Практична робота № 5

Варіант 13

Розробка простих нейронних мереж

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

Хід роботи:

Завдання 5.1: Створити простий нейрон.

Лістинг 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
```

```
C:\Python311\python.exe C:/Users/Ola/
0.9990889488055994
```

Рис.5.1. task-1.py

					ДУ «Житомирська політехніка».22.121.13.000 – Лр05		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Маковська О.Ю			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 10
Перевір.		Пулеко І. В.					
Керівник						ФІКТ Гр. ІПЗ-19-1[2]	
Н. контр.							
Зав. каф.							

Завдання 5.2: Створити просту нейронну мережу для передбачення статі людини.

Лістинг файлу task-2.py

```
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 MakovskaNeuralNetwork:
    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 = MakovskaNeuralNetwork()
    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
```

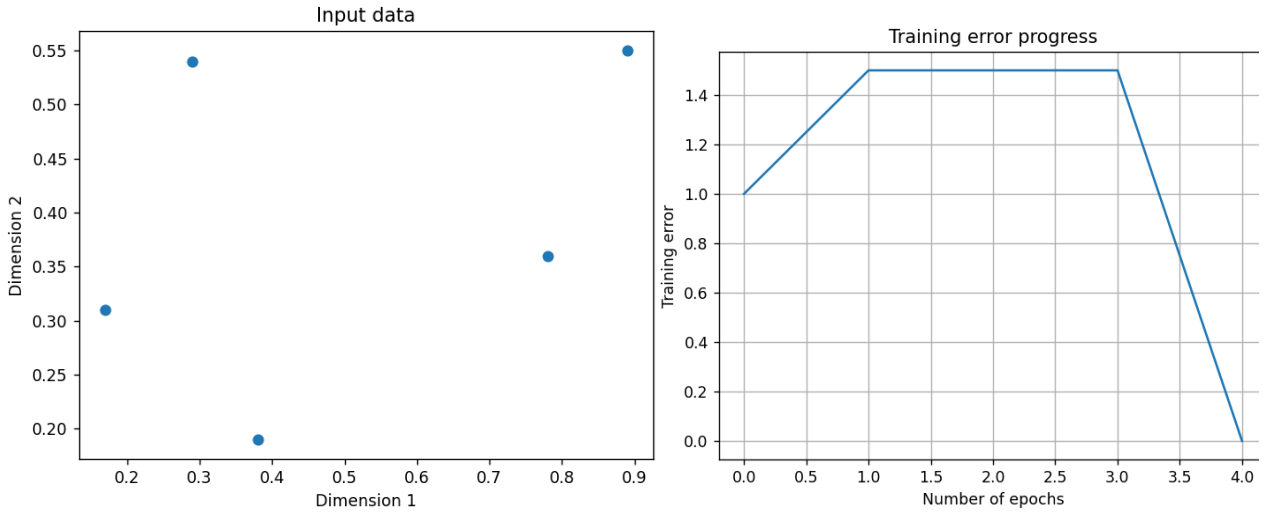
```
C:\Python311\python.exe C:/Users/Ola/Do
Epoch 0 loss: 0.316
Epoch 10 loss: 0.195
Epoch 20 loss: 0.123
Epoch 30 loss: 0.085
Epoch 40 loss: 0.062
Epoch 50 loss: 0.048
Epoch 60 loss: 0.039
Epoch 70 loss: 0.033
Epoch 80 loss: 0.028
Epoch 90 loss: 0.024
Epoch 940 loss: 0.002
Epoch 950 loss: 0.002
Epoch 960 loss: 0.002
Epoch 970 loss: 0.002
Epoch 980 loss: 0.002
Epoch 990 loss: 0.002
Emily: 0.964
Frank: 0.039
```

Рис.5.2. task-2.py

Завдання 5.3: Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab. Розробіть класифікатор на основі перцептрону з використанням бібліотеки NeuroLab для файлу даних.

Лістинг файлу task-3.py

```
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()
```



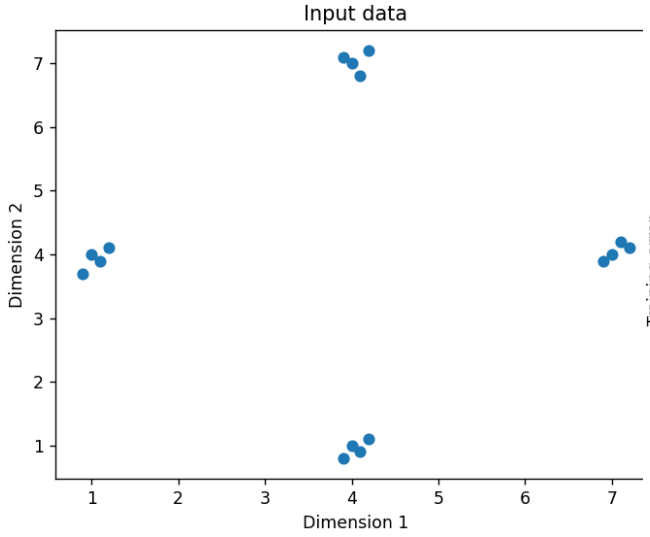

```
C:\Python311\python.exe C:/Users/Ola/Doc
The goal of learning is reached
```

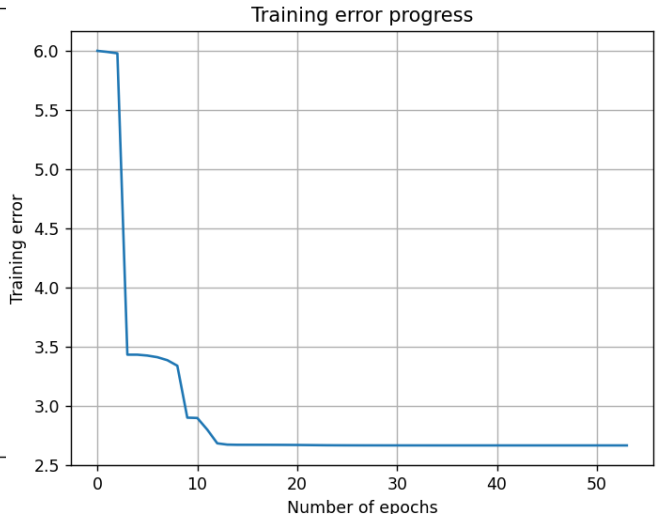
Рис.5.3. task-3.py

Завдання 5.4: Побудова одношарової нейронної мережі. Створіть одношарову нейронну мережу, що складається з незалежних нейронів, для вхідного файлу.

Лістинг 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])
```





```
C:\Python311\python.exe C:/Users/Ola/Documents/stu
Test results:
[0.4, 4.3] --> [0.33333335 0.66666673]
[4.4, 0.6] --> [ 9.99464944e-01 -5.75087279e-07]
[4.7, 8.1] --> [0.33333335 0.66666673]
```

Рис.5.4. task-4.py

Завдання 5.5: Побудова багатошарової нейронної мережі. Побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних.

Для отримання більш високої точності ми повинні надати більшу свободу нейронній мережі. Це означає, що нейронна мережа повинна мати більше одного шару для отримання базових закономірностей, що існують серед тестових даних.

Лістинг файлу task-5.py

```
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)

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('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()
```

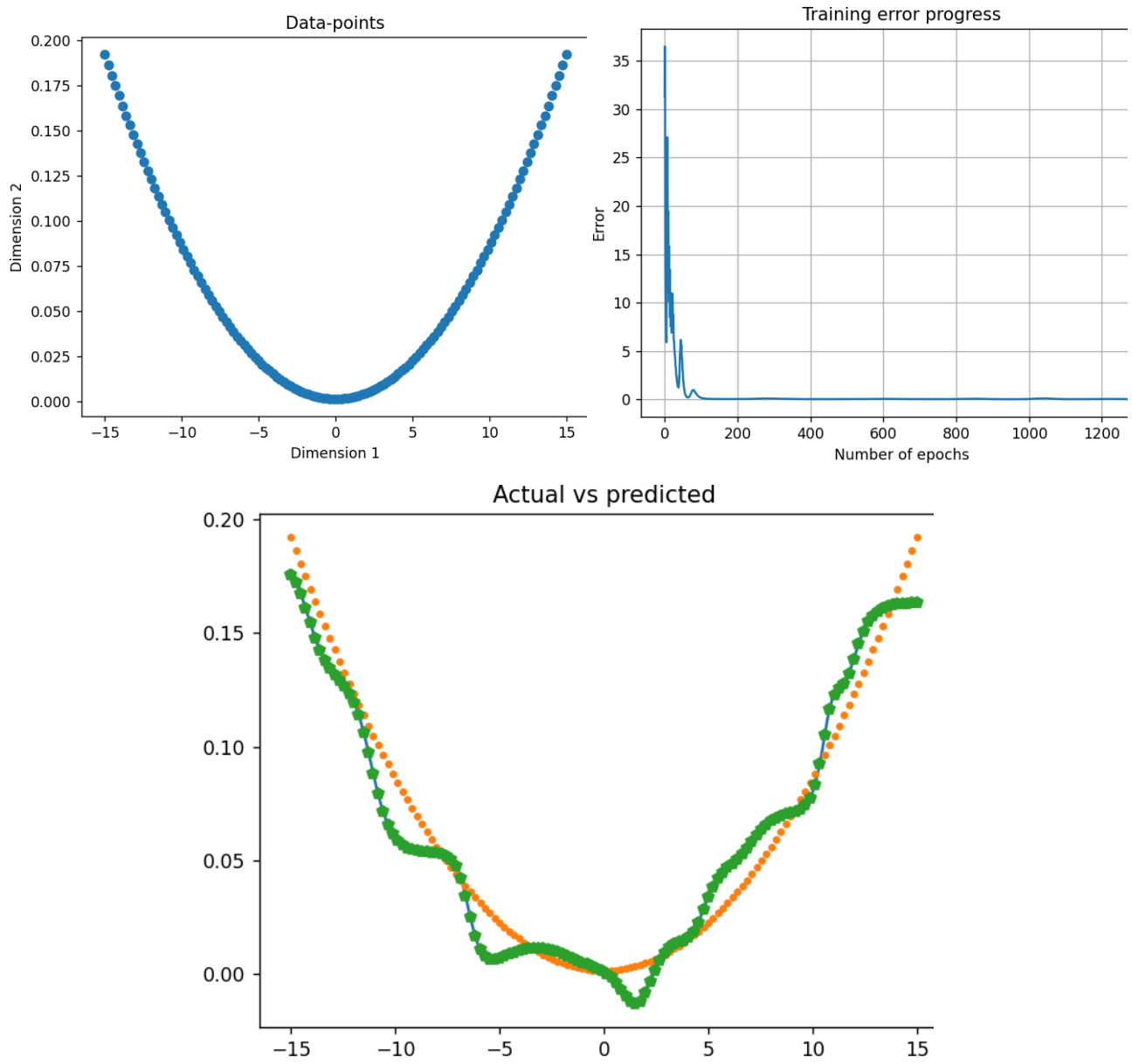


Рис.5.5. task-5.py

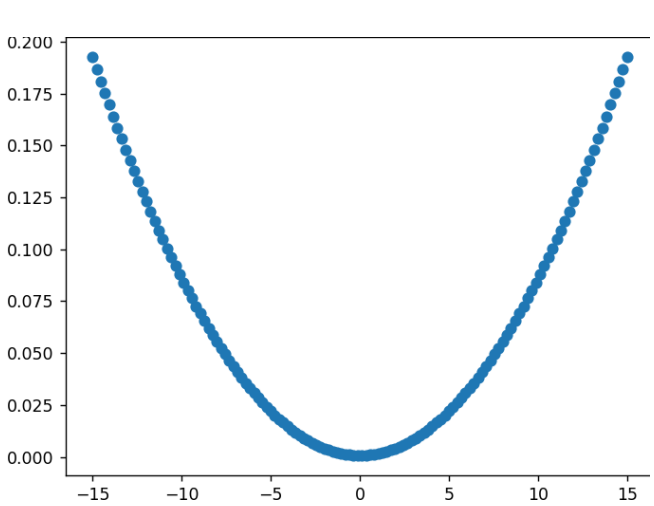
Завдання 5.6: Побудова багатошарової нейронної мережі для свого варіанту. По аналогії з попереднім завданням, побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних вашого варіанту. Варіант обирається згідно номеру за списком групи. Варіанти тестових даних указані в таблиці 1. Параметри багатошарової мережі указані в таблиці 2.

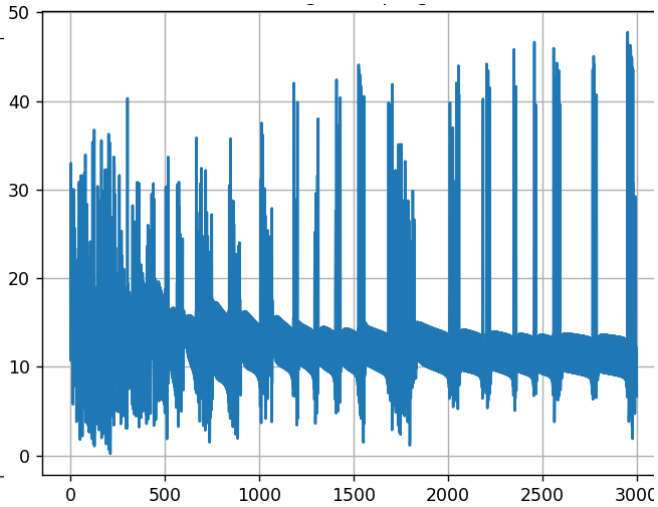
№ варіанта	Тестові дані
Варіант 13	$y = 5x^2+4$

Номер варіанта	Багатошаровий персептрон	
	Кількість шарів	Кількості нейронів у шарах
13	2	6-1

Лістинг файлу task-6.py

```
min_val = -15
max_val = 15
num_points = 130
x = np.linspace(min_val, max_val, num_points)
y = 5 * np.square(x) + 4
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]], [6, 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()
```





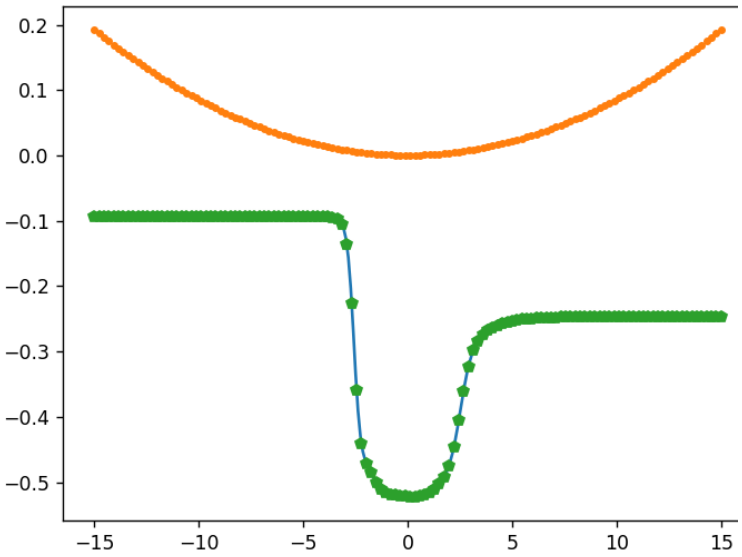


Рис.5.6. task-6.py

Завдання 5.7: Завдання 2.7. Побудова нейронної мережі на основі карти Кохонена, що само організується. По аналогії з попереднім завданням.

Лістинг 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()
```

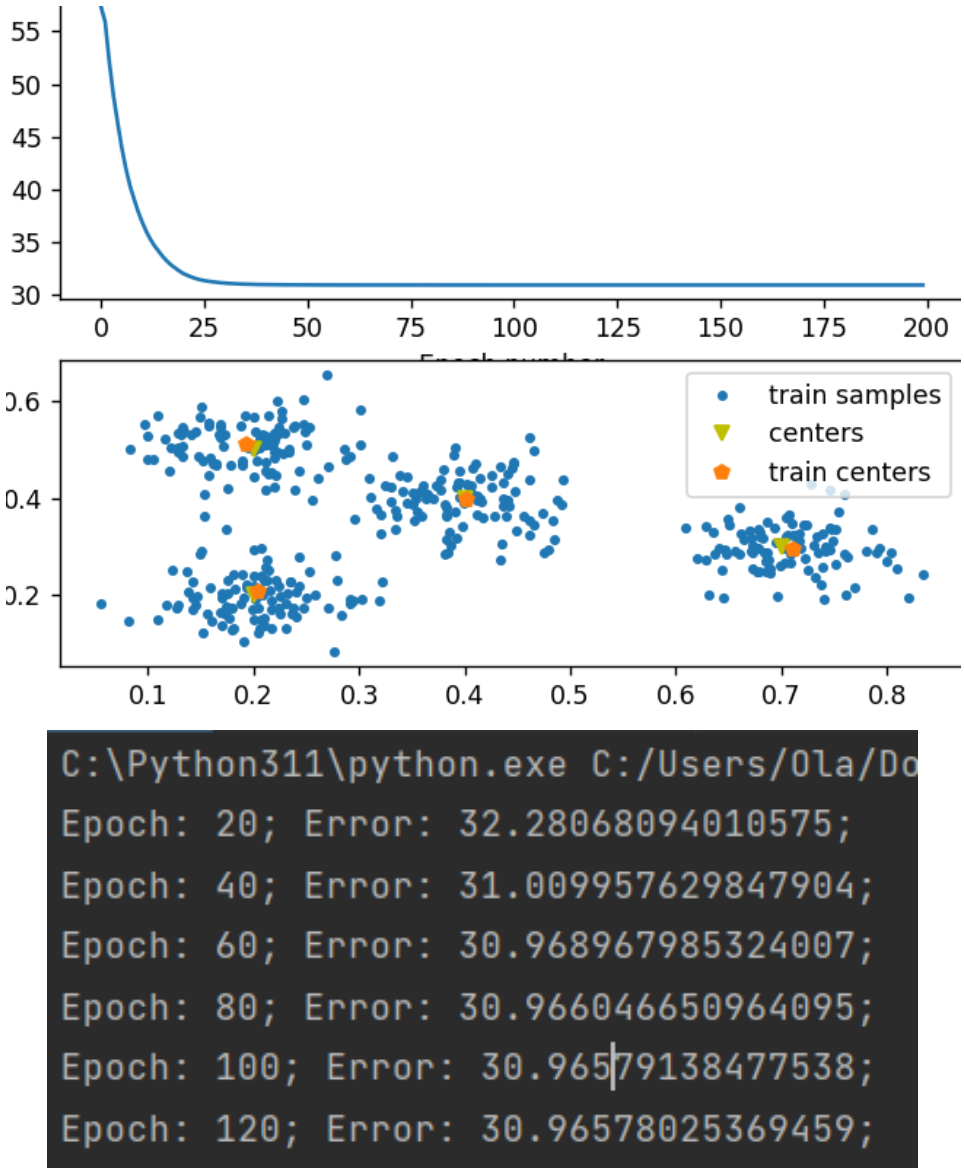


Рис.5.7. task-7.py

Завдання 5.8: Дослідження нейронної мережі на основі карти Кохонена, що самоорганізується.

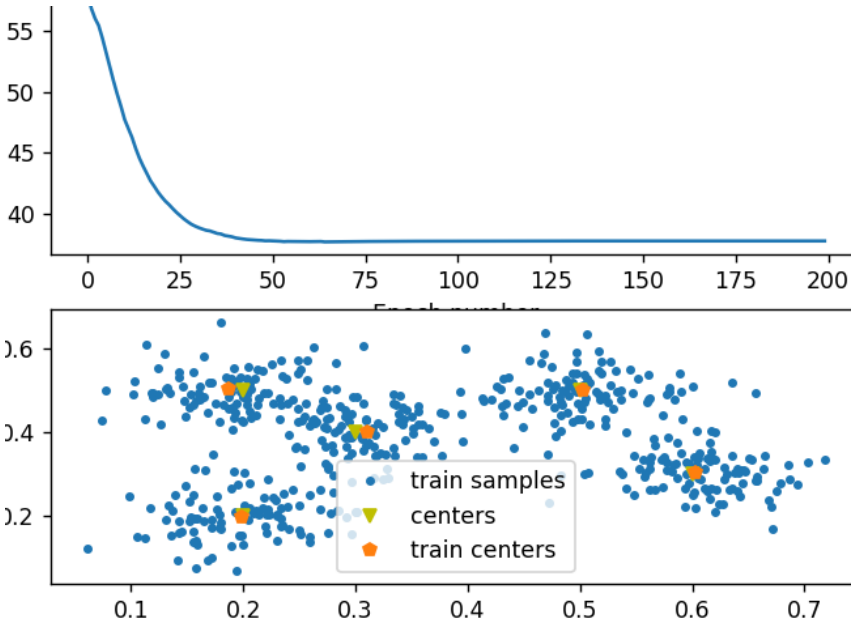
Проведіть дослідження по аналогії з попереднім завданням. Використовуючи готовий код внесіть зміни у вхідні данні згідно вашого варіанту у таблиці 3

Таблиця 3

№ варіанту	Центри кластера	skv
Варіант 13	[0.2, 0.2], [0.3, 0.4], [0.6, 0.3], [0.2, 0.5], [0.5, 0.5]	0,05

Лістинг файлу task-8.py

```
skv = .05
center = np.array([[0.2, 0.2], [0.3, 0.4], [0.6, 0.3], [0.2, 0.5], [0.5, 0.5]])
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]], 5)
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()
```




```
C:\Python311\python.exe C:/Users/Ola/Documen
Epoch: 20; Error: 41.75166376160144;
Epoch: 40; Error: 38.06902441802473;
Epoch: 60; Error: 37.66863640738843;
Epoch: 80; Error: 37.693553728676186;
Epoch: 100; Error: 37.716648463108456;
Epoch: 120; Error: 37.72416244491108;
```

Рис.5.8. task-8.py

https://github.com/avrorilka/AI_Python

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