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-01
Романецький М.С.

Викладач:
Професор кафедри ПМА
Терейковський І. А.

Київ — 2023

Зміст

Теоретичні відомості

Ймовірнісна нейронна мережа PNN - це вид штучних нейронних мереж, який використовує Баєсову статистику для виконання певних завдань, таких як класифікація або регресія. Ймовірнісна нейронна мережа була розроблена Дональдом Спехтом у 1990 році. Основна ідея PNN полягає в тому, що виходи мережі можна інтерпретувати як оцінки ймовірності належності вхідного об’єкта певному класу або значенню функції. Для цього PNN використовує метод ядерних оцінок густини ймовірності, який базується на використанні Гаусових функцій.

Структура PNN складається з чотирьох шарів: вхідного, прихованого, сумувального та вихідного. Вхідний шар містить нейрони, які передають вхідний вектор до прихованого шару. Прихований шар містить нейрони, які обчислюють відстань від вхідного вектора до кожного з навчальних прикладів, які належать до певного класу. Сумувальний шар містить нейрони, які підсумовують виходи прихованого шару для кожного класу. Вихідний шар містить нейрони, які нормалізують виходи сумувального шару та видають оцінки ймовірності для кожного класу.

Для навчання PNN не потрібно використовувати жодних алгоритмів оптимізації, таких як градієнтний спуск або зворотне поширення помилки. Навчання PNN полягає в тому, що для кожного навчального прикладу створюється нейрон у прихованому шарі, який має центр у цьому прикладі та параметр ширини, який визначається емпірично. Таким чином, PNN має просту та швидку процедуру навчання, але водночас вимагає багато пам’яті та обчислювальних ресурсів для зберігання та обробки всіх навчальних прикладів.

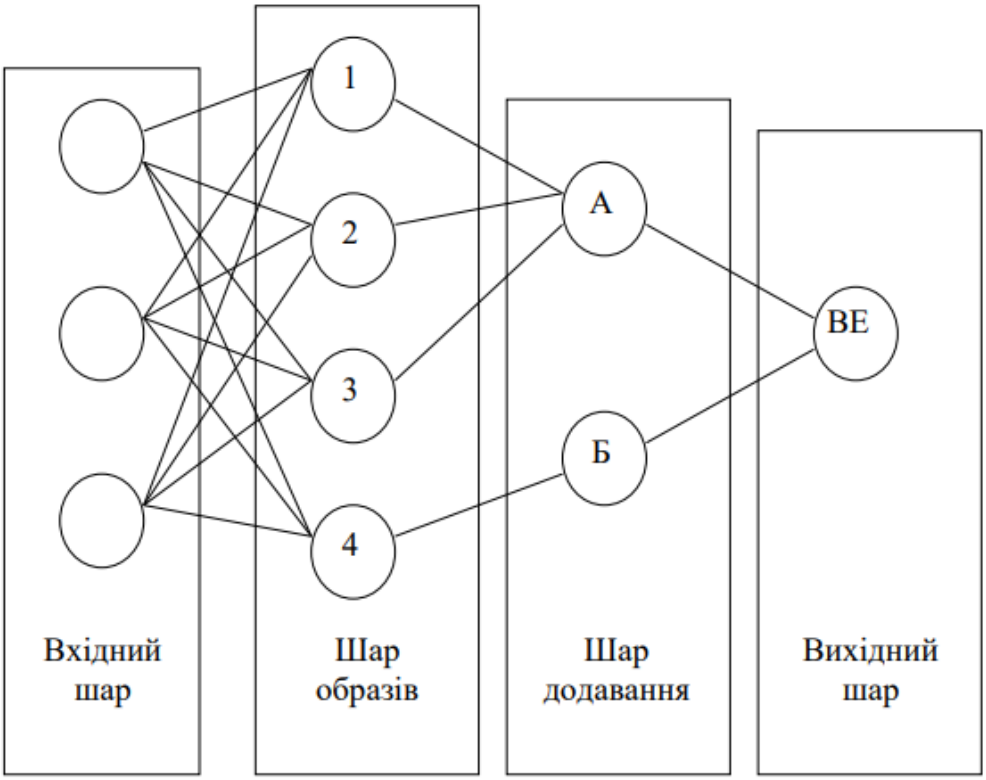

Рис. 1 – Архітектурна схема мережі PNN

https://ela.kpi.ua/bitstream/123456789/50135/1/ANN.pdf ст. 64

Основна частина

Завдання: розробити програмне забезпечення для реалізації мережі PNN, що призначена для апроксимації функції: $y = x_1 + x_2$

Передбачити режими навчання та розпізнавання.

Для написання програми використаємо мову програмування *Python*. Імпортуємо ряд бібліотек:

- *Numpy* для математичних розрахунків
- *Pandas* для зручного виводу інформації
- Та деякі функції з модуля *typing* для анотації типів у функціях

Створимо функцію *train_pnn*, яка прийматиме на вхід:

- *X_train* – вектор вхідних даних для тренування моделі
- *Y_train* – вектор міток (класи A та B) для тренування моделі
- Значення гіперпараметра *delta* для керування чутливістю до точок даних. (За замовчанням дельта = 1.0)

Ця функція проходиться циклом по навчальним даним та їх міткам. Обчислює ймовірності класів за формулою ядра Гауса. Повертає словник, що містить ймовірності класів.

Створимо функцію *predict_pnn*, яка прийматиме на вхід:

- *Class_probs* – словник ймовірностей класів, отриманий у результаті навчання
- *X_predict* – точка даних для розпізнавання
- *X_train* – характеристики навчальних даних
- Значення гіперпараметра *delta* для керування чутливістю до точок даних. (За замовчанням дельта = 1.0)

Ця функція проходиться циклом по словнику ймовірностей класів та перевіряє до якого класу найбільше підходять дані для розпізнавання. Повертає мітку класу.

Далі визначаємо навчальні дані, налаштовуємо гіперпараметр. Передаємо ці дані в функцію тренування, щоб модель навчилась на навчальних даних.

Визначаємо дані для розпізнавання. Та тестуємо модель на тих даних яких вона ще не бачила. В результаті отримуємо, що дана модель впоралась непогано, розпізнавши правильно 7 із 9 заданих варіантів.

Покращити результат можна шляхом додавання більшої кількості даних у навчальну вибірку та налаштування гіперпараметра дельта не емпіричним шляхом, а наприклад, кроссвалідацією.

Перелік посилань

1. Навчальний посібник «Основні концепції нейроних мереж» Роберт Каллан – стор. 158-164.
2. https://ela.kpi.ua/bitstream/123456789/50135/1/ANN.pdf
3. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=242d1ed56ff65d548da3e21b6a0aa53fc246b10b

Додаток А – Код програми

```
import numpy as np
import pandas as pd
from typing import List, Dict, Optional
```

```
def train_pnn(X_train: np.ndarray,
              Y_train: List[str],
              delta: float = 1.0) -> Dict[str, float]:
    """
    Фунція навчить імовірнісну нейронну мережу (PNN) за навчальними даними.

    Аргументи:
        - X_train (numpy.ndarray):  Характеристики навчальних даних.
        - Y_train (numpy.ndarray):  Відповідні мітки класів.
        - delta (float):            Гіперпараметр для керування чутливістю до точок
даних.

    Повертає значення:
        dict:                       Словник, що містить ймовірності класів.
    """
    class_probs = {}  #  Ініціалізувати порожній словник для зберігання
ймовірностей класів

    #  У циклі перебираємо навчальні дані та мітки
    for x, class_label in zip(X_train, Y_train):
        if class_label not in class_probs:  #  Якщо мітка класу відсутня у
словнику, ініціалізуйте її значенням 0.0
            class_probs[class_label] = 0.0
        """Обчислюємо ймовірності класів за формулою ядра Гауса. Ядро Гауса вимірює
схожість між точками даних.
        Воно базується на евклідовій відстані між X_train та x, зваженій на
дельту"""
        class_probs[class_label] += np.exp(-np.sum((X_train - x) ** 2, axis=1) / (2
* delta ** 2))

    return class_probs


def predict_pnn(class_probs: Dict[str, float],
                X_predict: np.ndarray,
                X_train: np.ndarray,
                delta: float = 1.0) -> Optional[str]:
    """
    Функція передбачає клас для заданої точки даних за допомогою навченого PNN.
```

```
    Аргументи:
        - class_probs (dict):            Словник ймовірностей класів, отриманий у
результаті навчання.
        - X_predict (numpy.ndarray):     Точка даних для розпізнавання.
        - X_train (numpy.ndarray):       Характеристики навчальних даних.
        - delta (float):                 Гіперпараметр для керування чутливістю до
точок даних.

    Повертається:
        str:                             Розпізнана мітка класу. (Може повернути
None)
    """
    max_prob = 0  #  Ініціалізуємо максимальну ймовірність рівну 0
    predicted_class = None  #  Ініціалізуємо розпізнаний клас значенням None

    for class_label, prob in class_probs.items():
        """У циклі обчислюємо схожість між X_train та X_predict за формулою ядра
Гауса
        Ядро Гауса базується на евклідовій відстані між X_train та X_predict,
зваженій на дельту."""
        similarity = np.exp(-np.sum((X_train - X_predict) ** 2, axis=1) / (2 *
delta ** 2))
        class_probs[class_label] += similarity

        if np.max(prob) > max_prob:  #  Порівняєм максимальне значення імовірності
            max_prob = np.max(prob)
            predicted_class = class_label

    return predicted_class
```

```
X_train = np.array([[100, 300], [200, 100], [5.0, 1.0], [300, 200], [8.0, 1.0],
[6.0, 6.0]])
Y_train = np.array(['A', 'A', 'B', 'A', 'B', 'B'])
```

```
delta = 250.0
```

```
class_probs = train_pnn(X_train, Y_train, delta)
```

```
X_predict = np.array([3.0, 1.2])
```

```
predicted_class = predict_pnn(class_probs, X_predict, X_train, delta)

print(f'Data {X_predict.tolist()} | Predicted class: {predicted_class}')
```

```
class_probs = train_pnn(X_train, Y_train, delta)
x_t = [[150, 350], [250, 150], [5.5, 1.5], [350, 250], [8.5, 1.5], [6.5, 6.5],
[150, 250], [3.5, 4.5], [2.5, 3.5]]

results_df = pd.DataFrame(columns=["Data", "Predicted class"])

for x in x_t:
    x = np.array(x)
    pred_cl = predict_pnn(class_probs, x, X_train, delta)
    results_df = pd.concat([results_df, pd.DataFrame({"Data": [x.tolist()],
"Predicted class": [pred_cl]})],
                           ignore_index=True)

print(results_df)
```