НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Звіт

із лабораторної роботи 2

з дисципліни «Системи глибинного навчання»

на тему: “Розробка програмного забезпечення для реалізації ймовірнісної нейронної мережі PNN”

Виконав:
студент групи КМ-03
Шаповалов Г. Г.

Перевірив:
Терейковський І. А.

Київ — 2023

Зміст

Теоретичні відомості

Ймовірнісна нейронна мережа (PNN) - це прямопрогонна нейронна мережа, яка широко використовується у задачах класифікації та розпізнавання шаблонів. В алгоритмі PNN батьківська функція розподілу ймовірностей (PDF) кожного класу наближається за допомогою методу ядерних оцінок густини ймовірності, Гаусових функцій.

Операції в PNN організовані в багатошарову прямопрогонну мережу з чотирма шарами:

1. Вхідний шар: Кожен нейрон вхідного шару представляє змінну прогнозування.
2. Шар шаблонів: Цей шар містить один нейрон для кожного випадку в навчальному наборі даних.
3. Шар додавання: Для PNN є один нейрон шаблону для кожної категорії цільової змінної.
4. Вихідний шар: Вихідний шар порівнює зважені голоси для кожної цільової категорії, накопичені в шарі шаблонів, і використовує найбільший голос для прогнозування цільової категорії.

PNN часто використовуються в задачах класифікації. Вони були виведені з Байєсової мережі та статистичного алгоритму.

Основна частина

Програму реалізовано на мові Python. Спочатку імпортуємо бібліотек numpy для математичних розрахунків. Створюємо власну функцію pnn, яка буде отримувати на вхід тренувальні дані, тестувальні дані та значення дельти для ядра Гуаса. Фунція щоразу буде проходитись по навчальним даним і буде порівнювати до їх схожість із тестувальними даними. Повертатиме значення класу (A, B або C). Також визначаємо власноруч тренувальні та тестові дані, які будемо передавати у функцію. А потім за допомогою циклу тестуємо нейронну мережу. У результаті отримуємо такі класи для тестових даних:

[0.43, 0.88] is B class

[0.63, 0.15] is C class

[0.57, 0.54] is C class

[0.24, 0.25] is A class

[0.55, 0.51] is C class

[0.45, 0.82] is C class

Додаток А – Код програми

```python
import numpy as np


def pnn(X_train, Y_train, delta, X_test):
    # Створюємо словник для зберігання ймовірностей класів
    class_probs = {}
    # Циклом проходимо по навчальному набору даних
    for x_train, class_label in zip(X_train, Y_train):
        # Якщо клас ще не в словнику, додаємо його
        if class_label not in class_probs:
            class_probs[class_label] = 0
        # Обчислюємо схожість між тестовими даними та поточними навчальними даними
        similarity = np.exp(-np.sum((x_train - X_test) ** 2) / (2 * delta ** 2))
        # Додаємо обчислену схожість до загальної ймовірності поточного класу
        class_probs[class_label] += similarity
    # Повертаємо клас з найбільшою ймовірністю
    return max(class_probs, key=class_probs.get)


if __name__ == '__main__':
    # навчальний набір
    X_train = np.array([[0.10, 0.30], [0.20, 0.10], [0.50, 0.10],
                        [0.30, 0.20], [0.80, 0.10], [0.60, 0.60],
                        [0.28, 0.15], [0.86, 0.54], [0.30, 0.61],
                        [0.85, 0.13], [0.68, 0.98], [0.00, 0.43]])
    # мітки навчального набору
    Y_train = np.array(['A', 'B', 'C',
                        'A', 'B', 'C',
                        'A', 'B', 'C',
                        'A', 'B', 'C'])
    # тестувальний набір
    X_test = np.array([[0.43, 0.88], [0.63, 0.15],
                       [0.57, 0.54], [0.24, 0.25],
                       [0.55, 0.51], [0.45, 0.82]])
    # параметр точності
    delta = 0.1
    # тестування моделі
    for x_test in X_test:
        pred_class = pnn(X_train, Y_train, delta, x_test)
        print(f'{x_test.tolist()} is {pred_class} class')
```