Санкт-Петербургский государственный морской технический университет

Отчёт

по практической работе №2

“Проектирование нейронной сети на персептронах Розенблатта”

Выполнил: Студент 1 курса

Факультета Цифровых

Промышленных Технологий

группы 20221 Хохлов Д.Р.

Преподаватель:

Кафедра киберфизических систем

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

# Введение

Цель работы: проектирование нейронной сети на основе персептронов Розенблатта для реализации заданной логической схемы.

Задача: необходимо составить таблицу истинности, разработать архитектуру нейронной сети, написать программу для обучения и тестирования отдельных блоков сети, а также провести проверку её корректной работы на всех вариантах входных данных.

# 

# Основная часть

## Постановка задачи

Задача данной лабораторной работы заключается в проектировании и реализации нейронной сети на основе персептронов Розенблатта, которая должна выполнять функции, соответствующие логическим элементам заданной схемы. Для этого необходимо:

1. Составить таблицу истинности для соответствующей схемы.
2. Разработать архитектуру нейронной сети, состоящей из блоков персептронов, каждый из которых реализует отдельные логические функции.
3. Реализовать программу для обучения каждого блока сети с использованием алгоритма обучения персептрона.
4. Провести проверку работы сети на тестовых данных.

## Таблица истинности

Таблица истинности реализуется на основании схемы (Рисунок 1), соответствующей варианту задания, была составлена таблица истинности, которая представляет собой список всех возможных входных состояний системы и соответствующих им выходных состояний. Эта таблица является основой для дальнейшего построения нейронной сети.

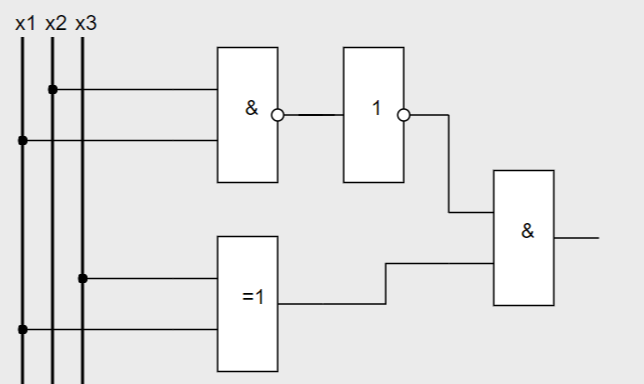


Рисунок 1 - Логическая схема

В данном случае таблица (Таблица 1) истинности состоит из 3 логических операций, а именно:

* NAND - логическое отрицающее И
* NOT - логическое отрицание
* XOR - логическое исключающее ИЛИ

Таблица 1 - Таблица истинности

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **x1** | **x2** | **x3** | **¬(x1∧x2)** | **¬(¬(x1∧x2))** | **x1⊕x2** | **¬(x1∧x2)∧x1⊕x2** |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |

## Проектирование нейронной сети

Нейронная сеть, которую необходимо спроектировать, состоит из персептронов, каждый из которых представляет собой простую модель искусственного нейрона. Персептрон Розенблатта используется для решения задач классификации и построения логических схем.

Для каждого логического элемента схемы создается отдельный блок сети (персептрон), который обучается для выполнения соответствующей логической функции. Эти блоки соединяются между собой таким образом, чтобы итоговая сеть соответствовала заданной схеме.

Однако, для логическая функция XOR (Рисунок 2) является довольно сложной функцией для нашего персептрона, т.к. персептрон должен разделять область сразу двумя линиями. Поэтому данная логическая функция состоит из 3-х персептронов, которые имеют 3 логические функции, а именно:

* NAND - логическое отрицающее И
* AND - логическое И
* OR - логическое ИЛИ

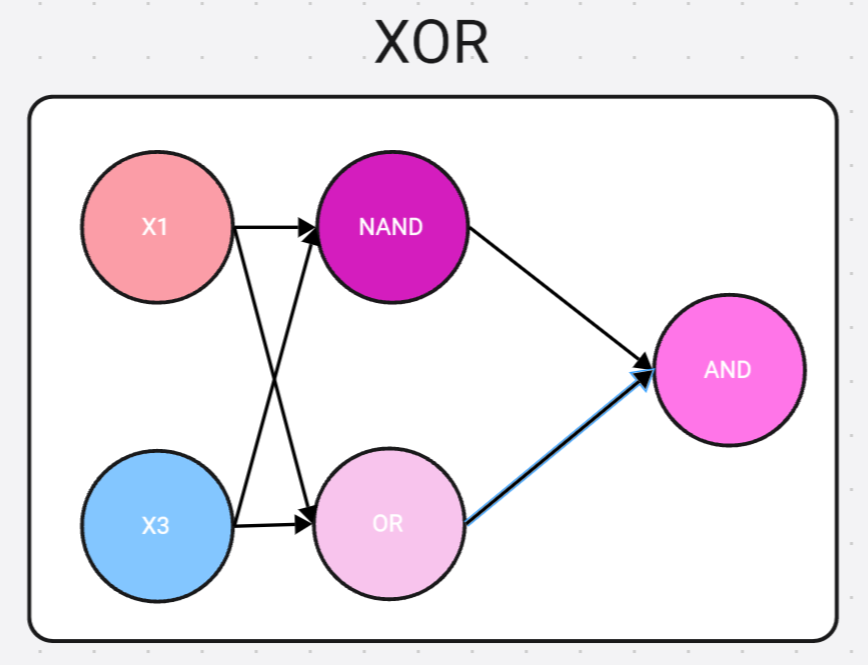


Рисунок 2 - Блок для функции XOR

Таким образом в нашей нейросети (Рисунок 3) будет присутствовать 9 нейронов:

* 3 нейрона (+bias) - входные
* 3 нейрона(+bias) - 1 NAND и 2 для XOR - NAND и OR
* 2 нейрона(+bias) - NOT и 1 для XOR - AND
* И выходной нейрон - AND

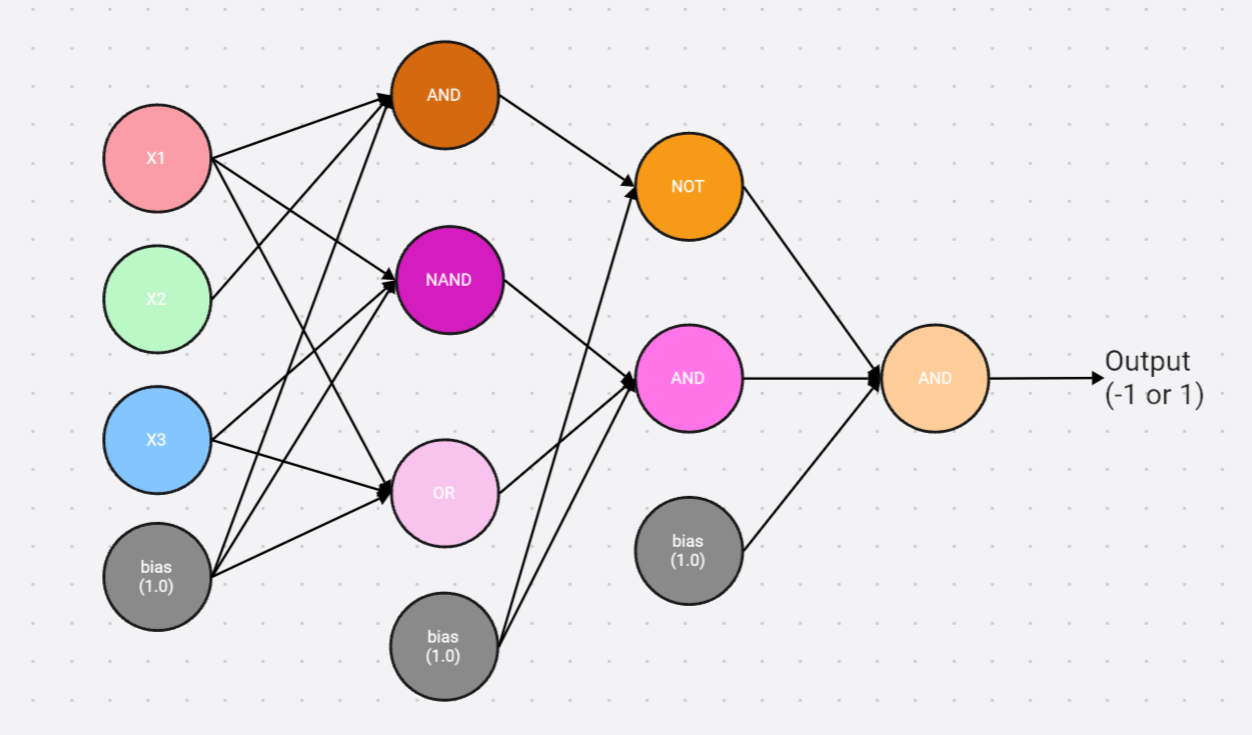


Рисунок 3 - Схема нейросети

Несмотря на то, что всего нейронов 9 (не считая bias) достаточно обучить всего 4 персептрона на определенную задачу, а именно

* NAND
* AND
* OR
* NOT

Причем, если 3 первых функции принимают на вход x1 и x2(+ bias), то NOT принимает всего лишь x1(+bias), так таблица истинности для данной функции(Таблица 2) принимает только 1 аргумент

Таблица 2 - Таблица истинности для функции NOT

|  |  |
| --- | --- |
| **x** | **¬x** |
| 0 | 1 |
| 1 | 0 |

## Реализация программы

Программа для реализации нейронной сети была написана с использованием библиотеки NumPy, для линейной алгебры, библиотеки PrettyTable, для вывода таблицы истинности и Matplotlib, для вывода графиков (Листинг 3.1). Каждый блок сети обучался отдельно на основе таблицы истинности (Листинг 3.3). Алгоритм обучения персептрона использует скалярное произведение весов и входных данных для вычисления результата и обновления весов на каждой итерации (Листинг 3.2).

Программа выводила результаты обучения в виде графиков и текстовых данных (Рисунок 4) на каждой итерации, чтобы отслеживать процесс обучения каждого блока сети (Листинг 3.4).

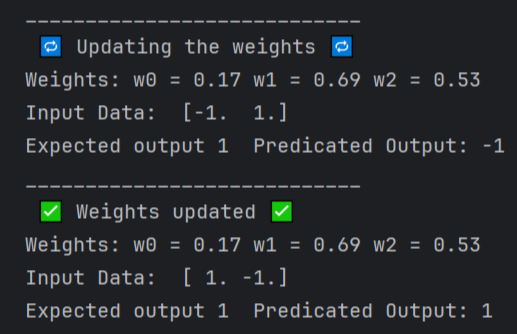


Рисунок 4 - Вывод обучения нейросети на каждой итерации

## 2.5 Результаты обучения

В процессе обучения весовые коэффициенты каждого персептрона изменялись в зависимости от ошибок на выходе. По мере уменьшения ошибок сеть постепенно обучалась правильно воспроизводить таблицу истинности. На каждом шаге обучения выводились промежуточные результаты.

## Тестирование сети

После завершения обучения была проведена проверка работы нейронной сети на тестовых данных(Листинг 3.4). Для этого использовались все возможные комбинации входных данных из таблицы истинности. Программа выводила полученные выходные значения и сравнивала их с ожидаемыми результатами. В результате тестирования была построена итоговая таблица истинности(Рисунок 5), которая подтвердила корректную работу сети.

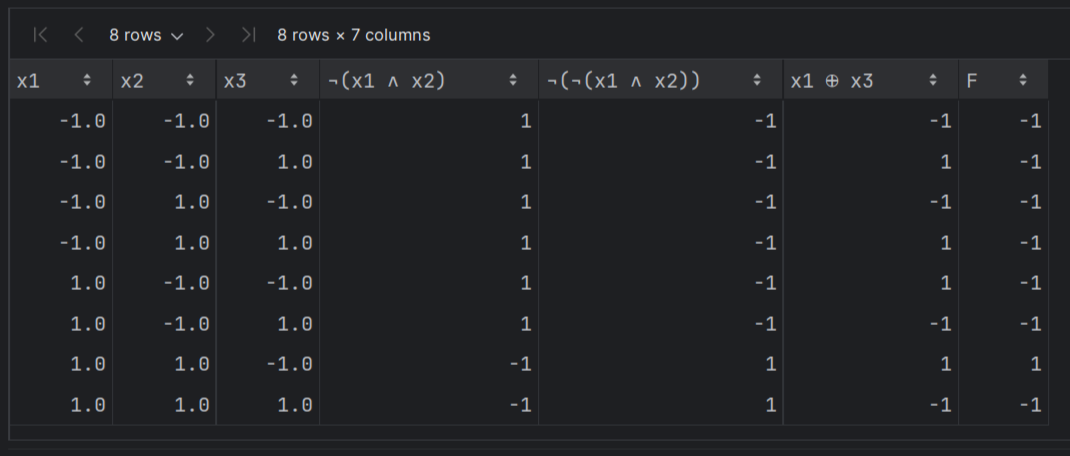


Рисунок 5 - Таблица истинности от нейросети (Готовый результат)

Из-за особенности обучения такой нейросети, она не может на входе принимать значение 0, поэтому 0 заменён на -1

# Приложение А

**Листинг программы**

## Листинг 1 - Установка и импорт библиотек

#--Установка библиотек–#

!pip install numpy

!pip install matplotlib

!pip install prettytable

#--Импорт библиотек–#

import numpy as np

import random

from prettytable import PrettyTable

from matplotlib import pyplot as plt

## Листинг 2 - Объект “Perceptron”, его свойства и методы

class Perceptron:

#---ИНИЦИАЛИЗАЦИЯ ДАННЫХ---#

def \_\_init\_\_(self,input\_data:np.array, output\_data, learning\_rate):

np.random.seed(7)

self.\_x\_train = input\_data

print(self.\_x\_train)

self.\_y\_train = output\_data

self.\_w = np.random.uniform(size=(1,self.\_x\_train.shape[1]))

self.\_all\_train = {

"input\_data":[],

"p\_out": [],

"w": []

}

self.\_lr = learning\_rate

self.\_index\_list = list(range(input\_data.shape[0]))

#---МЕТОД ВЫВОДА ИНФОРМАЦИИ НА ЭКРАН---#

def \_\_show\_train\_info(self,x,y,p\_out):

outprint\_W = ""

for i in range(len(self.\_w)):

for j in range(len(self.\_w[i])):

outprint\_W += (f"w{j} ={self.\_w[i][j]:5.2f} ")

print(f"""Weights: {outprint\_W}

Input Data: {x[1:]}

Expected output {int(y)} Predicated Output: {int(p\_out)}""")

def \_\_draw\_point(self,input\_data, p\_out):

#print(len(input\_data))

#input\_data = np.insert(input\_data,-1,input\_data[1]) if len(input\_data) == 2 else None

if p\_out == -1:

plt.scatter(input\_data[2],input\_data[1], color='blue', marker='\_')

else:

plt.scatter(input\_data[2],input\_data[1], color='red', marker='+')

def \_\_draw\_line(self, w ,color="green"):

x1 = [-1.0, 1.0]

#w = np.insert(w,-1,w[1]) if len(w) == 2 else None

x2 = [-w[0][1]\* x1[i]/w[0][2] - w[0][0] / w[0][2] for i in range(len(x1))]

plt.axline(\*list(zip(x2,x1)), color=color)

#---РАЗЛИЧНЫЕ ФУНКЦИИ АКТИВАЦИИ---#

# def sigmoid(self, x:float): # Сигмоидная функция активации

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

#

# def relu(self, x:float): # Выпрямитель (RELu)

# return np.maximum(0, x)

def step\_function(self,x:float): # Жесткая пороговая функция активации

if x < 0: return -1

else: return 1

#---ОБУЧЕНИЕ ПЕРСЕПТРОНА---#

def train(self):

all\_correct = False

while not all\_correct:

all\_correct = True

random.shuffle(self.\_index\_list)

for i in self.\_index\_list:

x = self.\_x\_train[i]

y = self.\_y\_train[0][i]

z = np.dot(self.\_w, x) # Скалярное произведение входа и весов

p\_out = self.step\_function(z) # Применение функции активации, в данном случае применяем пошаговую функцию активации

if y != p\_out: # Обновить веса, когда неправильно

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n 🔁 Updating the weights 🔁")

for j in range(0, len(self.\_w[0])):

self.\_w[0][j] += (y \* self.\_lr \* x[j]) # Процесс поправления весов (Процесс "обучения")

all\_correct = False

self.\_\_show\_train\_info(x,y,p\_out) # Вывод информации (Во время обучения)

self.\_all\_train["input\_data"].append(x)

self.\_all\_train["p\_out"].append(p\_out)

self.\_all\_train["w"].append(self.\_w.copy())

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n ✅ Weights updated ✅")

self.\_\_show\_train\_info(x,y,p\_out) # Вывод информации (Обученного персептрона)

#---МЕТОД "ПРОГНОЗИРОВАНИЯ"---#

def predict(self, input\_data:np.array):

z = np.dot(self.\_w, input\_data[0])

return self.step\_function(z)

def show\_all\_training(self): #Вывод графика всего обучения

plt.ylabel("x1")

plt.xlabel("x2")

plt.xlim([round(np.min(self.\_x\_train,axis=0)[2])-0.1,round(np.max(self.\_x\_train, axis=0)[2])+0.1])

plt.ylim([round(np.min(self.\_x\_train, axis=0)[1])-0.1,round(np.max(self.\_x\_train, axis=0)[1])+0.1])

for i in range(0,len(self.\_all\_train["p\_out"])):

self.\_\_draw\_point(self.\_all\_train["input\_data"][i],self.\_all\_train["p\_out"][i])

for i in range(0,len(self.\_all\_train["w"])):

color = "green" if i == len(self.\_all\_train["w"]) - 1 else "red"

self.\_\_draw\_line(self.\_all\_train["w"][i], color=color)

plt.show()

def show\_final\_training(self): #Вывод финального графика

plt.ylabel("x1")

plt.xlabel("x2")

plt.xlim([round(np.min(self.\_x\_train,axis=0)[2])-0.1,round(np.max(self.\_x\_train, axis=0)[2])+0.1])

plt.ylim([round(np.min(self.\_x\_train, axis=0)[1])-0.1,round(np.max(self.\_x\_train, axis=0)[1])+0.1])

if self.\_x\_train.shape[1] != 2:

plt.xlim([round(np.min(self.\_x\_train,axis=0)[2])-0.1,round(np.max(self.\_x\_train, axis=0)[2])+0.1 ])

else:

plt.xlim([0,1])

plt.ylim([round(np.min(self.\_x\_train, axis=0)[1])-0.1,round(np.max(self.\_x\_train, axis=0)[1])+0.1])

for i in range(0,len(self.\_y\_train[0])):

self.\_\_draw\_point(self.\_x\_train[i],self.\_y\_train[0][i])

self.\_\_draw\_line(self.\_w)

plt.show()

## 

## Листинг 3 - Создание и обучение персептронов

#--Константы и тренировочные данные--#

LEARNING\_RATE = 0.01

x\_train = np.array([(1.0,-1.0,-1.0),

(1.0,-1.0,1.0),

(1.0,1.0,-1.0),

(1.0,1.0,1.0)])

#--Создание персептронов под каждую задачу--#

nand\_perceptron = Perceptron(x\_train, np.array([[1.0,1.0,1.0,-1.0]]), LEARNING\_RATE)

or\_perceptron = Perceptron(x\_train, np.array([[-1.0,1.0,1.0,1.0]]), LEARNING\_RATE)

and\_perceptron = Perceptron(x\_train, np.array([[-1.0,-1.0,-1.0,1.0]]), LEARNING\_RATE)

not\_perceptron = Perceptron(np.array([[1.0, -1.0],[1.0, 1.0]]), np.array([[1.0, -1.0]]), LEARNING\_RATE)

#--Обучаем каждый персептрон--#

not\_perceptron.train() #NOT\_BLOCK

#NAND\_BLOCK#

nand\_perceptron.train()

nand\_perceptron.show\_final\_training()

nand\_perceptron.show\_all\_training()

#AND\_BLOCK#

and\_perceptron.train()

and\_perceptron.show\_final\_training()

and\_perceptron.show\_all\_training()

#OR\_BLOCK#

or\_perceptron.train()

or\_perceptron.show\_final\_training()

or\_perceptron.show\_all\_training()

## Листинг 4 - Тестирование нейросети и решение главной задачи

x\_test = np.array([(1.0,-1.0,-1.0,-1.0),

(1.0,-1.0,-1.0,1.0),

(1.0, -1.0,1.0,-1.0),

(1.0, -1.0,1.0,1.0),

(1.0, 1.0,-1.0,-1.0),

(1.0, 1.0,-1.0,1.0),

(1.0, 1.0, 1.0, -1.0),

(1.0, 1.0, 1.0, 1.0)]) #(<bias>, <x1>, <x2>, <x3>)

pt = PrettyTable(["x1", "x2", "x3","¬(x1 ∧ x2)","¬(¬(x1 ∧ x2))", "x1 ⊕ x3","F"]) #Создание таблицы с соответствующими названиями столбцов

output = []

#--Создание логической цепи--#

for i in range(len(x\_test)):

not\_block = not\_perceptron.predict(np.array([(1.0,

nand\_block := nand\_perceptron.predict(np.array([(x\_test[i][0], x\_test[i][1], x\_test[i][2])])))

])

)

xor\_block = and\_perceptron.predict(np.array([(1.0,

nand\_perceptron.predict(np.array([(x\_test[i][0], x\_test[i][1], x\_test[i][3])])),

or\_perceptron.predict(np.array([(x\_test[i][0], x\_test[i][1], x\_test[i][3])])))

])

)

output.append(and\_perceptron.predict(np.array([(1.0, not\_block, xor\_block)])))

pt.add\_row([x\_test[i][1], x\_test[i][2],x\_test[i][3], nand\_block,not\_block, xor\_block, output[i]])

print(pt) #Вывод таблицы