**ЛАБОРАТОРНА РОБОТА № 5**

**РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ**

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

**Хід роботи**

**Посилання на GitHub:** <https://github.com/Max2002/AI_IPZ-19-3_LMV>

**Завдання 1.** Створити простий нейрон

Лістинг коду файлу LR\_5\_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])  
 bias = 4  
 n = Neuron(weights, bias)  
  
 x = np.array([2, 3])  
 print(n.feedforward(x))

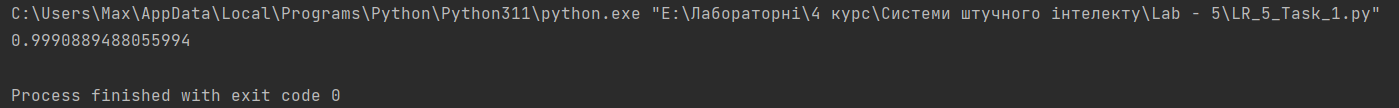


Рис.1. Результат роботи нейрона

**Завдання 2.** Створити просту нейронну мережу для передбачення статі людини

Лістинг коду файлу LR\_5\_Task\_2.py:

import numpy as np  
from LR\_5\_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

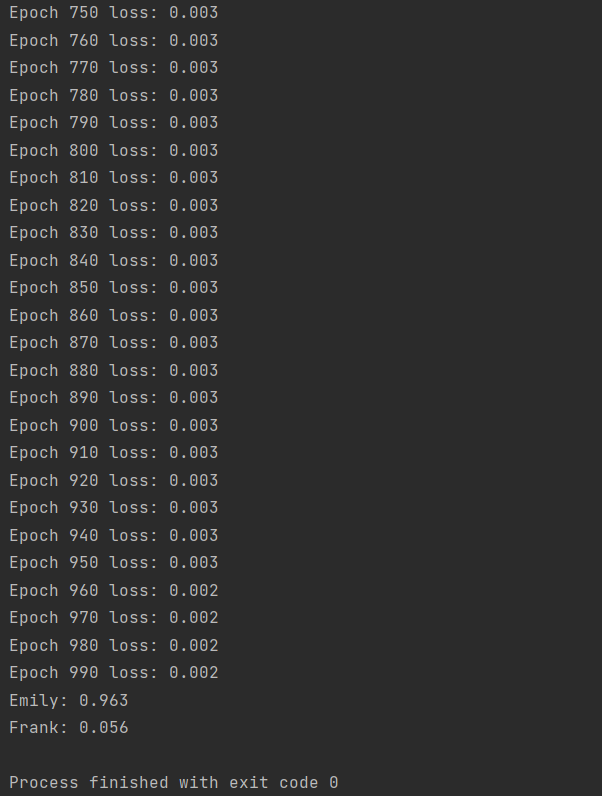


Рис.2. Результат навчання нейронної мережі

Функція активації використовується 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 LR\_5\_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()

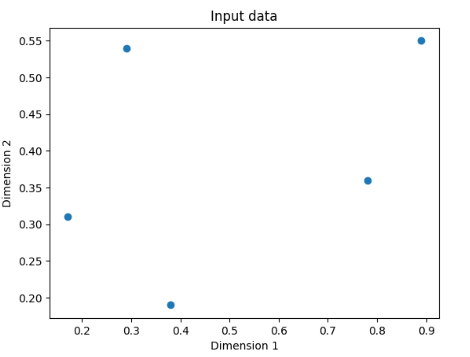


Рис.3. Вхідні дані до перцептрону



Рис.4. Навчання перцептрону

**Завдання 4.** Побудова одношарової нейронної мережі

Лістинг коду файлу LR\_5\_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])

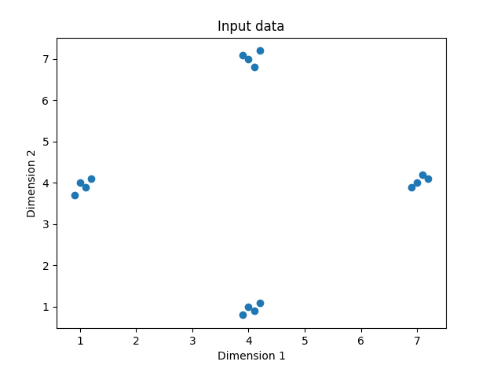


Рис.5. Вхідні дані до нейронної мережі

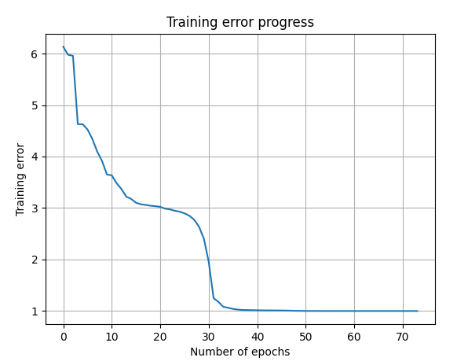


Рис.6. Навчання мережі

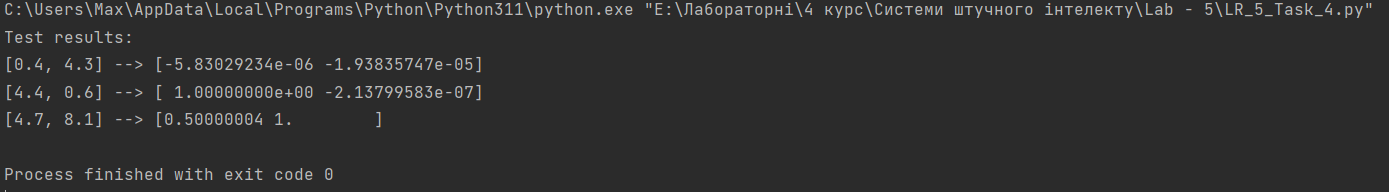


Рис.7.Тестові результати

**Завдання 5.** Побудова багатошарової нейронної мережі

Лістинг коду файлу LR\_5\_Task\_5.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 = 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('Dimension 1')  
plt.ylabel('Dimension 2')  
plt.title('Data-points')  
plt.show()  
  
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()

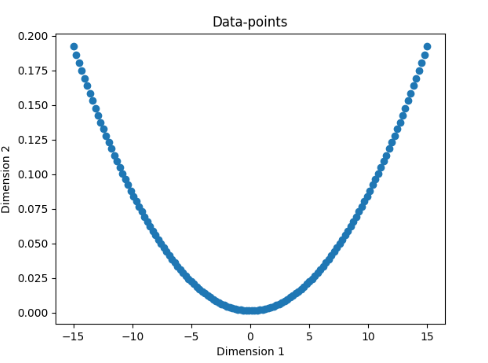


Рис.8. Дані рівняння 3x2+5

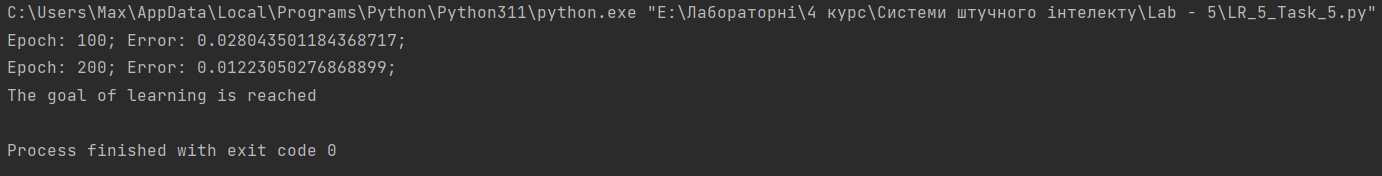


Рис.9. Звітність про навчання по епохам

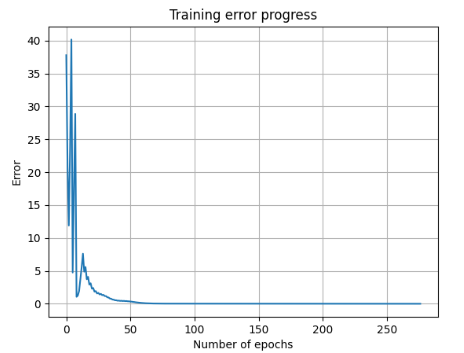


Рис.10. Графік навчання мережі

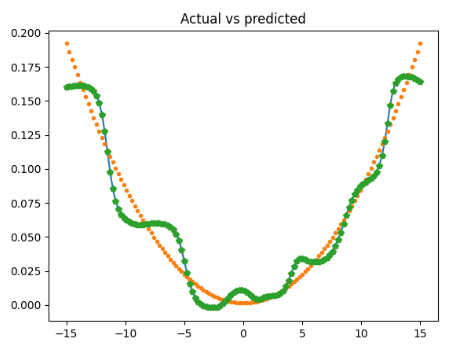


Рис.11. Графік-порівняння істинних та отриманих даних

**Завдання 6.** Побудова багатошарової нейронної мережі для свого варіанту

Варіант 9, дані: y = 3x2+9, кількість шарів: 3, кількість нейронів: 3-5-1

Лістинг коду файлу LR\_5\_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 = 3 \* np.square(x) + 9  
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]], [3, 5, 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 \* 3)  
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()

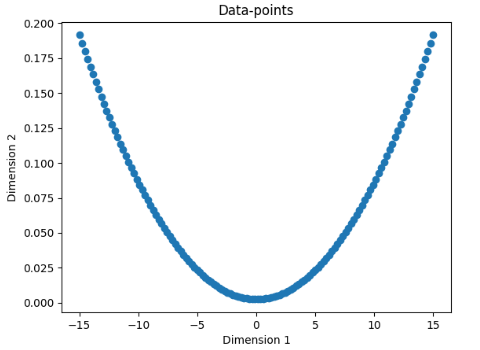


Рис.12. Графік вхідних даних по варіанту

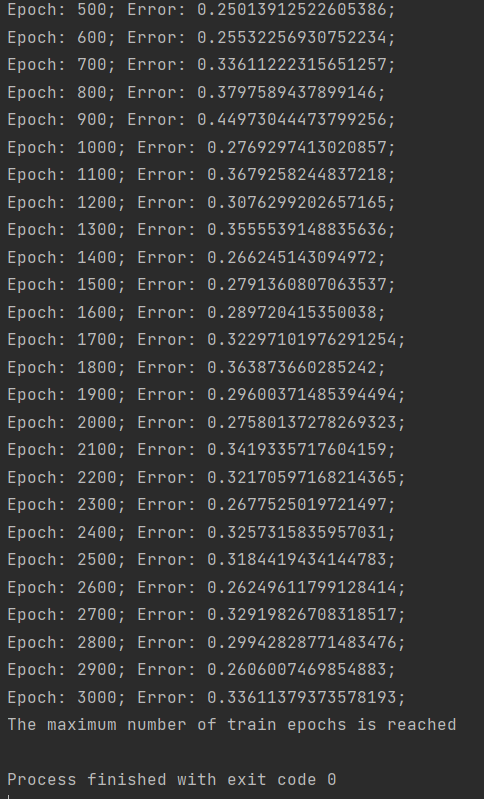


Рис.13. Звітність навчання по епохам (вивід 100х епох)



Рис.14. Прогрес помилковості при навчанні

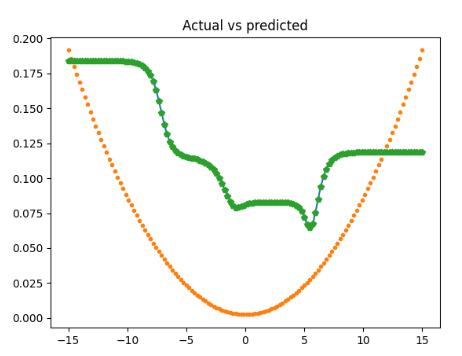


Рис.15. Графік-порівняння дійсних та передбачених даних

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 LR\_5\_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()

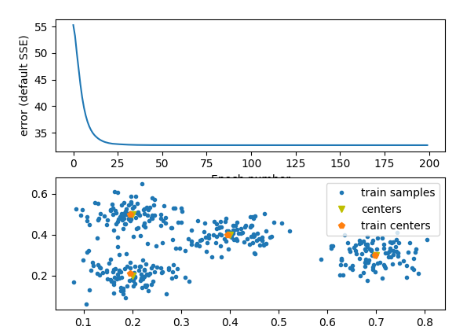


Рис.18. Графік помилковості по епохам та класифікація центрів

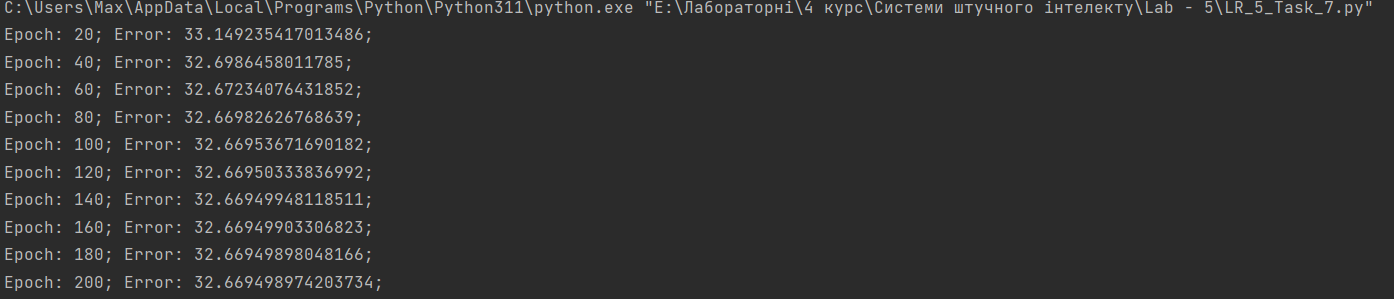


Рис.17. Звітність навчання

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

Варіант 9, центри: [0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.1, 0.5], [0.4, 0.5], skv = 0.04.

Лістинг коду файлу LR\_5\_Task\_8.py:

import numpy as np  
import numpy.random as rand  
import neurolab as nl  
import pylab as pl  
  
skv = .04  
center = np.array([[0.2, 0.3], [0.4, 0.4], [0.7, 0.3], [0.1, 0.5], [0.4, 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()

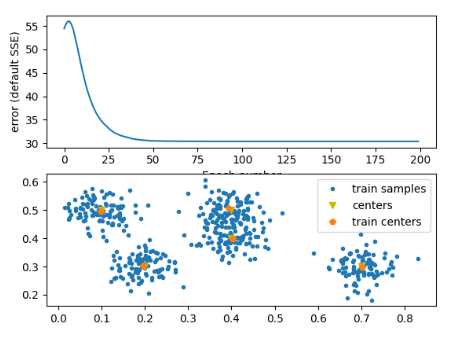


Рис.18. Графік навчання та класифікації за 4 нейрона

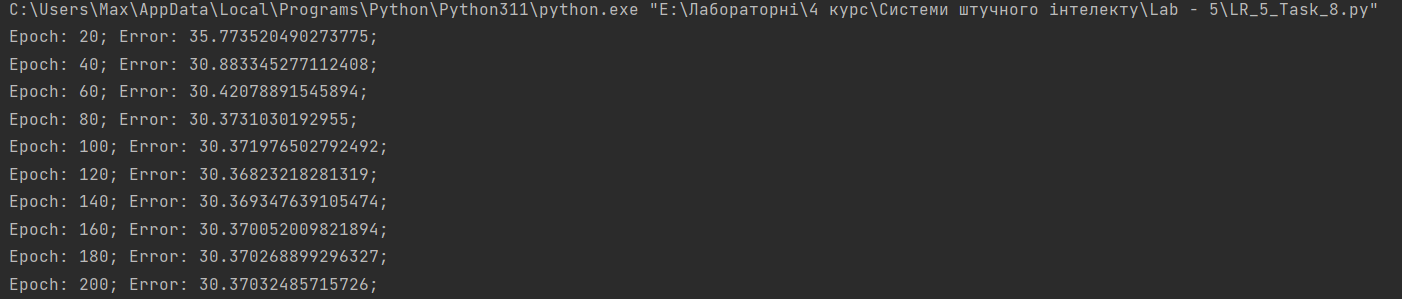


Рис.19. Звітність за 4х нейронів

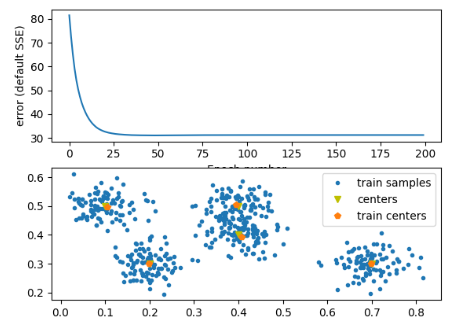


Рис.22. Графік навчання та класифікація за 5ти нейронів

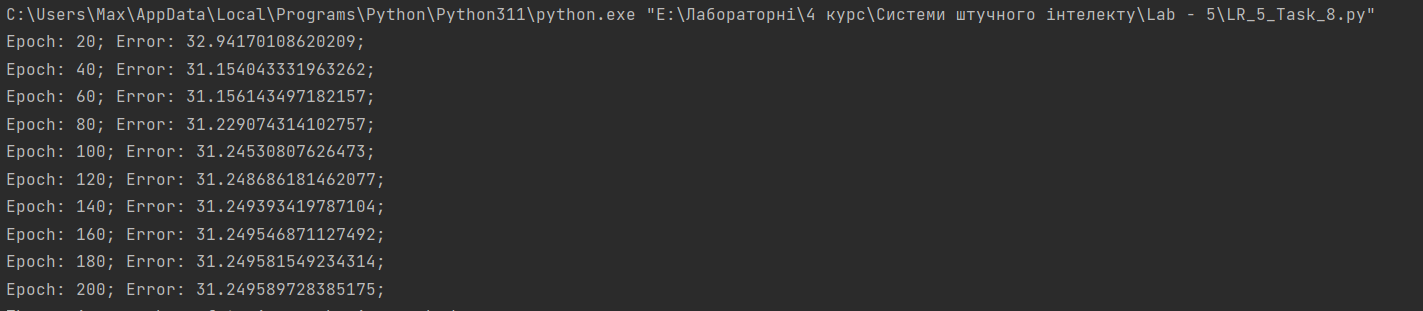


Рис.21. Звітність за 5ти нейронів

***Висновок:*** було отримано навички зі створення та застосовування простих нейронних мереж використовуючи спеціалізовані бібліотеки та мову програмування Python.