НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики
Кафедра прикладної математики

Звіт
із лабораторної роботи
із дисципліни «СИСТЕМИ ГЛИБИННОГО НАВЧАННЯ»
на тему:

Розробка програмного забезпечення для реалізації ймовірнісної нейронної мережі PNN

Виконав:
студент групи КМ-13
Онищенко В.С.

Перевірив:
Терейковський І. А.

Київ — 2024

ЗМІСТ

1 Теорія

Ймовірнісна нейронна мережа (Probabilistic Neural Network, PNN) є спеціалізованою моделлю, яка використовує статистичні принципи для апроксимації розподілу ймовірностей. В основі PNN лежить байєсівська теорема, а для обчислення подібності між зразками використовується Гаусове ядро. Мережа PNN складається з чотирьох основних шарів:

- **Вхідний шар**, який приймає значення вхідних параметрів.
- **Шаблонний (pattern) шар**, який обчислює значення Гаусового ядра для кожного зразка у тренувальному наборі, визначаючи схожість вхідного зразка з іншими.
- **Сумаційний шар**, який підсумовує виходи шаблонного шару, забезпечуючи розрахунок середньозваженого значення.
- **Вихідний шар**, що надає підсумковий результат, який інтерпретується як передбачення або оцінка функції.

Сигмоїдальна функція активації та похідна використовуються для згладжування вихідних даних, а параметр σ (сигма) визначає ширину Гаусового ядра. Зазвичай PNN застосовується для задач класифікації, але вона також ефективно працює з задачами апроксимації, такими як розрахунок нелінійних функцій. У даній роботі PNN використовується для апроксимації функції $y = x_1 + x_2$.

2 Програма

2.1 Ініціалізація та Гаусова функція ядра

PNN реалізована на Python, де бібліотека numpy використовується для обчислень. Ініціалізація параметра згладжування σ відбувається під час створення екземпляру класу **PNN**.

```
import numpy as np

class PNN:
    def __init__(self, sigma=0.1):
        self.sigma = sigma
        self.train_data = None
        self.train_targets = None
```

Функція ядра Гауса **gaussian_kernel** обчислює подібність між вхідним зразком **x** і кожним зразком **x_i** з тренувального набору.

```
    def gaussian_kernel(self, x, x_i):
        return np.exp(-np.linalg.norm(x - x_i) ** 2 / (2 * self.sigma ** 2
```

2.2 Навчання

Метод **train** зберігає тренувальні дані та цільові значення. PNN не потребує традиційного процесу навчання, тому навчання тут полягає лише у збереженні даних для подальшого використання.

```
    def train(self, X, y):
        self.train_data = X
        self.train_targets = y
```

2.3 Передбачення

Метод **predict** розраховує передбачення для кожного вхідного зразка, використовуючи середньозважене значення на основі обчислених ваг (подібностей) між новим зразком і всіма зразками з тренувального набору.

```
    def predict(self, X):
        predictions = []

        for x in X:
            weights = np.array([self.gaussian_kernel(x, x_i) for x_i
                                        in self.train_data])
            weighted_sum = np.dot(weights, self.train_targets)
            prediction = weighted_sum / np.sum(weights)
            predictions.append(prediction)

        return np.array(predictions)
```

3 Результати роботи програми

Для тренування були використані такі дані:

$$X_{\text{train}} = \begin{bmatrix} 0 & 0 \\ 0 & 2 \\ 1.5 & 0 \\ 1 & 1 \\ 0.3 & 0.7 \\ 0.2 & 1.8 \\ 1.7 & 0.2 \\ 1 & 2 \\ 2 & 1 \end{bmatrix}$$

та відповідні значення цільової функції $y_{\text{train}} = x_1 + x_2$.

Для тестування моделі було використано такі вектори та отримано наступні передбачення:

```
Тестовий вектор 1: [0.5 1.5], передбачення: 2.068081038174548
Тестовий вектор 2: [1.  1.5], передбачення: 2.477909561881604
Тестовий вектор 3: [0.3 1.3], передбачення: 1.712032059415510
Тестовий вектор 4: [1.8 0.2], передбачення: 1.783855803923885
Тестовий вектор 5: [0.9 1.1], передбачення: 1.954366446582369
Тестовий вектор 6: [1.1 1.9], передбачення: 2.975820937504303
Тестовий вектор 7: [2. 2.], передбачення: 2.9980678466628623
Тестовий вектор 8: [1.5 1. ], передбачення: 2.466931257419757
Тестовий вектор 9: [0.7 1.3], передбачення: 1.970310908345931
```

4 Висновок

У ході виконання лабораторної роботи було реалізовано ймовірнісну нейронну мережу (PNN) для апроксимації функції $y = x_1 + x_2$. Модель була успішно навчена на невеликому наборі даних, і результати тестування показали, що модель здатна приблизно передбачати значення функції з певною похибкою.

Аналіз результатів показує, що:

- PNN здатна точно апроксимувати лінійні залежності, такі як $y = x_1 + x_2$, однак точність передбачень залежить від параметра σ.
- Модель демонструє хорошу здатність до узагальнення, особливо для тестових даних, що близькі до тренувальних.

Для покращення точності можна:

- Тонко налаштувати параметр σ, який впливає на ширину ядра Гауса і, відповідно, на чутливість до віддалених зразків.
- Використовувати більший та різноманітніший набір тренувальних даних, що допоможе покращити точність передбачень на нових вхідних значеннях.

Виконана робота дозволила отримати базові знання та практичні навички роботи з ймовірнісними нейронними мережами та показала їх можливості для задач апроксимації.

5 Лістинг програми

```
import numpy as np

class PNN:
    def __init__(self, sigma=0.1):
        self.sigma = sigma
        self.train_data = None
        self.train_targets = None

    def gaussian_kernel(self, x, x_i):
        return np.exp(-np.linalg.norm(x - x_i) ** 2 /
                              (2 * self.sigma ** 2))

    def train(self, X, y):
        self.train_data = X
        self.train_targets = y

    def predict(self, X):
        predictions = []

        for x in X:
            weights = np.array([self.gaussian_kernel(x, x_i) for x_i
                                        in self.train_data])
            weighted_sum = np.dot(weights, self.train_targets)
            prediction = weighted_sum / np.sum(weights)
            predictions.append(prediction)

        return np.array(predictions)

if __name__ == "__main__":
    # Різноманітні тренувальні приклади
```

```
    X_train = np.array([
        [0, 0],
        [0, 2],
        [1.5, 0],
        [1, 1],
        [0.3, 0.7],
        [0.2, 1.8],
        [1.7, 0.2],
        [1, 2],
        [2, 1]
    ])
    y_train = np.array([np.sum(x) for x in X_train])

    pnn = PNN(sigma=0.3)
    pnn.train(X_train, y_train)

    # Різноманітні тестові приклади
    X_test = np.array([
        [0.5, 1.5],
        [1, 1.5],
        [0.3, 1.3],
        [1.8, 0.2],
        [0.9, 1.1],
        [1.1, 1.9],
        [2, 2],
        [1.5, 1],
        [0.7, 1.3]
    ])

    predictions = pnn.predict(X_test)

    for i, (x, pred) in enumerate(zip(X_test, predictions)):
```

```
63          print(f"Тестовий вектор {i+1}: {x}, передбачення: {pred}")
```