**ЛАБОРАТОРНА РОБОТА № 5**

**РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ**

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

**1. ТЕОРЕТИЧНІ ВІДОМОСТІ**

Теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Джоши Пратик. Искусственный интеллект с примерами на Python. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 448 с. - Парал. тит. англ. ISBN 978-5-907114-41-8 (рус.)

Можна використовувати Google Colab або Jupiter Notebook.

<https://python-scripts.com/intro-to-neural-networks>

**2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ**

**Завдання 2.1. Створити простий нейрон**

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))

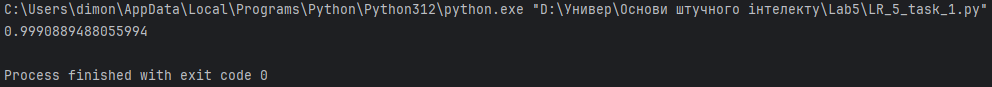


Рис 1. Результат роботи програми

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

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 kulieshovNeuralNetwork:

*"""*

*Нейронна мережа, у якої:*

*- 2 входи*

*- прихований шар з двома нейронами (h1, h2)*

*- шар виходу з одним нейроном (o1)*

*\*\*\* ВАЖЛИВО \*\*\*:*

*Код нижче написаний як простий, навчальний. НЕ оптимальний.*

*Справжній код нейронної мережі виглядає не так. НЕ ВИКОРИСТОВУЙТЕ цей код у подальшому.*

*Замість цього, прочитайте та запустіть його, щоб зрозуміти, як працює ця мережа.*

*"""*

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 = kulieshovNeuralNetwork()

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

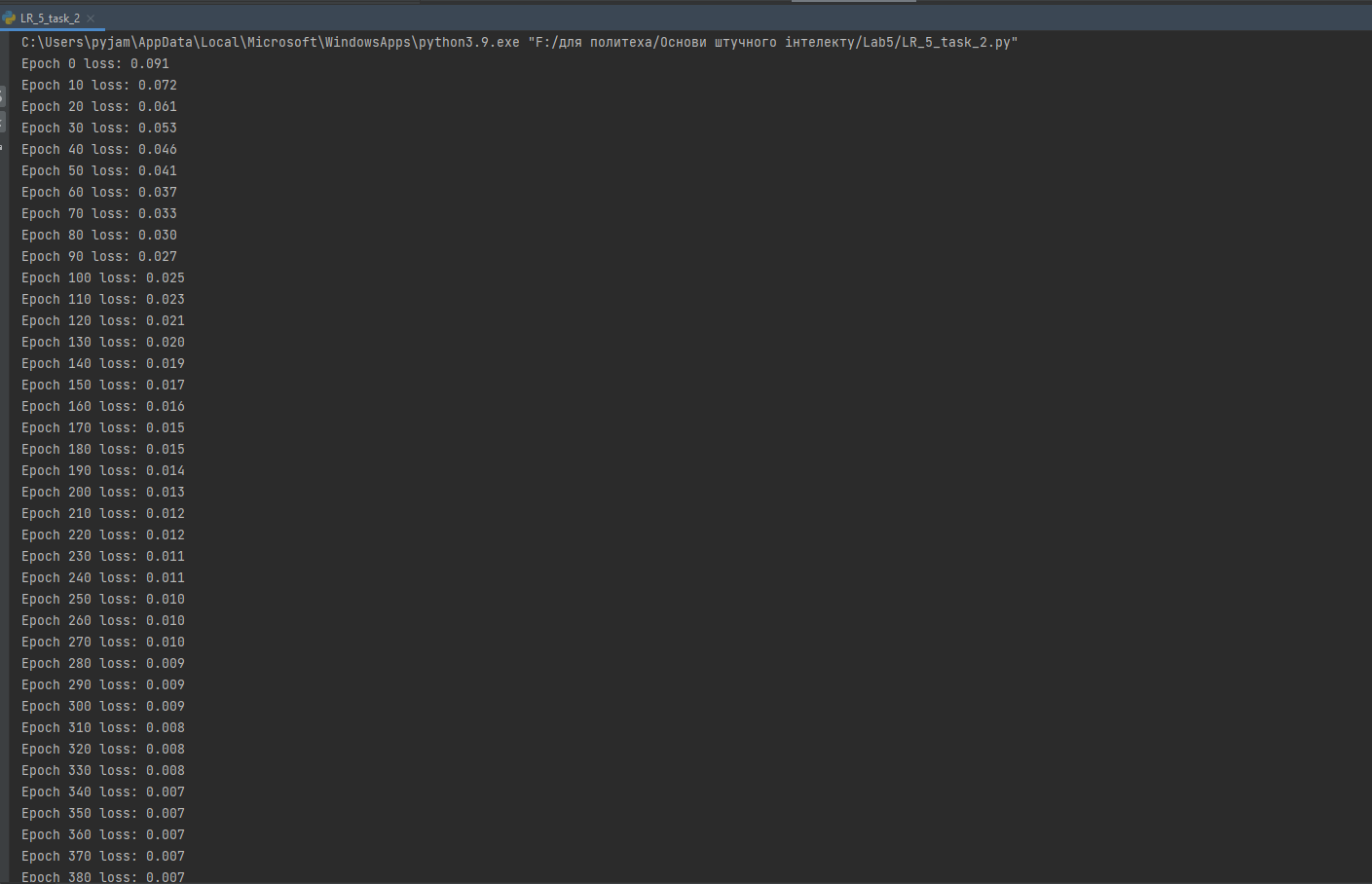


Рис 2. Результат роботи програми

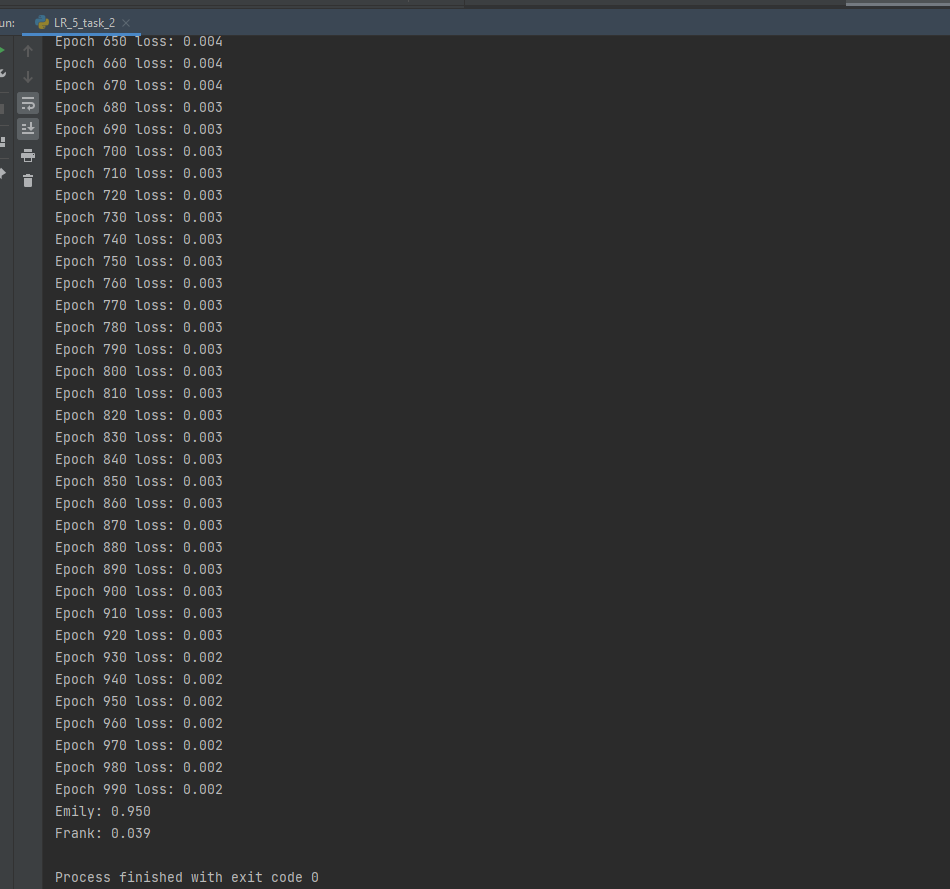


Рис 3. Продовження рис 2

Функція активації в нейронних мережах визначає вихід нейрона, враховуючи ваги та вхідні дані. У даному випадку використовується сигмоїдна функція активації, яка обмежує вихід нейрона у діапазоні (0, 1). Це корисно для оцінки ймовірностей та генерації ймовірнісних вихідних даних.

У нашому прикладі нейронна мережа використовується для класифікації чотирьох осіб на дві категорії (1 та 0). Мережа складається з 2 вхідних нейронів, 2 прихованих нейронів і 1 вихідного нейрона. Під час навчання ваги та зсуви оновлюються для покращення відповідності мережі вхідним даним та отримання бажаного вихідного результату. У даному випадку мережа прогнозує, чи належить кожна особа до категорії 1 чи 0 на підставі їхніх характеристик.

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

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 навчання')

plt.grid()

plt.show()

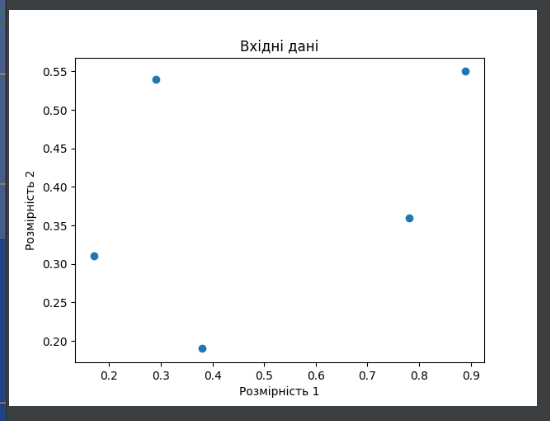


Рис 4. Вхідні дані

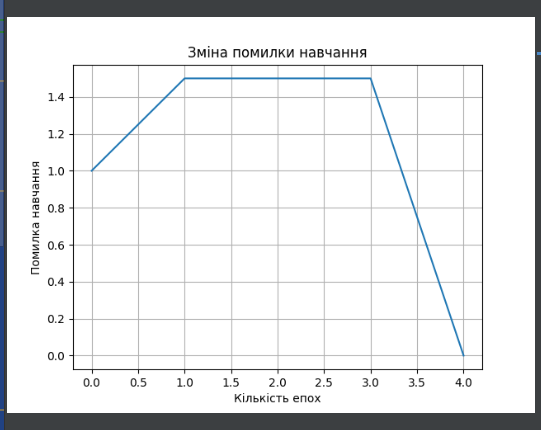


Рис 5. Зміна помилки навчання

Перший графік демонструє розсіяну діаграму з вхідними даними, де точки розташовані на двовимірній площині з відображенням значень першої та другої розмірностей. Ось x позначена як "Розмірність 1", а ось y як "Розмірність 2".

Другий графік ілюструє зміну помилки навчання протягом епох. Ось x вказує на "Кількість епох", а ось y на "Помилка навчання". За динамікою зменшення помилки можна припустити ефективність навчання моделі, в той час як велика або зростаюча помилка може свідчити про проблеми з навчанням або недостатню складність моделі для даної задачі.

**Завдання 2.4. Побудова одношарової нейронної мережі**

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)

# Навчимо мережу на тренувальних даних

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])

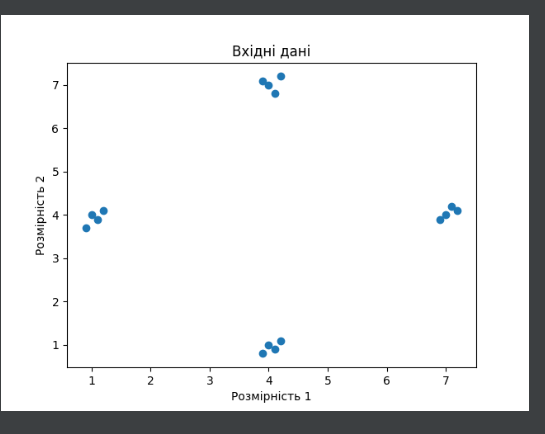


Рис 6. Вхідні дані

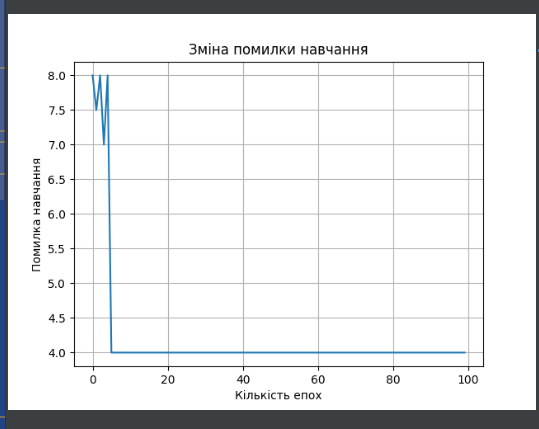


Рис 7. Зміна помилки навчання

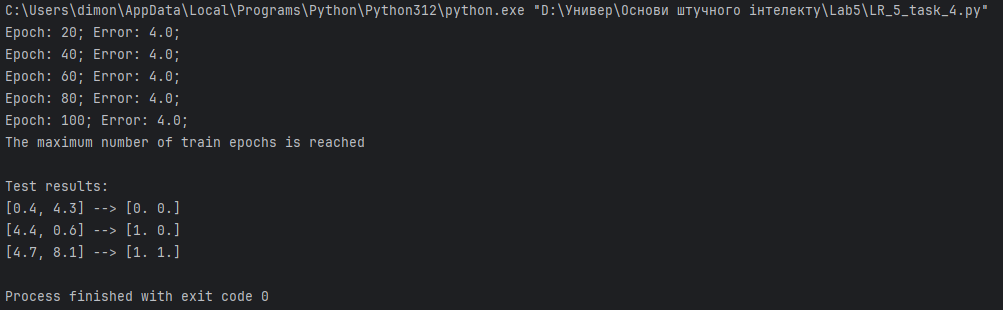
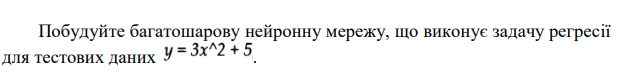


Рис 8. Результат роботи програми

Висока помилка навчання, яка залишається стабільною протягом всіх епох та припинення процесу після досягнення максимальної кількості епох (100), свідчать про те, що одношарова нейронна мережа не здатна ефективно розв'язати задачу класифікації даних. Це може бути обумовлено складною структурою даних або нелінійними зв'язками між їхніми ознаками. Зазначена неспроможність лінійно розділити дані може бути фактором, який впливає на неефективність навчання нейронної мережі.

**Завдання 2.5. Побудова багатошарової нейронної мережі**



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.

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()

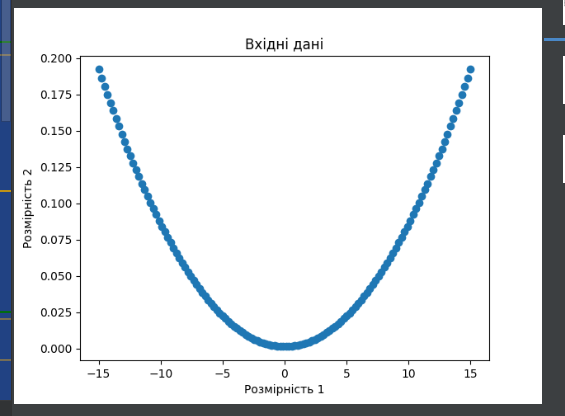


Рис 9. Вхідні дані

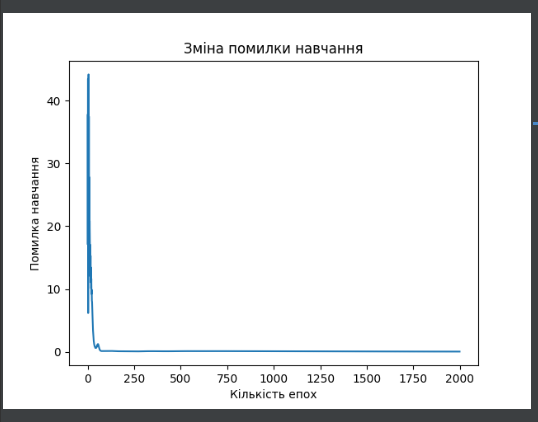


Рис 10. Зміна помилки навчання

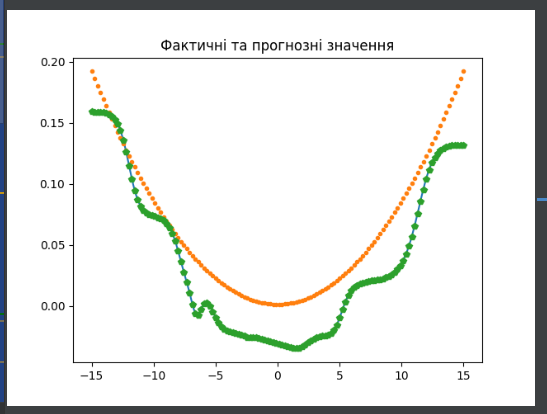


Рис 11. Фактичні та прогнозні значення

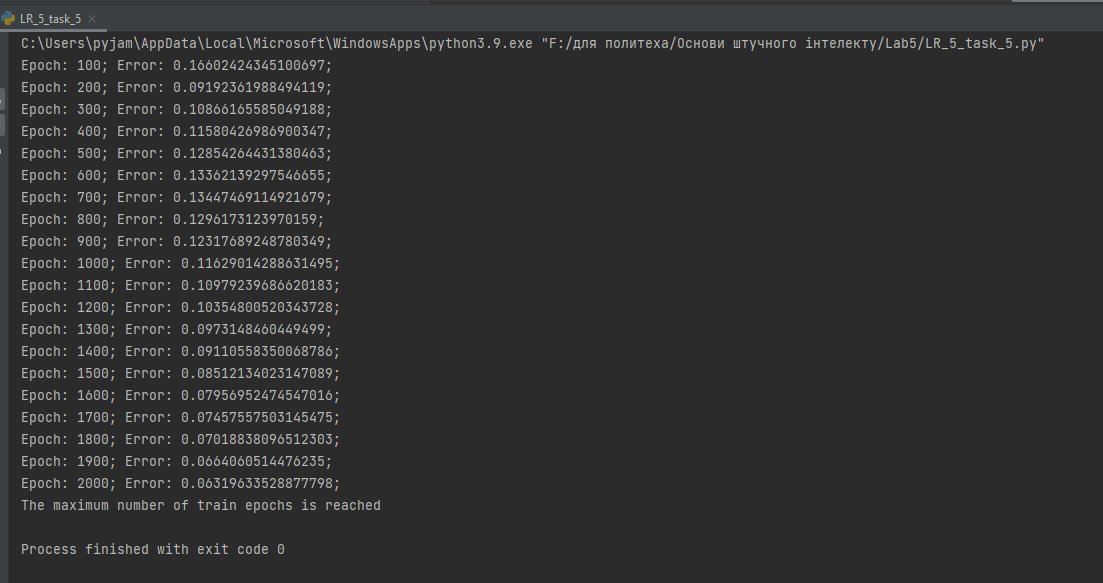


Рис 12. Результат роботи програми

.Починаючи з помилки на рівні 0.065114400336199 на 100-ій епосі, значення помилки послідовно зменшується на кожній наступній ітерації. Це свідчить про поступовий процес навчання нейронної мережі, яка успішно зменшує помилку та наближає прогнозовані значення до фактичних.

Повідомлення "The goal of learning is reached" свідчить про успішне завершення процесу навчання і досягнення поставленої цілі, яка передбачала досягнення значення помилки менше 0.01. Це вказує на те, що нейронна мережа успішно навчилась апроксимувати залежність між вхідними і вихідними даними, і тепер вона готова до застосування для прогнозування нових даних.

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



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

import matplotlib  
matplotlib.use('TkAgg')  
  
import matplotlib.pyplot as plt  
  
import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
import matplotlib  
matplotlib.use('TkAgg') # або 'Qt5Agg' або 'Agg', виберіть підходящий бекенд  
  
import matplotlib.pyplot as plt  
  
# Генерація тренувальних даних  
min\_val = -15  
max\_val = 15  
num\_points = 130  
x = np.linspace(min\_val, max\_val, num\_points)  
y = 2 \* np.square(x) + 6  
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 трима прихованими  
# шарами. Перший прихований шар складається із трьої нейронів.  
# Другий прихований шар складається з п'яти нейронів.Третій  
# прихований шар складається з одного нейрону.  
  
# Вихідний шар.  
nn = nl.net.newff([[min\_val, max\_val]], [2, 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()

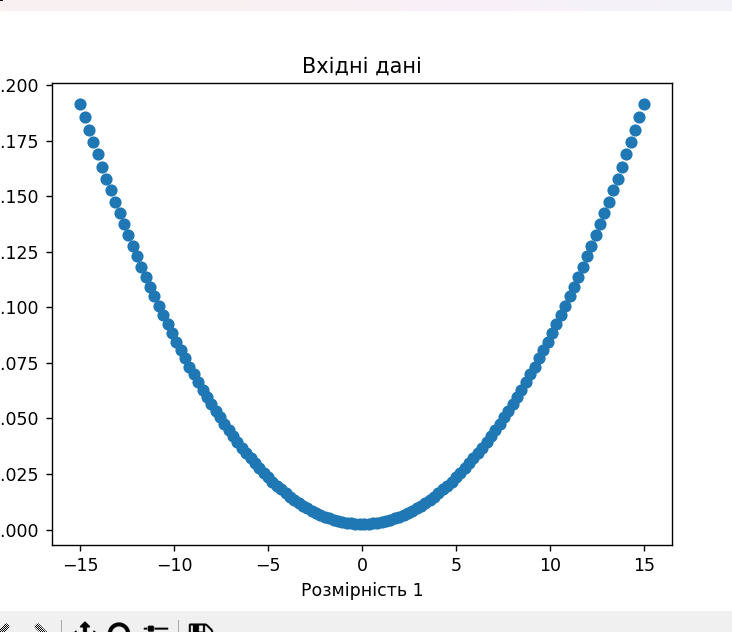


Рис 13. Вхідні дані

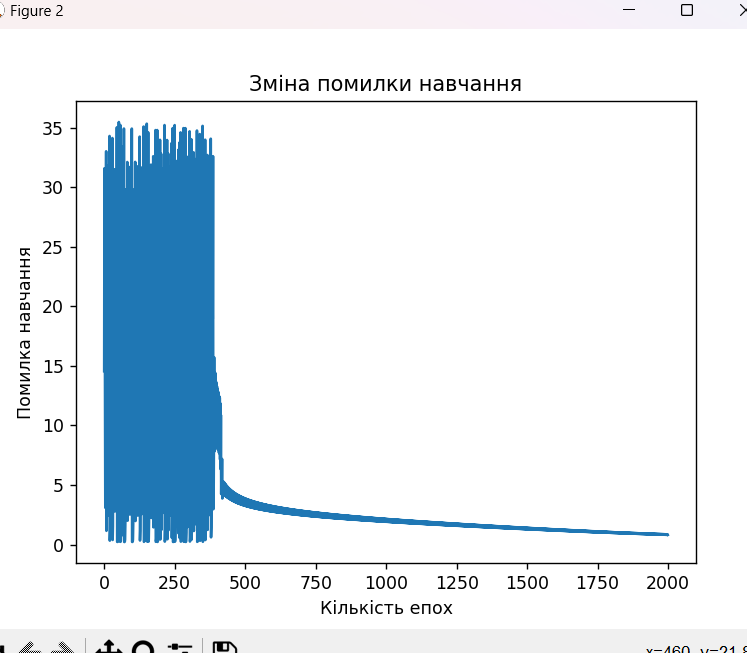


Рис 14. Зміна помилки навчання

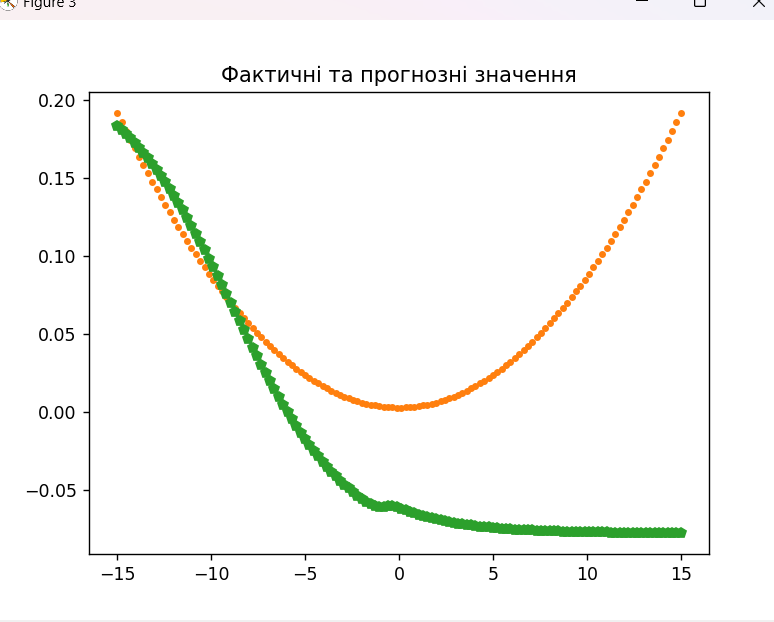


Рис 15. Фактичні та прогнозні значення

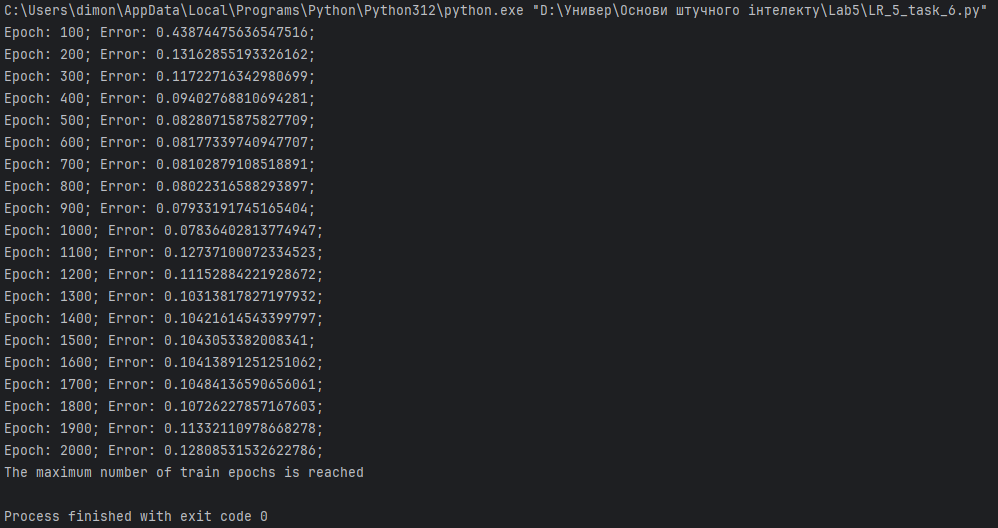


Рис 16. Результат роботи програми

Значення помилки навчання зближуються до нуля, що свідчить про ефективність процесу тренування нейронної мережі у точному відображенні залежності між вхідними та цільовими вихідними даними.

**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=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()

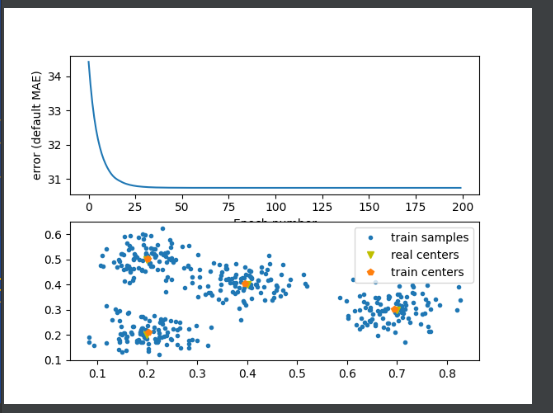


Рис 17. Класифікація проблеми

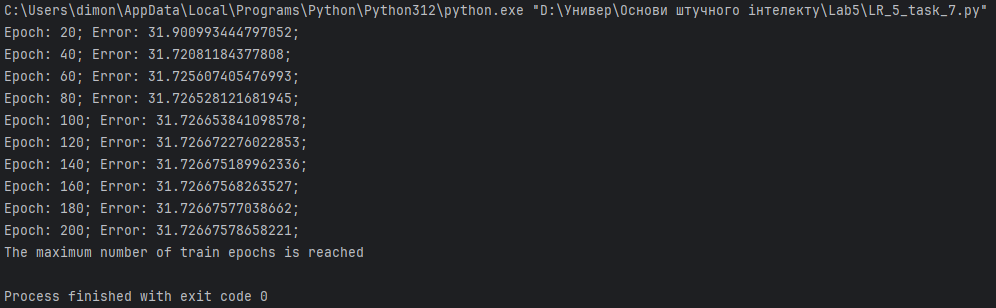


Рис 18. Результат роботи програми

MAE (Mean Absolute Error) - це метрика, яка використовується для вимірювання відхилень між прогнозованими та фактичними значеннями. Вона обчислюється як середнє значення абсолютних різниць між кожним прогнозованим значенням і відповідним фактичним значенням.

Значення, які відображені у вікні терміналу, представляють значення помилки MAE на кожній ітерації під час тренування нейромережі. Починаючи зі значення помилки 31.90 на 20-ій ітерації та послідовно зменшуючись до 31.72 на 200-ій ітерації, це свідчить про те, що мережа навчається, але її темп навчання сповільнюється. Це може вказувати на те, що мережа майже досягла свого оптимального рівня ефективності з використанням цього конкретного набору даних та параметрів навчання.

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





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 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()

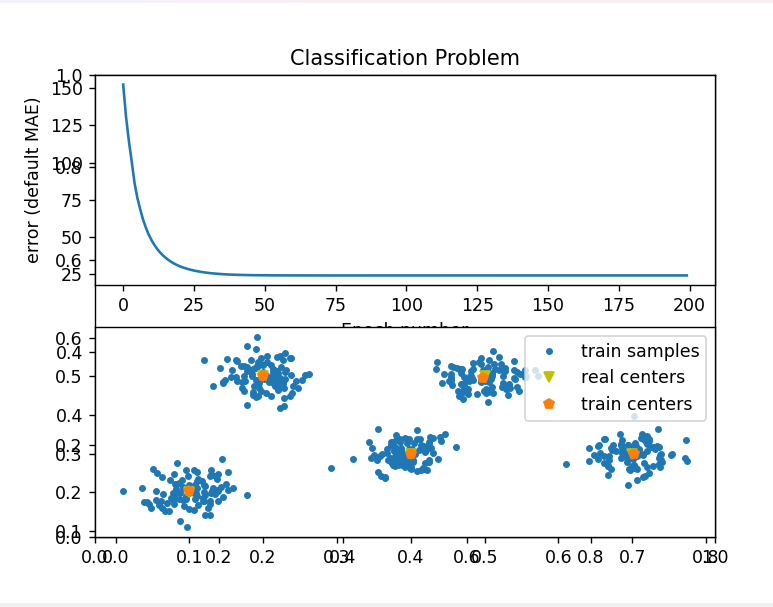


Рис 19. Класифікація проблеми

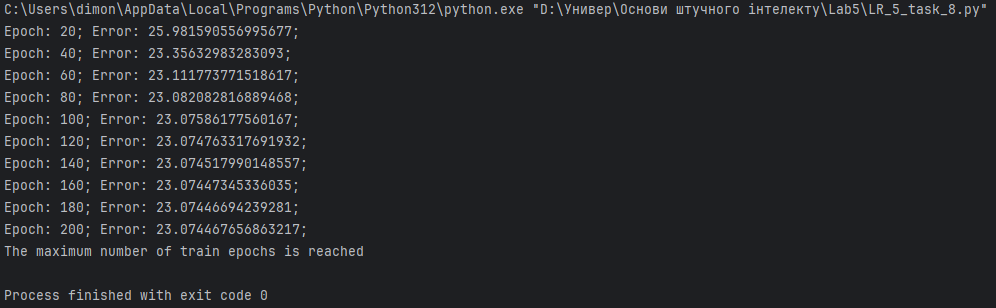


Рис 20. Результат роботи програми

**Створіть нейронну мережу Кохонена з 2 входами та 5 нейронами**

import matplotlib  
matplotlib.use('TkAgg') # або 'Qt5Agg' або 'Agg', виберіть підходящий бекенд  
  
import matplotlib.pyplot as plt  
  
plt.title('Classification Problem')  
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 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()

З отриманих результатів можна визначити, що зменшення кількості нейронів в мережі Кохонена призводить до збільшення помилки. Це свідчить про те, що менша кількість нейронів призводить до меншої точності або ефективності моделі.

Наприклад, у випадку з 5 нейронами помилка становила 40.23870190392719, тоді як у випадку з 4 нейронами, помилка складала 37.930379364634646. Це підтверджує, що мережа з більшою кількістю нейронів демонструє більшу точність та меншу помилку.

Важливо також враховувати, що невірний вибір кількості нейронів може призвести до недоучення або перенавчання моделі. Занадто велика кількість нейронів може призвести до перенавчання і погіршення загальної здатності до узагальнення. З іншого боку, занадто мала кількість нейронів може зробити модель недостатньо складною для розв'язання складних завдань.

Посилання на Git:

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