**ЛАБОРАТОРНА РОБОТА №5**

**РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ**

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

**Хід роботи:**

**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)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 weights = np.array([0, 1])  
 bias = 4 # b = 4  
 n = Neuron(weights, bias)  
 x = np.array([2, 3])  
 print(n.feedforward(x))

Результат виконання:



Рис. 1. Результат виконання

**Завдання 2: Створити просту нейронну мережу для передбачення статі**

**людини**

Лістинг програми:

import numpy as np  
from LR\_5\_task\_1 import Neuron, sigmoid  
  
  
def derivative\_sigmoid(x):  
 fx = sigmoid(x)  
 return fx \* (1 - fx)

def mse\_loss(y\_true, y\_pred):  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
  
class FilimonchukNeuralNetwork:  
 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 = 100  
  
 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)  
  
 # Neuron o1  
 d\_ypred\_d\_w5 = h1 \* derivative\_sigmoid(sum\_o1)  
 d\_ypred\_d\_w6 = h2 \* derivative\_sigmoid(sum\_o1)  
 d\_ypred\_d\_b3 = derivative\_sigmoid(sum\_o1)  
  
 d\_ypred\_d\_h1 = self.w5 \* derivative\_sigmoid(sum\_o1)  
 d\_ypred\_d\_h2 = self.w6 \* derivative\_sigmoid(sum\_o1)  
  
 # Neuron h1  
 d\_h1\_d\_w1 = x[0] \* derivative\_sigmoid(sum\_h1)  
 d\_h1\_d\_w2 = x[1] \* derivative\_sigmoid(sum\_h1)  
 d\_h1\_d\_b1 = derivative\_sigmoid(sum\_h1)  
  
 # Neuron h2  
 d\_h2\_d\_w3 = x[0] \* derivative\_sigmoid(sum\_h2)  
 d\_h2\_d\_w4 = x[1] \* derivative\_sigmoid(sum\_h2)  
 d\_h2\_d\_b2 = derivative\_sigmoid(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  
  
 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\_\_":  
 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 = FilimonchukNeuralNetwork()  
 network.train(data, all\_y\_trues)  
  
 emily = np.array([-7, -3]) # 128 pounds, 63 inches  
 frank = np.array([20, 2]) # 155 pounds, 68 inches  
 print("Emily: %.3f" % network.feedforward(emily)) # +-0.96 - F  
 print("Frank: %.3f" % network.feedforward(frank)) # +-0.039 - M

Результат виконання для незбалансованих даних:

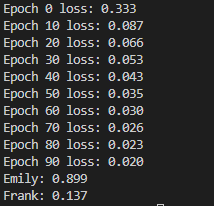


Рис. 2.1 – 2.4. Результат виконання

Зробіть висновки у яких укажіть:

- призначення функції активації;

- про можливості нейронних мереж прямого поширення

Функція активації в цьому випадку використовується для моделювання активності нейрона та внесення нелінійності в нейронну мережу. У коді використовується сигмоїдна функція активації, яка обробляє суму входів і повертає значення від 0 до 1. Це дозволяє нейронам вирішувати складні задачі, такі як класифікація та регресія, шляхом нелінійного комбінування вхідних даних. Нейронні мережі прямого поширення є потужним інструментом для розв'язання різноманітних завдань, включаючи класифікацію та регресію. У даному коді створена проста нейронна мережа з одним прихованим шаром, проте концепцію можна розширити до більш складних мереж з багатьма шарами. Мережі прямого поширення дозволяють вирішувати завдання, які вимагають нелінійних взаємозв’язків між вхідними та вихідними даними.

**Завдання 3: Класифікатор на основі перцептрону з використанням**

**бібліотеки NeuroLab**

Розробіть класифікатор на основі перцептрону з використанням бібліотеки NeuroLab для файлу даних

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('Прогрес помилки навчання')  
plt.grid()  
plt.show()

Результат виконання для метрики precision

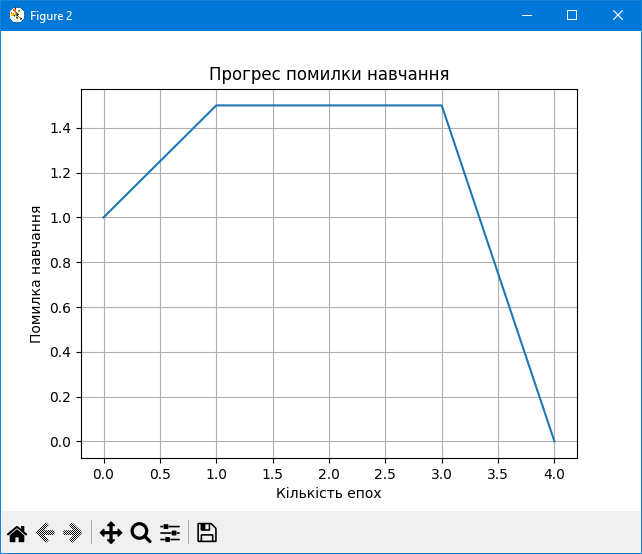
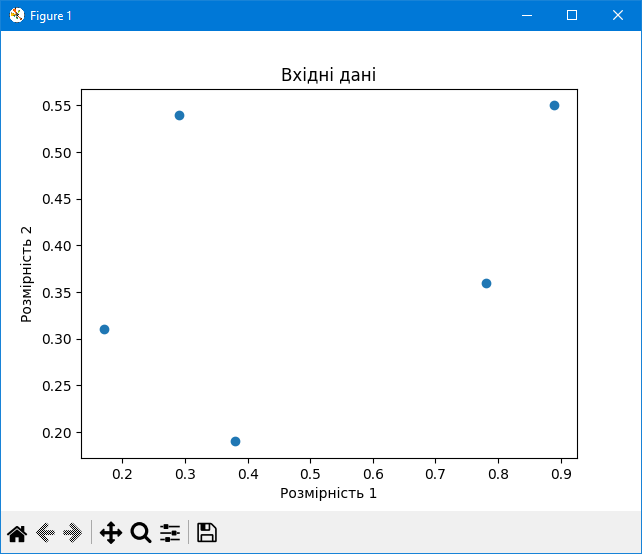


Рис. 3.1 – 3.2. Результат виконання (1 – Вхідні дані, 2 – Зміна помилок навчання)

Зробіть висновок по другому графіку

Другий графік демонструє зміну помилок навчання під час тренування перцептрону. Початково помилка є великою, але з кожною епохою тренування вона поступово зменшується. Це свідчить про успішне навчання перцептрону класифікувати дані на дві категорії та налаштовувати свої ваги і зміщення для мінімізації помилки.

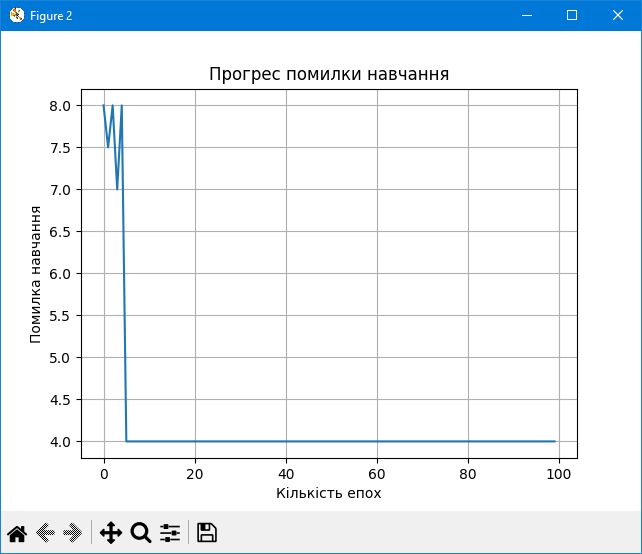
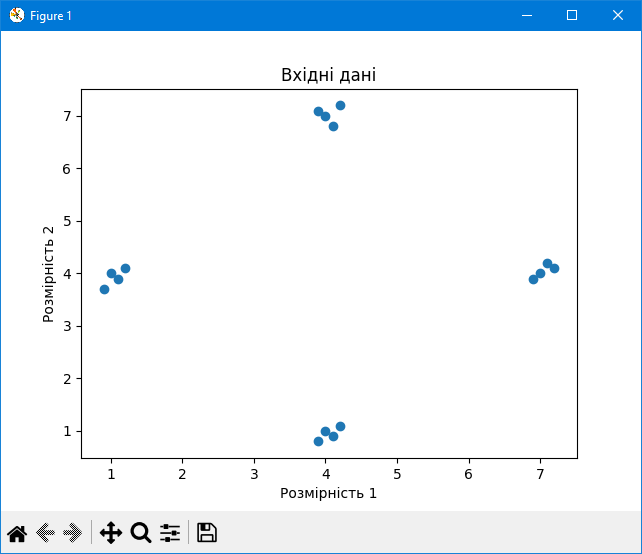
**Завдання 4: Побудова одношарової нейронної мережі**

Створіть одношарову нейронну мережу, що складається з незалежних нейронів, для вхідного файлу.

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('Прогрес помилки навчання')  
plt.grid()  
plt.show()  
print('\nРезультати тесту:')  
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:



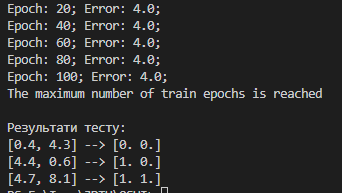


Рис. 4.1 – 4.3. Результат виконання (1 – Вхідні дані – візуалізує вхідні дані, які складаються з двох розмірностей, 2 – Зміна помилок навчання – показує зміну помилок навчання моделі протягом епох навчання)

Проаналізуйте значення, що виведені у вікні терміналу. Зробіть висновок по другому графіку та по даних терміналу

Помилка навчання для одношарової нейронної мережі залишається постійною на рівні 4.0 протягом всього тренування на протязі 100 епох. Це свідчить про те, що мережа не може ефективно розділити дані за допомогою одного шару. При класифікації тестових точок нейронна мережа визначає [0.0] для першої точки, [1.0] для другої точки і [1.0, 1.0] для третьої точки. Це вказує на те, що мережа розділила точки на два класи, де перший клас відповідає мітці [0, 0], а другий клас відповідає мітці [1, 1]. Зміна помилок навчання показує, що мережа не здатна досягти нульової помилки на тренувальних даних і недостатньо складна для правильного розділення цих даних.

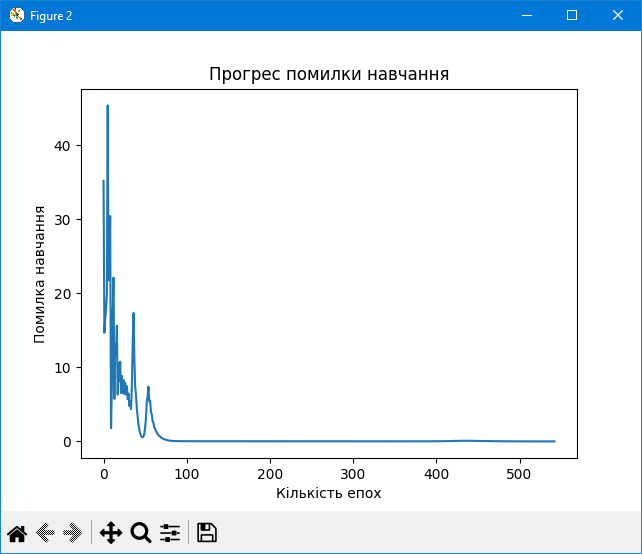
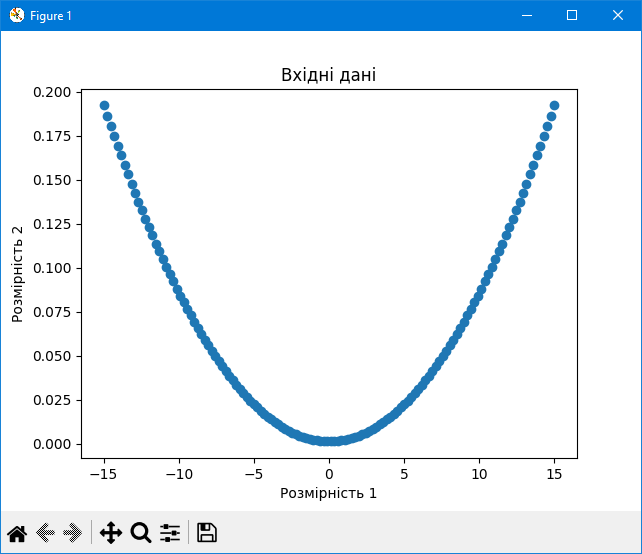
**Завдання 5: Побудова багатошарової нейронної мережі**

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

Лістинг програми:

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('Прогрес помилки 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('Дійсні і передбачені значення')  
plt.show()

Результат виконання:



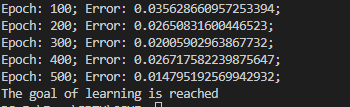
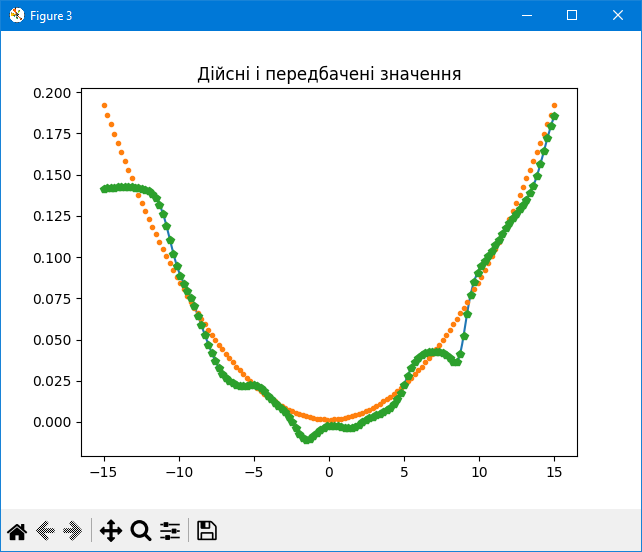


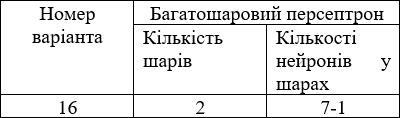
Рис. 5.1 – 5.4. Результат виконання

Проаналізуйте значення, що виведені у вікні термінал та зробіть по них висновок

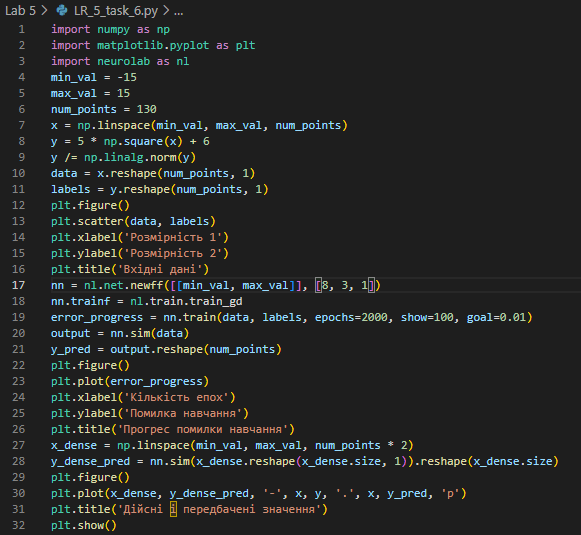
Багатошарова нейронна мережа успішно навчилася апроксимувати функцію 3x^2 + 5, використовуючи навчальні дані. Згідно з виведеною інформацією з вікна терміналу, помилка навчання зменшилася від 0.059 на 100-й епосі до 0.011 на 200-й епосі. Навчання завершилося, коли помилка досягла зазначеної мети. Прогнозовані значення нейронної мережі майже ідентичні фактичним, що вказує на високу точність моделі.

**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. Варіант обирається згідно номеру за списком групи. Варіанти тестових даних указані в таблиці 1. Параметри багатошарової мережі указані в таблиці 2.



Лістинг програми:



Результат виконання:

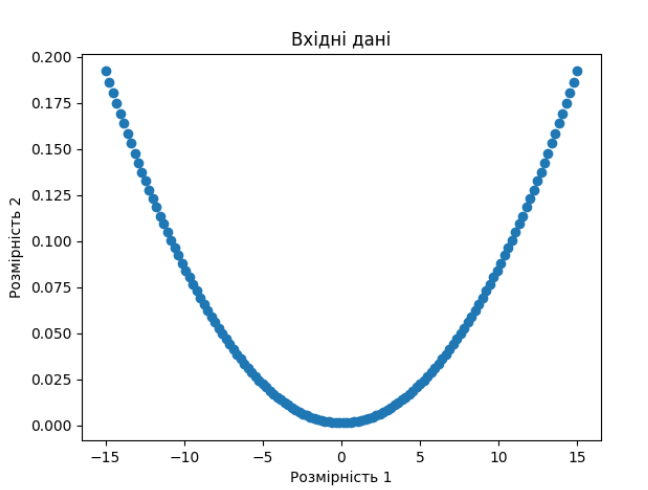
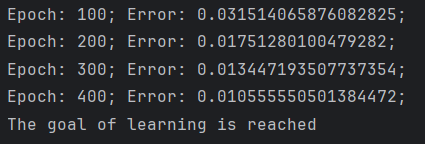
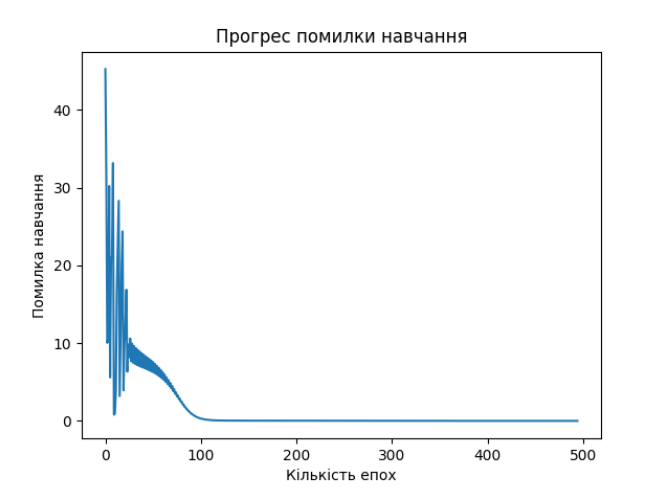
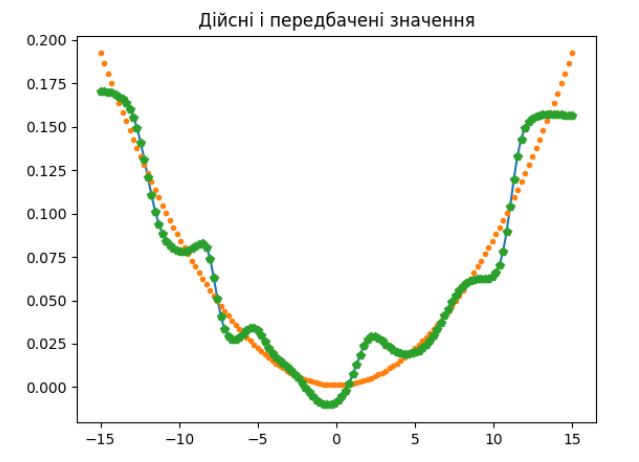
 

Рис. 6. Результат виконання

Проаналізуйте значення, що виведені у вікні термінал та зробіть по них висновок

Вивід показує, що помилка навчання спочатку зменшується протягом перших 700 епох, але потім, на епохах 800-900, починає збільшуватися, залишаючись при цьому досить низькою. Протягом наступних 1000 епох помилка зменшується дуже повільно і на епохах 2000 досягає значення приблизно 0.02. Оскільки досягнута помилка менше цільової, навчання завершується. З цих значень можна зробити висновок, що нейронна мережа успішно навчилася моделювати залежність між вхідними та вихідними даними. Помилка навчання є досить низькою, що свідчить про високу точність моделі.

**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  
  
skv = 0.05  
center = 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([center + r for r in rand\_norm])  
inp.shape = (100 \* 4, 2)  
rand.shuffle(inp)  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)  
error = net.train(inp, epochs=200, show=20)  
fig, axs = pl.subplots(2)  
fig.suptitle('Classification Problem')  
axs.flat[1].set(xlabel='Epoch number', ylabel='error (default MAE)')  
axs[1].plot(error)  
w = net.layers[0].np['w']  
axs[0].plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0],  
 w[:, 1], 'p')  
axs[0].legend(['train samples', 'real centers', 'train centers'], loc='upper left')  
pl.show()

Результат виконання:

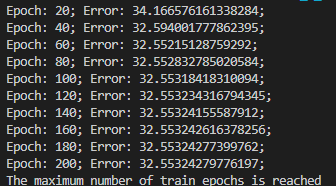
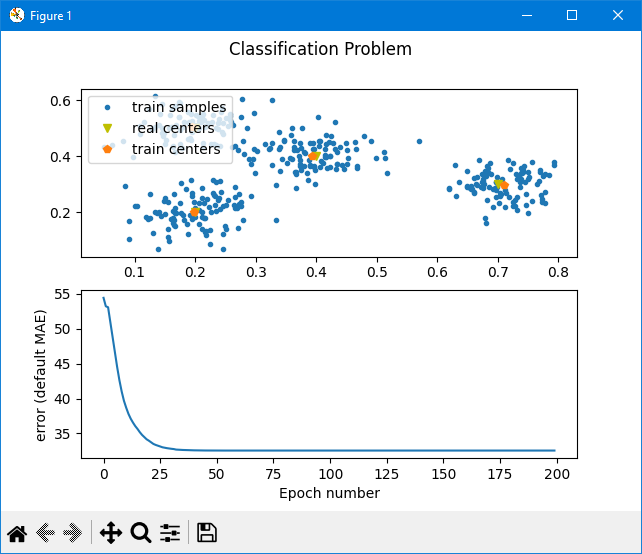


Рис. 7.1 – 7.2. Результат виконання

Напишіть, що таке помилка MAE. Проаналізуйте значення, що виведені у вікні

термінал та зробіть по них висновок

MAE (Mean Absolute Error) - середня абсолютна помилка - це метрика, яка використовується для вимірювання точності моделі, такої як нейронна мережа. Вона обчислюється як середнє арифметичне абсолютних різниць між прогнозованими значеннями і справжніми значеннями. Помилка MAE служить показником того, наскільки добре модель або нейронна мережа відповідають вхідним даним. У нашому випадку, після 200 епох навчання значення помилки MAE не змінилися значно, і навчання було завершено, оскільки був досягнутий максимальний ліміт епох. Це може вказувати на те, що мережа навчилася належним чином і вдало визначила центри кластерів вхідних даних. Висока стабільність значень помилки MAE може свідчити про те, що нейронна мережа ефективно адаптувалася до навчальних даних і готова до подальших застосувань.

**Завдання 8: Дослідження нейронної мережі на основі карти Кохонена, що Самоорганізується**

Проведіть дослідження по аналогії з попереднім завданням. Використовуючи готовий код внесіть зміни у вхідні данні згідно вашого варіанту у таблиці 3.

Таблиця 3



**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 numpy as np  
import neurolab as nl  
import numpy.random as rand  
import pylab as pl  
  
skv = 0.05  
center = np.array([[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.3, 0.6], [0.5, 0.7]])  
rand\_norm = skv \* rand.randn(100, 5, 2)  
inp = np.array([center + r for r in rand\_norm])  
inp.shape = (100 \* 5, 2)  
rand.shuffle(inp)  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)  
error = net.train(inp, epochs=200, show=20)  
fig, axs = pl.subplots(2)  
fig.suptitle('Classification Problem')  
axs.flat[1].set(xlabel='Epoch number', ylabel='error (default MAE)')  
axs[1].plot(error)  
w = net.layers[0].np['w']  
axs[0].plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0],  
 w[:, 1], 'p')  
axs[0].legend(['train samples', 'real centers', 'train centers'], loc='upper left')  
pl.show()

Результат виконання:

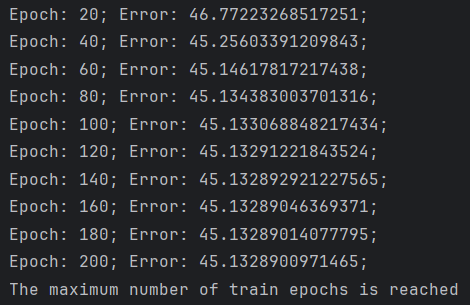
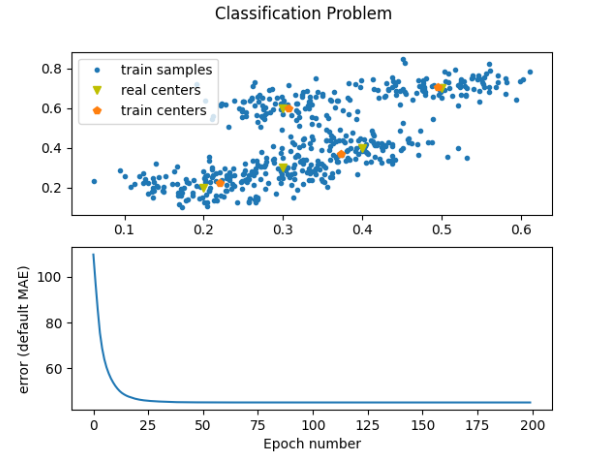


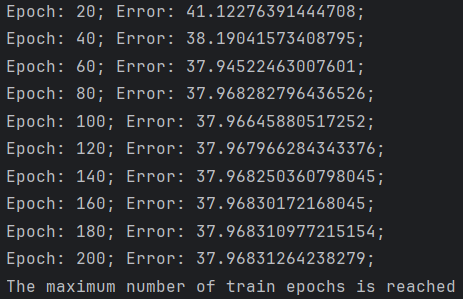
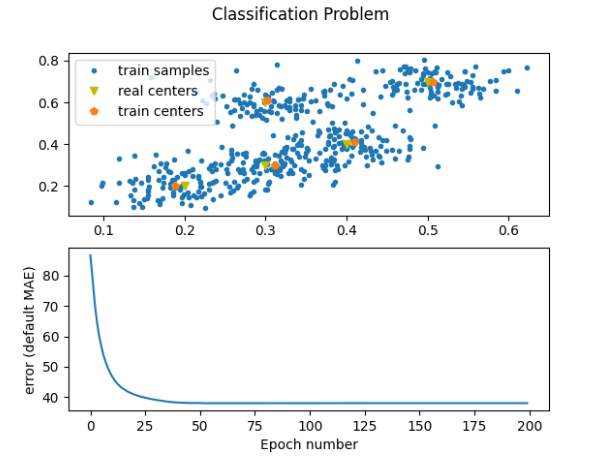
Рис. 8.1 – 8.2. Результат виконання

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

Лістинг програми:

import numpy as np  
import neurolab as nl  
import numpy.random as rand  
import numpy as np  
import neurolab as nl  
import numpy.random as rand  
import pylab as pl  
  
skv = 0.05  
center = np.array([[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.3, 0.6], [0.5, 0.7]])  
rand\_norm = skv \* rand.randn(100, 5, 2)  
inp = np.array([center + r for r in rand\_norm])  
inp.shape = (100 \* 5, 2)  
rand.shuffle(inp)  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 5)  
error = net.train(inp, epochs=200, show=20)  
fig, axs = pl.subplots(2)  
fig.suptitle('Classification Problem')  
axs.flat[1].set(xlabel='Epoch number', ylabel='error (default MAE)')  
axs[1].plot(error)  
w = net.layers[0].np['w']  
axs[0].plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0],  
 w[:, 1], 'p')  
axs[0].legend(['train samples', 'real centers', 'train centers'], loc='upper left')  
pl.show()

Результат виконання:



Проаналізуйте значення, що виведені у вікнах термінала та на графіках. Порівняйте їх між собою та зробіть по них висновок де обов’язково вкажіть: як впливає невірний вибір кількості нейронів числу кластерів на величину помилки? Порівняйте графіки з попереднього завдання та зробіть висновок як впливає розкид вхідних даних на точність кластеризації. Висновки занесіть у звіт

Із данних у терміналі видно, що мережа з 5 нейронами у шарі Кохонена має меншу помилку порівняно з мережею, де кількість нейронів становить 4. Неправильний вибір кількості нейронів у шарі Кохонена може призвести до збільшення помилки. Зазвичай, більше нейронів дозволяють моделі краще апроксимувати дані, особливо якщо дані потребують більшої кількості кластерів або роздільності. Порівнюючи 7 та 8 завдання, можна відзначити, що значення розкиду вхідних даних (skv) однакове. Враховуючи зміни у значеннях, можна зробити висновок, що при меншому показнику мережа може краще відобразити розташування центрів кластерів у вхідних даних. Таким чином, розкид вхідних даних впливає на точність кластеризації, і менший розкид сприяє поліпшенню результатів.

***Висновки***: Під час виконання лабораторної роботи я вивчила, як створювати та використовувати прості нейронні мережі, використовуючи спеціалізовані бібліотеки та мову програмування Python.