**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



Рис.5.1. task-1.py

**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

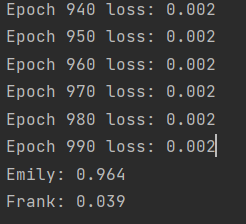
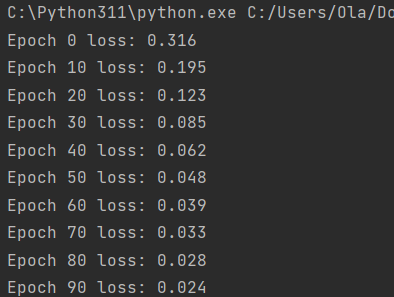


Рис.5.2. task-2.py

**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()

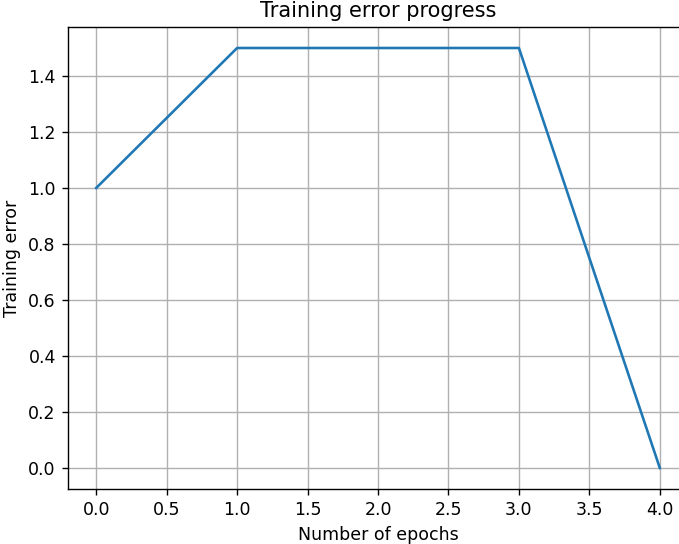
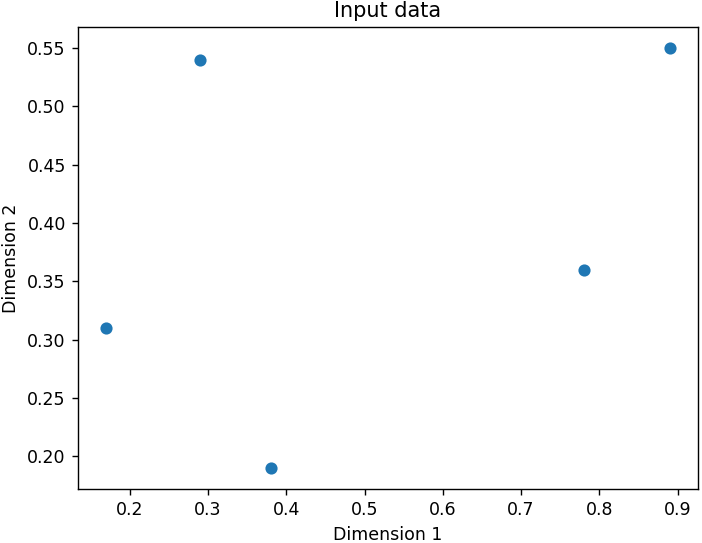


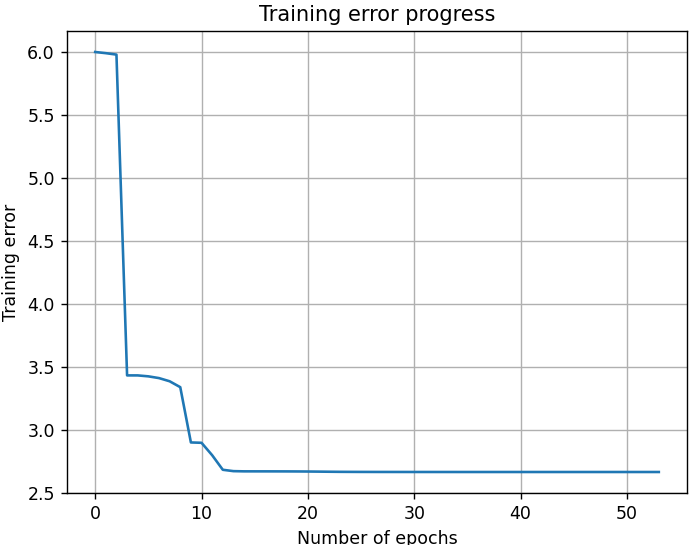
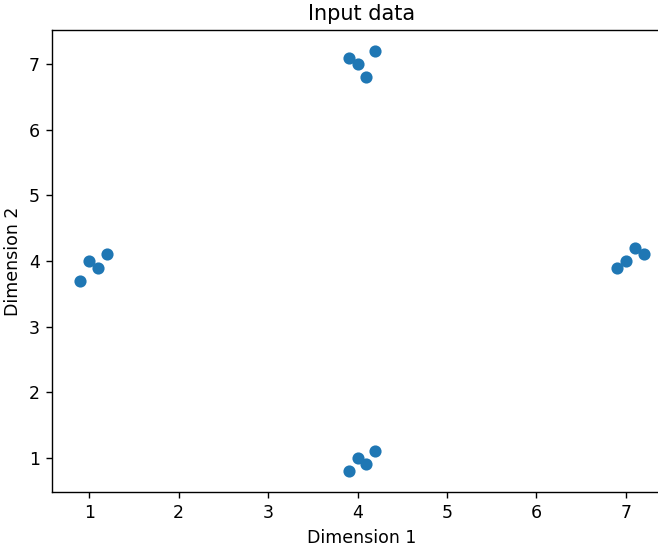


Рис.5.3. task-3.py

**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])



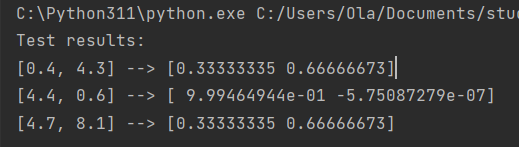


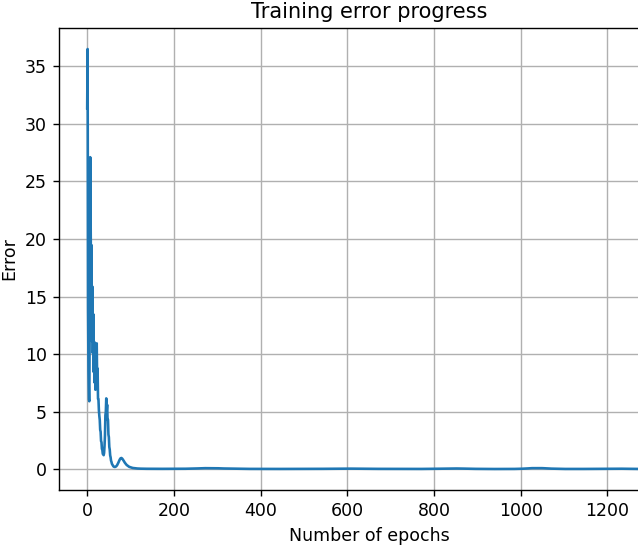
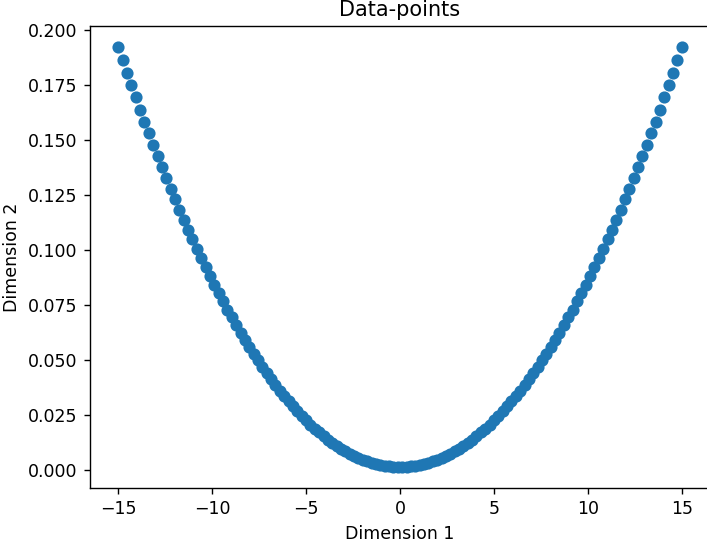
Рис.5.4. task-4.py

**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()



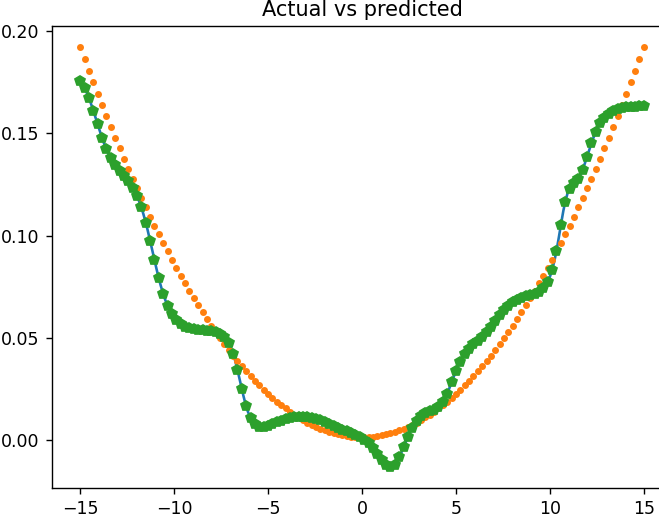


Рис.5.5. task-5.py

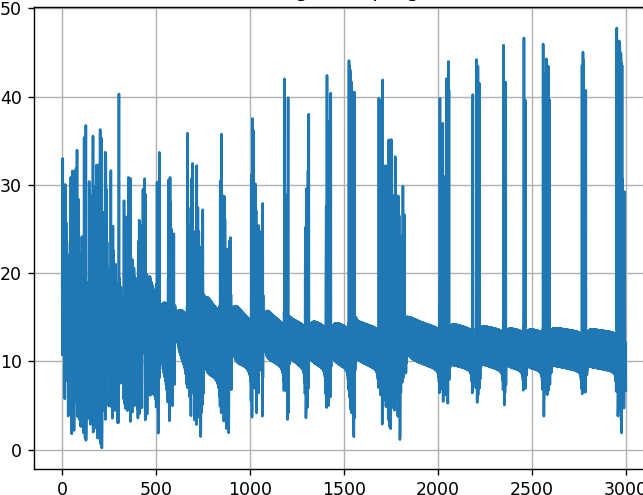
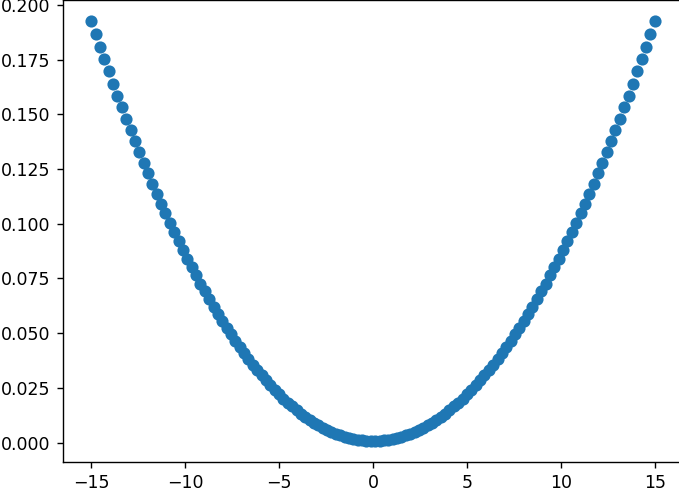
**Завдання 5.6:** Побудова багатошарової нейронної мережі для свого варіанту. По аналогії з попереднім завданням, побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних вашого варіанту. Варіант обирається згідно номеру за списком групи. Варіанти тестових даних указані в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

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

|  |  |
| --- | --- |
| № варіанта | Тестові дані |
| Варіант 13 | y = 5x2+4 |

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()



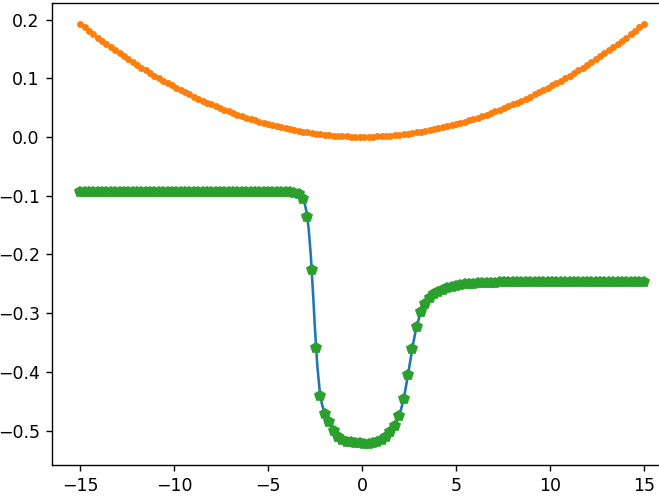
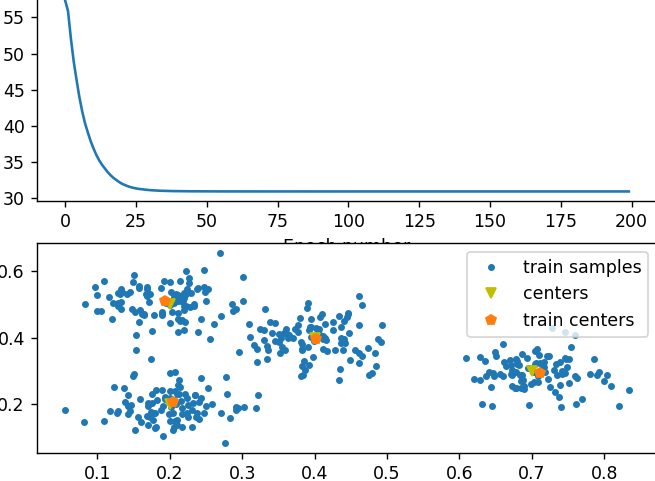


Рис.5.6. task-6.py

**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()



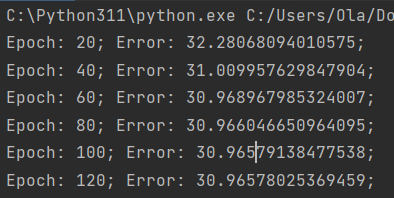


Рис.5.7. task-7.py

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

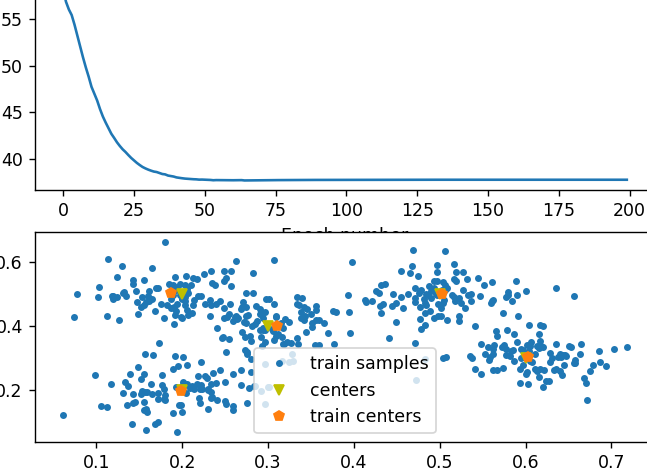
Проведіть дослідження по аналогії з попереднім завданням. Використовуючи готовий код внесіть зміни у вхідні данні згідно вашого варіанту у таблиці 3

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()



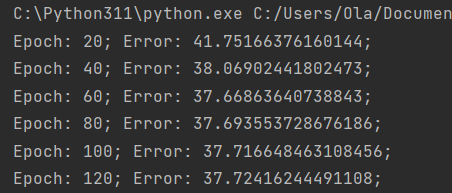


Рис.5.8. task-8.py

[**https://github.com/avrorilka/AI\_Python**](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