**Лабораторна робота 5**

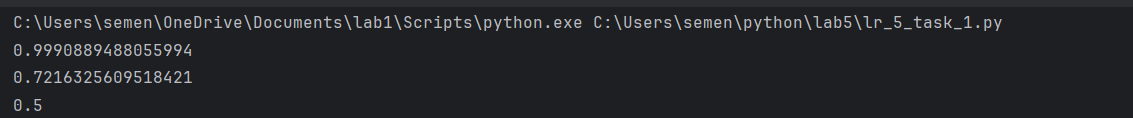
РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

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

**Завдання 2.1. Створити простий нейрон**

import numpy as np  
  
  
def sigmoid(x):  
 # function: 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):  
 # Weight inputs, add bias, then use the activation function  
 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))  
  
  
class NovichkovNeuralNetwork:  
 def \_\_init\_\_(self):  
 weights = np.array([0, 1])  
 bias = 0  
 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)  
  
 out\_o1 = self.o1.feedforward(np.array([out\_h1, out\_h2]))  
 return out\_o1

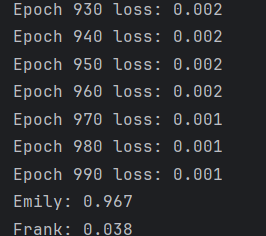
network = NovichkovNeuralNetwork()  
x = np.array([2, 3])  
print(network.feedforward(x))  
  
  
def mse\_loss(y\_true, y\_pred):  
 # y\_true and y\_pred are numpy arrays of the same length  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
  
y\_true = np.array([1, 0, 0, 1])  
y\_pred = np.array([0, 0, 0, 0])  
  
print(mse\_loss(y\_true, y\_pred))



**Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини**

import numpy as np  
  
def sigmoid(x):  
 # function: f(x) = 1 / (1 + e^(-x))  
 return 1 / (1 + np.exp(-x))  
  
def sigmoid\_derivative(x):  
 # derivative of sigmoid: f'(x) = f(x) \* (1 - f(x))  
 fx = sigmoid(x)  
 return fx \* (1 - fx)  
  
def mse\_loss(y\_true, y\_pred):  
 # y\_true and y\_pred are numpy arrays of the same length  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
class NovichkovNeuralNetwork:  
 *"""  
 A neural network with:  
 - 2 inputs  
 - a hidden layer with 2 neurons (h1, h2)  
 - an output layer with 1 neuron (o1)  
 Each neuron has the same weights and bias:  
 - w = [0, 1]  
 - b = 0  
 """* def \_\_init\_\_(self):  
 # weights  
 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()  
   
 # biases  
 self.b1 = np.random.normal()  
 self.b2 = np.random.normal()  
 self.b3 = np.random.normal()  
   
 def feedforward(self, x):  
 # x is a numpy array with 2 elements.  
 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  
   
 for epoch in range(epochs):  
 for x, y\_true in zip(data, all\_y\_trues):  
 # --- Do a feedforward (we'll need these values later)  
 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  
   
 # --- Calculate partial derivatives.  
 # --- Naming: d\_L\_d\_w1 represents "partial L / partial w1"  
 d\_L\_d\_ypred = -2 \* (y\_true - y\_pred)  
   
 # Neuron o1  
 d\_ypred\_d\_w5 = h1 \* sigmoid\_derivative(sum\_o1)  
 d\_ypred\_d\_w6 = h2 \* sigmoid\_derivative(sum\_o1)  
 d\_ypred\_d\_b3 = sigmoid\_derivative(sum\_o1)  
   
 d\_ypred\_d\_h1 = self.w5 \* sigmoid\_derivative(sum\_o1)  
 d\_ypred\_d\_h2 = self.w6 \* sigmoid\_derivative(sum\_o1)  
   
 # Neuron h1  
 d\_h1\_d\_w1 = x[0] \* sigmoid\_derivative(sum\_h1)  
 d\_h1\_d\_w2 = x[1] \* sigmoid\_derivative(sum\_h1)  
 d\_h1\_d\_b1 = sigmoid\_derivative(sum\_h1)  
   
 # Neuron h2  
 d\_h2\_d\_w3 = x[0] \* sigmoid\_derivative(sum\_h2)  
 d\_h2\_d\_w4 = x[1] \* sigmoid\_derivative(sum\_h2)  
 d\_h2\_d\_b2 = sigmoid\_derivative(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  
   
 # --- Calculate total loss at the end of each epoch  
 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))  
   
# Define dataset  
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  
])  
  
# Train our neural network!  
network = NovichkovNeuralNetwork()  
network.train(data, all\_y\_trues)  
  
# Make some predictions  
emily = np.array([-7, -3]) # 128 pounds, 63 inches  
frank = np.array([20, 2]) # 155 pounds, 68 inches  
print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F  
print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M

**Результат:**

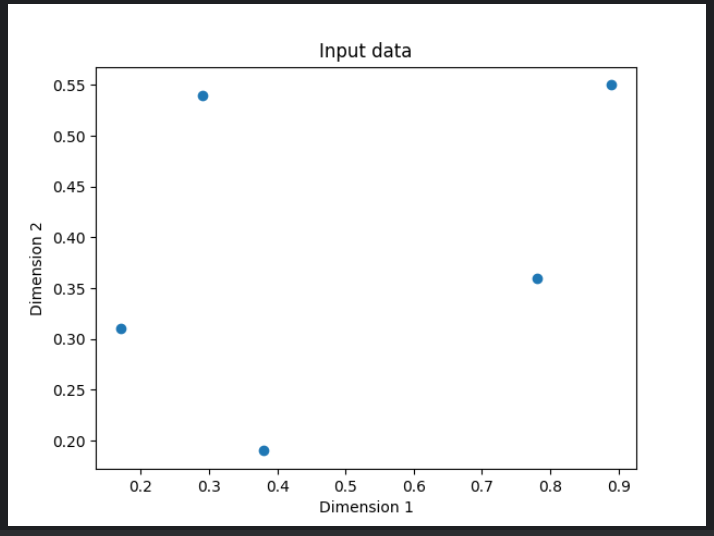
****

Завдання 2.3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

Лістинг:

import numpy as np  
import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
# Load input data  
text = np.loadtxt('data\_perceptron.txt')  
  
# Separate datapoints and labels  
data = text[:, :2]  
labels = text[:, 2].reshape((text.shape[0], 1))  
  
# Plot input data  
plt.figure()  
plt.scatter(data[:,0], data[:,1])  
plt.xlabel('Dimension 1')  
plt.ylabel('Dimension 2')  
plt.title('Input data')  
  
# Minimum and maximum values for each dimension  
dim1\_min, dim1\_max, dim2\_min, dim2\_max = 0, 1, 0, 1  
  
# Number of neurons in the output layer  
num\_output = labels.shape[1]  
  
# Define a perceptron with 2 input neurons (because we have 2 dimensions in the input data)  
dim1 = [dim1\_min, dim1\_max]  
dim2 = [dim2\_min, dim2\_max]  
perceptron = nl.net.newp([dim1, dim2], num\_output)  
  
# Train the perceptron using the data  
error\_progress = perceptron.train(data, labels, epochs=100, show=20, lr=0.03)  
  
# Plot the training progress  
plt.figure()  
plt.plot(error\_progress)  
plt.xlabel('Number of epochs')  
plt.ylabel('Training error')  
plt.title('Training error progress')  
plt.grid()  
plt.show()

Результат:



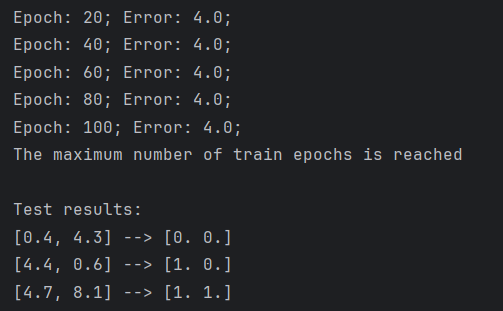


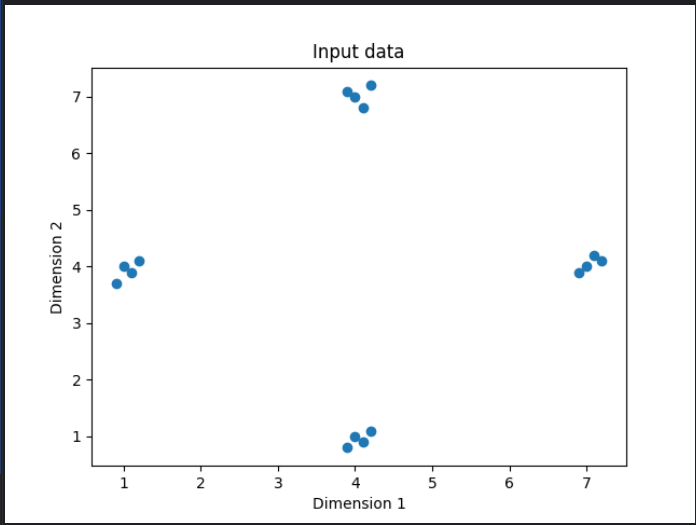
**Завдання 2.4. Побудова одношарової нейронної мережі**

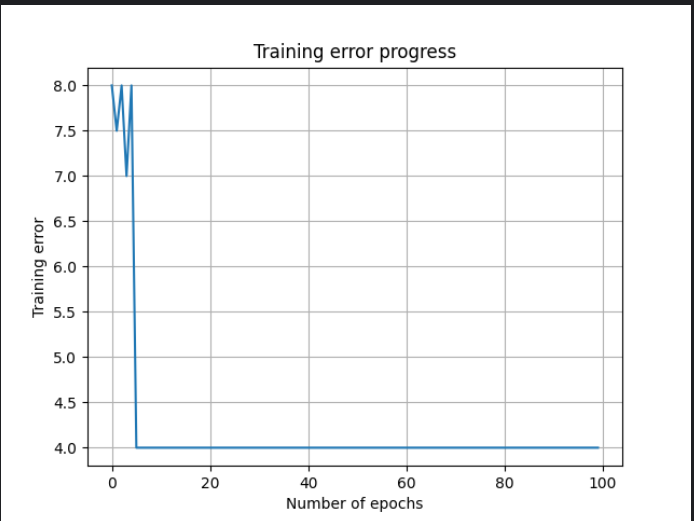
Лістинг:

from os import error  
import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
# Load input data  
text = np.loadtxt('data\_simple\_nn.txt')  
  
# Separate it into datapoints and labels  
data = text[:, 0:2]  
labels = text[:, 2:]  
  
# Plot input data  
plt.figure()  
plt.scatter(data[:,0], data[:,1])  
plt.xlabel('Dimension 1')  
plt.ylabel('Dimension 2')  
plt.title('Input data')  
  
# Minimum and maximum values for each dimension  
dim1\_min, dim1\_max = data[:,0].min(), data[:, 0].max()  
dim2\_min, dim2\_max = data[:,1].min(), data[:, 1].max()  
  
# Define the number of neurons in the output layer  
num\_output = labels.shape[1]  
  
# Define a single-layer neural network with 2 neurons in the input layer (because we have 2 dimensions in the input data) and 10 neurons in the output layer (because we have 10 labels)  
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)  
  
# Plot the training progress  
plt.figure()  
plt.plot(error\_progress)  
plt.xlabel('Number of epochs')  
plt.ylabel('Training error')  
plt.title('Training error progress')  
plt.grid()  
plt.show()  
  
# Run the classifier on test datapoints  
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:



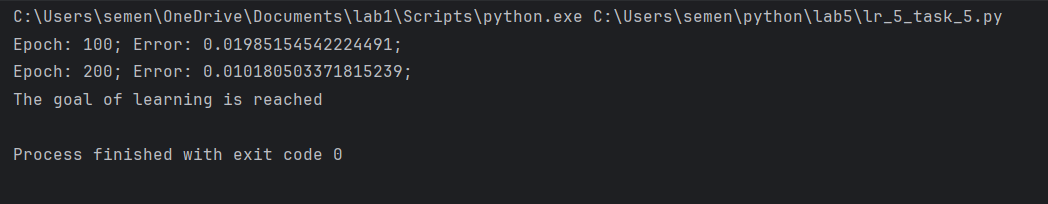


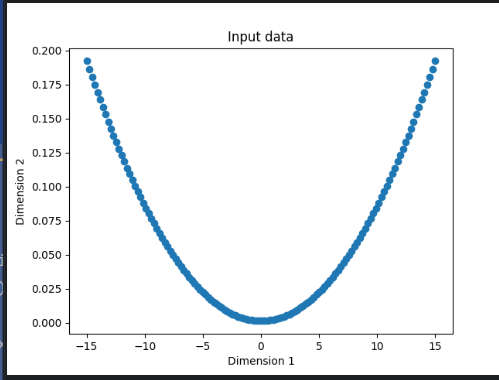


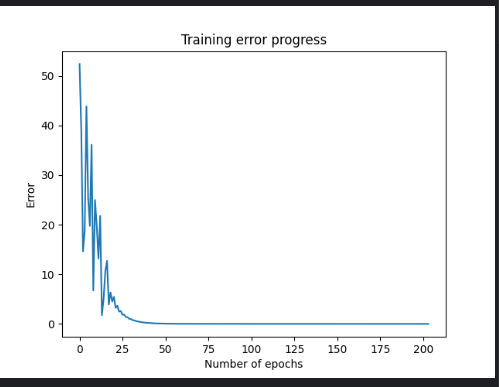
**Завдання 2.5. Побудова багатошарової нейронної мережі**

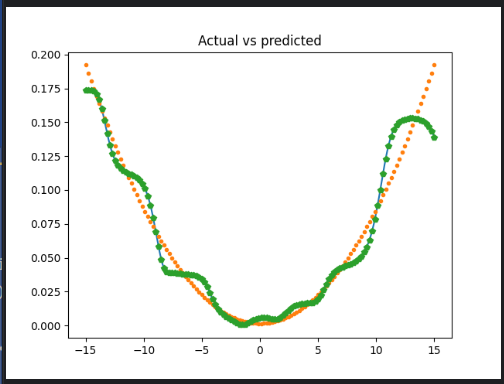
import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
# Generate some training data  
min\_value = -15  
max\_value = 15  
num\_datapoints = 130  
x = np.linspace(min\_value, max\_value, num\_datapoints)  
y = 3 \* np.square(x) + 5  
y /= np.linalg.norm(y)  
  
# Create data and labels  
data = x.reshape(num\_datapoints, 1)  
labels = y.reshape(num\_datapoints, 1)  
  
# Plot input data  
plt.figure()  
plt.scatter(data, labels)  
plt.xlabel('Dimension 1')  
plt.ylabel('Dimension 2')  
plt.title('Input data')  
  
# Define a multilayer neural network with 2 hidden layers; the first hidden layer has 10 neurons and the second one has 6 neurons, and the output layer has 1 neuron  
nn = nl.net.newff([[min\_value, max\_value]], [10, 6, 1])  
  
# Set the training algorithm to gradient descent  
nn.trainf = nl.train.train\_gd  
  
# Train the neural network  
error\_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)  
  
# Execute the neural network on the training data  
output = nn.sim(data)  
y\_pred = output.reshape(num\_datapoints)  
  
# Plot training error  
plt.figure()  
plt.plot(error\_progress)  
plt.xlabel('Number of epochs')  
plt.ylabel('Error')  
plt.title('Training error progress')  
  
# Plot output  
x\_dense = np.linspace(min\_value, max\_value, num\_datapoints \* 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()

Результат:









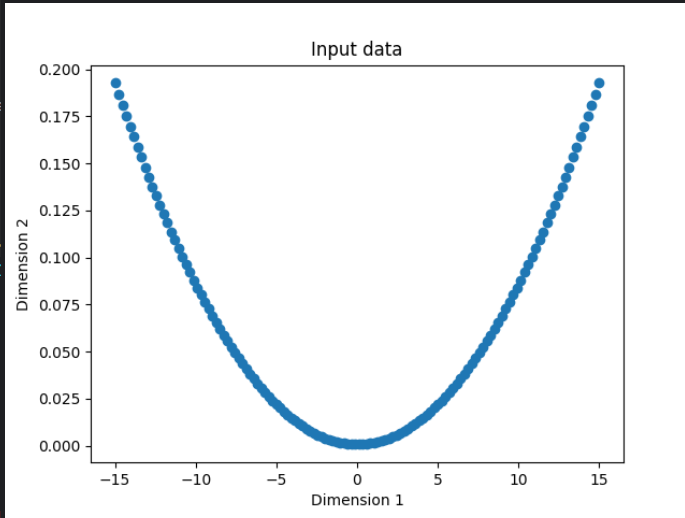
Оскільки навчальна помилка в процесі навчання нейронної мережі зменшувалася (графік 1 та інформація в консолі), навчання апроксимувати криву залежність можна назвати успішним. Мережа припинила навчання коли досягла цільової помилки 0.01. На графіку 2 видно порівняння прогнозованих та реальних даних, що показує що дані є схожими.

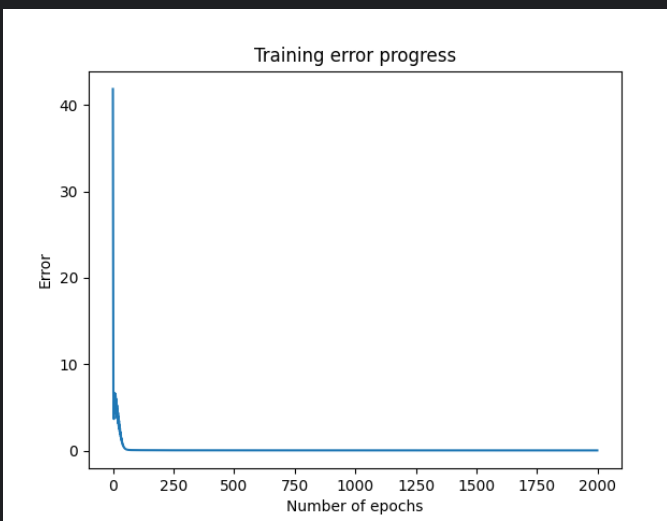
**Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту**

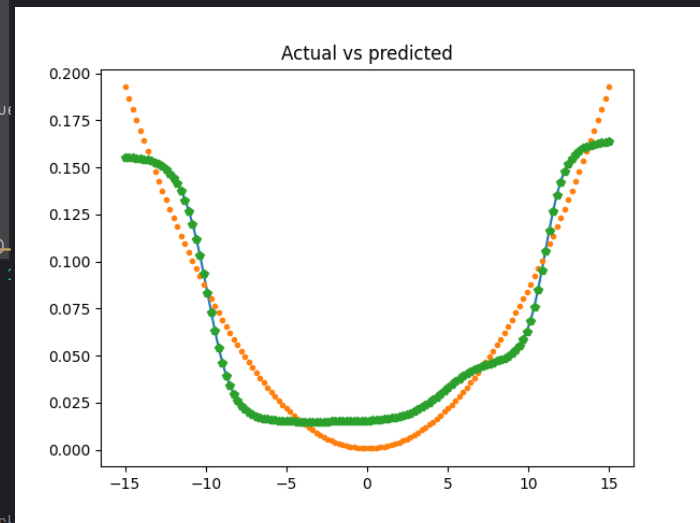
Лістинг:

# Варіант 13  
  
import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
# Generate some training data  
min\_value = -15  
max\_value = 15  
num\_datapoints = 130  
x = np.linspace(min\_value, max\_value, num\_datapoints)  
y = 5 \* np.square(x) + 4  
y /= np.linalg.norm(y)  
  
# Create data and labels  
data = x.reshape(num\_datapoints, 1)  
labels = y.reshape(num\_datapoints, 1)  
  
# Plot input data  
plt.figure()  
plt.scatter(data, labels)  
plt.xlabel('Dimension 1')  
plt.ylabel('Dimension 2')  
plt.title('Input data')  
  
# Define a multilayer neural network with 2 hidden layers; the first hidden layer has 6 neurons and the second one has 1 neuron  
nn = nl.net.newff([[min\_value, max\_value]], [6, 1, 1])  
  
# Set the training algorithm to gradient descent  
nn.trainf = nl.train.train\_gd  
  
# Train the neural network  
error\_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)  
  
# Execute the neural network on the training data  
output = nn.sim(data)  
y\_pred = output.reshape(num\_datapoints)  
  
# Plot training error  
plt.figure()  
plt.plot(error\_progress)  
plt.xlabel('Number of epochs')  
plt.ylabel('Error')  
plt.title('Training error progress')  
  
# Plot output  
x\_dense = np.linspace(min\_value, max\_value, num\_datapoints \* 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()

Результат:





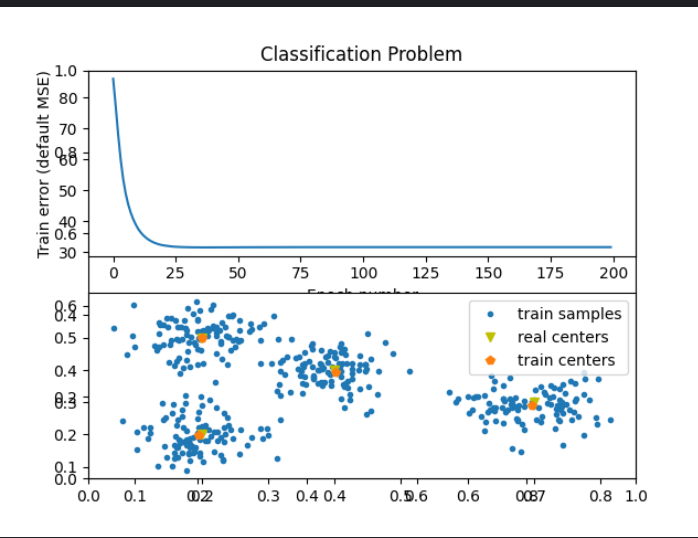


**Завдання 2.7. Побудова нейронної мережі на основі карти Кохонена, що самоорганізується**

Лістинг

import numpy as np   
import neurolab as nl  
import numpy.random as rand  
import pylab as pl  
  
# Standard deviation   
skv = 0.05  
  
# Centers of clusters  
center = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5]])  
  
# Generate random data around the centers  
rand.seed(0)  
rand\_normal = skv \* rand.randn(100, 4, 2)  
input = np.array([center + r for r in rand\_normal])  
input.shape = (100 \* 4, 2)  
  
# Shuffle data  
rand.shuffle(input)  
  
# 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 (CWTA)  
error = net.train(input, epochs=200, show=20)  
  
# Plot results  
pl.title('Classification Problem')  
pl.subplot(211)  
pl.plot(error)  
pl.xlabel('Epoch number')  
pl.ylabel('Train error (default MSE)')  
w = net.layers[0].np['w']  
pl.subplot(212)  
pl.plot(input[:, 0], input[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0], w[:, 1], 'p')  
pl.legend(['train samples', 'real centers', 'train centers'])  
pl.show()

Результат:



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

Лістинг:

import numpy as np   
import neurolab as nl  
import numpy.random as rand  
import pylab as pl  
  
# Standard deviation   
skv = 0.05  
  
# Centers of clusters  
center = np.array([[0.2, 0.2], [0.3, 0.4], [0.6, 0.3], [0.2, 0.5], [0.5, 0.5]])  
  
# Generate random data around the centers  
rand.seed(0)  
rand\_normal = skv \* rand.randn(100, 5, 2)  
input = np.array([center + r for r in rand\_normal])  
input.shape = (100 \* 5, 2)  
  
# Shuffle data  
rand.shuffle(input)  
  
# Create net with 2 inputs and 5 neurons  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 5)  
  
# Train with rule: Conscience Winner Take All (CWTA)  
error = net.train(input, epochs=200, show=20)  
  
# Plot results  
pl.title('Classification Problem')  
pl.subplot(211)  
pl.plot(error)  
pl.xlabel('Epoch number')  
pl.ylabel('Train error (default MSE)')  
w = net.layers[0].np['w']  
pl.subplot(212)  
pl.plot(input[:, 0], input[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0], w[:, 1], 'p')  
pl.legend(['train samples', 'real centers', 'train centers'])  
pl.show()

Результат:

