НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

лабораторної роботи №1

з дисципліни «[СИСТЕМИ ГЛИБИННОГО НАВЧАННЯ](https://classroom.google.com/c/NjIxMzk4NjAyNjI4)»

на тему

«Розробка програмного забезпечення для реалізації  
двошарового персептрону з сигмоїдальною функцією активації»

|  |  |
| --- | --- |
| Виконав:  Студент групи КМ-03  Мітченко А.Д. | Перевірив:  доцент  Терейковський І. А. |
|  |  |
|  |  |
|  |  |

Київ — 2023

ЗМІСТ

[Теоретичні відомості 3](#_Toc147935922)

[Частина 1: Реалізація класичного нейрону 3](#_Toc147935923)

[Частина 2: Реалізація елементарного двошарового персептрону (1-1-1) 3](#_Toc147935924)

[Частина 3: Реалізація двошарового персептрону зі структурою 2-3-1 4](#_Toc147935925)

[Скріншоти та опис програми 5](#_Toc147935926)

[ДОДАТОК А ТЕКСТ ПРОГРАМИ 10](#_Toc147935927)

[Завдання 1: 10](#_Toc147935928)

[Інший варіант виконання 1го завдання 11](#_Toc147935929)

[Завдання 2: 14](#_Toc147935930)

[Завдання 3: 16](#_Toc147935931)

# Теоретичні відомості

## Частина 1: Реалізація класичного нейрону

В цій частині нам потрібно реалізувати класичний нейрон, який має один вхід і один вихід. Класичний нейрон використовує функцію активації (наприклад, сигмоїдальну функцію) для обчислення вихідного значення на основі вхідного сигналу та ваг.

Тут основні концепції:

**Вхід і ваги:** Класичний нейрон має один вхід та ваги, які використовуються для обчислення вихідного значення. Ваги визначають, наскільки важливий кожен вхід для вихідного сигналу.

**Функція активації:** Функція активації приймає взважену суму вхідних значень та регулює це значення для створення вихідного сигналу. Сигмоїдальна функція активації широко використовується в класичних нейронах.

**Навчання:** Під час навчання нейрону, ваги оновлюються, щоб вони відповідали бажаному вихідному значенню на основі навчального прикладу. Оновлення ваг виконується за допомогою методу спуску градієнта або інших алгоритмів навчання.

## Частина 2: Реалізація елементарного двошарового персептрону (1-1-1)

У цій частині нам потрібно створити простий двошаровий персептрон зі структурою 1-1-1. Це означає, що у нас є один вхідний нейрон, один прихований нейрон та один вихідний нейрон. Ось ключові поняття:

**Вхідний шар (1-й шар):** Вхідний шар містить один нейрон, який приймає вхідні дані.

**Прихований шар (2-й шар):** Другий шар містить один прихований нейрон, який використовує функцію активації для обчислення свого вихідного значення на основі входу.

**Вихідний шар (3-й шар):** Вихідний шар також містить один нейрон, який генерує вихідний сигнал.

**Функція активації:** Функція активації використовується в прихованому нейроні та вихідному нейроні для обчислення їх вихідних значень. В нашому випадку це сигмоїда.

**Навчання:** Під час навчання персептрону, ваги вагонів оновлюються за допомогою методу спуску градієнта або інших алгоритмів навчання для досягнення бажаного вихідного значення на основі навчального прикладу.

## Частина 3: Реалізація двошарового персептрону зі структурою 2-3-1

У цій частині нам потрібно створити двошаровий персептрон зі структурою 2-3-1. Це означає, що у нас є два вхідні нейрони, три прихованих нейрона та один вихідний нейрон. Основні концепції аналогічні до частини 2, але цей персептрон має більше входів і нейронів, що робить його потужнішим для навчання та розпізнавання складніших залежностей в даних.

# Скріншоти та опис програми

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

# ДОДАТОК А ТЕКСТ ПРОГРАМИ

## Завдання 1:

розробити програмне забезпечення для реалізації

класичного нейрону. Передбачити режим навчання на одному навчальному прикладі

та режим розпізнавання.

# Імпорт бібліотек

import numpy as np

import math

import random

from numpy import exp, array, random, dot

# Визначення вхідних даних для навчального набору

training\_set\_inputs = array([[0, 0, 1]])

# Визначення вихідних даних для навчального набору та транспонування їх

training\_set\_outputs = array([[0]]).T

# Ініціалізація вагових коефіцієнтів нейронної мережі випадковими значеннями

random.seed(1)

synaptic\_weights = 2 \* random.random((3, 1)) - 1

# Навчання нейронної мережі (в даному випадку - 10000 ітерацій)

for iteration in range(10000):

# Обчислення вихідних значень нейрону за допомогою функції сигмоїди

output = 1 / (1 + exp(-(dot(training\_set\_inputs, synaptic\_weights))))

# Коригування вагових коефіцієнтів нейрону на основі отриманих результатів

synaptic\_weights += dot(training\_set\_inputs.T, (training\_set\_outputs - output) \* output \* (1 - output))

# Виведення прогнозу для нових даних

new\_input = array([0, 0, 1])

predicted\_output = 1 / (1 + exp(-(dot(new\_input, synaptic\_weights))))

print(predicted\_output)

## Інший варіант виконання 1го завдання

# Даємо визначення сигмоїдної функції активації

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Визначаємо похідну сигмоїдної функції

def sigmoid\_derivative(x):

return x \* (1 - x)

class ClassicNeuron:

def \_\_init\_\_(self):

# Ініціалізуємо випадкові ваги та поріг

self.weights = np.random.uniform(size=(1, 1))

self.bias = np.random.uniform()

def train(self, X\_train, y\_train, learning\_rate, epochs):

for epoch in range(epochs):

# Проходимо через всі навчальні дані

for i in range(len(X\_train)):

# Вираховуємо вагований вхід та відповідь

weighted\_sum = np.dot(X\_train[i], self.weights) + self.bias

prediction = sigmoid(weighted\_sum)

# Рахуємо помилку

error = y\_train[i] - prediction

# Обчислюємо внутрішню помилку (похідна помножена на помилку)

d\_error = error \* sigmoid\_derivative(prediction)

# Оновлюємо ваги та поріг

self.weights += learning\_rate \* np.dot(X\_train[i].reshape(-1, 1), d\_error)

self.bias += learning\_rate \* d\_error

def predict(self, X\_test):

weighted\_sum = np.dot(X\_test, self.weights) + self.bias

prediction = sigmoid(weighted\_sum)

return prediction

# Навчальні дані (приклад: x=4, y=1)

X\_train = np.array([[4]])

y\_train = np.array([[1]])

# Створюємо класичний нейрон

neuron = ClassicNeuron()

# Навчаємо нейрон

neuron.train(X\_train, y\_train, learning\_rate=0.1, epochs=10000)

# Режим розпізнавання

X\_test = np.array([[6]]) # Вхід для розпізнавання

prediction = neuron.predict(X\_test)

print(f"For input {X\_test[0][0]}, predicted output: {prediction[0][0]:.2f}")

#У цьому коді ми створюємо клас ClassicNeuron, який містить методи для навчання та розпізнавання. Ми навчаємо класичний нейрон на одному навчальному прикладі (x=2, y=0) і після навчання використовуємо його для розпізнавання в режимі розпізнавання (для x=3).

# Функція для здійснення прогнозу з нейронної мережі

# x: Вхідні дані (вектор)

def predict(x):

# Обчислення входу до прихованого шару

hidden\_layer\_input = np.dot(x, weights\_input\_hidden) + bias\_hidden

# Застосування активаційної функції (сигмоїда) до входу прихованого шару

hidden\_layer\_output = sigmoid(hidden\_layer\_input)

# Обчислення входу до вихідного шару

output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output

# Застосування активаційної функції (сигмоїда) до входу вихідного шару

output\_layer\_output = sigmoid(output\_layer\_input)

# Повернення вихідних значень вихідного шару (прогноз)

return output\_layer\_output

## Завдання 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.

# Визначимо навчальні дані: вхідні (x) і цільові (y) значення

X\_train = np.array([[0.5]]) # Input (x)

y\_train = np.array([[1]]) # Target (y) for y = 2x

# Встановлюємо випадкове початкове число для відтворюваності

np.random.seed(42)

# Ініціалізація ваги та зміщення випадковим чином

input\_neurons = 1

hidden\_neurons = 1

output\_neurons = 1

weights\_input\_hidden = np.random.uniform(size=(input\_neurons, hidden\_neurons))

weights\_hidden\_output = np.random.uniform(size=(hidden\_neurons, output\_neurons))

bias\_hidden = np.random.uniform(size=(1, hidden\_neurons))

bias\_output = np.random.uniform(size=(1, output\_neurons))

# Встановлюємо швидкість навчання

learning\_rate = 0.1

# Тренування персептрона

epochs = 10000

for epoch in range(epochs):

# Forward propagation

hidden\_layer\_input = np.dot(X\_train, weights\_input\_hidden) + bias\_hidden

hidden\_layer\_output = sigmoid(hidden\_layer\_input)

output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output

output\_layer\_output = sigmoid(output\_layer\_input)

# Calculate the error

error = y\_train - output\_layer\_output

# Зворотне поширення

d\_output = error \* sigmoid\_derivative(output\_layer\_output)

error\_hidden\_layer = d\_output.dot(weights\_hidden\_output.T)

d\_hidden\_layer = error\_hidden\_layer \* sigmoid\_derivative(hidden\_layer\_output)

# Оноввлюємо ваги та зміщення

weights\_hidden\_output += hidden\_layer\_output.T.dot(d\_output) \* learning\_rate

weights\_input\_hidden += X\_train.T.dot(d\_hidden\_layer) \* learning\_rate

bias\_output += np.sum(d\_output, axis=0, keepdims=True) \* learning\_rate

bias\_hidden += np.sum(d\_hidden\_layer, axis=0, keepdims=True) \* learning\_rate

x\_test = np.array([[0.75]]) # Test input

prediction = predict(x\_test)

print(f"For input {x\_test[0][0]}, predicted output: {prediction[0][0]:.2f}")

## Завдання 3:

Завдання: розробити програмне забезпечення для реалізації двошарового персептрону із структурою 2-3-1. Передбачити режим навчання «ON-LINE» та режим розпізнавання.

Піддослідна функція х1+х2=у

X\_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Input features (x1 and x2)

y\_train = np.array([[0], [1], [1], [2]]) # Corresponding labels (y = x1 + x2)

np.random.seed(42)

input\_neurons = 2

hidden\_neurons = 3

output\_neurons = 1

weights\_input\_hidden = np.random.uniform(size=(input\_neurons, hidden\_neurons))

weights\_hidden\_output = np.random.uniform(size=(hidden\_neurons, output\_neurons))

bias\_hidden = np.random.uniform(size=(1, hidden\_neurons))

bias\_output = np.random.uniform(size=(1, output\_neurons))

learning\_rate = 0.1

epochs = 10000

for epoch in range(epochs):

for i in range(len(X\_train)):

input\_data = X\_train[i]

hidden\_layer\_input = np.dot(input\_data, weights\_input\_hidden) + bias\_hidden

hidden\_layer\_output = sigmoid(hidden\_layer\_input)

output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output

output\_layer\_output = sigmoid(output\_layer\_input)

error = y\_train[i] - output\_layer\_output

d\_output = error \* sigmoid\_derivative(output\_layer\_output)

error\_hidden\_layer = d\_output.dot(weights\_hidden\_output.T)

d\_hidden\_layer = error\_hidden\_layer \* sigmoid\_derivative(hidden\_layer\_output)

weights\_hidden\_output += np.outer(hidden\_layer\_output, d\_output) \* learning\_rate

weights\_input\_hidden += np.outer(input\_data, d\_hidden\_layer) \* learning\_rate

bias\_output += d\_output \* learning\_rate

bias\_hidden += d\_hidden\_layer \* learning\_rate

for i in range(len(X\_train)):

input\_data = X\_train[i]

prediction = predict(input\_data)

print(f"For input {input\_data}, predicted output: {prediction[0][0]:.2f} (Actual: {y\_train[i][0]})")