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

Звіт

із лабораторної роботи №1

з дисципліни «Системи глибинного навчання»

на тему

“Розробка програмного забезпечення для реалізації двошарового персептрону з сигмоїдальною функцією активації”

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

Викладач:
Професор кафедри ПМА
Терейковський І. А.

Київ — 2023

Зміст

Теоретичні відомості

Персептрон є одним з основних видів штучних нейронних мереж, який становить базу для багатьох інших архітектур глибокого навчання. Винайдений Френком Розенблаттом в 1957 році, персептрон представляє собою простий бінарний класифікатор, здатний вирішувати проблеми лінійної роздільної природи. Основними складовими персептрона є входи, ваги, суматор, функція активації та вихід. Він має здатність навчатися шляхом коригування вагів під час тренування, що дозволяє розв'язувати більш складні задачі.

Багатошаровий персептрон (БП) є розширенням концепції персептрона, що включає багато шарів штучних нейронів. Кожен штучний нейрон у першому шарі приймає вхідні дані, обробляє їх та передає результати наступному шару. Зазвичай, багатошаровий персептрон складається з трьох типів шарів: вхідний, прихований та вихідний. Вхідний шар отримує вхідні дані, прихований шар виконує обчислення, а вихідний шар генерує остаточний результат.

Важливою особливістю багатошарового персептрона є його здатність вирішувати складні нелінійні проблеми завдяки використанню нелинійних функцій активації в прихованих шарах. Це дозволяє персептрону набагато ефективніше моделювати взаємозв'язки в даних, що робить його потужним інструментом для вирішення різноманітних завдань, таких як класифікація, регресія та розпізнавання образів

Основна частина

Частина 1

Завдання: розробити програмне забезпечення для реалізації класичного нейрону. Передбачити режим навчання на одному навчальному прикладі та режим розпізнавання.

Оскільки треба виконувати багато математичних розрахунків, імпортуємо бібліотеку numpy та зафіксуємо генерацію випадкових чисел.

В якості функції активації будемо використовувати сигмоїду:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Створимо функцію тренування. У циклі буде викликатися функція активації з переданим вектором, обчислюватися помилка навчання, оновлюватися вагові коефіцієнти нейрона та виводитися на екран інформація про першу та останню ітерації. Критерій завершення тренування визначатиметься проходженням всіх ітерацій.

Створимо екземпляр класу і проведемо його тренування на навчальних даних. Після завершення тренування модель перейде в режим розпізнавання, приймаючи на вхід значення, які вона не бачила раніше, та очікуватиме виводу, що відповідає її попередньому навчанню. У даному випадку очікуване значення Y = 0.9.

Результат частини 1:

```
Виконав студент КМ-03 | Шаповалов Г. Г. | Лаб 1


Частина 1

ITERARION № 1:
        Weights: [0.9663 0.702  1.003 ]
        Current output: 0.8507
        Error: 0.0063
        Weight update: [0.0063 0.0025 0.0031]

ITERARION № 100:
        Weights: [1.2128 0.8006 1.1263]
        Current output: 0.8904
        Error: 0.0009
        Weight update: [0.0009 0.0004 0.0005]

Режим розпізнавання:
Вектор [1.  0.2 0.7] == 0.8967
```

Мережа оновлює ваги і в режимі розпізнавання видає результат 0.8967, що дуже близько до очікуваного значення 0.9

Частина 2

Завдання: розробити програмне забезпечення для реалізації елементарного двошарового персептрону із структурою 1-1-1. Передбачити режим навчання на одному навчальному прикладі та режим розпізнавання.

Створюємо функцію тренування нейронної мережі. Ця функція виконує прямий та зворотній прохід під час якого змінює ваги нейронів тим самим навчає мережу.

Створюємо функцію розпізнавання, яка використовує дані, які мережа не бачила для отримання результату на який навчена мережа. Критерій зупинки навчання проходження всіх ітерацій.

Результат частини 2:

```
Частина 2

Тренувальні дані:
x = 0.7
y = 0.5
w12 = 0.2201
w23 = 0.3611

Тренування завершено
w12 = 0.1997
w23 = 0.0594

Режим розпізавання:
X = 0.35
Отриманий y = 0.5077
```

Мережа у режимі навчання змінила вагові коефіцієнти і в режимі розпізнавання видає результат 0.5077, що близько до очікуваного значення 0.5

Частина 3

Завдання: розробити програмне забезпечення для реалізації двошарового персептрону із структурою 2-3-1. Передбачити режим навчання «ON-LINE» та режим розпізнавання. Піддослідна функція x1+x2=y

Створюємо функцію тренування нейронної мережі. Виводить результати першої та останньої ітерації, для переконання, що мережа навчається.

Створюємо функцію розпізнавання, яка отримує значення x1 та x2 і прогнозує вихід.

Результат частини 3:

```
Частина 3

Тренувальні дані:
x1 = 0.2
x2 = 0.3
y = 0.5

ITERATION № 1:
W01 =
[[0.5326 0.2468 0.6708]
 [0.5595 0.5418 0.8919]]
W2 = [0.8374 0.3041 0.6273];
W02 = [0.0926 0.1974 0.295 ]
W03 = 0.3926
Y = 0.8491

ITERATION № 3000:
W01 =
[[0.3735 0.1946 0.562 ]
 [0.3208 0.4636 0.7287]]
W2 = [0.3473 0.1215 0.2519];
W02 = [-0.7032 -0.0632 -0.2492]
W03 = -0.4032
Y = 0.5


Режим розпізнавання:
x1 = 0.4
x2 = 0.1
Очікуваний y = 0.5
Отриманий  y = 0.4966
Різниця = 0.0034
```

Тренувальні дані x1 = 0.2 ; x2 = 0.3 ; y = 0.5

Бачимо, що мережа явно змінює значення вагових коефіцієнтів і покращує вихідне значення. В результаті в наслідок 3000 ітерацій мережа навчилась непогано виконувати завдання досягнувши результату на тестових даних – 0.4966. Це всього на 0.0034 менше від реального значення

Додаток А – Код програми

```
import numpy as np
np.random.seed(24)


class Part_1:
    def __init__(self, input_vector, weights, target_output):
        self.input_vector = input_vector
        self.weights = weights
        self.target_output = target_output

    def activation_function(self, x):
        weighted_sum = np.dot(self.weights, x)
        return 1 / (1 + np.exp(-weighted_sum))

    def train(self, max_iterations=100):
        iteration = 1
        while iteration <= max_iterations:
            #  Обчислення поточного виходу нейрона за допомогою активаційної
функції
            current_output = self.activation_function(self.input_vector)
            #  Обчислення помилки навчання
            error = current_output * (1 - current_output) * (self.target_output -
current_output)
            #  Обчислення оновлення ваг за допомогою вхідного вектора та помилки
            weight_update = self.input_vector * error
            # Оновлення ваг нейрона
            self.weights += weight_update

            if iteration == 1 or iteration == max_iterations:
                # Вивід інформації про першу та останню ітерації
                print(f'ITERARION № {iteration}:')
                print(f'\tWeights: {np.round(self.weights, 4)}')
                print(f'\tCurrent output: {np.round(current_output, 4)}')
                print(f'\tError: {np.round(error, 4)}')
                print(f'\tWeight update: {np.round(weight_update, 4)}\n')

            iteration += 1


class Part_2:
    def __init__(self, input_value, weights, target_output):
        self.x = input_value
        self.w = weights
        self.y = target_output


    def activation_function(self, x):
        return 1 / (1 + np.exp(-x))
```

```
    def train(self, iterations=100):
        for _ in range(iterations):
            #  Прямий прохід
            xs2 = self.x * self.w[0]
            y2 = self.activation_function(xs2)
            xs3 = y2 * self.w[1]
            y3 = self.activation_function(xs3)

            #  Зворотній прохід
            d3 = y3 * (1 - y3) * (self.y - y3)
            dw23 = y2 * d3
            self.w[1] += dw23

            d2 = y2 * (1 - y2) * (d3 * self.w[1])
            dw12 = self.x * d2
            self.w[0] += dw12

        self.print_training_info()


    def predict(self, input_value):
        self.x = input_value
        xs2 = self.x * self.w[0]
        y2 = self.activation_function(xs2)
        xs3 = y2 * self.w[1]
        y3 = self.activation_function(xs3)
        return y3


    def print_training_info(self):
        print(f'Тренування завершено \nw12 = {round(self.w[0], 4)} \nw23 =
{round(self.w[1], 4)}')


class Part_3:
    def __init__(self):
        self.w1 = np.random.rand(2, 3)
        self.w2 = np.random.rand(3)
        self.w02 = np.array([0.1, 0.2, 0.3])
        self.w03 = 0.4
        self.x1 = None
        self.x2 = None
        self.y = None
        self.yr = None


    def activation_function(self, xsi):
        return 1 / (1 + np.exp(-xsi))
```

```
    def train(self, x1, x2, y, epochs=3000):
        self.x1, self.x2, self.y = x1, x2, y

        for iteration in range(epochs):
            xs1_2 = self.calculate_x2_1()
            xs2_2 = self.calculate_x2_2()
            xs3_2 = self.calculate_x2_3()
            y1_2, y2_2, y3_2 = map(self.activation_function, [xs1_2, xs2_2, xs3_2])
            xs1_3 = self.calculate_x3_1(y1_2, y2_2, y3_2)
            y1_3 = self.activation_function(xs1_3)

            d1_3 = y1_3 * (1 - y1_3) * (self.y - y1_3)
            d1_2, d2_2, d3_2 = self.calculate_d2(d1_3, y1_2, y2_2, y3_2)

            #  Оновлення вагових коефцієнтів
            self.update_weights(d1_2, d2_2, d3_2, y1_2, y2_2, y3_2)
            self.yr = y1_3  # Оновлення self.yr до поточного значення виходу моделі
            if iteration in [0, epochs-1]:
                self.print_iteration_info(iteration)

        self.yr = y1_3


    def predict(self, x1, x2):
        """
        Розпізнавання
        """
        self.x1, self.x2 = x1, x2
        xs1_2 = self.calculate_x2_1()
        xs2_2 = self.calculate_x2_2()
        xs3_2 = self.calculate_x2_3()
        y1_2, y2_2, y3_2 = map(self.activation_function, [xs1_2, xs2_2, xs3_2])
        xs1_3 = self.calculate_x3_1(y1_2, y2_2, y3_2)
        y1_3 = self.activation_function(xs1_3)
        return y1_3


    def calculate_x2_1(self):
        """
        Обчислення вагованої суми для першого прихованого шару
        """
        return self.w02[0] + np.sum(self.w1[0] * self.x1) + np.sum(self.w1[1] *
self.x2)


    def calculate_x2_2(self):
        """
        Обчислення вагованої суми для другого прихованого шару
        """
        return self.w02[1] + np.sum(self.w1[0] * self.x1) + np.sum(self.w1[1] *
self.x2)
```

```
    def calculate_x2_3(self):
        """
        Обчислення вагованої суми для третього прихованого шару
        """
        return self.w02[2] + np.sum(self.w1[0] * self.x1) + np.sum(self.w1[1] *
self.x2)


    def calculate_x3_1(self, y1_2, y2_2, y3_2):
        """
        Обчислення вагованої суми для вихідного шару
        """
        return self.w03 + np.sum(self.w2 * np.array([y1_2, y2_2, y3_2]))


    def calculate_d2(self, d1_3, y1_2, y2_2, y3_2):
        """
        Обчислення дельт для прихованого шару
        """
        d1_2 = y1_2 * (1 - y1_2) * d1_3 * self.w2[0]
        d2_2 = y2_2 * (1 - y2_2) * d1_3 * self.w2[1]
        d3_2 = y3_2 * (1 - y3_2) * d1_3 * self.w2[2]
        return d1_2, d2_2, d3_2


    def update_weights(self, d1_2, d2_2, d3_2, y1_2, y2_2, y3_2):
        """
        Оновлення вагових коефіцієнтів
        """
        self.w1[0] += np.array([d1_2 * self.x1, d2_2 * self.x1, d3_2 * self.x1])
        self.w1[1] += np.array([d1_2 * self.x2, d2_2 * self.x2, d3_2 * self.x2])
        self.w2 += np.array([d1_2 * y1_2, d2_2 * y2_2, d3_2 * y3_2])
        self.w02 += np.array([d1_2, d2_2, d3_2])
        self.w03 += d1_2


    def print_iteration_info(self, iteration):
        """
        W01 = [[w01_1, w11_1, w12_1],
               [w02_1, w21_1, w22_1]];
        W2  = [w01_2, w11_2, w21_2];
        W02 = [w01_3, w02_3, w03_3];
        W03 = w01_4;
        Y   = yr;
        """
        print(f'ITERATION № {iteration + 1}:\n'
              f'W01 = \n{np.round(self.w1, 4)}\n'    #  Ваги між вхідним шаром і
першим прихованим шаром
              f'W2 = {np.round(self.w2, 4)};\n'    #  Ваги між прихованим шаром і
вихідним шаром
```

```
            f'W02 = {np.round(self.w02, 4)}\n'  #  Ваги для зсуву в першому
прихованому шарі
            f'W03 = {np.round(self.w03, 4)}\n'  #  Зсув в другому прихованому
шарі
            f'Y = {np.round(self.yr, 4)}\n')    #  Вихід моделі


if __name__ == '__main__':
    print(f'\n\nВиконав студент КМ-03 | Шаповалов Г. Г. | Лаб 1\n\n')


    '''
    Частина 1
    Завдання:
    Розробити програмне забезпечення для реалізації класичного нейрону.
    Передбачити режим навчання на одному навчальному прикладі та режим
розпізнавання.
    '''
    print(f'Частина 1\n')
    train_vector = np.array([1, 0.4, 0.5])  # Вектор навчальних даних
    weights_1 = np.random.rand(3)  # Ініціалізуємо ваги випадковим чином
    target_output_1 = 0.9  # Очікуване значення Y (повертається функцією активації)

    # Режим навчання
    model_1 = Part_1(train_vector, weights_1, target_output_1)
    model_1.train()

    # Режим розпізнавання
    predict_vector = np.array([1, 0.2, 0.7])

    # Вивід
    output = model_1.activation_function(predict_vector)
    print(f'Режим розпізнавання: \nВектор {predict_vector} == {np.round(output,
4)}\n\n\n')


    '''
    Частина 2
    Завдання:
    Розробити програмне забезпечення для реалізації елементарного двошарового
персептрону із структурою 1-1-1.
    Передбачити режим навчання на одному навчальному прикладі та режим
розпізнавання.
    '''
    print(f'Частина 2\n')
    x = 0.7
    y = 0.5
    w = np.random.rand(2)

    print(f'Тренувальні дані: \nx = {x} \ny = {y} \nw12 = {round(w[0], 4)} \nw23 =
{round(w[1], 4)} \n')
```

```
    # Режим навчання
    model_2 = Part_2(x, w, y)
    model_2.train()

    # Режим розпізнавання
    predict_x = 0.35
    predict_y = model_2.predict(predict_x)
    print(f'\nРежим розпізавання: \nX = {predict_x} \nОтриманий y =
{round(predict_y, 4)}\n\n\n')


    '''
    Частина 3
    Завдання:
    Розробити програмне забезпечення для реалізації двошарового персептрону із
структурою 2-3-1
    Передбачити режим навчання «ON-LINE» та режим розпізнавання.
    Піддослідна функція x1+x2=y
    '''
    print(f'Частина 3\n')
    # Початкові вагові коефіцієнти
    w1 = np.random.rand(2, 3)
    w2 = np.random.rand(3)
    w02 = np.array([0.1, 0.2, 0.3])
    w03 = 0.4


    # Тренувальні дані
    x1 = 0.2
    x2 = 0.3
    y = 0.5

    print(f'Тренувальні дані: \nx1 = {x1} \nx2 = {x2} \ny = {y}\n')
    model_3 = Part_3()
    model_3.train(x1, x2, y)


    # Тестові дані
    x1 = 0.4
    x2 = 0.1
    print(f'\nРежим розпізнавання: \nx1 = {x1} \nx2 = {x2}')
    y_real = x1 + x2
    print(f'Очікуваний y = {y_real}')
    y_pred = model_3.predict(x1, x2)
    print(f'Отриманий  y = {round(y_pred, 4)}')
    print(f'Різниця = {round(abs(y_real - y_pred), 4)}')
```