ЛАБОРАТОРНА РОБОТА № 5

***Тема: «***РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ*»*

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

**Хід роботи**

Посилання на програмнй код на Github:

<https://github.com/dengaevsky/Labs_AI/tree/main/lab5>

***Завдання 1.* Створити простий нейрон**

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])  
bias = 4 # b = 4  
n = Neuron(weights, bias)  
  
x = np.array([2, 3])  
print(n.feedforward(x))

Результат виконання програми:

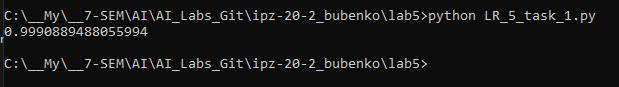


Рис. 1. Результат виконання програми

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

Реалізуємо пряме розповсюдження (feedforward) ваг по відношенню до нейронної мережі.

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])  
bias = 4 # b = 4  
n = Neuron(weights, bias)  
  
x = np.array([2, 3])  
print(n.feedforward(x))  
  
  
class BubenkoNeuralNetwork:  
  
 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 = BubenkoNeuralNetwork()  
x = np.array([2, 3])  
print(network.feedforward(x)) # 0.7216325609518421

Результат виконання програми:

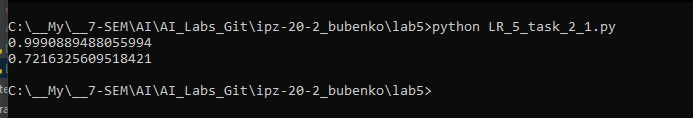


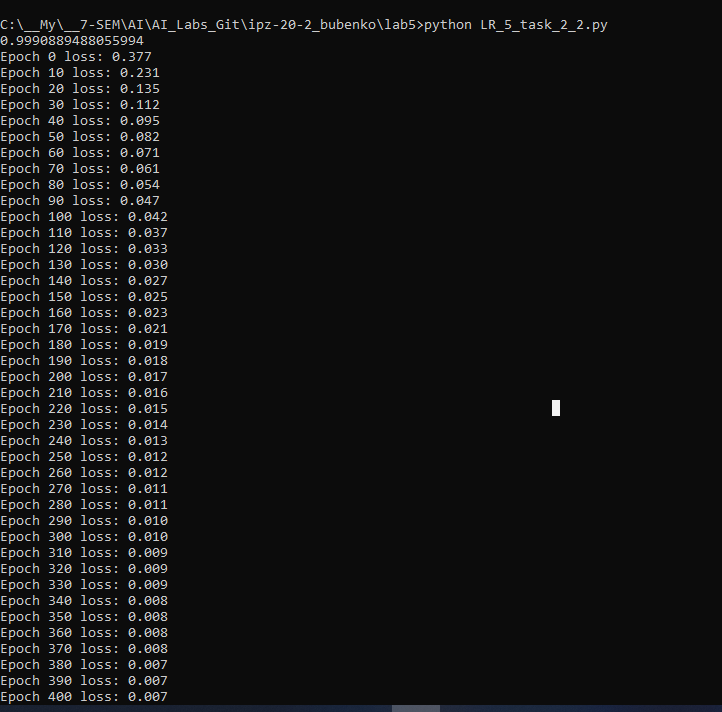
Рис. 2.1. Результат виконання програми

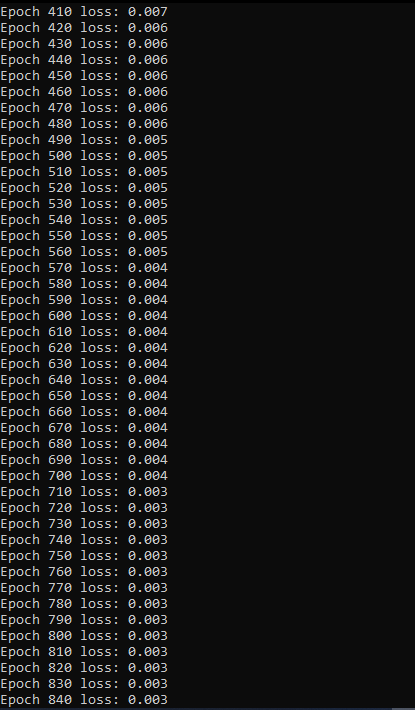
Тепер створимо нейронну мережу для передбачення статі.

Лістинг програми:

import numpy as np  
  
from LR\_5\_task\_1 import sigmoid  
  
  
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 BubenkoNeuralNetwork:  
 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):  
 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  
  
 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\_\_":  
 # 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 = BubenkoNeuralNetwork()  
 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 дюймів  
 print("Emily: %.3f" % network.feedforward(emily)) # +-0.966 - F  
 print("Frank: %.3f" % network.feedforward(frank)) # +-0.038 - M

Результат виконання програми:





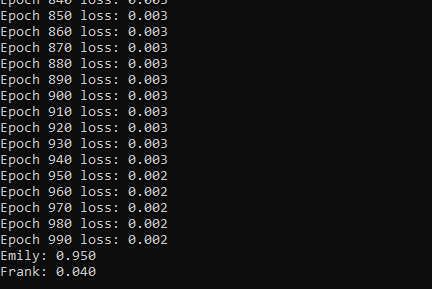


Рис. 2.2. – 2.4. Результат виконання програми

***Висновки щодо створення простої нейронної мережі***

Функція активації в нейронних мережах використовується для надання нелінійності та створення складних зв'язків між вхідними та вихідними даними. Найчастіше використовується функція сигмоїди. Її призначення полягає в перетворенні вагованої суми вхідних даних та зсуву в вихідне значення, що лежить в діапазоні від 0 до 1. Ця нелінійність дозволяє нейронам ефективно моделювати складні відношення між вхідними та вихідними даними.

Наша нейронна мережа прямого поширення успішно вирішує класифікаційні завдання, які вимагають розділення об'єктів на класи 1 чи 0 на основі вхідних даних. Вона навчилася адаптувати свої ваги та зсуви для точного прогнозування, наближеного до правильних відповідей. Однак нейронні мережі прямого поширення знайшли широке застосування в сферах комп'ютерного бачення та розпізнавання мовлення, де складна класифікація цільових класів виявляється складним завданням. Ці типи мереж демонструють ефективну роботу навіть з зашумленими даними, що свідчить про їхню потужність та гнучкість в аналізі складних наборів інформації.

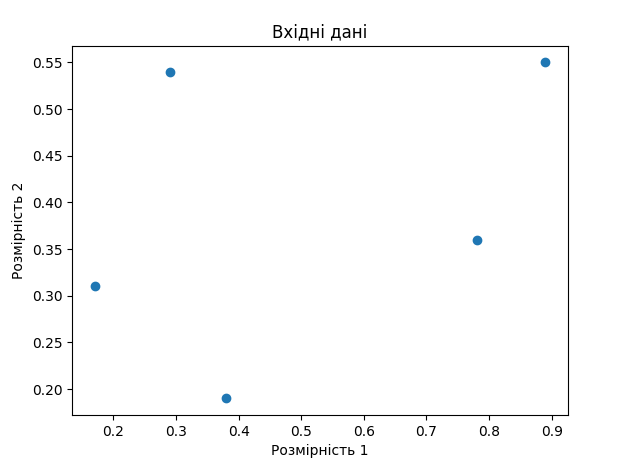
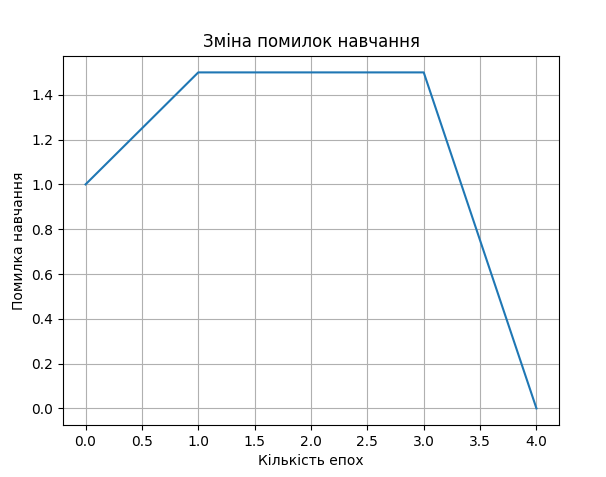
***Завдання 3.* Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab**

Розробіть класифікатор на основі перцептрону з використанням бібліотеки NeuroLab для файлу даних data\_perceptron.txt.

Лістнг програми:

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 помилок навчання')  
plt.grid()  
plt.show()

Результат виконання програми:

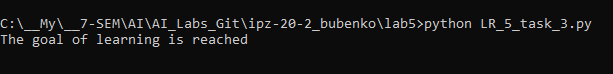


Рис. 3.1 - 3.3. Результати виконання програми

***Висновки щодо рисунку 3.2.***

На другому графіку представлений процес навчання, який оцінюється за допомогою метрики помилок. На осі X кількість епох, а на осі Y – значення помилки навчання на кожній епосі. Під час першої епохи помилки коливалися від 1.0 до 1.5. Протягом двох наступних епох вони залишалися на рівні близько 1.5. Однак у четвертій епохі ми спостерігали поступове зменшення помилок. Це сталося через те, що ми успішно навчили перцептрон, використовуючи тренувальні дані.

***Завдання 4.* Побудова одношарової нейронної мережі.**

Створіть одношарову нейронну мережу, що складається з незалежних нейронів, для вхідного файлу data\_simple\_nn.txt.

Лістинг програми:

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 помилок 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 програми:

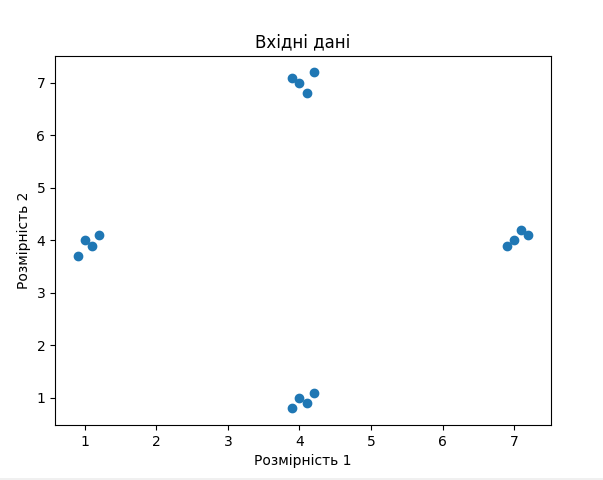


Рис. 4.1. Графік вхідних даних

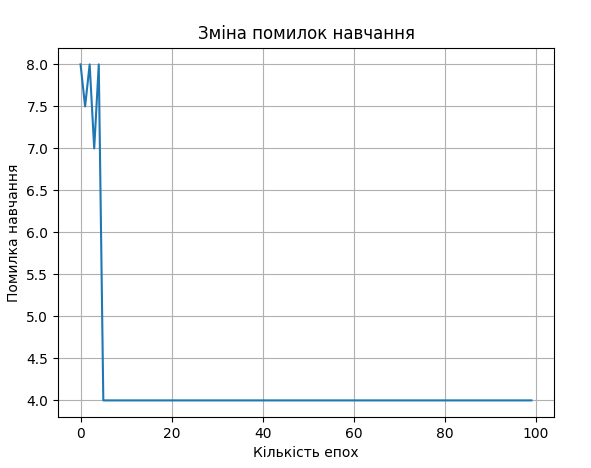


Рис. 4.2. Графік зміни помилок

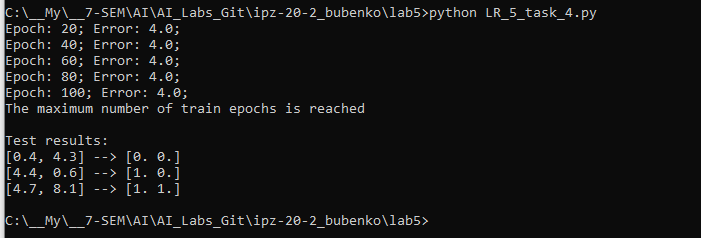


Рис. 4.3. Результат виконаня програми в консолі

***Висновки щодо побудови одношарової нейронної мережі***

На рисунку 4.1. зображено вхідні дані двох розмірностей.

На рисинку 4.2. зображено зміну помилок навчання моделі протягом епох навчання. Як бачимо помилка навчання 4.0 залишається незмінною (на 20, 40, 60, 80, 100 епохах), що вказує на те, що нейромережа досягла мінімуму втрат для навчального набору даних.

На рисунку 4.3. зображено процес навчання моделі. Модель досягла помилки 4.0. Модель про тестовано на трьох точках, а результати виведено у форматі векторів. Тобто точку [4.7, 8.1] модель віднесла до класу [1. 1.]

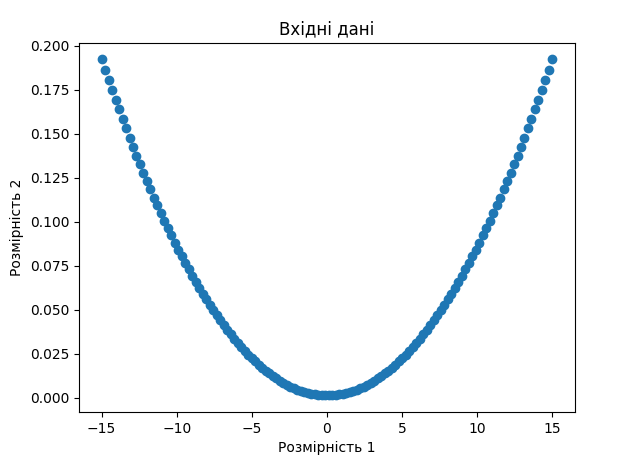
***Завдання 5.* Побудова багатошарової нейронної мережі**

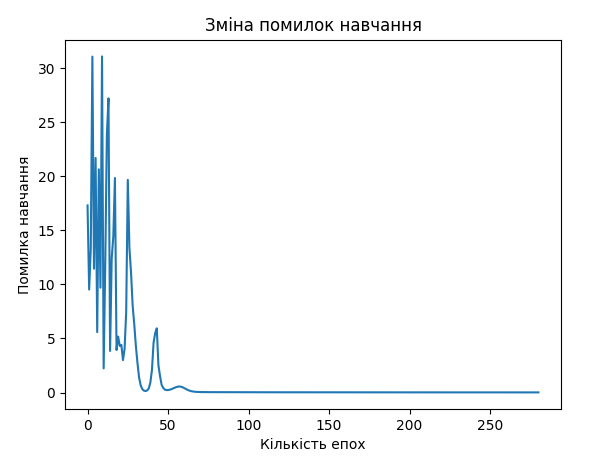
Побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних 

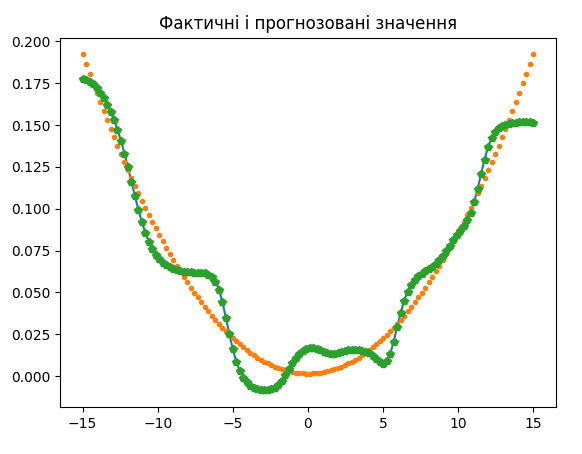
Лістинг програми:

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 помилок 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 і прогнозовані значення')  
plt.show()

Результат виконання програми:







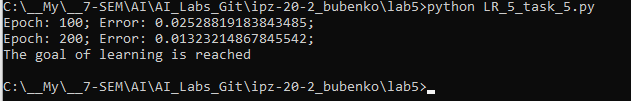


Рис. 5.1. – 5.4. Результат виконання програми

***Висновки щодо побудови багатошарової нейронної мережі***

На рисунку 5.4. зображено процес навчання багатошарової нейронної мережи. Як бачимо, помилка зменшувалась і досягла значення 0,0132. Цілі навчання були досягнуті після 200 епох, коли помилка досягла значення менше за 0.01. Після цього навчання завершилось.

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

По аналогії з попереднім завданням, побудуйте багатошарову нейронну мережу, що виконує задачу регресії для тестових даних вашого варіанту.

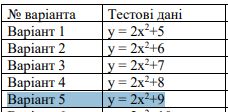
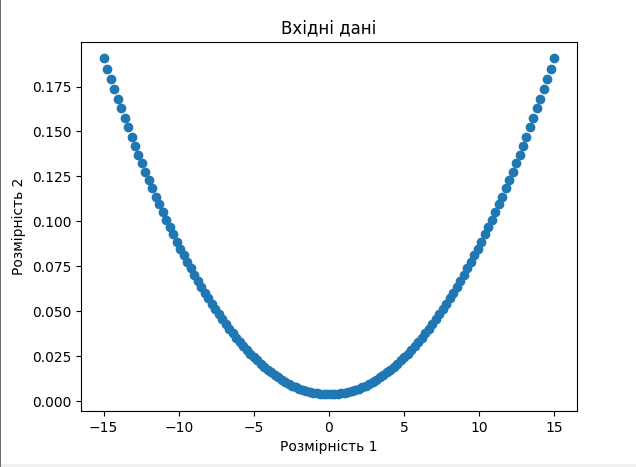
 

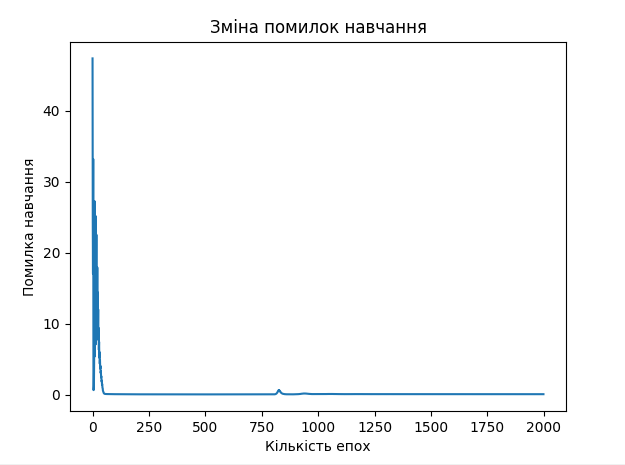
Рис. 6.1. - 6.2. Тестові дані та параметри багатошарової мережі

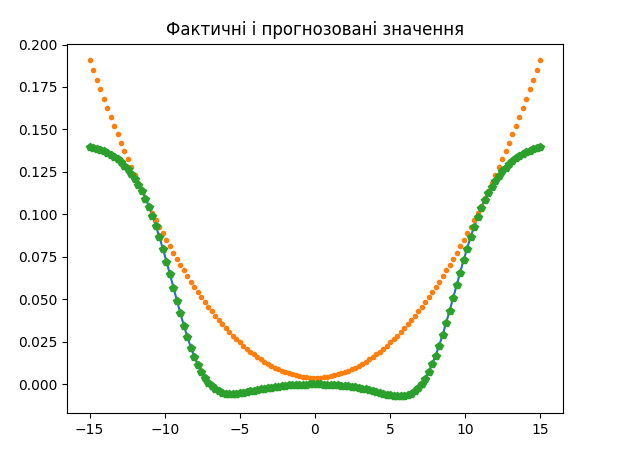
Лістинг програми:

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 = 2 \* np.square(x) + 9  
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  
nn = nl.net.newff([[min\_val, max\_val]], [2, 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 помилок 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 і прогнозовані значення')  
plt.show()

Результат виконання програми:







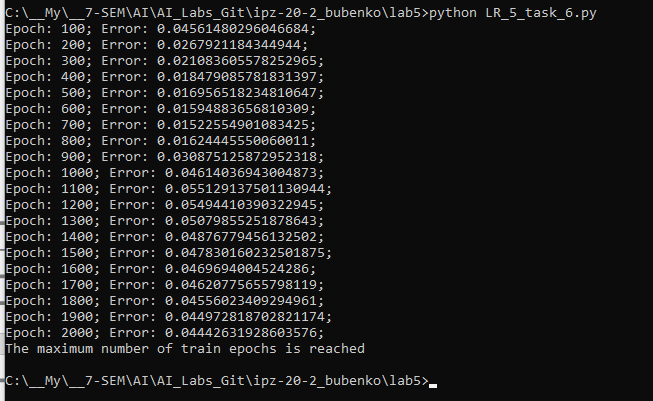


Рис. 6.1 – 6.4. Результат виконання програми

***Висновки щодо побудови багатошарової нейронної мережі для свого варіанту***

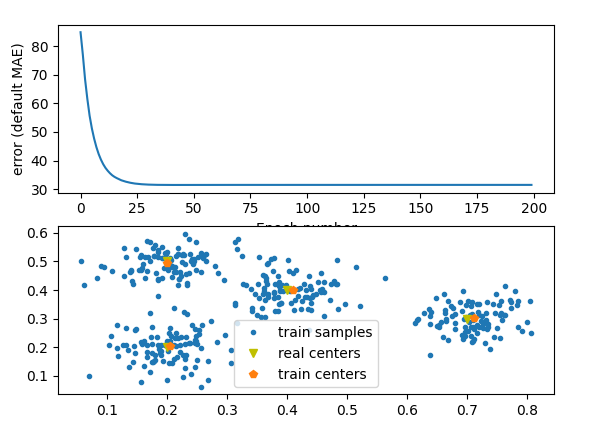
На рисунку 6.4. зображено процес навчання багатошарової нейронної мережі. Як бачимо, помилка, то збільшується, то зменшується, але не наближається до цільового значення 0,01 навіть після 2000 епох навчання. Також при тестуванні для 20000 епох цільового значення для помилки не було досягнуто. Тому можна дійти висновку, що побудована нейронна мережа не здатна зменшити поилку і навчання не просувається вперед після перших 800 епох.

***Завдання 7.* Побудова нейронної мережі на основі карти Кохонена, що самоорганізується**

Лістинг програми:

import numpy as np  
import neurolab as nl  
import numpy.random as rand  
import pylab as pl  
  
# Генерація даних  
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)  
  
# Створення мережі з 2 вхідними нейронами і 4 нейронами в одному шарі  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)  
  
# Тренування мережі  
error = net.train(inp, epochs=200, show=100)  
  
# Побудова графіків  
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 програми:



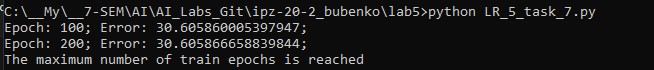


Рис. 7.1. – 7.2. Результат виконання програми

***Висновки щодо побудови нейронної мережі на основі карти Кохонена, що самоорганізується***

MAE (Mean Absolute Error) — це середня абсолютна похибка. Середньою абсолютною похибкою називають середнє абсолютне значення різниці між фактичними та прогнозованими значеннями. Ця метрика використовується для вимірювання точності в задачах регресії.

Як бачимо з рисунку 7.2 після 200 епох навчання помилка не змінюється та залишаєтсья на рівні 30,60586. Аналогічні результат для 2000 епох. Тому можна зробити висновок, що нейронна мережа досягла свого максимуму навчання і покращення результатів не є можливим.

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

Проведіть дослідження по аналогії з попереднім завданням.

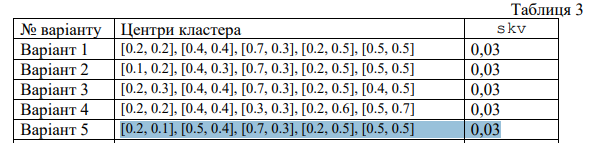


Рис. 8.1. Параметри за варіантом

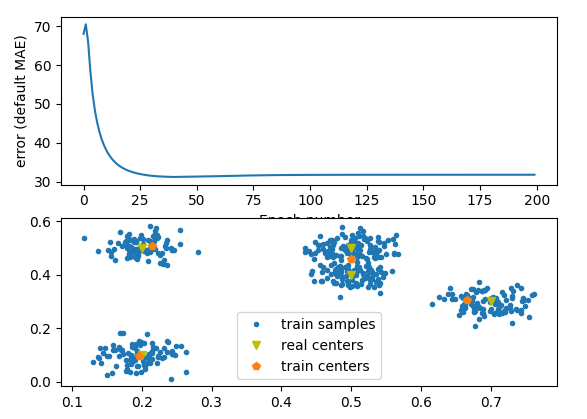
***Створіть нейронну мережу Кохонена з 2 входами та 4 нейронами***

Лістинг програми:

**import** numpy **as** np  
**import** neurolab **as** nl  
**import** numpy.random **as** rand  
**import** pylab **as** pl

# Генерація даних з новими параметрами  
skv = 0.03  
centr = np.array([[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]])  
  
# Додано нові центри кластерів  
rand\_norm = skv \* rand.randn(100, 4, 2)  
inp = np.array([centr + r **for** r **in** rand\_norm])  
inp.shape = (100 \* 5, 2)  
rand.shuffle(inp)  
  
# Створення мережі з 2 вхідними нейронами і 4 нейронами в одному шарі  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)  
  
# Тренування мережі  
error = net.train(inp, epochs=200, show=100)  
  
# Побудова графіків  
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 програми:



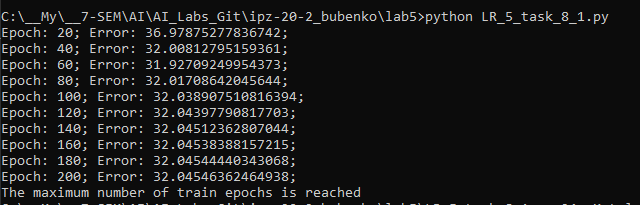


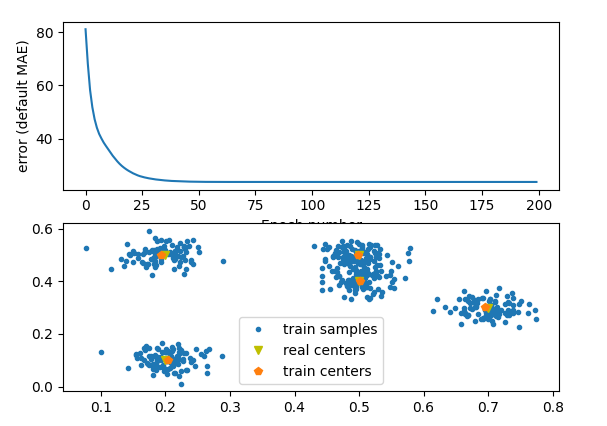
Рис. 8.1 – 8.2. Результат виконання програми

***Створіть нейронну мережу Кохонена з 2 входами та 4 нейронами***

Лістинг програми:

**import** numpy **as** np  
**import** neurolab **as** nl  
**import** numpy.random **as** rand  
**import** pylab **as** pl  
  
# Генерація даних з новими параметрами  
skv = 0.03  
centr = np.array([[0.2, 0.1], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5]])  
  
# Додано нові центри кластерів  
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)  
  
# Створення мережі з 2 вхідними нейронами і 5 нейронами в одному шарі  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 5)  
  
# Тренування мережі  
error = net.train(inp, epochs=200, show=100)  
  
# Побудова графіків  
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 програми:



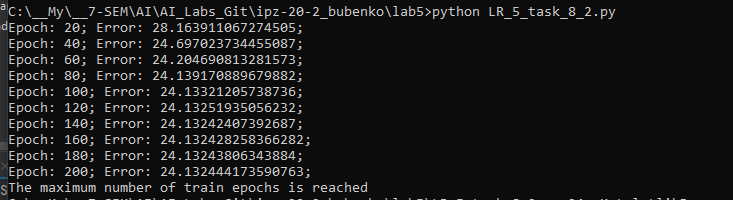


Рис. 8.3. – 8.4. Результат виконання програми

***Висновки щодо отриманих результатів***

Найголовніший висновок полягає в тому, що мережа з 5 нейронами в шарі Кохонена має меншу похибку порівняно з мережею з 4 нейронами. А саме 24,13 проти 32,05.

Невірний вибір кількості нейронів у нейронній мережі Кохонена може призвести до неправильної кластеризації даних. Якщо нейронів надто мало, то мережа може недостатньо точно відобразити структуру даних, збільшуючи помилку. Натомість, якщо нейронів забагато, це може привести до зайвого підрозділення кластерів та збільшення внутрішньокластерної варіабельності. Оптимальний результат досягається, коли кількість нейронів відповідає кількості кластерів у даних.

Зменшення розкиду вхідних даних поліпшує точність кластеризації. У восьмому завданні з меншим skv = 0.03, мережа ефективніше відображає розташування кластерів. Отже, розкид вхідних даних суттєво впливає на точність кластеризації.

Top of Form

***Висновок:*** у ході виконання лабораторної роботи я, використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовувати прості нейронні мережі.