt.grid()
plt.show()
```

Результат виконання



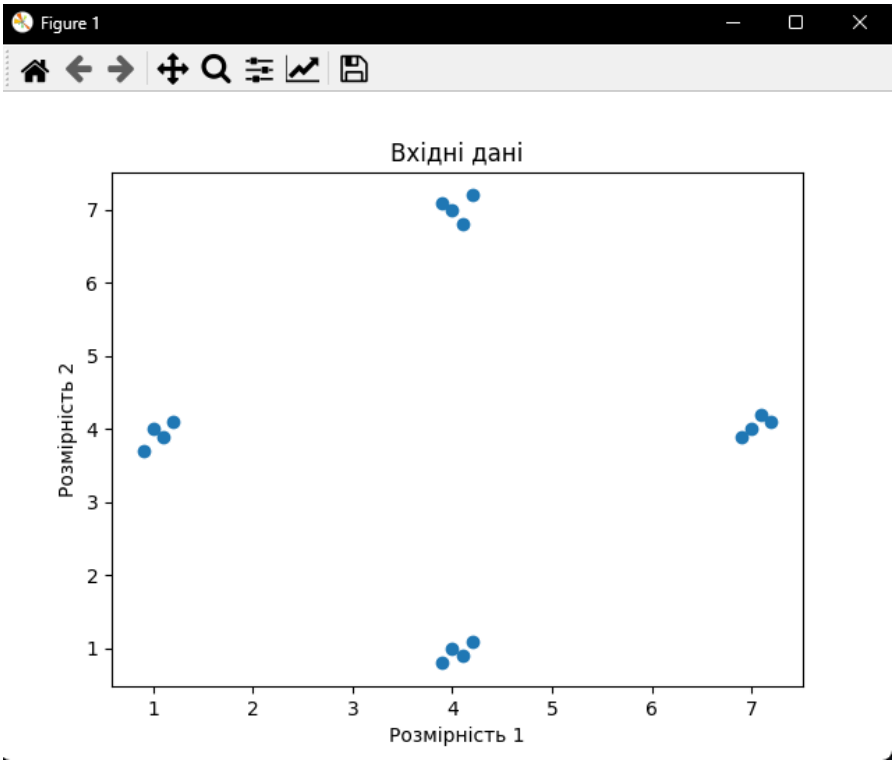
Завдання 2.4. Побудова одношарової нейронної мережі

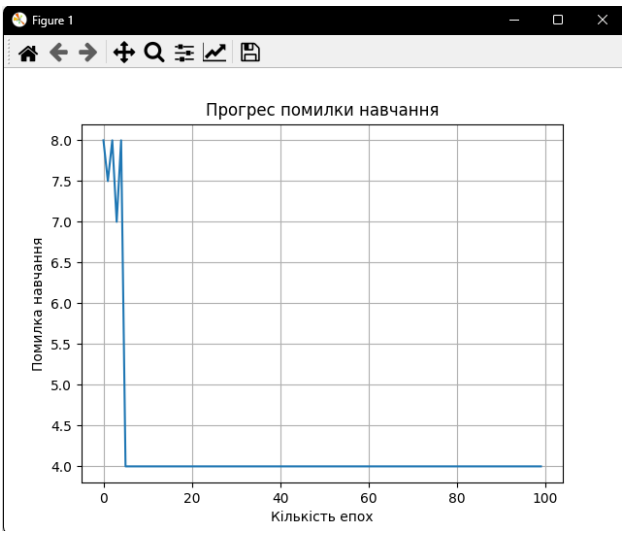
Лістинг програми

```
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')
plt.show()
# 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()
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)
# Побудова графіка просування процесу навчання
plt.figure()
```

```
plt.plot(error_progress)
plt.xlabel('Кількість епох')
plt.ylabel('Помилка навчання')
plt.title('Прогрес помилки навчання')
plt.grid()
plt.show()
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






```
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.]
PS E:\labs\WIKI\lab6>
```

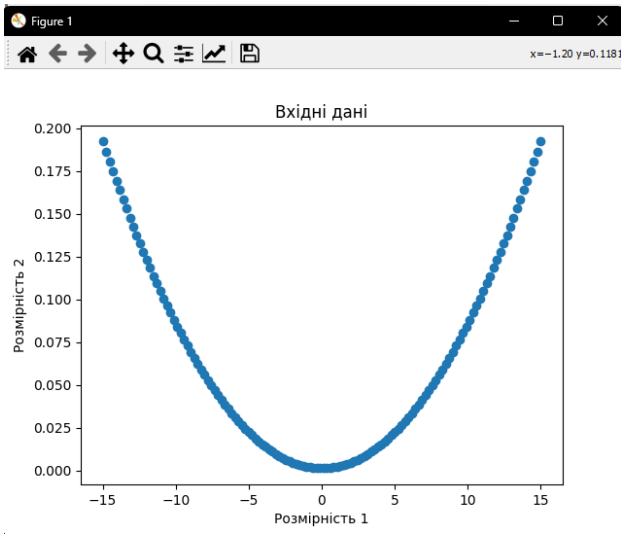
Завдання 2.5. Побудова багатошарової нейронної мережі

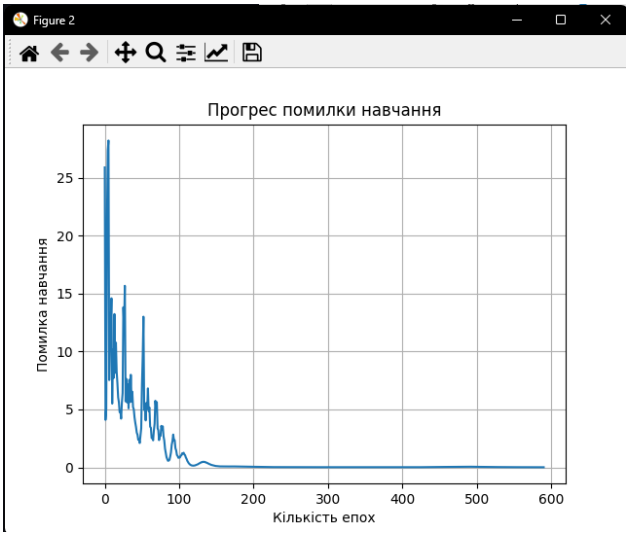
Лістинг програми

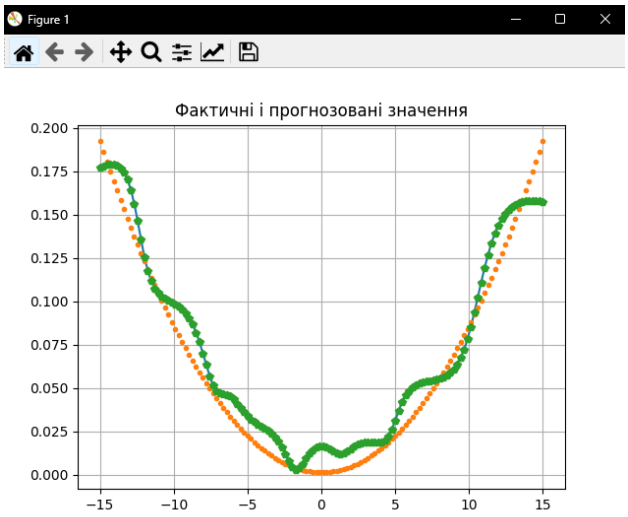
```python
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# 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)
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('Прогрес помилки навчання')
plt.grid()
plt.show()
# Побудова графіка результатів
x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred = nn.sim(x_dense.reshape(x_dense.size, 1)).re-
shape(x_dense.size)
plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
plt.title('Фактичні і прогнозовані значення')
plt.grid()
plt.show()
```

Результат виконання








```
Epoch: 100; Error: 0.8182485053247603;
Epoch: 200; Error: 0.04240091211658453;
Epoch: 300; Error: 0.014264281042282436;
Epoch: 400; Error: 0.011649244695865514;
Epoch: 500; Error: 0.048323737530608646;
The goal of learning is reached
PS E:\labs\ШIKI\lab6>
```

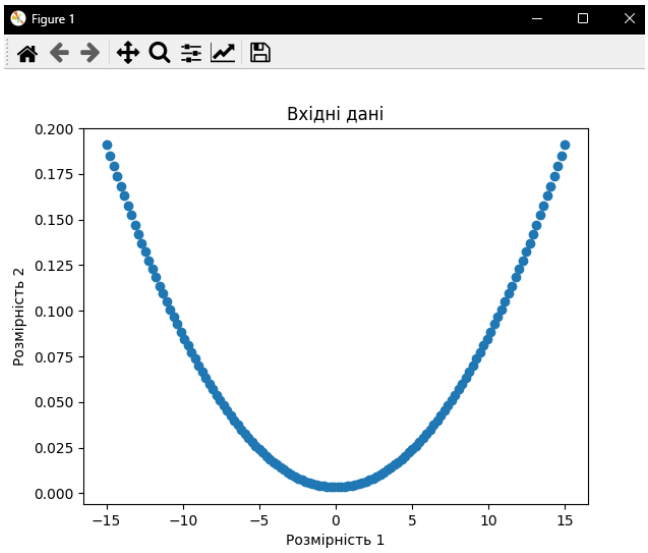
Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

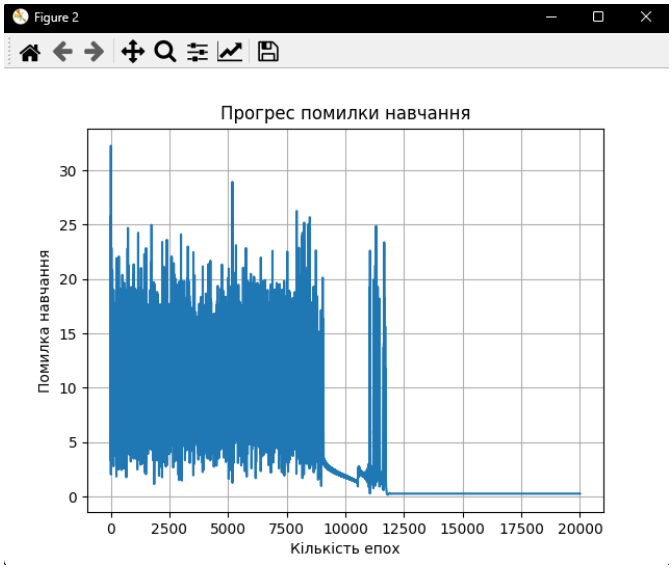
Лістинг програми

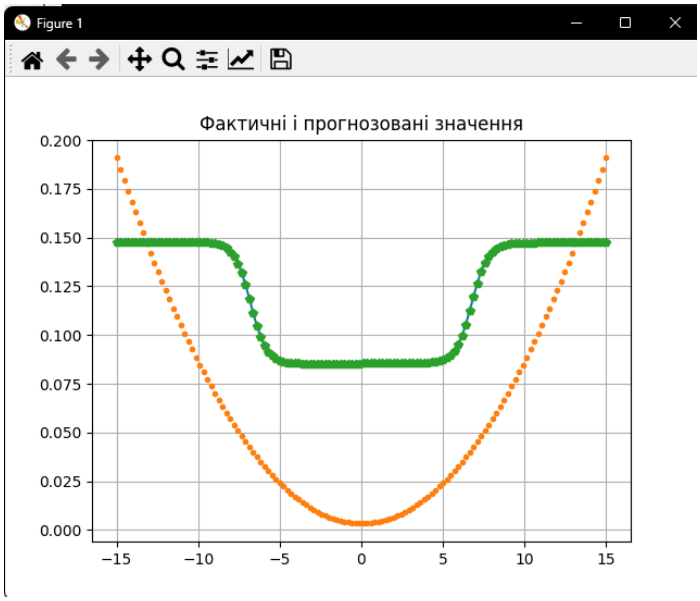
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

min_val = -15
max_val = 15
num_points = 130
```

```
x = np.linspace(min_val, max_val, num_points)
y = 2 * 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]], [5, 1])
nn.trainf = nl.train.train_gd
# Тренування нейронної мережі
error_progress = nn.train(data, labels, epochs=20000, show=1000, 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('Прогрес помилки навчання')
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)).re-
shape(x_dense.size)
plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
plt.title('Фактичні і прогнозовані значення')
plt.grid()
plt.show()
```

Результат виконання








```
Epoch: 1000; Error: 7.8852386276906685;
Epoch: 2000; Error: 8.813776707714528;
Epoch: 3000; Error: 8.137017777745692;
Epoch: 4000; Error: 6.571569108868559;
Epoch: 5000; Error: 6.967763442031128;
Epoch: 6000; Error: 2.582323684220567;
Epoch: 7000; Error: 9.040368598924669;
Epoch: 8000; Error: 6.99651529916205;
Epoch: 9000; Error: 3.8174824750241685;
Epoch: 10000; Error: 1.738118354068051;
Epoch: 11000; Error: 1.0016367088015428;
Epoch: 12000; Error: 0.25493253339339594;
Epoch: 13000; Error: 0.26334436853686816;
Epoch: 14000; Error: 0.2633733840679719;
Epoch: 15000; Error: 0.26338648097252404;
Epoch: 16000; Error: 0.26339576344158333;
Epoch: 17000; Error: 0.26340365378803543;
Epoch: 18000; Error: 0.2634109511044233;
Epoch: 19000; Error: 0.2634179685282896;
Epoch: 20000; Error: 0.26342483475870965;
The maximum number of train epochs is reached
```

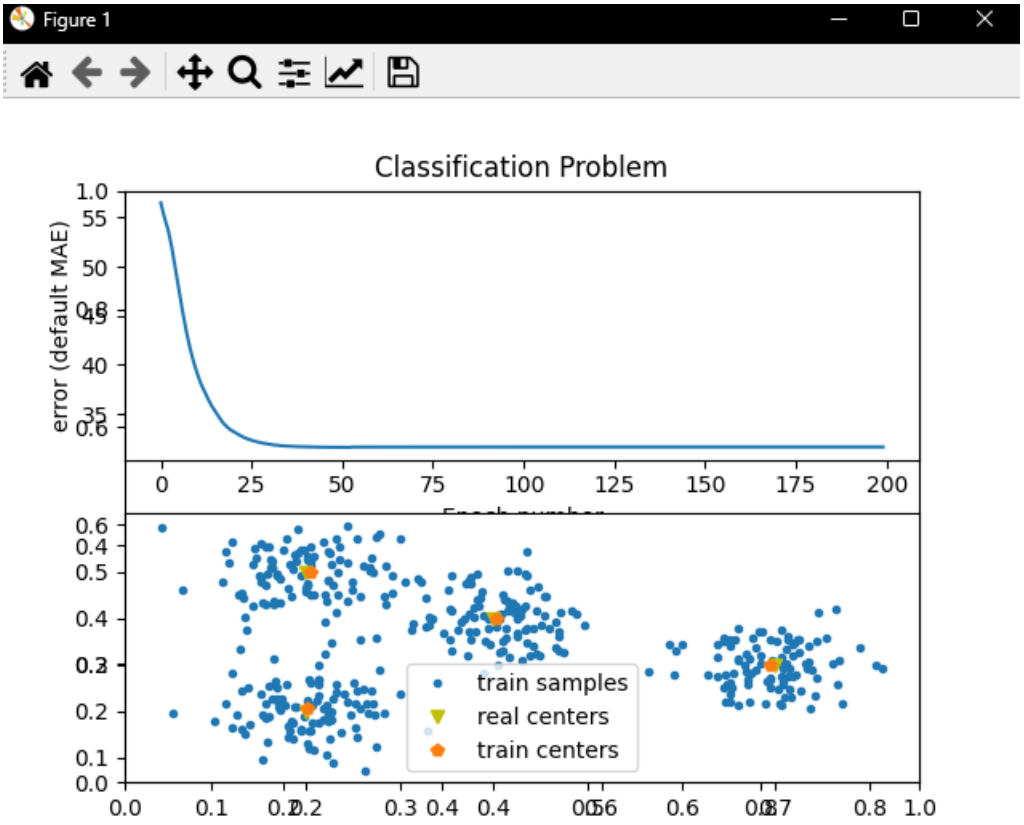
Завдання 2.7. Побудова нейронної мережі на основі карти Кохонена, що самоорганізується

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=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()
```

Результат виконання




```
Epoch: 20; Error: 33.43816140935912;
Epoch: 40; Error: 31.668802038807318;
Epoch: 60; Error: 31.642494568034344;
Epoch: 80; Error: 31.650123024067476;
Epoch: 100; Error: 31.646558939603615;
Epoch: 120; Error: 31.647119800120123;
Epoch: 140; Error: 31.647281230656375;
Epoch: 160; Error: 31.647316901989342;
Epoch: 180; Error: 31.647323994201223;
Epoch: 200; Error: 31.64732532873553;
The maximum number of train epochs is reached
```

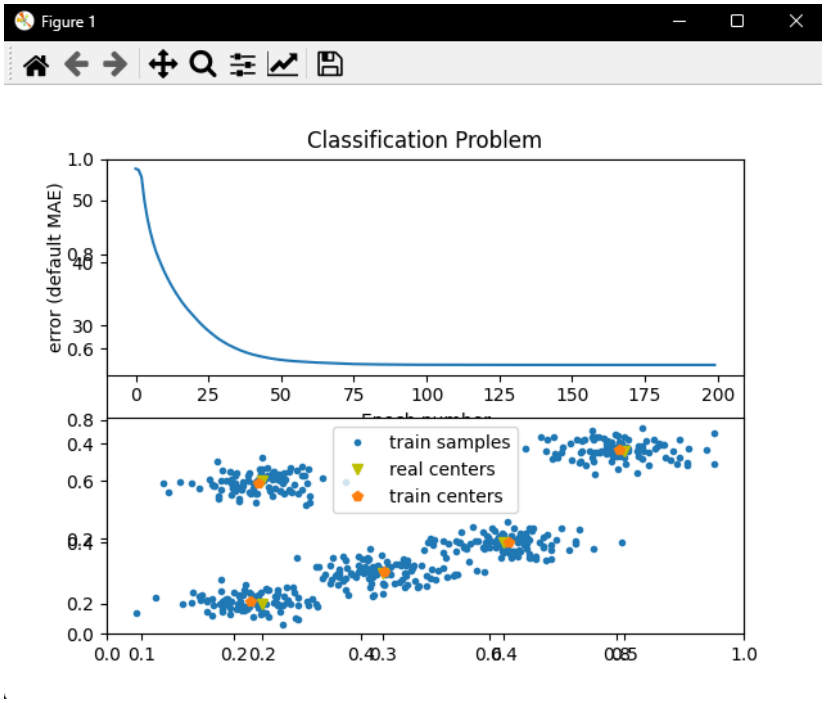
Завдання 2.8. Дослідження нейронної мережі на основі карти Кохонена, що самоорганізується

Лістинг програми

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

```
skv = 0.03
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5,
0.7]])
rand_norm = skv * rand.randn(100, 5, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 5, 2)
rand.shuffle(inp)
# Create net with 2 inputs and 4 neurons
net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 5)
# train with rule: Conscience Winner Take All algoritm (CWTA)
error = net.train(inp, epochs=200,  show=20)
# Plot results:
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()
```

Результат виконання




```
Epoch: 20; Error: 31.949456273753594;
Epoch: 40; Error: 25.483286337894747;
Epoch: 60; Error: 24.105595953918154;
Epoch: 80; Error: 23.75468960486186;
Epoch: 100; Error: 23.67430664979983;
Epoch: 120; Error: 23.65259929336198;
Epoch: 140; Error: 23.645749670171988;
Epoch: 160; Error: 23.643837946903464;
Epoch: 180; Error: 23.643419079695246;
Epoch: 200; Error: 23.6432986395616;
The maximum number of train epochs is reached
```

Посилання на GitHub - https://github.com/MischenchukMykola/lab6

Висновок: виконуючи цю лабораторну роботу я використовуючи спеціалізовані бібліотеки та мову програмування Python навчився створювати та застосовувати прості нейронні мережі**.**