Міністерство освіти і науки України

Державний університет ,,Житомирська політехніка”

Кафедра: Фікт

Група: ІПЗ-20-1

# Лабораторна робота №5

«РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ»

Виконав: Денисюк Н. Г.

Прийняв: Голенко М. Ю.

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

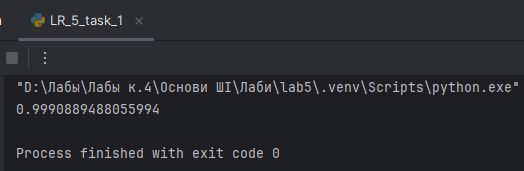
**Виконання роботи:**

**Завдання №1:** Створити простий нейрон

Код:

import numpy as np  
  
  
def sigmoid(x):  
 # Наша функція активації: f(x) = 1 / (1 + e^(-x))  
 return 1 / (1 + np.exp(-x))  
  
  
class Neuron:  
 def \_\_init\_\_(self, weights, bias):  
 self.weights = weights  
 self.bias = bias  
  
 def feedforward(self, inputs):  
 # Вхідні дані про вагу, додавання зміщення  
 # і подальше використання функції активації  
  
 total = np.dot(self.weights, inputs) + self.bias  
 return sigmoid(total)  
  
  
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))

**Результат виконання:**

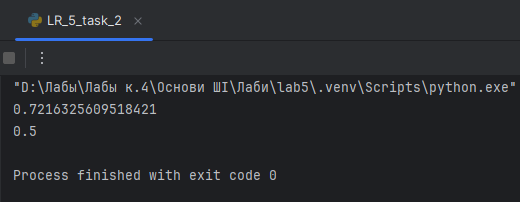


**Завдання №2:** Створити просту нейронну мережу для передбачення статі людини

Код:

import numpy as np  
  
  
def sigmoid(x):  
 # Наша функція активації: f(x) = 1 / (1 + e^(-x))  
 return 1 / (1 + np.exp(-x))  
  
  
class Neuron:  
 def \_\_init\_\_(self, weights, bias):  
 self.weights = weights  
 self.bias = bias  
  
 def feedforward(self, inputs):  
 # Вхідні дані про вагу, додавання зміщення  
 # і подальше використання функції активації  
  
 total = np.dot(self.weights, inputs) + self.bias  
 return sigmoid(total)  
  
  
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  
  
class DenysiukNeuralNetwork:  
 def \_\_init\_\_(self):  
 weights = np.array([0, 1])  
 bias = 0  
  
 # Класс Neuron із попереднього завдання  
 self.h1 = Neuron(weights, bias)  
 self.h2 = Neuron(weights, bias)  
 self.o1 = Neuron(weights, bias)  
  
 def feedforward(self, x):  
 out\_h1 = self.h1.feedforward(x)  
 out\_h2 = self.h2.feedforward(x)  
  
 # Входи для о1 є виходами h1 и h2  
 out\_o1 = self.o1.feedforward(np.array([out\_h1, out\_h2]))  
  
 return out\_o1  
  
  
network = DenysiukNeuralNetwork()  
x = np.array([2, 3])  
print(network.feedforward(x)) # 0.7216325609518421  
  
  
def mse\_loss(y\_true, y\_pred):  
 # y\_true и y\_pred є масивами numpy з однаковою довжиною  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
  
y\_true = np.array([1, 0, 0, 1])  
y\_pred = np.array([0, 0, 0, 0])  
  
print(mse\_loss(y\_true, y\_pred)) # 0.5

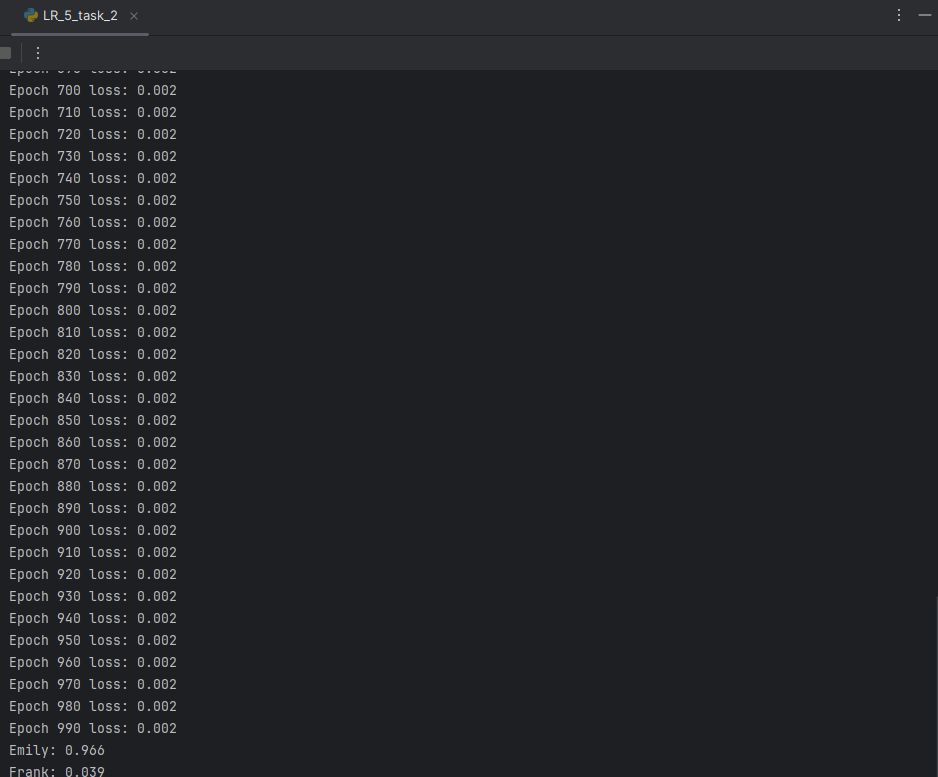
**Результат виконання:**



Код:

import numpy as np  
  
  
def sigmoid(x):  
 # Функція активації sigmoid:: f(x) = 1 / (1 + e^(-x))  
 return 1 / (1 + np.exp(-x))  
  
  
def deriv\_sigmoid(x):  
 # Похідна від sigmoid: f'(x) = f(x) \* (1 - f(x))  
 fx = sigmoid(x)  
 return fx \* (1 - fx)  
  
  
def mse\_loss(y\_true, y\_pred):  
 # y\_true и y\_pred є масивами numpy з одинаковою довжиною  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
  
class DenysiukNeuralNetwork:  
  
  
 def \_\_init\_\_(self):  
 # Вага  
 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()  
  
 # Зміщення  
 self.b1 = np.random.normal()  
 self.b2 = np.random.normal()  
 self.b3 = np.random.normal()  
  
 def feedforward(self, x):  
 # x є масивом numpy з двома елементами  
 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):  
 *"""  
 - data is a (n x 2) numpy array, n = # of samples in the dataset.  
 - all\_y\_trues is a numpy array with n elements.  
 Elements in all\_y\_trues correspond to those in data.  
 """* learn\_rate = 0.1  
 epochs = 1000 # кількість циклів у всьому наборі даних  
  
 for epoch in range(epochs):  
 for x, y\_true in zip(data, all\_y\_trues):  
 # --- Виконуємо зворотній зв'язок (ці значання нам потрібні в подальшому )  
 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  
 # --- Найменування: d\_L\_d\_w1 означає "частково L / частково w1"  
 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  
  
 # --- Підраховуємо загальні втрати в кінці кожної фази  
 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))  
  
  
# Задання набору даних  
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  
])  
  
# Тренуємо вашу нейронну мережу!  
network = DenysiukNeuralNetwork()  
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 дюймов  
print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F  
print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M

**Результат виконання:**



- Функція активації в нейронних мережах, така як sigmoid, виконує дуже важливу роль. Вона додає нелінійність до моделі, що дозволяє нейронній мережі вчитися з більш складних даних і вирішувати не просто лінійні задачі. Функція sigmoid обмежує вихідне значення в діапазоні від 0 до 1, що робить її корисною для задач, де потрібно передбачити ймовірність або коли вихід має бути у вигляді ймовірності.

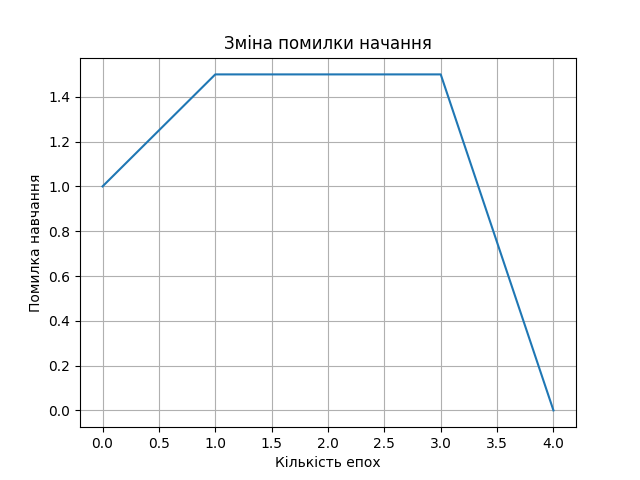
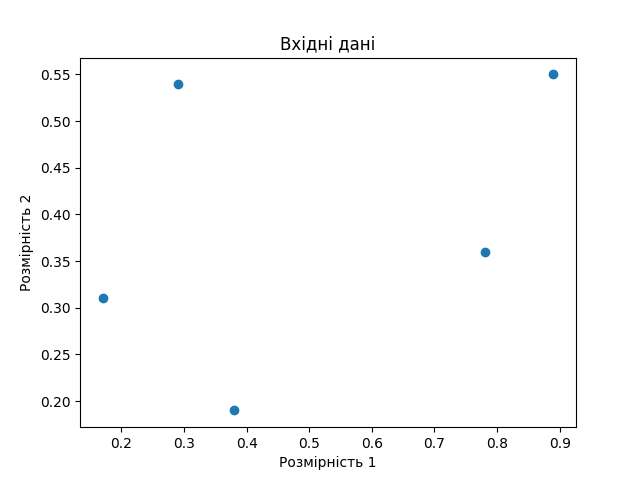
- Нейронні мережі прямого поширення, такі як та, що представлена у вашому коді, мають здатність апроксимувати будь-яку функцію, що робить їх універсальними апроксиматорами. Це означає, що вони можуть моделювати складні відносини між вхідними і вихідними даними. Завдяки процесу навчання, який включає коригування ваг і зміщень на основі помилок передбачення (як показано у функції train), нейронні мережі можуть покращувати свої передбачення і досягати високої точності.

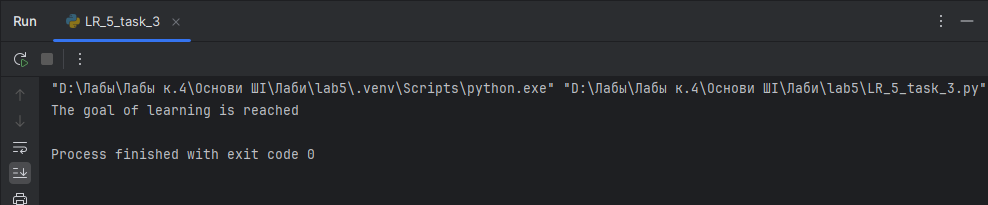
**Завдання №3:** Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

Код:

import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
# Завантаження вхідних даних  
text = np.loadtxt ('data\_perceptron.txt')  
  
# Поділ точок даних та міток  
data = text[:, :2]  
  
labels = text[:, 2].reshape((text.shape[0], 1))  
  
# Побудова графіка вхідних даних  
plt.figure()  
plt.scatter(data[:, 0], data[:, 1])  
plt.xlabel('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
  
# Визначення максимального та мінімального значень для кожного виміру  
dim1\_min, dim1\_max, dim2\_min, dim2\_max = 0, 1, 0, 1  
  
# Кількість нейронів у вихідному шарі  
num\_output = labels.shape[1]  
  
# Визначення перцептрону з двома вхідними нейронами (оскільки  
# Вхідні дані - двовимірні)  
dim1 = [dim1\_min, dim1\_max]  
dim2 = [dim2\_min, dim2\_max]  
perceptron = nl.net.newp([dim1, dim2], num\_output)  
  
# Тренування перцептрону з використанням наших даних  
error\_progress = perceptron.train(data, labels, epochs=100,  
 show=20, lr=0.03)  
  
# Побудова графіка процесу навчання  
plt.figure()  
plt.plot(error\_progress)  
plt.xlabel('Кількість епох')  
plt.ylabel('Помилка навчання')  
plt.title('Зміна помилки начання')  
plt.grid()  
plt.show()

**Результат виконання:**



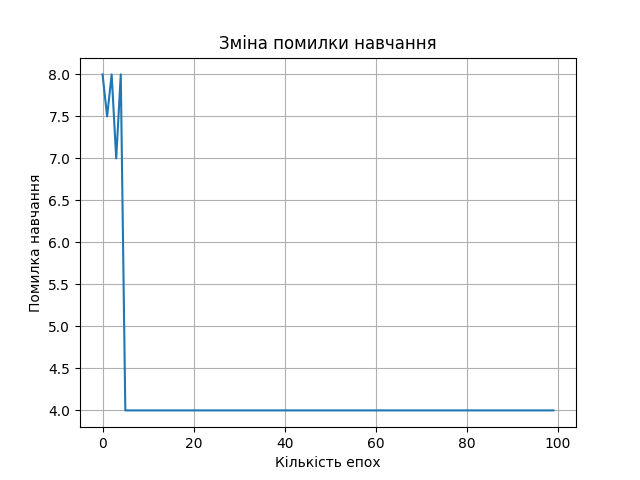
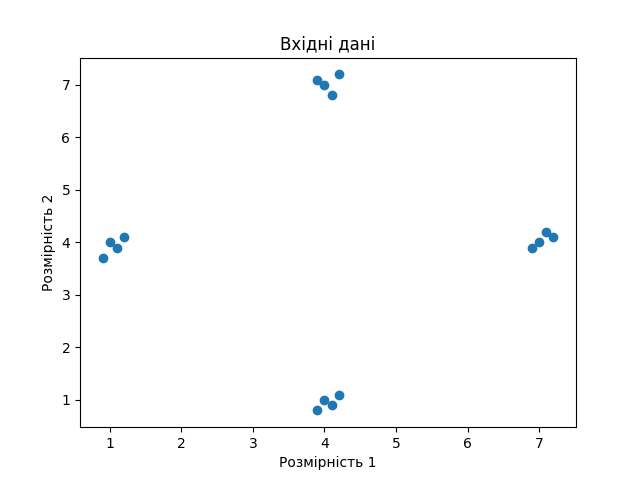
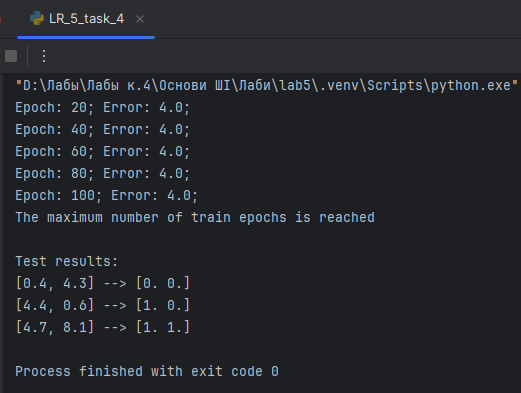


**Завдання №4:** Побудова одношарової нейронної мережі

Код:

import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
# Завантаження вхідних даних  
text = np.loadtxt('data\_simple\_nn.txt')  
  
# Поділ даних на точки даних та мітки  
data = text[:, 0:2]  
labels = text[:, 2:]  
  
# Побудова графіка вхідних даних  
plt.figure()  
plt.scatter(data[:,0], data[:,1])  
plt.xlabel('Розмірність 1')  
plt.ylabel ('Розмірність 2')  
plt.title('Вхідні дані')  
  
# 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()  
  
# Визначення кількості нейронів у вихідному шарі  
num\_output = labels.shape[1]  
  
# Визначення одношарової нейронної мережі  
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('Зміна помилки навчання')  
plt.grid()  
plt.show()  
  
# Виконання класифікатора на тестових точках даних  
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:**

**** ****

У вікні терміналу виведено результати навчання одношарової нейронної мережі. Значення помилки, яке залишається стабільним і дорівнює 4.0 протягом усіх епох навчання, свідчить про те, що мережа не покращує свої передбачення з кожною епохою. Це може бути ознакою того, що мережі не вистачає складності для вирішення задачі, або що параметри навчання (наприклад, швидкість навчання) не оптимальні.

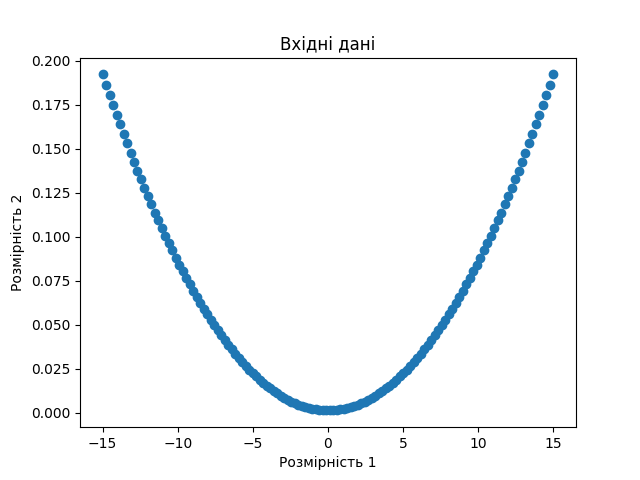
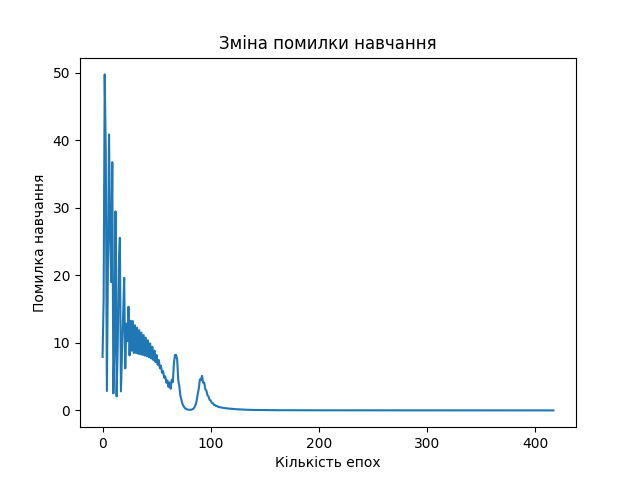
На другому графіку показано зміну помилки навчання від кількості епох

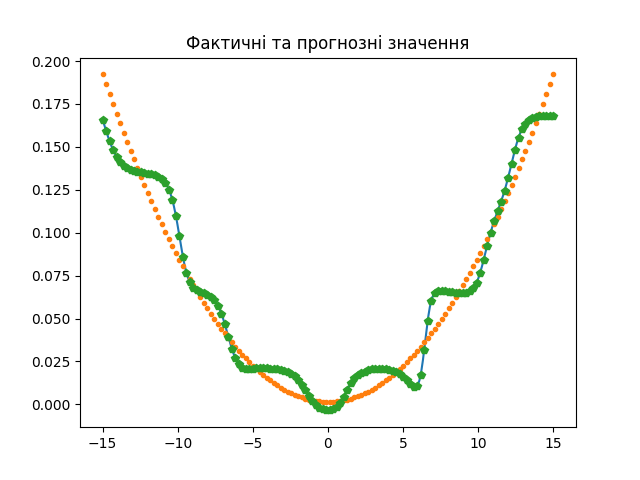
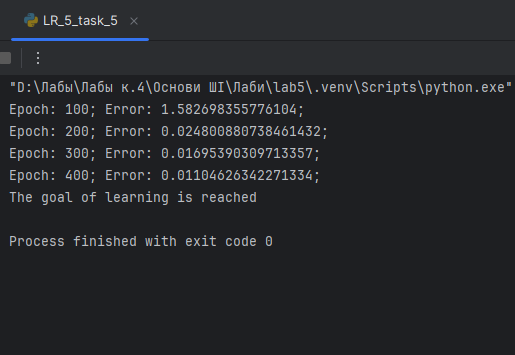
**Завдання №5:** Побудова багатошарової нейронної мережі

Код:

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)  
  
# Створення даних та міток  
data = x.reshape(num\_points, 1)  
labels = y.reshape(num\_points, 1)  
  
# Побудова графіка вхідних даних  
plt.figure()  
plt.scatter(data, labels)  
plt.xlabel('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
  
# Визначення багатошарової нейронної мережі з двома прихованими  
# шарами. Перший прихований шар складається із десяти нейронів.  
# Другий прихований шар складається з шести нейронів.  
# Вихідний шар складається з одного нейрона.  
nn = nl.net.newff([[min\_val, max\_val]], [10, 6, 1])  
  
# Завдання градієнтного спуску як навчального алгоритму  
nn.trainf = nl.train.train\_gd  
  
# Тренування нейронної мережі  
error\_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)  
  
# Виконання нейронної мережі на тренувальних даних  
output = nn.sim(data)  
y\_pred = output.reshape(num\_points)  
  
# Побудова графіка помилки навчання  
plt.figure()  
plt.plot(error\_progress)  
plt.xlabel('Кількість епох')  
plt.ylabel('Помилка навчання')  
plt.title('Зміна помилки навчання')  
  
# 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)).reshape(x\_dense.size)  
  
plt.figure()  
plt.plot(x\_dense, y\_dense\_pred, '-', x, y, '.', x, y\_pred, 'p')  
plt.title('Фактичні та прогнозні значення')  
plt.show()

**Результат виконання:**

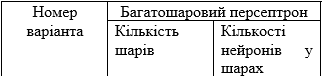
 

**Завдання №6:** Побудова багатошарової нейронної мережі для свого варіанту





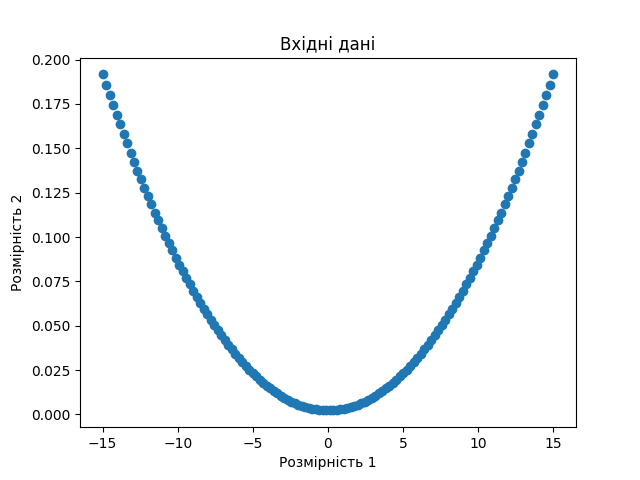
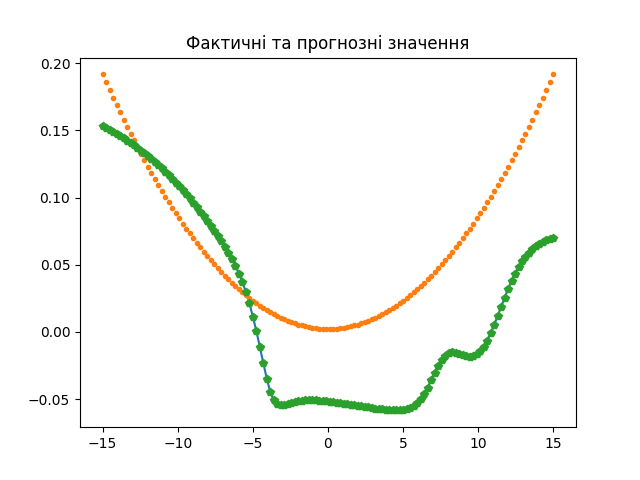
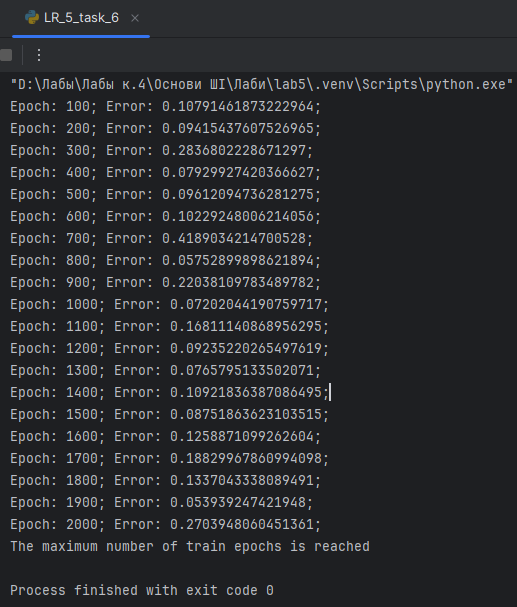




Код:

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) + 8  
y /= np.linalg.norm(y)  
  
# Створення даних та міток  
data = x.reshape(num\_points, 1)  
labels = y.reshape(num\_points, 1)  
  
# Побудова графіка вхідних даних  
plt.figure()  
plt.scatter(data, labels)  
plt.xlabel('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
  
# Визначення багатошарової нейронної мережі з двома прихованими шарами  
nn = nl.net.newff([[min\_val, max\_val]], [5, 5, 1])  
  
# Завдання градієнтного спуску як навчального алгоритму  
nn.trainf = nl.train.train\_gd  
  
# Тренування нейронної мережі  
error\_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)  
  
# Виконання нейронної мережі на тренувальних даних  
output = nn.sim(data)  
y\_pred = output.reshape(num\_points)  
  
# Побудова графіка помилки навчання  
plt.figure()  
plt.plot(error\_progress)  
plt.xlabel('Кількість епох')  
plt.ylabel('Помилка навчання')  
plt.title('Зміна помилки навчання')  
  
# 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)).reshape(x\_dense.size)  
  
plt.figure()  
plt.plot(x\_dense, y\_dense\_pred, '-', x, y, '.', x, y\_pred, 'p')  
plt.title('Фактичні та прогнозні значення')  
  
plt.show()

**Результат виконання:**

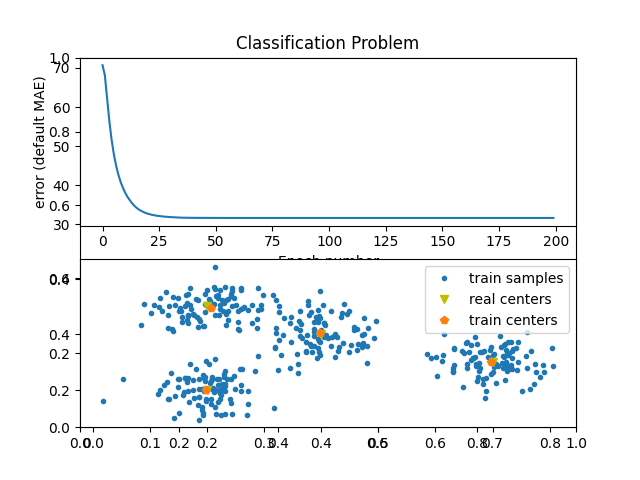
   

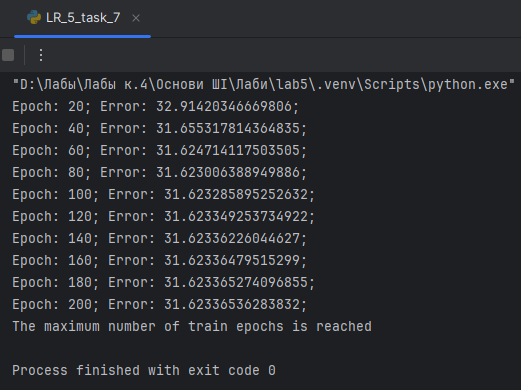
**Завдання №7:** Побудова нейронної мережі на основі карти Кохонена, що самоорганізується

Код:

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:**

****

****

**Завдання №8:** Дослідження нейронної мережі на основі карти Кохонена, що самоорганізується

Код:

2 входи 4 нейрони:

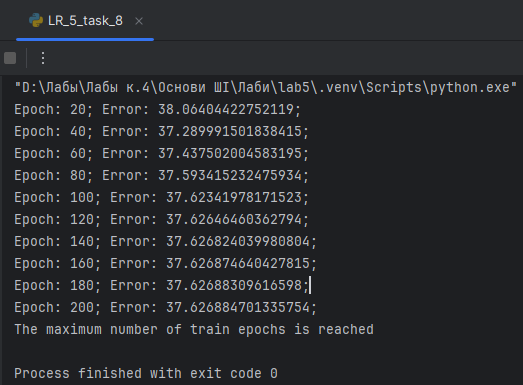
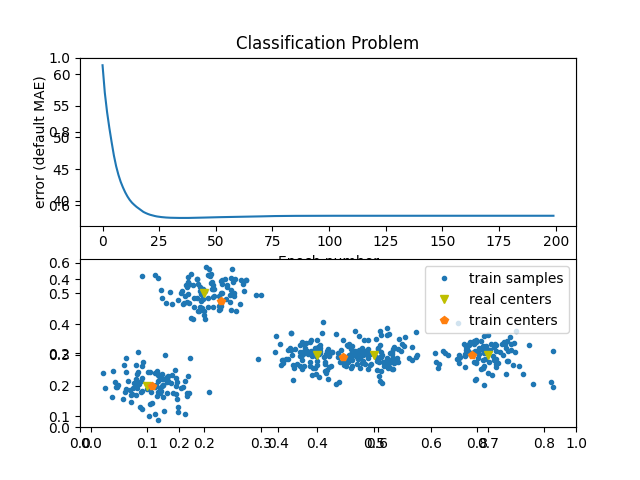
import numpy as np  
import neurolab as nl  
import numpy.random as rand  
  
skv = 0.04  
centr = np.array([[0.1, 0.2], [0.4, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.3]])  
rand\_norm = skv \* rand.randn(100, 5, 2)  
inp = np.array([centr + r for r in rand\_norm])  
inp.shape = (100 \* 5, 2)  
rand.shuffle(inp)  
  
# Create net with 2 inputs and 4 neurons  
net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 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()

2 входи 5 нейронів:

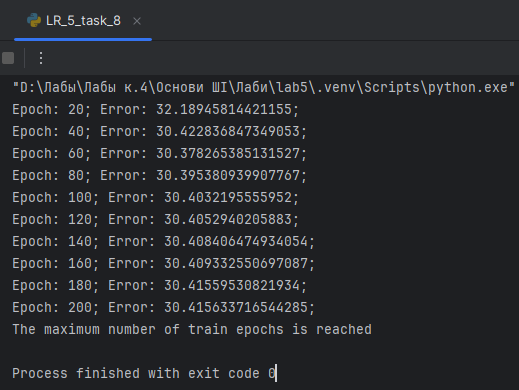
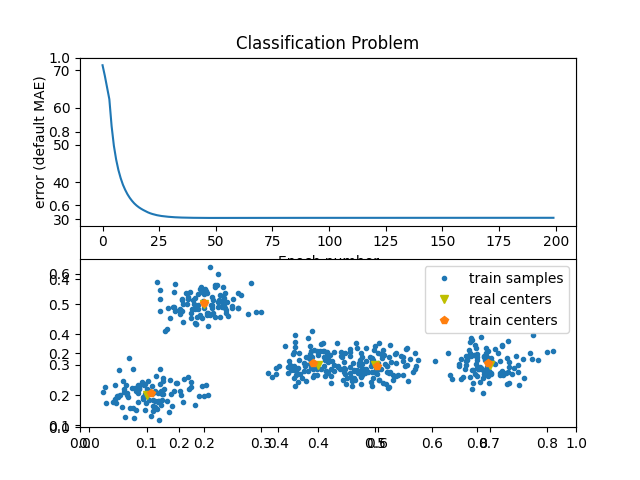
import numpy as np  
import neurolab as nl  
import numpy.random as rand  
  
skv = 0.04  
centr = np.array([[0.1, 0.2], [0.4, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.3]])  
rand\_norm = skv \* rand.randn(100, 5, 2)  
inp = np.array([centr + r for r in rand\_norm])  
inp.shape = (100 \* 5, 2)  
rand.shuffle(inp)  
  
# 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:**

2 входи 4 нейрони:



2 входи 5 нейронів:



Коли кількість нейронів у мережі менша за кількість кластерів (2 входи, 4 нейрони), помилка вища порівняно з конфігурацією, де кількість нейронів дорівнює або перевищує кількість кластерів (2 входи, 5 нейронів). Це свідчить про те, що недостатня кількість нейронів не може адекватно представити всі кластери, що призводить до вищої помилки.

У обох випадках помилка зменшується на початкових етапах тренування, але потім стабілізується і не показує значного подальшого зниження. Це може вказувати на досягнення мережею своєї максимальної здатності до класифікації з даними параметрами.

**Висновок:** використав спеціалізовані бібліотеки та мову програмування Python навчився створювати та застосовувати прості нейронні мережі.