**Лабораторна робота 5**

**Тема*:*** РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

**Мета*:*** використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовуватипрості нейронні мережі**.**

**Хід роботи**

**Завдання 1.** Створити простий нейрон

**Лістинг програми:**

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):  
 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))



**Рис. 1** Результат виконання програми

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

**Лістинг програми**

import numpy as np  
  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))

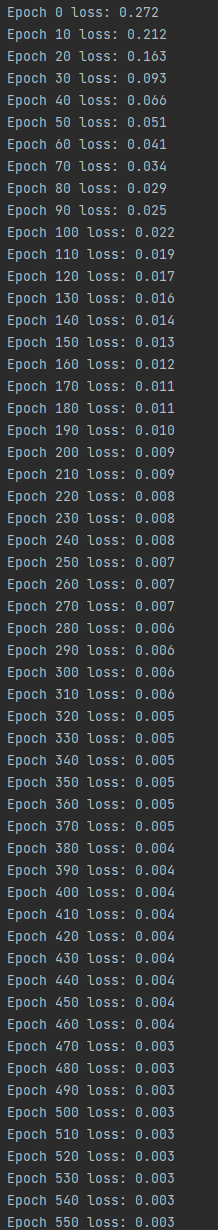
class Neuron:  
 def \_\_init\_\_(self, weights, bias):  
 self.weights = weights  
 self.bias = bias  
  
 def feedforward(self, inputs):  
 total = np.dot(self.weights, inputs) + self.bias  
 return sigmoid(total)  
  
  
weights = np.array([0, 1]) # w1 = 0, w2 = 1  
bias = 4 # b = 4  
n = Neuron(weights, bias)  
  
x = np.array([2, 3]) # x1 = 2, x2 = 3  
  
class VoitkoNeuralNetwork:  
  
 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 = VoitkoNeuralNetwork()  
x = np.array([2, 3])  
print(network.feedforward(x)) # 0.7216325609518421



**Рис. 2** Результат виконання програми

**Лістинг програми**

import numpy as np  
  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
  
def deriv\_sigmoid(x):  
 fx = sigmoid(x)  
 return fx \* (1 - fx)  
  
  
def mse\_loss(y\_true, y\_pred):  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
  
class VoitkoNeuralNetwork:  
  
 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)  
  
 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)  
  
 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)  
  
 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)  
  
 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  
  
 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  
  
 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))  
  
  
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 = VoitkoNeuralNetwork()  
network.train(data, all\_y\_trues)  
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

****

**Рис. 3** Результат виконання програми

Зображення, що містить текст

Автоматично згенерований опис

**Рис. 4** Результат виконання програми

**Висновок**

Функція активації, або передавальна функція штучного нейрона — залежність вихідного сигналу штучного нейрона від вхідного. Більшість видів нейронних мереж для функції активації використовують сигмоїди.

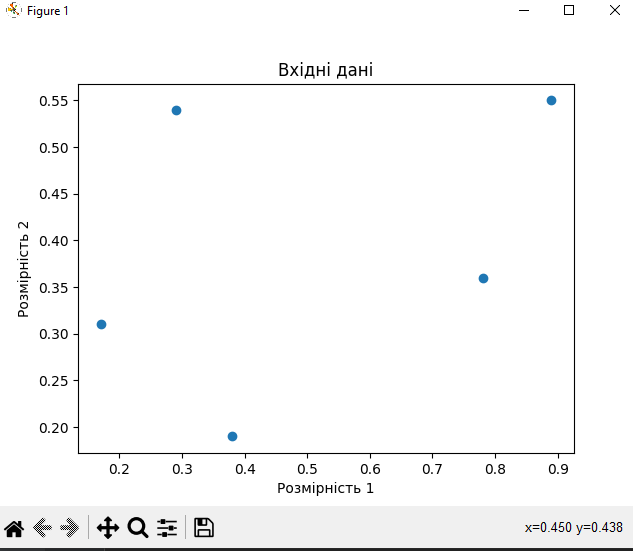
Можливості нейроннних мереж прямого поширення полягають в тому, що сигнали поширюються в одному напрямку, починаючи від вхідного шару нейронів, через приховані шари до вихідного шару і на вихідних нейронах отримується результат опрацювання сигналу. В мережах такого виду немає зворотніх зв’язків.

Нейронні мережі прямого поширення знаходять своє застосування в задачах комп'ютерного бачення та розпізнаванні мовлення, де класифікація цільових класів ускладнюється. Такі типи нейронних мереж добре справляються із зашумленими даними.

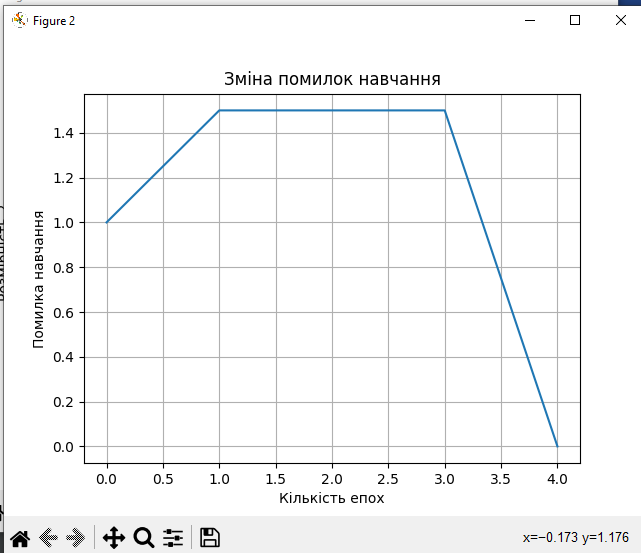
**Завдання 2.3.** Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

**Лістинг програми**

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('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
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('Кількість епох')  
plt.ylabel('Помилка навчання')  
plt.title('Зміна помилок навчання')  
plt.grid()  
plt.show()



**Рис. 5** Результат виконання програми



**Рис. 6** Результат виконання програми

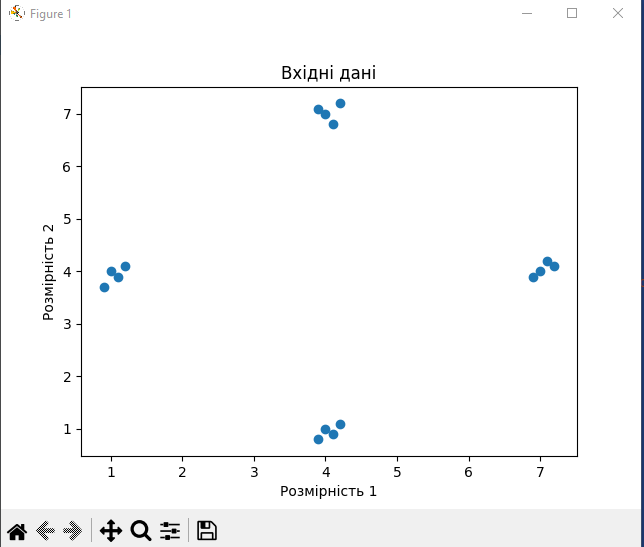
**Висновок**

На другому графіку я відобразив процес навчання, використовуючи метрику помилки. Під час першої епохи відбулося від 1.0 до 1.5 помилок, під час наступних двох епох відбулось 1.5 помилок. Потім під час 4 епохи помилки почались зменшувтась, тому що ми навчили перцептрон за допомогою тренувальних даних.

**Завдання 2.4.** Побудова одношарової нейронної мережі

**Лістинг програми**

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('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
dim1\_min, dim1\_max = data[:, 0].min(), data[:, 0].max()  
dim2\_min, dim2\_max = data[:, 1].min(), data[:, 1].max()  
num\_output = labels.shape[1]  
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)  
plt.figure()  
plt.plot(error\_progress)  
plt.xlabel('Кількість епох')  
plt.ylabel('Помилка навчання')  
plt.title('Зміна помилок навчання')  
plt.grid()  
plt.show()  
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])



**Рис. 7** Графік вхідних даних

Зображення, що містить стіл

Автоматично згенерований опис

**Рис. 8** Графік вхідних даних

Зображення, що містить текст

Автоматично згенерований опис

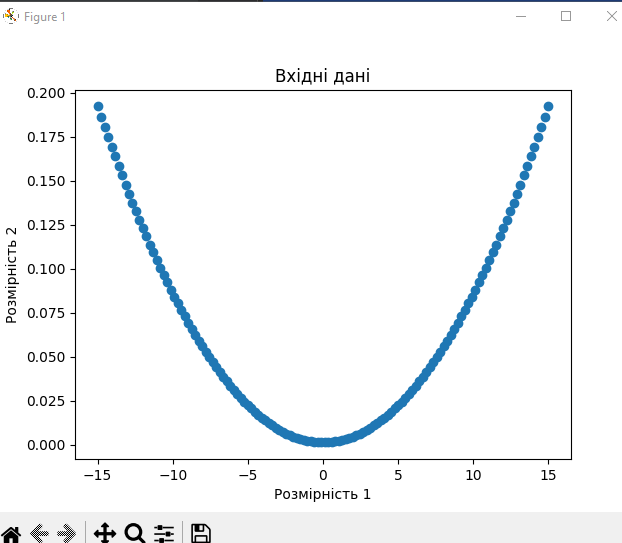
**Рис. 9** Результат виконання програми

**Висновок**

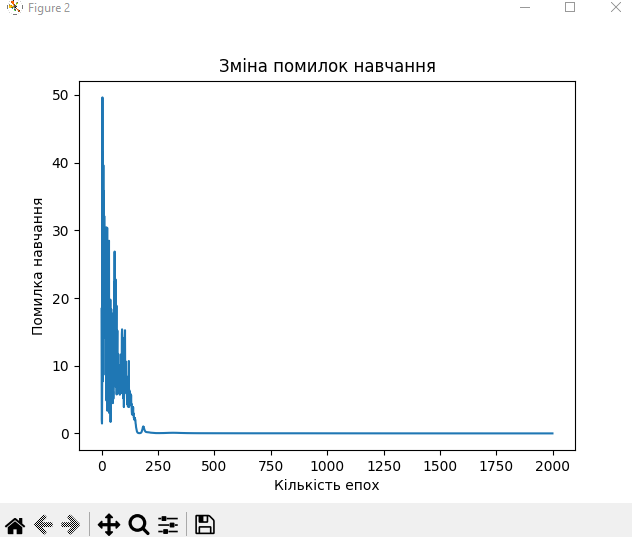
На рис. 20 зображено процес навчання мережі. На 20 епосі відбулось 4 помилки, аналогічно на 40, 60, 80 та 100. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування. Ми вирішили визначити вибіркові тестові точки даних та запустили для них нейронну мережу. І це його результат.

**Завдання 2.5**. Побудова багатошарової нейронної мережі

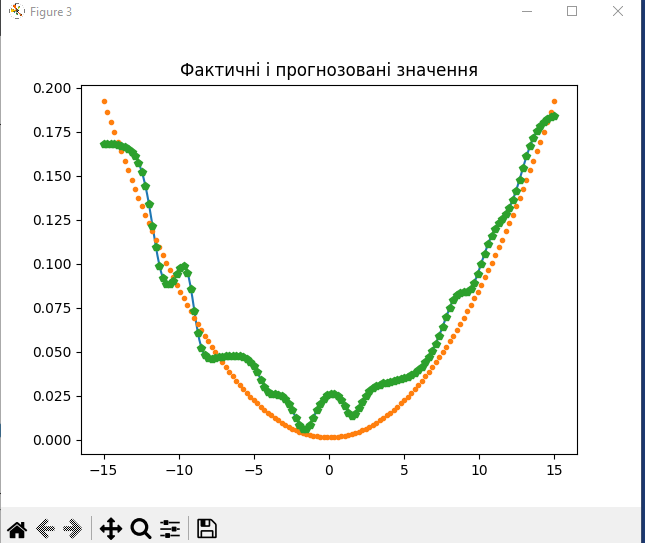
import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
min\_val = -15  
max\_val = 15  
num\_points = 130  
x = np.linspace(min\_val, max\_val, num\_points)  
y = 3 \* np.square(x) + 5  
y /=np.linalg.norm(y)  
data = x.reshape(num\_points, 1)  
labels = y.reshape(num\_points, 1)  
plt.figure()  
plt.scatter(data, labels)  
plt.xlabel('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
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('Кількість епох')  
plt.ylabel('Помилка навчання')  
plt.title('Зміна помилок навчання')  
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 і прогнозовані значення')  
plt.show()



**Рис. 10** Результат виконання програми



**Рис. 11** Результат виконання програми



**Рис. 12** Результат виконання програми

Зображення, що містить текст

Автоматично згенерований опис

**Рис. 13** Результат виконання програми

**Висновок**

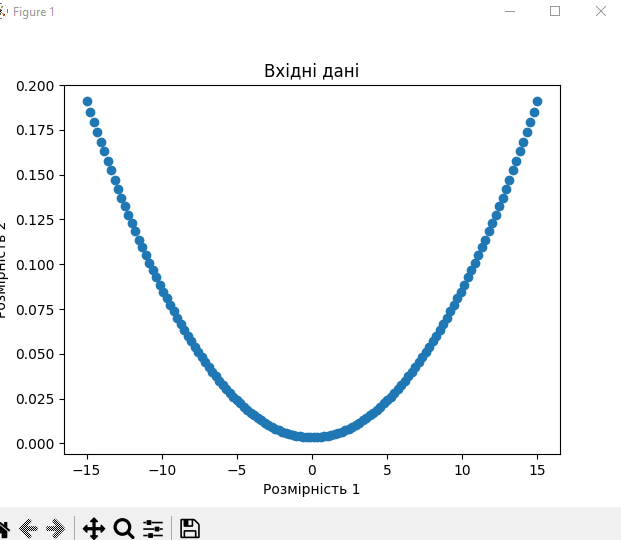
На рис. 13 зображено процес навчання мережі. Відносно кожної епосі відбувались помилки. На 100 3.87 помилки. На 2000 0.01. Потім вивелось повідомлення, що ми досягли цілі навчання.

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

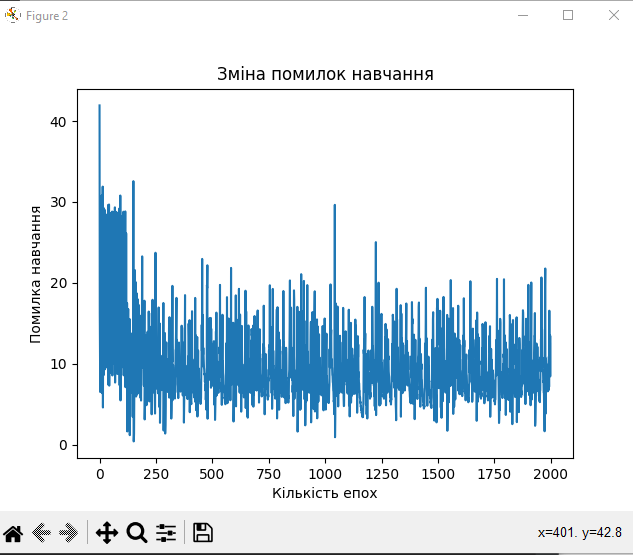
****



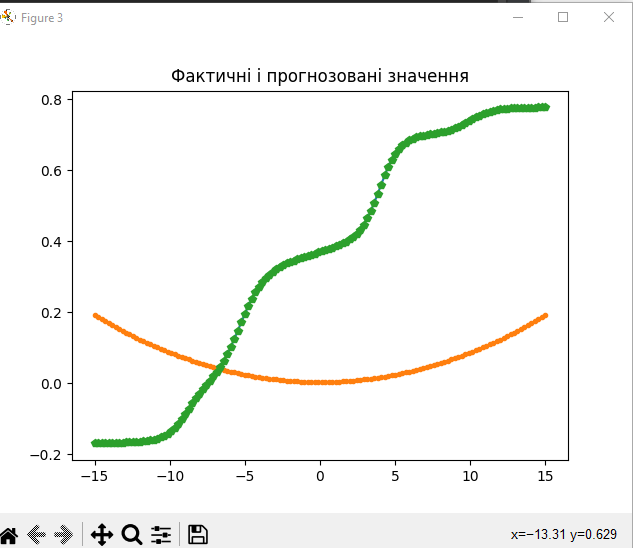
import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
min\_val = -15  
max\_val = 15  
num\_points = 130  
x = np.linspace(min\_val, max\_val, num\_points)  
y = 2 \* x \* x + 6  
y /=np.linalg.norm(y)  
data = x.reshape(num\_points, 1)  
labels = y.reshape(num\_points, 1)  
plt.figure()  
plt.scatter(data, labels)  
plt.xlabel('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
nn = nl.net.newff([[min\_val, max\_val]], [2, 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('Кількість епох')  
plt.ylabel('Помилка навчання')  
plt.title('Зміна помилок навчання')  
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 і прогнозовані значення')  
plt.show()



**Рис. 14** Результат виконання програми



**Рис. 15** Результат виконання програми



**Рис. 16** Результат виконання програми

Зображення, що містить текст

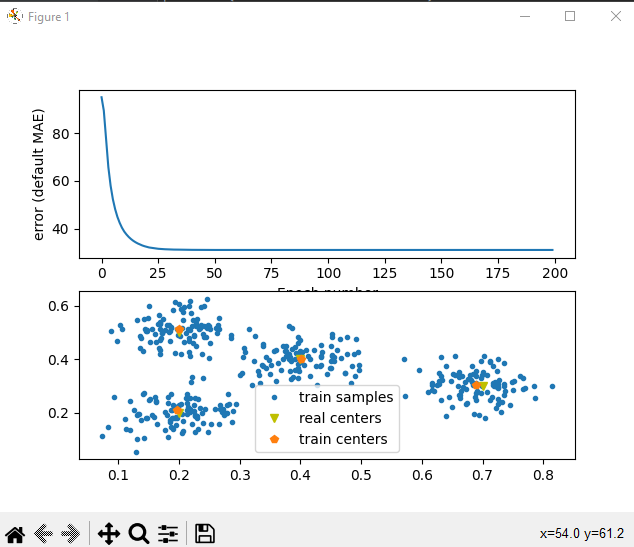
Автоматично згенерований опис

**Рис. 17** Результат виконання програми

На рис. 17 зображено процес навчання мережі. На 100 епосі відбулось 13.45 помилки, на 200 епосі відбулось 17.80 помилки, і так далі, на 2000 епосі відбулось 13.36 помилки,. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

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

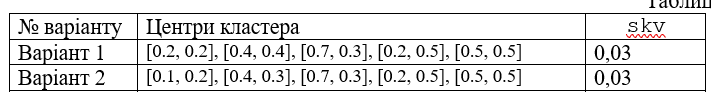
import numpy as np  
import neurolab as nl  
import numpy.random as rand  
  
skv = 0.05  
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5]])  
rand\_norm = skv \* rand.randn(100, 4, 2)  
inp = np.array([centr + r for r in rand\_norm])  
inp.shape = (100 \* 4, 2)  
rand.shuffle(inp)  
  
# 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=100)  
  
# 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()



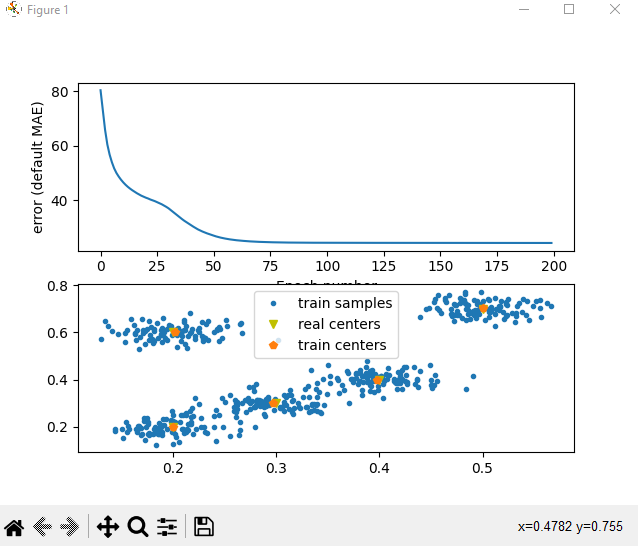
**Рис. 18** Результат виконання програми

Помилка MAE - [Средня абсолютна помилка (Mean Absolute Error). Середньою абсолютною похибкою називають середнє арифметичне з абсолютних похибок усіх вимірювань.](https://wiki.loginom.ru/articles/mae.html)

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



import numpy as np  
import neurolab as nl  
import numpy.random as rand  
  
skv = 0.03  
centr = np.array([[0.1, 0.2], [0.4, 0.3], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]])  
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 5 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()



**Рис. 19** Результат виконання програми

Зображення, що містить текст

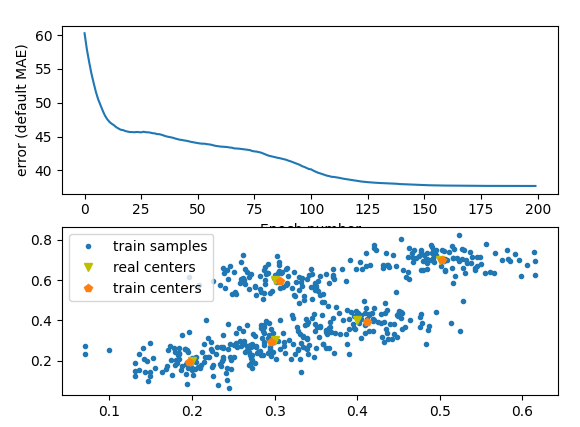
Автоматично згенерований опис

**Рис. 20** Результат виконання програми

На рис. 20 зображено процес навчання мережі. На 20 епосі відбулось 41.28 помилки, помилки і так далі, на 200 епосі відбулось 24.26 помилки,. Потім вивелось повідомлення, що ми досягли максимальної кількості епох для тренування.

Зображення, що містить текст

Автоматично згенерований опис



**Рис. 21** Результат виконання програми

Зображення, що містить текст

Автоматично згенерований опис

**Рис. 22** Результат виконання програми

Якщо порівнювати нейронну мережу Кохонена з 4 нейронами та 5 нейронами, можна зробити такі висновки. При 4 нейронах Помилка МAE повільніше зменшується, ніж з 5 нейронами,також з 5 нейронами ця помилка нижча. З 5 нейронами обоє центрів збігаються майже в одні точці. Число нейронів в шарі Кохонена має відповідати числу класів вхідних сигналів. Тобто в нашому випадку нам давалось 5 вхідних сигналів, значить у нас має бути 5 нейронів, а не 4.Отже, невірний вибір кількості нейронів числу кластерів впливає на величину помилки ускладнюючи навчання мережі і швидкості, тому на рис. 33 набагато гірші результати, ніж на рис. 36

**Висновок:** під час виконання лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовуватипрості нейронні мережі**.**

**Репозиторій:** **https://github.com/BOYYYKO/ai/tree/main/lr5**