МIНIСТЕРСТВО ОСВIТИ I НАУКИ УКРАЇНИ

НАЦIОНАЛЬНИЙ ТЕХНIЧНИЙ УНIВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛIТЕХНIЧНИЙ IНСТИТУТ»

Кафедра прикладної математики

Звіт

із лабораторної роботи №1

з дисципліни «Методи штучного інтелекту»

на тему:

«Багатошаровий персептрон»

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студент групи КМ-63 | *Терейковська Л. О.* |
| *Артеменко Я.К.* |  |

Київ — 2019

# **ЗМІСТ**

[**ВСТУП** 3](#_Toc20169821)

[**2 ОСНОВНА ЧАСТИНА** 4](#_Toc20169822)

[**2.1 Завдання** 4](#_Toc20169823)

[**2.2 Описання алгоритму** 5](#_Toc20169824)

[**ВИСНОВКИ** 6](#_Toc20169825)

[**ДОДАТКИ** 7](#_Toc20169826)

[**Додаток А (код програми)** 7](#_Toc20169827)

[**Додаток Б (результати роботи)** 11](#_Toc20169828)

# **ВСТУП**

Дана лабораторна робота складається з трьох частин, кожна з якої має відмінність в алгоритмі, оскільки для кожної частини лабораторної роботи використовуються різні схеми нейронів.

А першій яастині проводиться навчання одного нейрону, використовуючи вектор Х вхідних даних. Друга частина реалізує персептрон 1-1-1, третя частина реаліхує схему 2-3-1.

Потрібно запрограмувати запропоновані алгоритми для поставлених задач.

# **2 ОСНОВНА ЧАСТИНА**

# **2.1 Завдання**

Для першої частини лабораторної роботи на вхід подається вектор Х, який дорівнює [1,7,4,5] та задано Y очікуване, з яких потім очікуємо вихід. Для кожного з X задається своя вага. Потім дані обчислюються за допомогою певних формул, послідовно по заданому алгоритму.

Для другої частини лабораторної на вхід подається один X і є три нейрони. Один нейрон являється вхідним і він приймає значення ікс без змін. Другий нейрон множить значення ікса на відповідну вагу і так далі по заданому алгоритму.

Розробити програмне забезпечення для реалізації елементарного двошарового персептрону з структурою 2-3-1. Передбачити режим навчання online та режим розпізнавання.

Рахуємо помилку відносно вихідного нейрону для схованого нейрону:

def beck(self):

res = []  
 sigma = self.y \* (1 - self.y) \* (y - self.y)  
 for i in range(3):  
 res.append(sigma \* self.w[i])  
 return res

Визначаємо ваги для нерону схованого шару:

def \_\_init\_\_(self, counr, i):  
 self.i = i  
 # w = [[-0.1, 0.1, 0.2], [-0.2, 0.5, 0.3], [-0.1, 0.1, 0.2]]  
 w = []  
 for i in range(counr + 1):  
 w.append(uniform(-1, 1))  
 self.w = w

Кількість епох:

for i in range(500000):  
 ww = []  
 print("Епоха " + str(i + 1))

Навчання нейронів:

y1.append(neuron\_1\_1.training(x[j]))  
y1.append(neuron\_1\_2.training(x[j]))  
y2.append(neuron\_2\_1.training(y1))  
y2.append(neuron\_2\_2.training(y1))  
y2.append(neuron\_2\_3.training(y1))  
y3.append(neuron\_3\_1.training(y2))

Зворотній прохід:

neuron\_3\_1.correction()  
neuron\_2\_1.correction()  
neuron\_2\_2.correction()  
neuron\_2\_3.correction()

Тестовий приклад без корекції:

x = [[23, 29]]  
y = (x[0][0] + x[0][1]) / 100  
y1 = [1]  
y2 = [1]  
y3 = []  
ww = []  
y1.append(neuron\_1\_1.training(x[0]))  
y1.append(neuron\_1\_2.training(x[0]))  
y2.append(neuron\_2\_1.training(y1))  
y2.append(neuron\_2\_2.training(y1))  
y2.append(neuron\_2\_3.training(y1))  
y3.append(neuron\_3\_1.training(y2))  
  
print(ww)

# **2.2 Описання алгоритму**

Суть алгоритму полягає в тому, що потрібно коректувати ваги нейронів до тих пір, поки дельта, яке ми рахуємо за формулою:

(1)

не буде меншою за дельту допустиму, яка задана спочатку і дорівнює 0,1.

ymodel знаходиться за допомогою формули:

(2)

Коли дельта буде меншим за дельта допустиме (тобто умова буде задовольнятись), то тоді обчислення припиняються. Якщо ні, то ваги коректуються по відповідним формулам і все відбувається спочатку, але вже з новими вагами.

Для першої частини ці дії виконуються з вектором Х та значенням У очікуваного.

Для другої частини на вхід подається один Х та навчальний У. В даній частині перший нейрон є вхідним і він саме значення Х лінійно передає на наступний шар. Другий нейрон домножує на вагу W1,2, а потім, пропускаючи його через функцію (1), передаються дані на останній нейрон. Останній нейрон використовує вже вагу W2,3, і в результаті на виході ми отримуємо У модельне. Потім іде порівняння з У очікуваним, відбуважться перерахунок ваг, починаючи з останнього за допомогою певних формул. Всі дії виконуються до тих пір, поки не буде виконана умова.

Для третьої частини подається два Х та навчальний У, після чого починається обчислення. В даній схемі також існує зв’язок «сам з собою», де ваговий коефіцієнт є заданим, а Х береться за одиницю. Існує 500000 епох по 4 ітерації в кожній. Після всіх ітерацій буде показано у.

# 

# **ВИСНОВКИ**

В даній лабораторній роботі було організовано три різних варіанти нейронних мереж, де всі обчислення проводяться за певними формулами та алгоритм підрахунку відбувається також за відповідним алгоритмом.

# **ДОДАТКИ**

# **Додаток А (код програми)**

Частина №1:

import math  
  
  
x = [1.,7.,4.,5.]  
y = 0.3  
w = [1, 0.4, 0.7, 0.3]  
  
delt\_w = [0., 0., 0., 0.]  
  
delta\_dop = 0.1  
delt\_i = 1.  
b\_i = 0  
count = 1  
  
while delt\_i > delta\_dop:  
 x\_sum = 0  
 for i in range(0, len(x)):  
 x\_sum += x[i] \* w[i]  
  
 y\_model = 1 / (1 + math.exp(- x\_sum))  
  
 delt\_i = math.fabs((y\_model - y) / y)  
  
 if delt\_i < delta\_dop:  
 print("i = ", count, "| y\_model = ", round(y\_model, 5), "| DELTA = ", round(delt\_i, 3))  
 break  
 else:  
 b\_i = y\_model \* (1 - y\_model) \* (y - y\_model)  
 for j in range(0, len(delt\_w)):  
 delt\_w[j] = x[j] \* b\_i  
 for k in range(0, len(w)):  
 w[k] = w[k] + delt\_w[k]  
 print("i = ", count, "| y\_model = ", round(y\_model, 5), "| DELTA = ", round(delt\_i, 3))  
 count += 1  
  
  
  
  
x\_test = [1., 4., 3., 8.]  
y\_test = []  
x\_sum = 0  
  
for i in range(0, len(w)):  
 x\_sum += x[i] \* w[i]  
y\_model = 1 / (1 + math.exp(- x\_sum))  
y\_test.append(w[i] \* x\_test[i])  
  
print("\n\nx\_test = ", x\_test)  
print("y\_test = ", y\_test)

Частина №2

import math  
  
x = 5.  
y = 0.5  
w = [0.4, 0.7]  
  
delt\_w = [0., 0.]  
  
delta\_dop = 0.1  
delt\_i = 1.  
  
count = 1  
  
x\_sum = [0., 0., 0.,]  
y\_model = [y, 0., 0.]  
  
while delt\_i > delta\_dop:  
 x\_sum[1] = y\_model[0] \* w[0]  
  
 y\_model[1] = 1 / (1 + math.exp(- x\_sum[1]))  
 x\_sum[2] = y\_model[1] \* w[1]  
  
 y\_model[2] = 1 / (1 + math.exp(- x\_sum[2]))  
  
 delt\_i = math.fabs((y\_model[2] - y) / y)  
  
 if delt\_i < delta\_dop:  
 print("i = ", count, "| y\_model = ", y\_model, "| DELTA = ", round(delt\_i, 3))  
 break  
 else:  
 b\_3 = y\_model[2] \* (1 - y\_model[2]) \* (y - y\_model[2])  
 delt\_w[1] = y\_model[1] \* b\_3  
 w[1] = w[1] + delt\_w[1]  
  
 b\_2 = y\_model[1] \* (1 - y\_model[1]) \* b\_3 \* w[1]  
 delt\_w[0] = x \* b\_2  
 w[0] = w[1] + delt\_w[0]  
 print("i = ", count, "| y\_model = ", y\_model, "| DELTA = ", round(delt\_i, 3))  
  
 count += 1  
  
  
  
  
x\_test = 5.  
y\_test = []  
x\_summa = [0., 0., 0.,]  
x\_summa[1] = x\_test \* w[0]  
y\_test.append(1 / (1 + math.exp(- x\_summa[1])))  
x\_summa[2] = y\_test[0] \* w[1]  
y\_test.append((1 / (1 + math.exp(- x\_summa[2]))))  
  
print('\n\nx\_test = ', x\_test)  
print("y\_test = ",y\_test[len(y\_test)-1])

Частина №3

from math import exp  
from random import uniform  
  
delte = 0.001  
  
count\_of\_neurons = [2, 3, 1]  
  
ww = []  
  
class Neuron\_I:  
 def \_\_init\_\_(self, number):  
 self.n = number  
  
 def activation(self):  
 self.y = self.y \* 1  
  
 def training(self, x):  
 self.y = x[self.n]  
 self.activation()  
 return self.y  
  
  
class Neuron\_M:  
 def \_\_init\_\_(self, counr, i):  
 self.i = i  
 # w = [[-0.1, 0.1, 0.2], [-0.2, 0.5, 0.3], [-0.1, 0.1, 0.2]]  
 w = []  
 for i in range(counr + 1):  
 w.append(uniform(-1, 1))  
 self.w = w  
  
  
 def sumo(self):  
 sums = 0  
 for i in range(len(self.x)):  
 sums += float(self.x[i]) \* self.w[i]  
 self.y = sums  
 ww.append(self.w)  
  
 def activation(self):  
 self.y = 1 / (1 + exp(-1 \* self.y))  
  
 def correction(self):  
 dop = neuron\_3\_1.beck()  
 sigma = self.y \* (1 - self.y) \* dop[self.i]  
 # sigma = sig  
 delta\_w = []  
 for i in range(len(self.x)):  
 delta\_w.append(self.x[i] \* sigma \* 0.01)  
 self.w[i] = self.w[i] + delta\_w[i]  
  
 def training(self, x):  
 self.x = x  
 self.sumo()  
 self.activation()  
 return self.y  
  
  
class Neuron\_U:  
 def \_\_init\_\_(self, counr):  
 # w = [0.5, 0.1, -0.1, 0.2]  
 w = []  
 for i in range(counr + 1):  
 w.append(uniform(-1, 1))  
 self.w = w  
  
 def sumo(self):  
 sums = 0  
 for i in range(len(self.x)):  
 sums += float(self.x[i]) \* self.w[i]  
 self.y = sums  
 ww.append(self.w)  
  
 def activation(self):  
 self.y = 1 / (1 + exp(-1 \* self.y))  
  
 def mistake(self):  
 self.delta = abs((self.y - y) / y)  
 return self.delta  
  
 def correction(self):  
 sigma = self.y \* (1 - self.y) \* (y - self.y)  
 delta\_w = []  
 for i in range(len(self.x)):  
 delta\_w.append(self.x[i] \* sigma \* 0.01)  
 self.w[i] = self.w[i] + delta\_w[i]  
  
 def training(self, x):  
 self.x = x  
 self.sumo()  
 self.activation()  
 # self.mistake()  
 print("Y - " + str(self.y))  
 return self.y  
  
 def beck(self):  
 res = []  
 sigma = self.y \* (1 - self.y) \* (y - self.y)  
 for i in range(3):  
 res.append(sigma \* self.w[i])  
 return res  
  
# Neurons  
# Neurons = []  
# for i in range(len(count\_of\_neurons)):  
# Neurons.append([])  
#  
# for i in range(len(Neurons)):  
# for j in count\_of\_neurons:  
  
  
neuron\_1\_1 = Neuron\_I(0)  
neuron\_1\_2 = Neuron\_I(1)  
neuron\_2\_1 = Neuron\_M(2, 0)  
neuron\_2\_2 = Neuron\_M(2, 1)  
neuron\_2\_3 = Neuron\_M(2, 2)  
neuron\_3\_1 = Neuron\_U(3)  
  
x = [[3, 4], [9, 11], [21, 29], [25, 35]]  
  
  
for i in range(500000):  
 ww = []  
 print("Епоха " + str(i + 1))  
 for j in range(4):  
 y1 = [1]  
 y2 = [1]  
 y3 = []  
 y = (x[j][0] + x[j][1]) / 100  
 print("Итерация " + str(j + 1))  
 y1.append(neuron\_1\_1.training(x[j]))  
 y1.append(neuron\_1\_2.training(x[j]))  
 y2.append(neuron\_2\_1.training(y1))  
 y2.append(neuron\_2\_2.training(y1))  
 y2.append(neuron\_2\_3.training(y1))  
 y3.append(neuron\_3\_1.training(y2))  
 neuron\_3\_1.correction()  
 neuron\_2\_1.correction()  
 neuron\_2\_2.correction()  
 neuron\_2\_3.correction()  
  
print("\n\nТестування")  
x = [[23, 29]]  
y = (x[0][0] + x[0][1]) / 100  
y1 = [1]  
y2 = [1]  
y3 = []  
ww = []  
y1.append(neuron\_1\_1.training(x[0]))  
y1.append(neuron\_1\_2.training(x[0]))  
y2.append(neuron\_2\_1.training(y1))  
y2.append(neuron\_2\_2.training(y1))  
y2.append(neuron\_2\_3.training(y1))  
y3.append(neuron\_3\_1.training(y2))  
  
print(ww)

# **Додаток Б (результати роботи)**

Частина №1:

При вхідних даних:

x = [1.,7.,4.,5.]  
y = 0.3  
w = [1, 0.4, 0.7, 0.3]

i = 1 | y\_model = 0.9997 | DELTA = 2.332

i = 2 | y\_model = 0.99969 | DELTA = 2.332

i = 3 | y\_model = 0.99968 | DELTA = 2.332

i = 4 | y\_model = 0.99968 | DELTA = 2.332

i = 5 | y\_model = 0.99967 | DELTA = 2.332

i = 6 | y\_model = 0.99966 | DELTA = 2.332

i = 7 | y\_model = 0.99966 | DELTA = 2.332

i = 8 | y\_model = 0.99965 | DELTA = 2.332

…

i = 71 | y\_model = 0.03861 | DELTA = 0.871

i = 72 | y\_model = 0.08851 | DELTA = 0.705

i = 73 | y\_model = 0.31446 | DELTA = 0.048

Частина №2:

При вхідних даних:

x = 5.  
y = 0.5  
w = [0.4, 0.7]

i = 1 | y\_model = [0.5, 0.549833997312478, 0.5950504735417613] | DELTA = 0.19

i = 2 | y\_model = [0.5, 0.5827228143151154, 0.5988240123308017] | DELTA = 0.198

i = 3 | y\_model = [0.5, 0.58104513329505, 0.5966138802867568] | DELTA = 0.193

i = 4 | y\_model = [0.5, 0.579492606407629, 0.5944761925660375] | DELTA = 0.189

i = 5 | y\_model = [0.5, 0.577972683741228, 0.5923936616565537] | DELTA = 0.185

i = 6 | y\_model = [0.5, 0.5764848633903034, 0.5903648120729406] | DELTA = 0.181

i = 7 | y\_model = [0.5, 0.5750284516027625, 0.5883881627693337] | DELTA = 0.177

…

i = 32 | y\_model = [0.5, 0.5470195836319344, 0.5522582239771099] | DELTA = 0.105

i = 33 | y\_model = [0.5, 0.5461725400330358, 0.5512230571077942] | DELTA = 0.102

i = 34 | y\_model = [0.5, 0.5453424108941578, 0.5502118965052541] | DELTA = 0.1

i = 35 | y\_model = [0.5, 0.5445288135632405, 0.5492240791476046] | DELTA = 0.098