Міністерство освіти і науки України

Національний технічний університет "ХПІ"

кафедра "Інформатика та інтелектуальна власність"

**ЗВІТ**

до лабораторної роботи № 4

Тема : «Розв'язання задачі класифікації за допомогою одного нейрона (одношарового перцептрону) мовою Python»

Варіант номер 18

з дисципліни "Основи штучного інтелекту"

Виконав:

студент групи КН-321В

Хома Д.М.

Перевірив:

Паржин Ю. В.

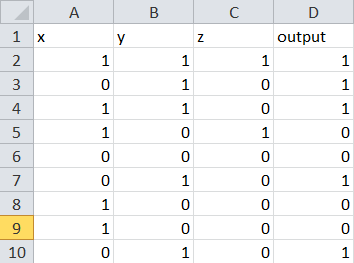
Харків 2023

**1. Архітектура мережі та вибрані гіперпараметри**

У мережі використовується один нейрон зі сигмоїдною активаційною функцією. Кількість вхідних вузлів (INPUT\_DIM) становить 3. Швидкість навчання (learning\_rate) встановлена на рівні 0.1. Ваги ініціалізуються випадковими значеннями.

**2. Алгоритм побудови навчальної та тестової вибірки**

Навчальний набір даних з 1000 прикладами зчитується з файлу '3D\_data.xlsx' на рис.2.1, де вхідні дані мають три ознаки і вихідне значення, яке має бути вивчено . Алгоритм визначення вихідного значення: =ЕСЛИ( B2=1; 1; 0). Тобто, якщо y = 1, то вихід = 1.



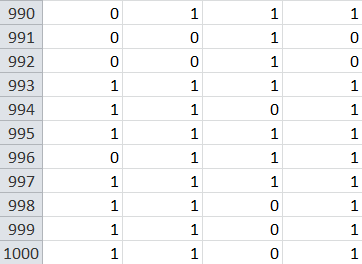
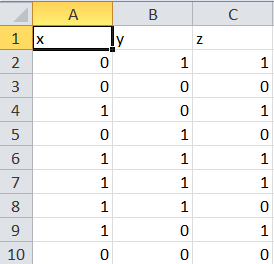


Рисунок 2.1 - '3D\_data.xlsx

Тестовий набір даних зчитується з файлу '3D\_data\_test.xlsx' на рис 2.2. Вхідні дані обробляються, і вихідні значення порівнюються з вихідом мережі для оцінки її продуктивності.



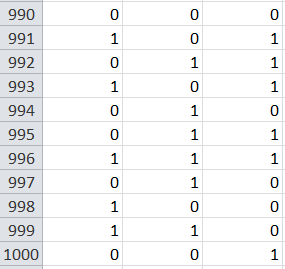


Рисунок 2.2 - 3D\_data\_test.xlsx

**3. Алгоритм навчання ШНМ**

Алгоритм навчання мережі використовує ітерації 10, 50, 100, 1000 епох та пакетний градієнтний спуск для оновлення вагових коефіцієнтів.

У мережі використано сигмоїдну активаційну функцію без порогу.

Для модифікації вагових коефіцієнтів використано значення зваженої помилки:

ΔW *= error* × *input × Grad(output)(1)*

Формула (1) дозволяє виконувати коригування ваги пропорційно до величини помилки (*error*). Далі множення відбувається на вхідне значення *input*. Якщо вхідне значення дорівнює 0, вага коригуватися не буде. Потім виконується множення на *градієнт* сигмоїдної кривої (*Grad (output)*).

1. Обчислюється сума входів і ваг.
2. Застосовується сигмоїдна активаційна функція до вагованої суми, отримуючи вихід мережі.
3. Обчислюється помилка, що полягає в різниці між очікуваним вихідним значенням і вихідним значенням мережі.
4. Обчислюється градієнт сигмоїдної функції.
5. Ваги оновлюються за допомогою формули (1), яка враховує помилку, вхід та градієнт.

**4. Текст програми**

import pandas as pd  
import numpy as np  
  
# Функція активації - сигмоїда  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))

# Функція, яка визначає очікуваний вихід для навчання  
def func(x):  
 return 1 if x == 1 else 0  
  
# Клас нейронної мережі  
class NeuralNetwork:  
 INPUT\_DIM = 3 # Кількість входів мережі  
 learning\_rate = 0.1 # Швидкість навчання  
 weights = np.random.rand(INPUT\_DIM) # Випадкова ініціалізація ваг  
  
 def \_\_init\_\_(self):  
 self.expected\_output = None  
  
 # Метод навчання мережі  
 def learn(self, filename):  
 training\_data = pd.read\_excel(filename)  
 self.expected\_output = training\_data['output']  
 training\_data = np.asarray(training\_data.drop('output', axis=1))  
  
 # Цикл навчання  
 for epoch in range(50):  
 for i in range(len(training\_data)):  
 output\_sum = np.sum(np.multiply(training\_data[i, :], self.weights))  
 output\_value = sigmoid(output\_sum)  
  
 error = self.expected\_output[i] - output\_value  
  
 grad = output\_value \* (1 - output\_value)  
  
 # Оновлення ваг відповідно до правила навчання  
 for n in range(self.INPUT\_DIM):  
 self.weights[n] = self.weights[n] + self.learning\_rate \* error \* training\_data[i, n] \* grad  
 print("Weights:", self.weights)  
  
 # Метод для тестування мережі на тестовому наборі даних  
 def testing(self, filename):  
  
 test\_data = np.asarray(pd.read\_excel(filename))  
 count\_errors = 0  
 lst\_errors = []  
  
 for i in range(len(test\_data)):  
 output\_sum = np.sum(np.multiply(test\_data[i, :], self.weights))  
 output\_value = sigmoid(output\_sum)  
  
 real = func(test\_data[i][1])  
  
 print("Input: {}, Output: {}, Real: {}".format(test\_data[i], output\_value, real))  
 if round(output\_value) == real:  
 continue  
 else:  
 lst\_errors.append((test\_data[i], output\_value, real))  
 count\_errors += 1  
  
 print('Помилка', count\_errors / len(test\_data))  
 print("Ваги:", self.weights)  
 if len(lst\_errors) :print(lst\_errors, sep='\n')

if \_\_name\_\_ == "\_\_main\_\_":  
 nn = NeuralNetwork()  
 nn.learn('3D\_data.xlsx')  
 nn.testing('3D\_data\_test.xlsx')

**5. Результати навчання**

Мінімальна помилка на навчальному наборі та підсумкові вагові коефіцієнти будуть варіюватися в залежності від конкретного набору даних та кількості епох навчання.

Нижче наведені результати навчання мережі для різної кількості епох:

**10 епох:**



**50 епох:**



**100 епох:**



**1000 епох:**



Бачимо що, вагові коефіцієнти змінюються з кожною епохою навчання. А саме 1 та 3 коефіцієнти майже рівні і зменшуються, а 2 коефіцієнт збільшується зі збільшенням кількості епох навчання.

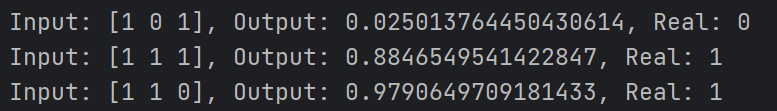
**6. Алгоритм тестування та результати тестування**

Алгоритм тестування використовує тестовий набір даних, де вхідні дані не беруть участі у навчанні. Для кожного тестового прикладу виконується наступне:

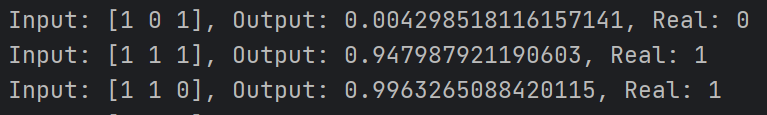
1. Обчислюється вагована сума входів і ваг.
2. Застосовується сигмоїдна активаційна функція до вагованої суми, отримуючи вихід мережі.
3. Порівнюється вихід мережі з реальним значенням, яке було визначено за допомогою функції func().
4. Виведення результатів, включаючи вхідні дані, виход мережі та реальне значення.
5. Збільшення лічильника помилок, якщо виникає, та збереження неправильно класифікованих прикладів.
6. Обчислення та виведення відсотка помилок на тестовому наборі.
7. Якщо вихід мережі близький до реального значення, то вважається, що тестування пройшло успішно.

Я обрав 3 рандомні приклади для демонстрації зміни похибки залежно від кількості епох навчання.

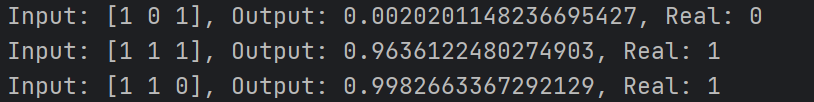
**10 епох:**



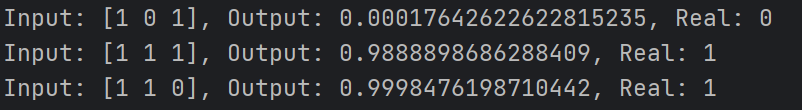
**50 епох:**



**100 епох:**



**1000 епох:**



Можемо побачити, що помилка зменшується залежно від епох навчання.

**6. Висновки**

1. Кількість нейронів та архітектура мережі:

У даному випадку використовується один нейрон, і це може бути досить ефективним для задачі класифікації, де вхідні дані мають невелику кількість ознак.

Однак у більш складних задачах із багатьма ознаками може знадобитися більша кількість нейронів або більша архітектура мережі для досягнення кращих результатів.

2. Швидкість навчання (learning rate):

Швидкість навчання є різним для різних задач. У даному випадку вона дорівнює 0.1 і може бути прийнятною, але для інших задач може бути необхідним експериментувати з різними значеннями швидкості навчання. Занадто велика швидкість навчання може призвести до незбалансованих вагових коефіцієнтів, а занадто мала - до повільного навчання.

3. Ініціалізація ваг:

Ваги ініціалізуються випадковими значеннями, що є звичайним підходом. Але існують інші методи ініціалізації ваг, такі як Xavier/Glorot.

4. Кількість епох навчання:

Для більшої точності може знадобитися більше епох. Важливо експериментувати з цим параметром для досягнення оптимальних результатів. Для моєї правильна класифікація відбувається і за 10 епох, але з великою похибкою. При 1000 епохах похибка майже зникає і дорівнює 1 % та менше.