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-01
Іваник Ю. П.

Викладач:
Професор кафедри ПМА
Терейковський І. А.

Київ — 2023

Зміст

Теоретичні відомості

Персептрон - це проста модель штучного нейрона, яка служить основою для багатьох моделей глибокого навчання. Введений Френком Розенблаттом в 1957 році, персептрон можна розглядати як математичну модель, що намагається імітувати роботу людського мозку. Він складається з одного штучного нейрона, який приймає вхідні сигнали, ваговує їх і видає вихід. Основна ідея полягає в тому, що ваги, які призначаються вхідним сигналам, визначають важливість кожного сигналу для вихідного результату.

Багатошаровий персептрон (БП) розширює ідею простого персептрона, дозволяючи використовувати багато штучних нейронів, організованих у шари. Він має вхідний шар, один або декілька прихованих шарів і вихідний шар. Кожен нейрон у кожному шарі пов'язаний з кожним нейроном на наступному шарі. Ваги, які призначаються цим зв'язкам, навчаються під час тренування мережі, де вона підбирає ваги так, щоб максимізувати точність прогнозів.

Багатошаровий персептрон дозволяє моделювати складні взаємозв'язки в даних, такі як нееліптичні рішення або взаємодії між різними функціями. Ця архітектура допомагає вирішити проблему лінійної роздільності, яка є обмеженням для простих персептронів. БП здатний вирішувати завдання класифікації та регресії, а також використовується у багатьох сучасних моделях глибокого навчання, таких як нейронні мережі.

Основна частина

Частина 1

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

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, в рамках якої буде викликатися активаційна функція з переданим вектором, обчислюватися помилка навчання, оновлюватися вагові коефіцієнти нейрона та виводитися інформація про початок та завершення кожної ітерації у циклі. Умова завершення тренування буде визначатися досягненням всіх ітерацій.

Створимо екземпляр класу і проведемо його тренування на навчальних даних. Після завершення тренування модель перейде у режим розпізнавання, приймаючи на вхід значення, які вона не зустрічала раніше, та очікуватиме виводу, що відповідає її попередньому навчанню.

```
Режим навчання НМ:
X = [1.  0.4 0.3]
y = 0.7

Ітерація 1:
        Ваги: [0.5036 0.2317 0.2582]
        Вихід: 0.6598
        Помилка: 0.009
        Оновлення вагів: [0.009  0.0036 0.0027]

Ітерація 2:
        Ваги: [0.512  0.2351 0.2607]
        Вихід: 0.6623
        Помилка: 0.0084
        Оновлення вагів: [0.0084 0.0034 0.0025]

Ітерація 99:
        Ваги: [0.642  0.287  0.2997]
        Вихід: 0.6999
        Помилка: 0.0
        Оновлення вагів: [0. 0. 0.]

Ітерація 100:
        Ваги: [0.642  0.287  0.2997]
        Вихід: 0.6999
        Помилка: 0.0
        Оновлення вагів: [0. 0. 0.]


Режим розпізнавання:
Вектор [0.1 0.6]  == 0.7007
```

Рис. 1 – Виконання коду частини 1

НМ справилась досить добре, досягнув результату 0.7007 при очікуваному 0.7

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.

Розробляємо функцію навчання нейронної мережі, яка здійснює як прямий, так і зворотний прохід, внаслідок чого змінюються ваги нейронів і, отже, навчає мережу.

Створюємо функцію розпізнавання, яка використовує дані, що не були використані мережею під час тренування, для отримання результату, на який вона була навчена. Умова завершення навчання визначається пройденим усіма ітераціями.

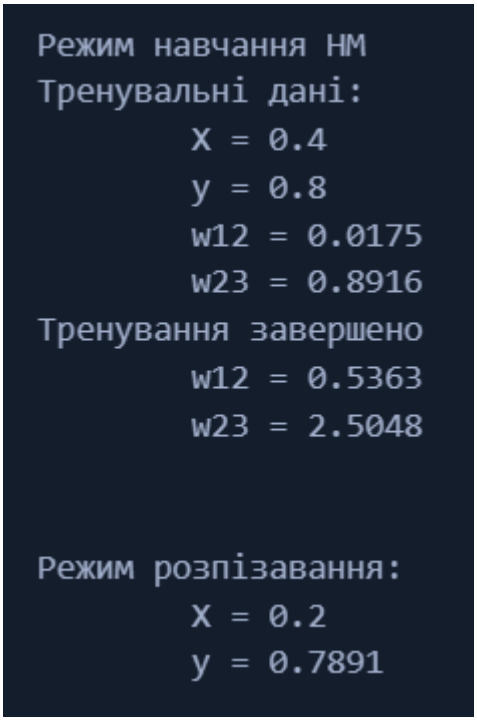

Рис. 2 - Виконання коду частини 2

НМ навчилась на тестовому прикладі. Це можна побачити по тому, що вагові коефіцієнти змінились та у режимі розпізнавання мережа видає значення 0.7891, що досить близько до очікуваного результату 0.8

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

Розробляємо функцію тренування нейронної мережі, яка виводить результати першої та останньої ітерації, щоб переконатися, що мережа вчиться.

Розробляємо функцію розпізнавання, яка приймає значення x1 та x2 і здійснює прогноз виходу.

```
Режим навчання НМ:
x1 = 0.1
x2 = 0.6
y = 0.7

Ітерація 1:
W01:
[[0.8507 0.73   0.1087]
 [0.8935 0.8571 0.165 ]]
W2 = [0.6318 0.0205 0.1166];
W02 = [0.0993 0.2    0.2999]
W03 = 0.3993
Y = 0.7356

Ітерація 2:
W01:
[[0.8507 0.73   0.1087]
 [0.8931 0.8571 0.165 ]]
W2 = [0.6312 0.0204 0.1166];
W02 = [0.0986 0.2    0.2998]
W03 = 0.3986
Y = 0.7354
```

```
Ітерація 999:
W01:
[[0.8416 0.7297 0.1073]
 [0.8387 0.8555 0.1563]]
W2 = [0.5592 0.0182 0.1045];
W02 = [0.008  0.1973 0.2853]
W03 = 0.308
Y = 0.7

Ітерація 1000:
W01:
[[0.8416 0.7297 0.1073]
 [0.8387 0.8555 0.1563]]
W2 = [0.5592 0.0182 0.1045];
W02 = [0.008  0.1973 0.2853]
W03 = 0.308
Y = 0.7



Режим розпізнавання:
x1 = 0.3
x2 = 0.4
Predicted  y = 0.6992
Очікуваний y = 0.7
Difference 0.0008
```

Тренувальні дані мають вигляд x1 = 0.1, x2 = 0.6, y = 0.7. Видно, що мережа модифікує значення вагових коефіцієнтів та поліпшує вихідний результат. Після проведення 1000 ітерацій мережа ефективно вивчила завдання, досягаючи результату на тестових даних - 0.6992 Це лише на 0.0008 менше за фактичне значення.

Додаток А – Код програми

```
# КМ-01 | Іваник Юрій | Лаб 1

import numpy as np

## Частина 1

### Завдання:

_Розробити програмне забезпечення для реалізації класичного нейрону._

_Передбачити режим навчання на одному навчальному прикладі та режим розпізнавання._

class Task1:
    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 in [1, 2, max_iterations-1, max_iterations]:
            print(f'Ітерація {iteration}:')
            print(f'\tВаги: {np.round(self.weights, 4)}')
            print(f'\tВихід: {np.round(current_output, 4)}')
            print(f'\tПомилка: {np.round(error, 4)}')
            print(f'\tОновлення вагів: {np.round(weight_update, 4)}\n')

        iteration += 1

np.random.seed(50)
train_vector = np.array([1, 0.4, 0.3])  #  Вектор навчальних даних
weights_1 = np.random.rand(3)  #  Ініціалізуємо ваги випадковим чином
target_output_1 = 0.7  #  Очікуване значення Y (повертається функцією активації)

print(f'Режим навчання НМ: \nX = {train_vector} \ny = {target_output_1}\n')
model_1 = Task1(train_vector, weights_1, target_output_1)
model_1.train()

#  Вхідні дані для нейрона (Режим розпізнавання)
predict_vector = np.array([1, 0.1, 0.6])
```

```
#  Вивід результату для нейрона
output = model_1.activation_function(predict_vector)
print(f'\nРежим розпізнавання: \nВектор [{predict_vector[1]} {predict_vector[-1]}] == {np.round(output, 4)}')
```

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.

```
class Task2:
    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=1000):
        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 \n\tw12 = {round(self.w[0], 4)} \n\tw23 = {round(self.w[1], 4)}')



np.random.seed(48)

x = 0.4
y = 0.8
w = np.random.rand(2)

print(f'Режим навчання НМ')
```

```
print(f'Тренувальні дані: \n\tX = {x} \n\ty = {y} \n\tw12 = {round(w[0], 4)} \n\tw23 = {round(w[1], 4)}')


model_2 = Task2(x, w, y)
model_2.train()


predict_x = 0.2
predict_y = model_2.predict(predict_x)
print(f'\n\nРежим розпізавання: \n\tX = {predict_x} \n\ty = {round(predict_y, 4)}')
```

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

```
class Task3:
    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=1_000):
        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)

            #  Зворотнє поширення помилки
            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, 1, epochs-2, epochs-1]:  #  друкуємо лише 1, 2 та останню ітерації
                self.print_iteration_info(iteration)

        self.yr = y1_3


    def predict(self, x1, x2):
```

```
    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):
    return self.w02[0] + np.sum(self.w1[0] * self.x1) + np.sum(self.w1[1] * self.x2)


def calculate_x2_2(self):
    return self.w02[1] + np.sum(self.w1[0] * self.x1) + np.sum(self.w1[1] * self.x2)


def calculate_x2_3(self):
    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):
    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):
    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):
        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'Ітерація {iteration + 1}:\n'      #  Номерація
              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
```

```
np.random.seed(11)
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 = 0.1
x2 = 0.6
y = 0.7

print(f'Режим навчання НМ: \nx1 = {x1} \nx2 = {x2} \ny = {y}\n')
model_3 = Task3()
model_3.train(x1, x2, y)


#  Тестувальні дані
x1 = 0.3
x2 = 0.4
print(f'\n\nРежим розпізнавання: \nx1 = {x1} \nx2 = {x2}')
y_pred = model_3.predict(x1, x2)
print(f'Predicted  y = {round(y_pred, 4)}')
y_real = x1 + x2
print(f'Очікуваний y = {y_real}')
print(f'Difference {round(abs(y_real - y_pred), 4)}')
```