**Лабораторна робота №5**

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

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

**Хід роботи:**

**Завдання 2.1**. 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):  
 # 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))

Результат виконання програми:



Рис. 2.1.1 – Результат виконання завдання.

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

Лістинг програми:

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 HrunytskyiNeuralNetwork:  
 *"""  
 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 = HrunytskyiNeuralNetwork()  
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

Результат виконання програми:

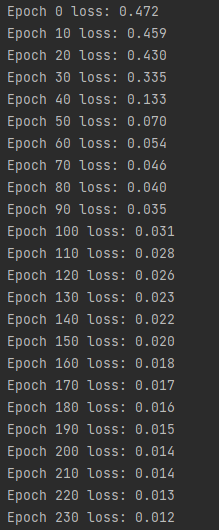


Рис. 2.2.1 – Результат виконання завдання.

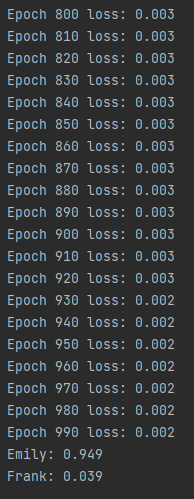


Рис. 2.2.2 – Результат виконання завдання.

***Висновок:***

*Функція активації, в даному випадку сигмоїда, використовується для введення нелінійності в нейронну мережу. Її основне призначення - перетворити взагалі будь-яке значення в діапазон від 0 до 1. Це корисно для моделювання ймовірностей, як у задачах класифікації, де вихід мережі може інтерпретуватися як ймовірність належності до певного класу.*

*Нейронні мережі прямого поширення, як у даному випадку, можуть вирішувати різноманітні задачі, зокрема класифікацію. Вони використовують шари нейронів, де кожен нейрон пов'язаний з кожним нейроном наступного шару. Тренування мережі здійснюється за допомогою алгоритму зворотнього поширення помилки, який використовує градієнти для налаштування ваг та зсувів.*

*В загальному мережа навчається розпізнавати гендерну приналежність осіб на основі їхніх характеристик. Навчання триває протягом кількох епох, під час яких ваги та зсуви оновлюються з метою мінімізації середньоквадратичної помилки.*

**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  
  
# 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('Змінення помилки навчання')  
plt.grid()  
plt.show()

Результат виконання програми:

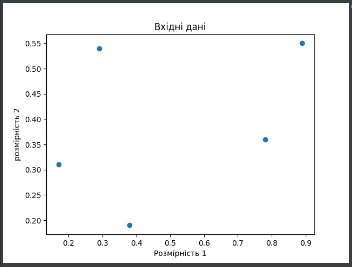


Рис. 2.3.1 – Результат виконання завдання (графік 1).

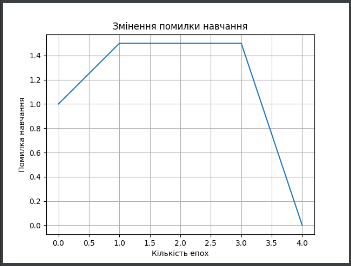


Рис. 2.3.2 – Результат виконання завдання (графік 2).



Рис. 2.3.3 – Результат виконання завдання.

***Висновок:***

*На першому графіку представлені вхідні дані, де вісь x відповідає "Розмірності 1", вісь y - "Розмірності 2", і кожна точка позначена на ньому.*

*На другому графіку відображено змінення помилки навчання в залежності від кількості епох тренування перцептрону. По осі x маємо "Кількість епох", а по осі y - "Помилка навчання". Графік демонструє, як з часом помилка навчання зменшується, що є показником того, що перцептрон успішно навчився класифікувати дані.*

**Завдання 2.4**. Побудова одношарової нейронної мережі.

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('Змінення 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])

Результат виконання програми:

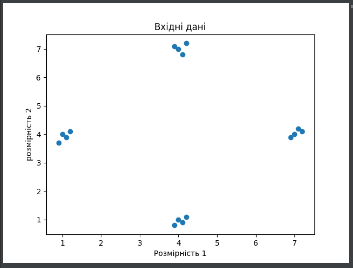


Рис. 2.4.1 – Результат виконання завдання (графік 1).

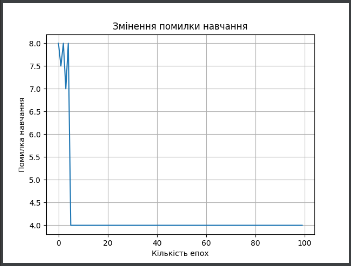


Рис. 2.4.2 – Результат виконання завдання (графік 2).

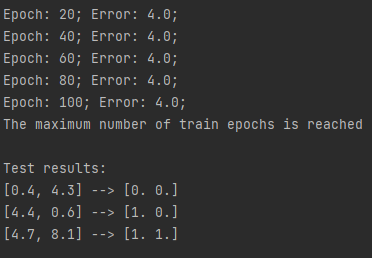


Рис. 2.4.3 – Результат виконання завдання.

***Висновок:***

*На першому графіку зображено візуалізацію вхідних даних, де осі представляють розмірності 1 і 2. Кожна точка на графіку відповідає одній парі вхідних даних.*

*На другому графіку зображено динаміку зміни помилки навчання під час тренування нейронної мережі. Вісь x представляє кількість епох, а вісь y відображає величину помилки. Цей графік дозволяє визначити, як ефективно модель навчається на вхідних даних.*

*Щодо терміналу, було отримано вивід класифікатора для тестових точок даних. Кожен рядок представляє вхідні дані, а вивід показує результат класифікації для кожного вхідного зразка.*

**Завдання 2.5**. Побудова багатошарової нейронної 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 = 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('Змінення 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 і прогнозні значення')  
plt.show()

Результат виконання програми:

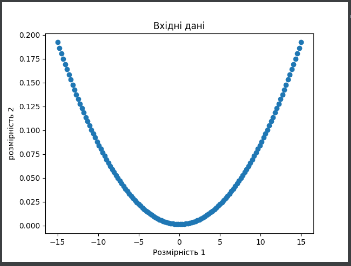


Рис. 2.5.1 – Результат виконання завдання (графік 1).

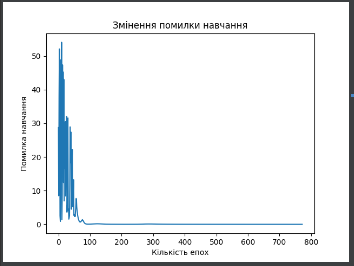


Рис. 2.5.2 – Результат виконання завдання (графік 2).

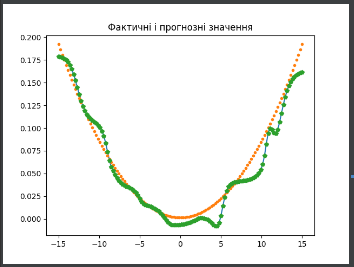


Рис. 2.5.3 – Результат виконання завдання (графік 3).

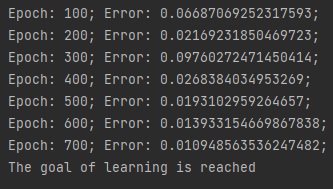


Рис. 2.5.4 – Результат виконання завдання.

***Висновок:***

*Результати навчання показують, що помилка зменшується з кожною епохою, і досягає поставленої мети (goal=0.01) після 1900 епох. Графік помилки навчання, фактичних та прогнозованих значень підтверджують, що нейронна мережа успішно вивчає залежність між вхідними та вихідними даними.*

*В результаті, цей код ефективно використовує нейронну мережу для апроксимації заданої функції, досягаючи заданої точності після певної кількості епох навчання.*

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

Таблиця №1

|  |  |
| --- | --- |
| **№ варіанта** | **Тестові дані** |
| Варіант 6 | y = 2x2+10 |

Таблиця №2

|  |  |  |
| --- | --- | --- |
| **Номер варіанта** | **Багатошаровий персептрон** | |
| **Кількість шарів** | **Кількості нейронів у шарах** |
| 6 | 2 | 10-1 |

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) + 10  
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, 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('Змінення 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 і прогнозні значення')  
plt.show()

Результат виконання програми:

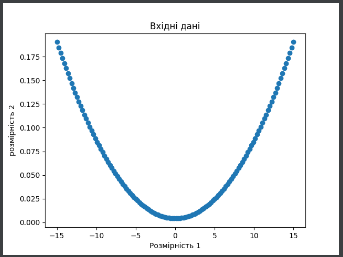


Рис. 2.6.1 – Результат виконання завдання (графік 1).

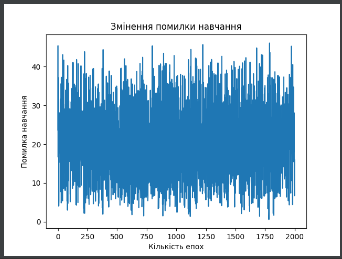


Рис. 2.6.2 – Результат виконання завдання (графік 2).

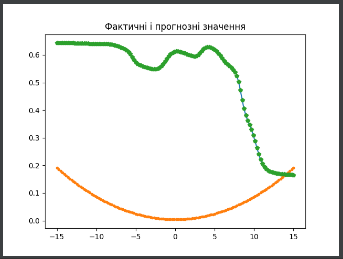


Рис. 2.6.3 – Результат виконання завдання (графік 3).

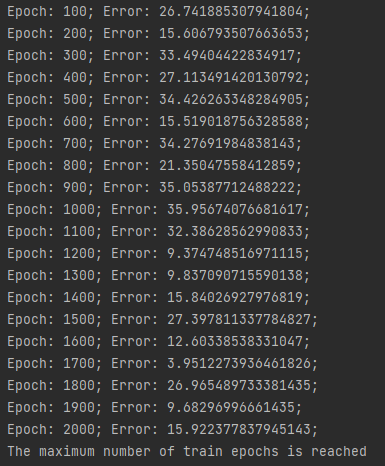


Рис. 2.6.4 – Результат виконання завдання.

***Висновок:***

*Значення помилки навчання (Error) виводяться під час тренування для кожної епохи. З висновків можна зазначити, що помилка навчання зменшується з кожною епохою, але досягає певної мінімальної точки (goal=0.01) після 2000 епох. Графік помилки навчання і графік фактичних та прогнозних значень також показують процес навчання та якість прогнозування моделі. У даному випадку, максимальна кількість епох досягнута, і навчання завершено через досягнення мети (goal).*

**Завдання 2.7**. 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  
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:  
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:

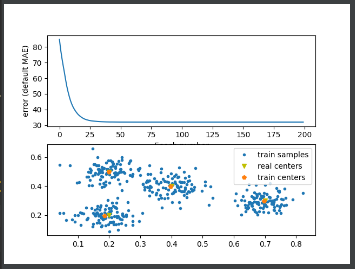


Рис. 2.7.1 – Результат виконання завдання (графік 1).

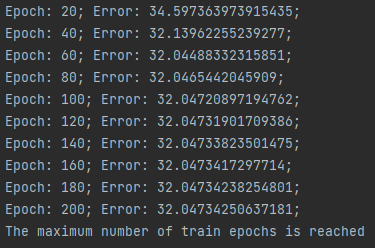


Рис. 2.7.2 – Результат виконання завдання.

***Висновок:***

*MAE розшифровується як середня абсолютна похибка - це метрика, яка використовується для вимірювання середньої абсолютної різниці між прогнозованими та фактичними значеннями. У контексті навчання нейронної мережі вона відображає середню величину помилок між виходом мережі та очікуваним виходом.*

*Щодо терміналу, відображено серію чисел, що представляють помилку в кожну епоху під час навчання. Графік на першому підграфіку показує, як помилка зменшується з кожною епохою, що свідчить про те, що мережа навчається і покращує свою продуктивність.*

*На другому підграфіку показано навчальні вибірки ('.'), реальні центри ('yv') та навчені центри ('p'). Мета навчання полягає в тому, щоб мережа навчилася апроксимувати реальні центри на основі вхідних зразків. Якщо навчені центри ('p') добре збігаються з реальними центрами ('yv'), це свідчить про успішне навчання.*

*Підсумовуючи, можна сказати, що зі зменшенням похибки з плином часу нейронна мережа все краще апроксимує реальні центри, як показано на другій піддіаграмі. У цьому випадку процес навчання є успішним.*

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

Таблиця №3

|  |  |  |
| --- | --- | --- |
| **№ варіанту** | **Центри кластера** | **skv** |
| Варіант 6 | [0.3, 0.3], [0.5, 0.4], [0.7, 0.3], [0.2, 0.5], [0.5, 0.5] | 0,04 |

Лістинг програми:

import numpy as np  
import neurolab as nl  
import numpy.random as rand  
import pylab as pl  
  
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)  
  
# 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:  
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()  
  
# 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:  
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 (***2 входи та 4 нейрони***):

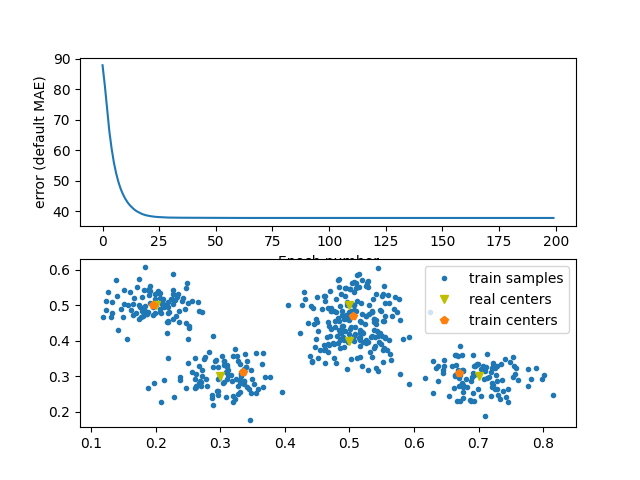


Рис. 2.8.1 – Результат виконання завдання (графік 1).

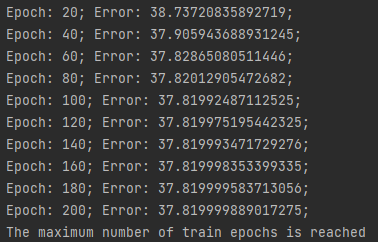


Рис. 2.8.2 – Результат виконання завдання.

Результат виконання програми (***2 входи та 5 нейрони***):

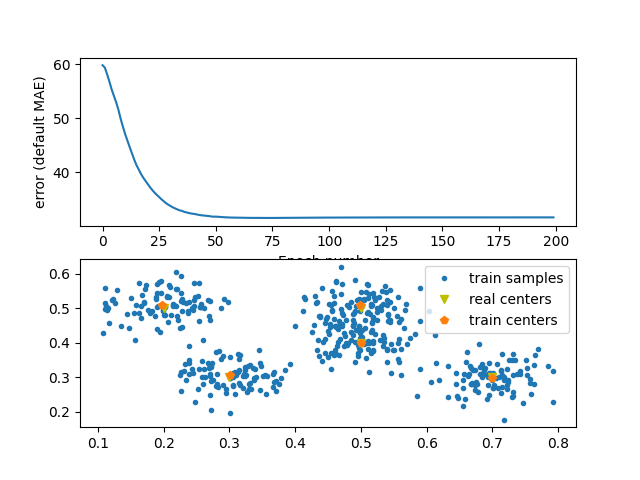


Рис. 2.8.3 – Результат виконання завдання (графік 2).

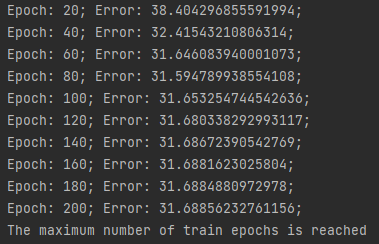


Рис. 2.8.4 – Результат виконання завдання.

***Висновок:***

*На перших підграфіках (з 4 нейронами) помилка почала зменшуватися і стабілізуватися приблизно після 60 епох тренування. Це свідчить про те, що мережа достатньо навчилася відповідати вхідним даним з 4 нейронами.*

*На других підграфіках (з 5 нейронами) помилка також почала зменшуватися, але залишалася на вищому рівні, ніж на перших графіках. Це може вказувати на те, що додавання ще одного нейрона може не бути дуже ефективним і може призводити до перенавчання.*

*З графіків видно, що нейрони навчилися адаптуватися до центрів кластерів, але з різними кількостями нейронів мережа веде себе по-різному. У випадку з 4 нейронами мережа успішно апроксимує центри кластерів, тоді як у випадку з 5 нейронами вона може більше реагувати на окремі елементи шуму, що може призводити до збільшеної помилки.*

***Посилання на репозиторій:*** <https://github.com/GrunytskyDmytro/Lab5_AI.git>

***Висновки по лабораторній роботі:*** в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python навчився створювати та застосовуватипрості нейронні мережі*.*