НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи №1-2

із дисципліни «Методи штучного інтелекту»

на тему

*Розробка програмного забезбепечення для реалізації двошарового персептрону х сигмоїдальною функцією активації*

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студентки групи КМ-81 | Доцент |
| *Верзун П. В.* | Терейковська Л.О. |

Київ — 2021

**Зміст**

[**Постановка задачі**](#_zfjl9hdpxc7o) **3**

[**Короткі теоретичні відомості**](#_tqcqjv38h9xn) **4**

[**Результат роботи програми**](#_vvk8gcp26g0t) **7**

[**Висновки**](#_otm11ky3mm47) **8**

# **Постановка задачі**

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

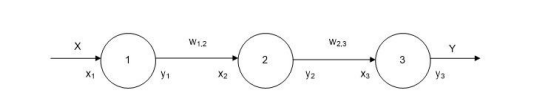
y\_r = 0.1

x = 2

dd = 0.1

# **Короткі теоретичні відомості**

В даній лабораторній роботі будемо реалізовувати двошаровий персептрон із структурою 1-1-1, який представлений на малюнку нижче:



Де активаційна функція в першому нейроні лінійна.

Потім до другого нейрону поступає сума зважених входів y1\*w12. Після чого в другому нейроні в якості активаційної функції застосовується сигмоїда. Далі до 3 нейрона подається сума зважених входів w23\*y2. Після чого на виході отримуємо результат від сигмоїди.

Під час режиму навчання буде застосовано зворотне розповсюдження помилки, яке здійснюється за правилом обчислення похідної функції,далі застосовується градієнтний спуск.

**Код програми з коментарями**

class TwoLayerPerceptron:

# this is embarassing

def \_\_init\_\_(self, x, y\_r, dd):

self.w1 = random()

self.w2 = random()

self.\_x = x

self.y\_r = y\_r

self.dd = dd

self.y2 = None

self.y3 = None

self.i = 1

@property

def x(self):

return self.\_x

@x.setter

def x(self, x):

self.\_x = x

@staticmethod

def sigmoid(x\_s):

return 1 / (1 + exp(-x\_s))

def train(self):

while True:

x2\_s = self.\_x \* self.w1

self.y2 = self.sigmoid(x2\_s)

x3\_s = self.y2 \* self.w2

self.y3 = self.sigmoid(x3\_s)

dn = abs((self.y\_r - self.y3) / self.y\_r)

if dn < self.dd:

print(f"Answer: {self.y3}")

return 0

else:

q3 = self.y3 \* (1 - self.y3) \* (self.y\_r - self.y3)

q2 = self.y2 \* (1 - self.y2) \* (q3 \* self.w2)

delta\_w2 = q3 \* self.y2

delta\_w1 = q2 \* self.\_x

self.w1 += delta\_w1

self.w2 += delta\_w2

self.i += 1

print(f"\n Iteration: {self.i} \n y3 = {self.y3} \n dn = {dn}")

В класі ініціалізації задаються всі необхідні параметри, а саме вхідні сигнали, допустиму похибку, очікуваний результат, випадково задані ваги для вхідних сигналів.

В функції активації реалізована сигмоїда.

Функція train() реалізує тренування двошарового персептрону. Рахуємо зважений вхід x2\_s, проганяємо через функцію активації. Рахуємо зважений вхід x3\_s, проганяємо через функцію активації. Оцінюємо похибку, в випадку невдачі, рахуємо q2, q3, дельти та вираховуємо нові ваги. В результаті на виході отримуємо натренований двошаровий персептрон та відповідь y3.

perceptron = TwoLayerPerceptron(2, 0.1, 0.1)

perceptron.train()

perceptron.x = 2.4

z1 = perceptron.x \* perceptron.w1

z2 = perceptron.sigmoid(z1) \* perceptron.w2

print(f"\n Detection: {perceptron.sigmoid(z2)}")

Створюємо екземпляр класу, тренуємо його. Задаємо нове значення х та перераховуємо зважені входи з вже натренерованими вагами. Вираховуємо значення функції активації та виводимо його в консоль.

# **Результат роботи програми**

Task2

...

Iteration: 228

y3 = 0.11047235478963312

dn = 0.10472354789633112

Iteration: 229

y3 = 0.11037406843023083

dn = 0.10374068430230823

Iteration: 230

y3 = 0.1102768509203501

dn = 0.10276850920350092

Iteration: 231

y3 = 0.11018068785088735

dn = 0.1018068785088734

Iteration: 232

y3 = 0.11008556506123311

dn = 0.10085565061233107

Answer: 0.10999146863389404

Detection: 0.1030316522580736

# **Висновки**

Розроблено програмне забезпечення, що реалізує логіку двошарового персептрона з схемою 1-1-1. Мережа навчилась за 232 ітерації. Кількість ітерацій напряму залежить від вагів, що в нашому випадку, задані рандомно при ініціалізації класу. Більш того, мережа повернула результат 0.10999, що є близьким до тестового запуску. а в режимі розпiзнавання 0.10303. dn = 0.10085565061233107