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: | Викладач: |
| студент групи КМ-01 | Професор кафедри ПМА |
| Романецький М.С. | Терейковський І. А. |

Київ — 2023

Зміст

[Теоретичні відомості 3](#_Toc150360145)

[Основна частина 5](#_Toc150360146)

[Перелік посилань 7](#_Toc150360147)

[Додаток А – Код програми 8](#_Toc150360148)

## 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.

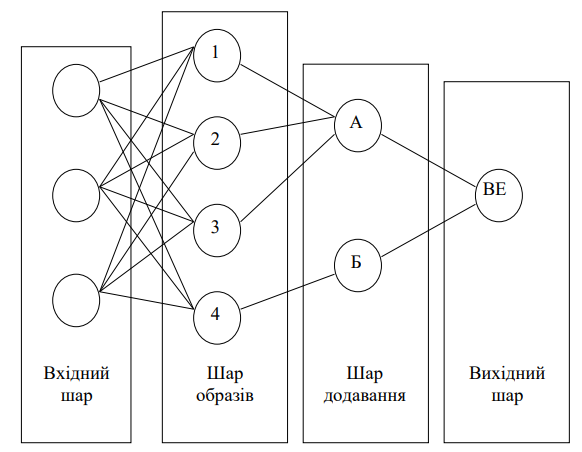


Рис. 1 – Архітектурна схема мережі PNN

<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:

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 А та В) 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 х, 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  *#  Ініціалізувуємо максимальну ймовірність рівну 0*

    predicted\_class = None  *#  Ініціалізувуємо 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)