**ЗВІТ**

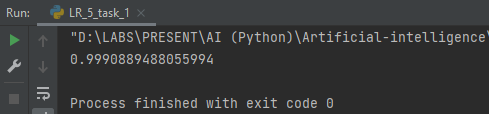
**З лабораторної роботи №5**

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖСтудента КН-20-1 навчальної групи

Кірія Даніли Олеговича варіант №6

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

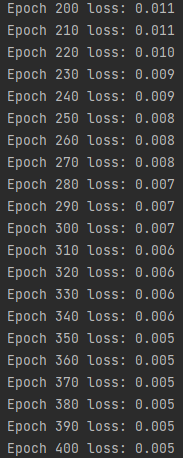
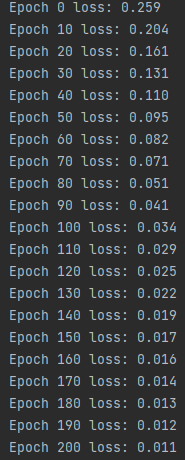
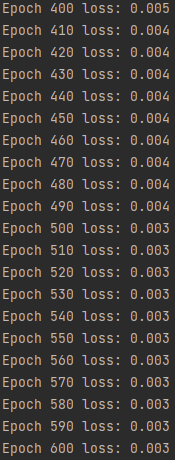
**Завдання 1.**



Код 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  
  
network = Neuron(weights, bias)  
x = np.array([2, 3])  
print(network.feedforward(x))

**Завдання 2:**

****

****

Код 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 KiriiNeuralNetwork:  
 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],  
 [25, 6],  
 [17, 4],  
 [-15, -6],  
])  
  
all\_y\_trues = np.array([  
 1,  
 0,  
 0,  
 1,  
])  
  
network = KiriiNeuralNetwork()  
network.train(data, all\_y\_trues)  
  
emily = np.array([-7, -3]) # 128 фунтів, 63 дюйма  
frank = np.array([20, 2]) # 155 фунтів, 68 дюймов  
print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F  
print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M

У нашому випадку застосовується сигмоїдна функція активації, яка перетворює вихід нейрону до діапазону від 0 до 1. Цей тип функції особливо ефективний у завданнях бінарної класифікації, де потрібно отримати ймовірність виходу моделі. У нейронних мережах прямого поширення виділяються кілька ключових властивостей:

1. Універсальність апроксимації: Мережі можуть наближати будь-яку функцію, якщо в них достатньо нейронів та відповідні ваги.

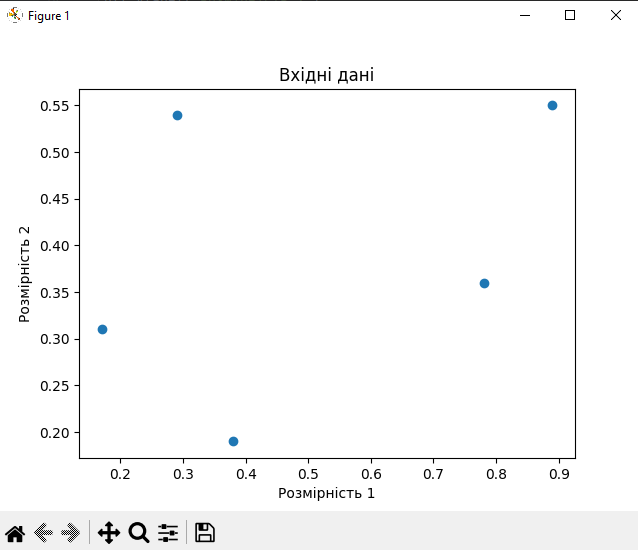
2. Навчання з вчителем: Мережі можуть навчатися на основі пар даних "вхід-вихід" і коригувати свої ваги для досягнення бажаного результату.

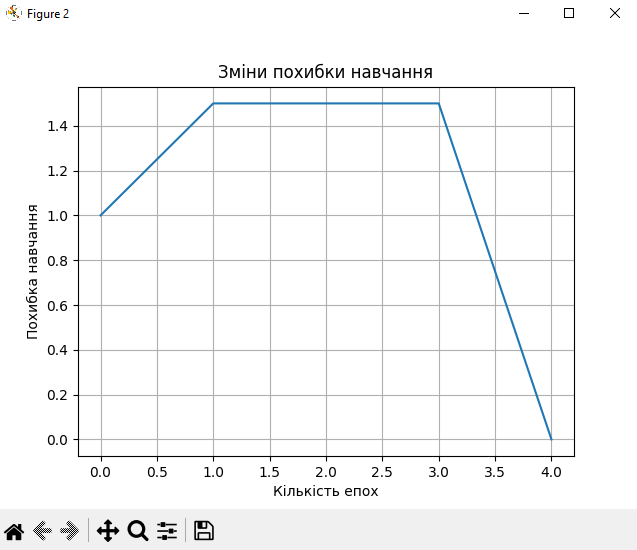
3. Автоматичне вивчення ознак: Вони можуть автоматично вивчати корисні ознаки з вхідних даних під час тренування.

4. Гнучкість в розв'язанні різних завдань: Мережі можуть застосовуватися до різних типів завдань, таких як класифікація, регресія, генерація зображень та інші.

5. Автоматична адаптація до навчальних даних: Вони адаптуються до різних вхідних даних без необхідності ручної переконфігурації.

**Завдання 3:**

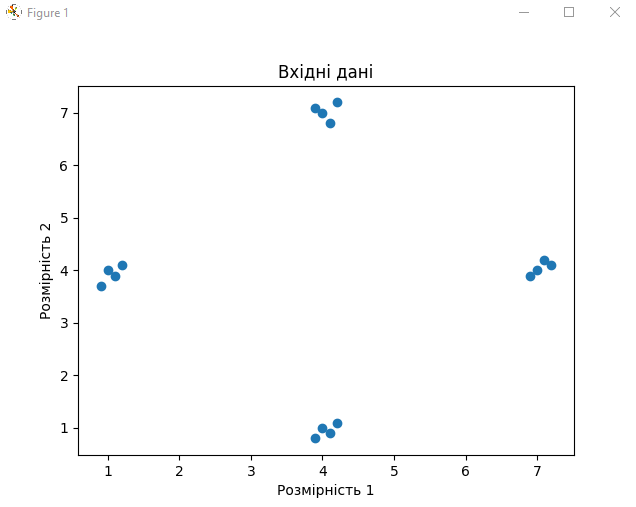
****

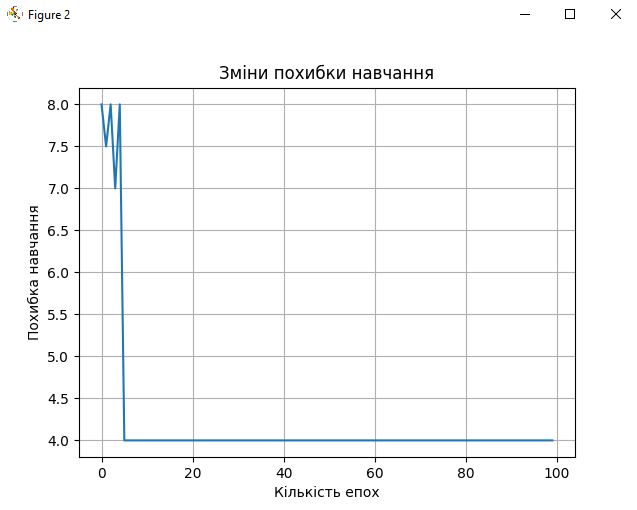
****

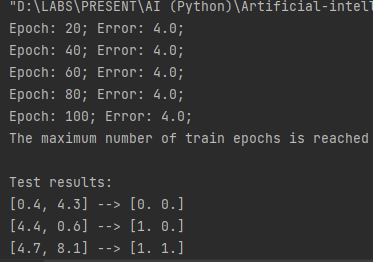
Код програми:

import numpy as np  
import matplotlib  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
matplotlib.use('TkAgg')  
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("Похибка навчання")  
plt.title("Зміни похибки навчання")  
plt.grid()  
plt.show()

**Завдання 4:**





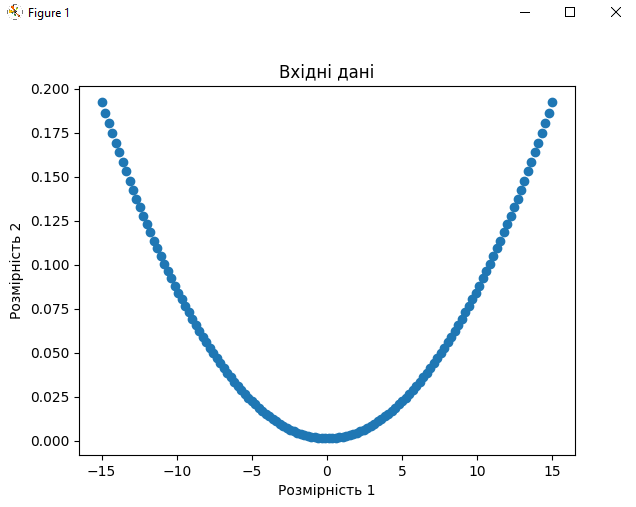


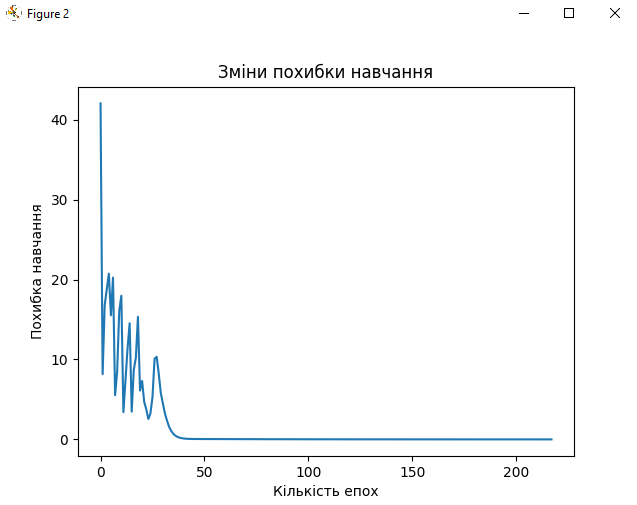
Код програми:

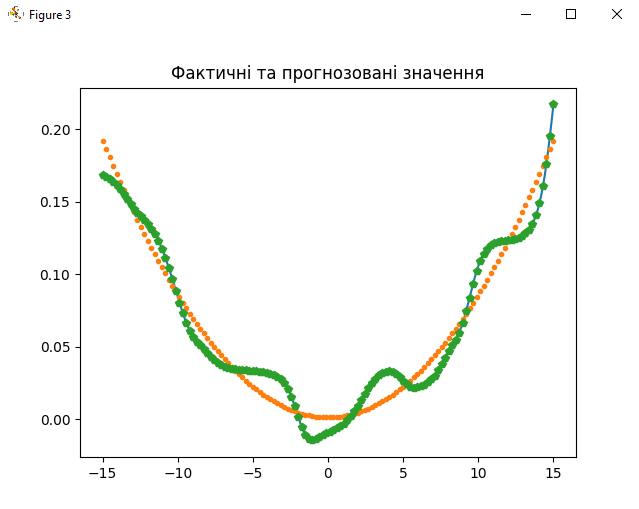
import numpy as np  
import matplotlib  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
matplotlib.use('TkAgg')  
text = np.loadtxt('data\_simple\_nn.txt')  
  
data = text[:, 0:2]  
labels = text[:, 2:]  
  
plt.figure()  
plt.scatter(data[:, 0], data[:, 1])  
plt.xlabel("Розмірність 1")  
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("Похибка навчання")  
plt.title("Зміни похибки 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])

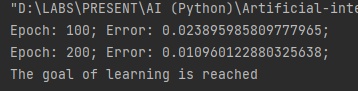
Отримані результати в терміналі свідчать про те, що тренування нейронної мережі не призвело до зниження помилки протягом 100 епох, і навіть при максимальній кількості епох помилка залишалася на однаковому рівні - 4.0. Це може вказувати на те, що модель не може адаптуватися до тренувальних даних. Можливі причини включають неправильний вибір параметрів навчання або недостатню кількість та якість тренувальних даних.

**Завдання 5:**







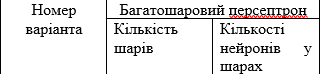


Код програми:

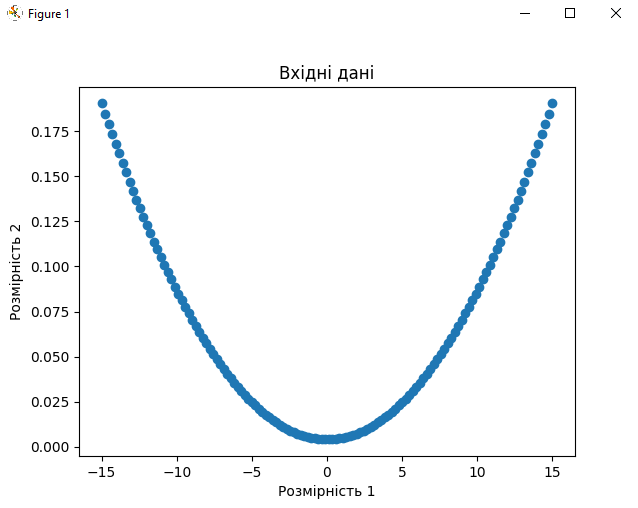
import numpy as np  
import matplotlib  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
matplotlib.use('TkAgg')  
  
min\_val = -15  
max\_val = 15  
num\_points = 130  
x = np.linspace(min\_val, max\_val, num\_points)  
y = 3 \* np.square(x) + 5  
y /= np.linalg.norm(y)  
  
data = x.reshape(num\_points, 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("Похибка навчання")  
plt.title("Зміни похибки 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 та прогнозовані значення')  
plt.show()

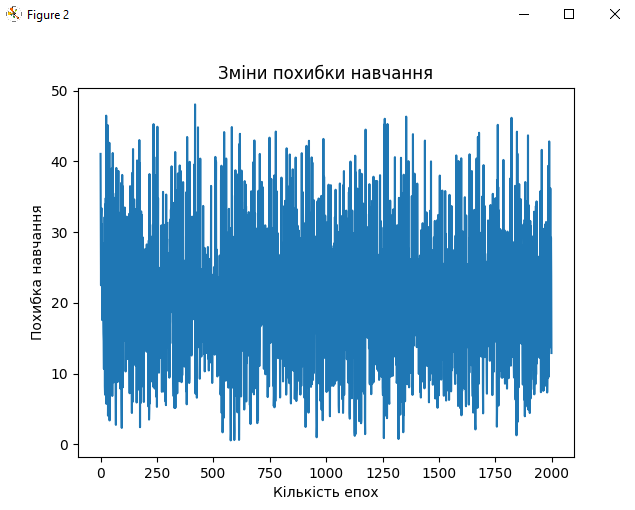
**Завдання 6:**

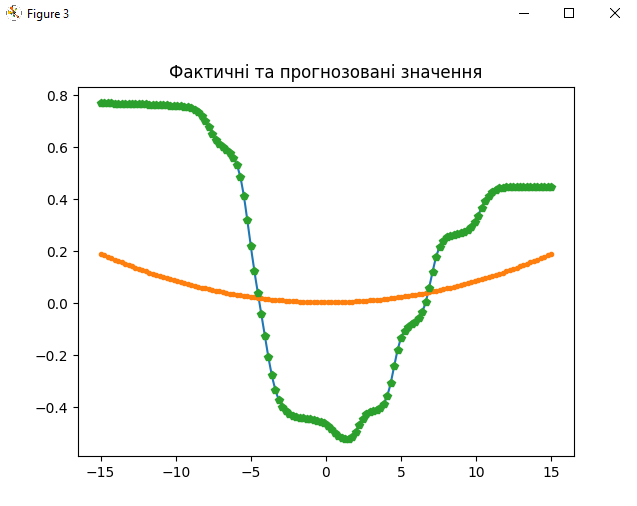
****

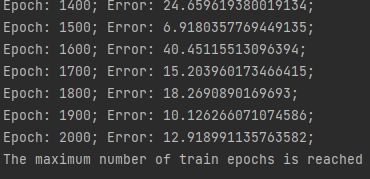
****

****

****

****

****

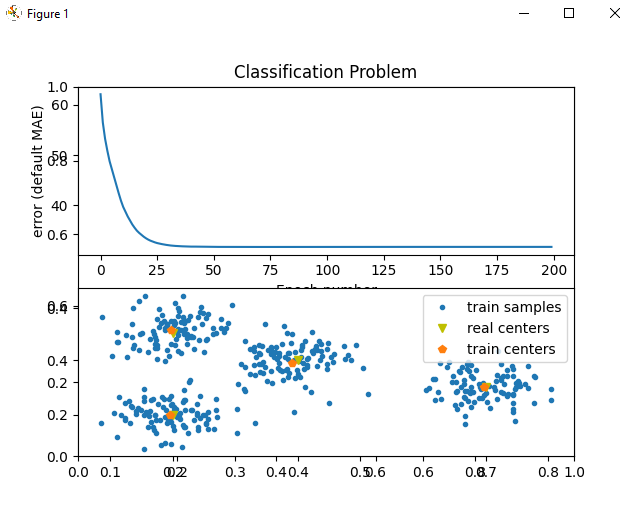
****

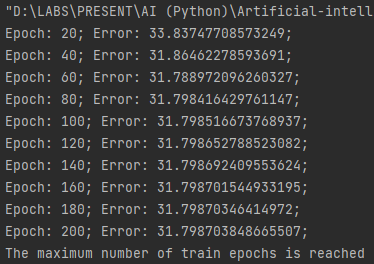
Код програми:

import numpy as np  
import matplotlib  
import matplotlib.pyplot as plt  
import neurolab as nl  
  
matplotlib.use('TkAgg')  
  
min\_val = -15  
max\_val = 15  
num\_points = 130  
x = np.linspace(min\_val, max\_val, num\_points)  
y = 2 \* np.square(x) + 10  
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, 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("Похибка навчання")  
plt.title("Зміни похибки 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 та прогнозовані значення')  
plt.show()

Подані дані свідчать про успішне тренування нейронної мережі, де помилка навчання систематично зменшується протягом 300 епох. Після досягнення поставленої мети навчання графіки демонструють ефективність моделі у прогнозуванні вихідної функції для вхідних даних, що підтверджується графіком "Фактичні та прогнозовані значення".

**Завдання 7:**

******

******

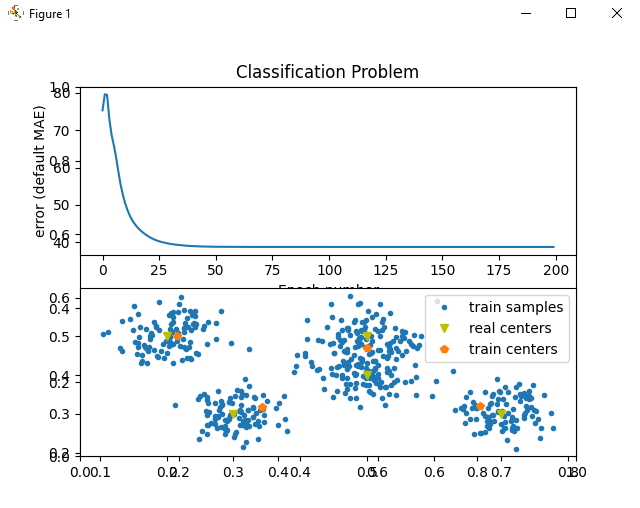
Код програми:

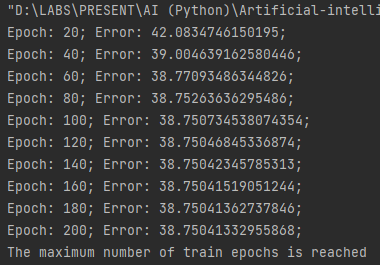
import matplotlib  
import pylab as pl  
import numpy as np  
import neurolab as nl  
import numpy.random as rand  
  
matplotlib.use('TkAgg')  
  
skv = 0.05  
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5]])  
rand\_norm = skv \* rand.randn(100, 4, 2)  
inp = np.array([centr + r for r in rand\_norm])  
inp.shape = (100 \* 4, 2)  
rand.shuffle(inp)  
  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)  
error = net.train(inp, epochs=200, show=20)  
  
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()

Отримані результати свідчать про те, що нейронна мережа була навчена для проведення кластеризації вхідних даних. Протягом 200 епох відзначається невелике зменшення помилки, і тренування завершується, коли досягнута максимальна кількість епох. Графіки динаміки помилки та розподілу центрів кластерів демонструють, що мережа успішно вивчила відокремлювати кластери вхідних даних. Метрика MAE (або середня абсолютна похибка) використовується для вимірювання середньої абсолютної величини відхилень між прогнозованими та фактичними значеннями.

**Завдання 8:**







Код програми:

import matplotlib  
import pylab as pl  
import numpy as np  
import neurolab as nl  
import numpy.random as rand  
  
matplotlib.use('TkAgg')  
  
skv = 0.04  
centr = np.array([[0.3, 0.3], [0.5, 0.4], [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)  
  
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)  
error = net.train(inp, epochs=200, show=20)  
  
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()

Висновок з результатів вказує на те, що тренування нейронної мережі для кластеризації було завершено, досягнувши максимальної кількості епох (200). Зменшення помилки відбулося лише на початкових етапах, а подальше тренування призвело до її стабільності.

• Вплив кількості нейронів і кластерів:

З погляду помилки тренування, перевищення кількості кластерів, які намагається навчити мережа, може спричинити збільшення помилки через надмірну адаптацію до навчальних даних. У цьому випадку може виникнути неправильний вибір кількості кластерів, оскільки мережа старається апроксимувати більше кластерів, ніж фактично присутні в даних.

• Вплив розкиду вхідних даних:

Розкид вхідних даних може впливати на точність кластеризації, і в даному випадку розкид є достатньо великим, що ускладнює відрізнення реальних центрів кластерів. Це може призвести до менш точної апроксимації кластерів мережею. Обидва ці фактори підкреслюють важливість вибору оптимальної кількості кластерів та уважності до характеристик вхідних даних при тренуванні нейронних мереж для кластеризації.

***Висновки:*** в ході виконання лабораторної роботи було, використовуючи спеціалізовані бібліотеки та мову програмування Python, отримано практичні навички зі створення та застосовування простих нейронних мереж.

GitHub: https://github.com/invicibleee/Artificial-intelligence.git