Санкт-Петербургский государственный морской технический университет

Отчёт

по практической работе №3

“ Анализ нейросети с алгоритмом обратного распространения ошибки”

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: исследовать процесс обучения двухслойной нейронной сети для реализации функции XOR с помощью алгоритма обратного распространения ошибки. Познакомиться с использованием функций активации (tanh и сигмоида), механизмом обновления весов и алгоритмом проверки сходимости.

Задание

1. Реализовать двухслойную нейронную сеть для выполнения функции XOR.
2. Описать принцип работы алгоритма обратного распространения.
3. Протестировать сеть на наборе обучающих данных до достижения сходимости.
4. Ответить на теоретические вопросы.

# Основная часть

## Постановка задачи

Главной задачей является исследование процесса обучения нейронной сети XOR c обучением построенном на алгоритме обратного распространения ошибки (backpropagation). Объяснить прикрепленный к этому заданию код.

## Работа нейросети XOR

Работа нейросети имеет следующую структуру:

* Инициализация данных
* Обучение
* Проверка сходимости

## Инициализация данных

Перед началом обучения инициализируются данные, такие как входные данные и выходные, для проверки обучения.

Входными данными для нейросети с задачей определения XOR, является таблица истинности для задачи XOR (Таблица 1)

Выходными данными является результат из таблицы истинности для XOR (Таблица 1)

Таблица 1 – Таблица истинности для XOR

|  |  |  |
| --- | --- | --- |
| **X1** | **X2** | **XOR** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## Обучение

Обучение нейросеть с алгоритмом backpropagation состоит из нескольких частей:

* Прямой проход
* Обратный проход
* Обновление весов

Прямым проходом здесь является работа нейросети от входа к выходу, как обычно и работают все нейросети. На этом этапе нейросеть просчитывает правильный ответ, но на первом прямом проходе процент получения правильного ответа очень мал.

После прохождения прямого прохода, происходит обратный проход. Нейросеть на этом этапе двигается от выхода к входу, вычисляя на каждом слое производную от функции активации. После вычисления производной происходит вычисление коэффициента ошибки на каждом нейроне слоя

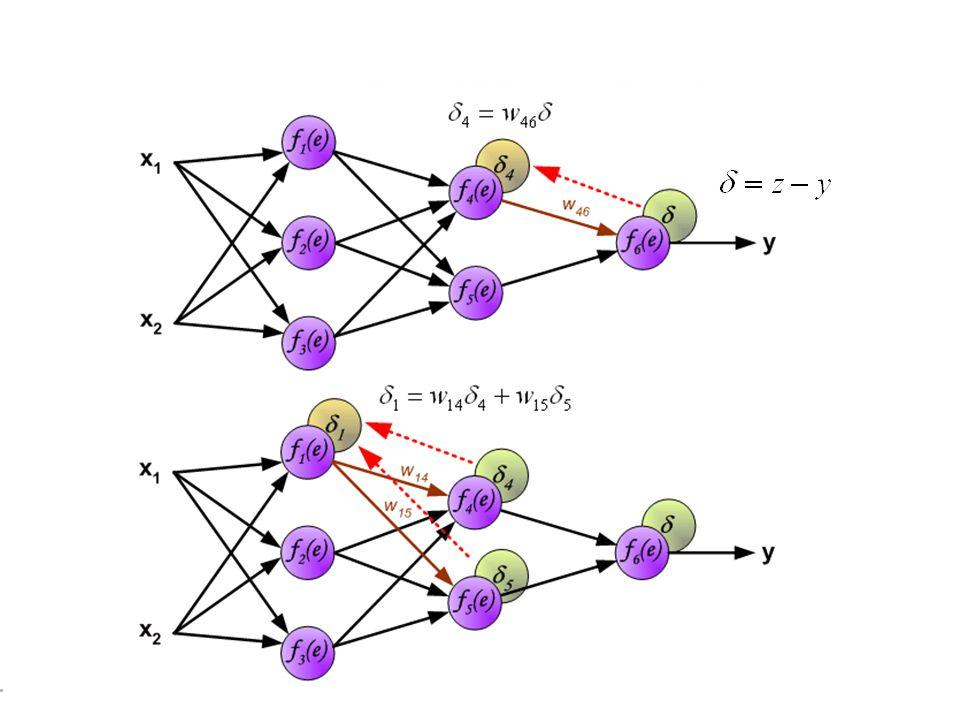


Рисунок 1 – Демонстрация процесса обучения

После вычисления коэффициента ошибки на каждом нейроне всех слоёв происходит обновления весов по формуле:

* x – входной сигнал нейрона

## Проверка сходимости

Для проверки сходимости берутся выходные данные из таблицы истинности XOR и выходные данные из нейросети и сравниваются.

Если нейросеть правильно определила 1 пример, в данной маленькой задаче можно предположить, что и остальные варианты, она в силе высчитать. В таком случае обучение останавливается.

В случае неправильных ответов от нейросети обучение идёт дальше, и следующим этапом является прямой проход. Обучение проходит циклично, до тех пор, пока не выдаст правильный ответ.

Блок-схема обучения

Таким образом имея ввиду все выше описанные этапы работы можно получить UML схему, где отображены все этапы обучения (Рисунок 2)

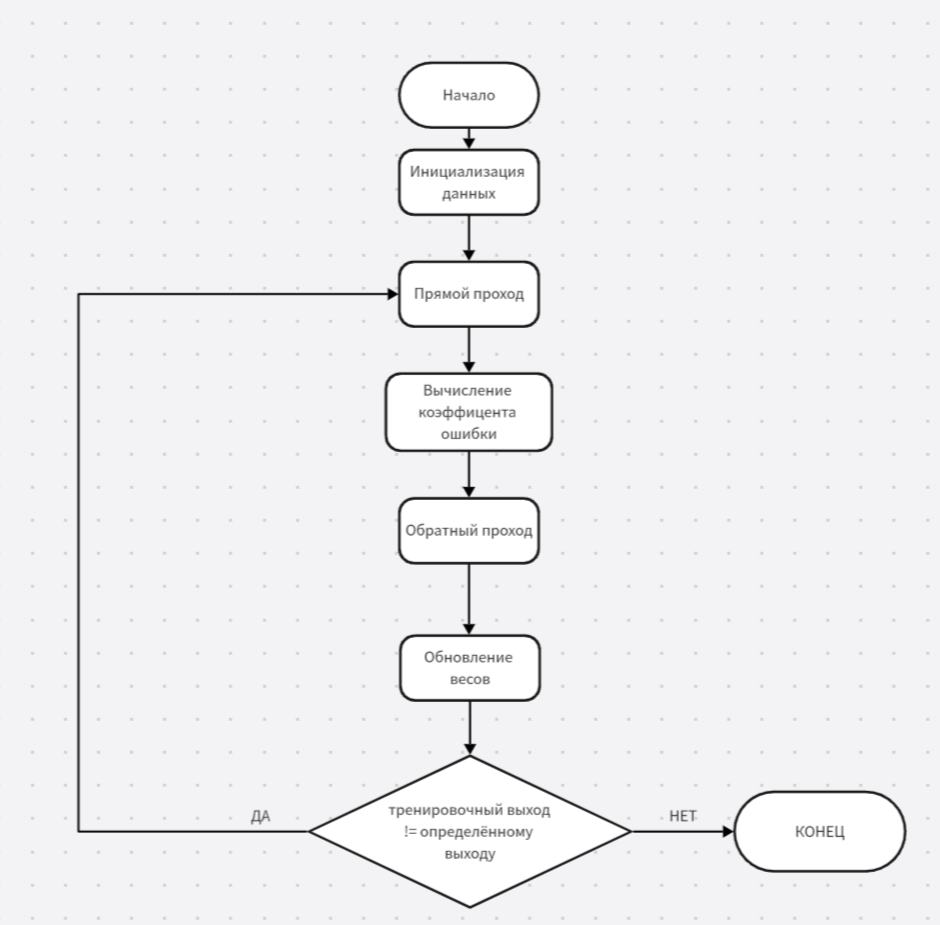


Рисунок 2 – Блок-схема нейросети

## Ответ на теоретические вопросы

В работе было задано несколько теоретических вопросов по нейросети XOR

Зачем используется np.random.seed вместо random.seed?

Функция random.seed() фиксирует начальное состояние генератора случайных чисел, чтобы последовательность случайных чисел оставалась одинаковой при каждом запуске программы. Это необходимо для воспроизводимости результатов и анализа влияния гиперпараметров на обучение. В отличии от random.seed(), np.random.seed() используется для тех же целей, но для работы с библиотекой NumPy.

**Почему истинное значение преобразуется в диапазон от 0.0 до 1.0?**

Выходной нейрон использует логистическую сигмоидальную функцию активации, которая возвращает значения в диапазоне ‎[0.0, 1.0]. Поэтому целевые значения (истинные метки) также приводятся к этому диапазону, чтобы сеть могла эффективно обучаться и корректно сравнивать свои прогнозы с истинными значениями.

## Результаты обучения нейросети

Результатами обучения нейросети является правильно определяющая XOR таблицу нейросеть.

После обучения нейросеть выдала обученные, изменённые веса (Рисунок 3)

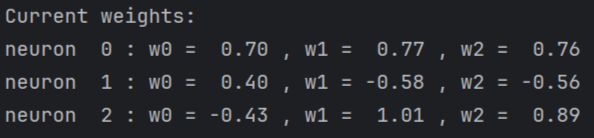


Рисунок 3 – Веса после обучения

После обучения можно проверить работоспособность нейросети и дать ей на вход всю таблицу XOR. Нейросеть должна правильно определить выход каждого варианта из таблицы (Рисунок 4)

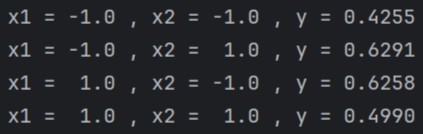


Рисунок 4 – Проверка нейросети

Нейросеть в этом случае выдаёт не точные результаты, их надо округлять. Всё что > 0,5 округляется до 1, если <0.5 то до 0.

Как видно на рисунке, нейросеть правильно обучилась и смогла определить таблицу истинности XOR.

# Заключение

В ходе лабораторной работы был проанализирован код 2-х слойной нейронной сети для определения выхода из таблицы XOR с использованием алгоритма обучения обратного распространения ошибки. Был проанализирован способ такого обучения, была построена UML-схема нейросети и были даны ответы на поставленные теоретические вопросы.

# Приложение А

Листинг программы

Листинг 1 – Инициализация данных и создание матрицы весов

import numpy as np  
  
np.random.seed(3) #задаём параметр генератору случайных значений  
LEARNING\_RATE = 0.1 #скорость обучения  
index\_list = [0, 1, 2, 3] #список для рандомизирования  
  
x\_train = [np.array([1.0, -1.0, -1.0]),  
 np.array([1.0, -1.0, 1.0]),  
 np.array([1.0, 1.0, -1.0]),  
 np.array([1.0, 1.0, 1.0])] #задание таблицы истинности  
y\_train = [0.0, 1.0, 1.0, 0.0] #выход таблицы истинности  
  
  
def neuron\_w(input\_count):  
 weights = np.zeros(input\_count+1) #создание нулевой матрицы  
 for i in range(1, (input\_count+1)): #цикл для прохода по нулевой матрице  
 weights[i] = np.random.uniform(-1.0, 1.0) #заполнение элементов матрицы рандомными значениями  
 return weights #возвращение весов  
  
n\_w = [neuron\_w(2), neuron\_w(2), neuron\_w(2)] #задание весов нейросети  
n\_y = [0, 0, 0]  
n\_error = [0, 0, 0] # задание списка для ошибок

Листинг 2 – Функции для обучения нейросети

#Функция для отображения весов нейросети, такая же была в Практической работе №1  
def show\_learning():  
 print('Current weights:')  
 for i, w in enumerate(n\_w):  
 print('neuron ', i, ': w0 =', '%5.2f' % w[0],  
 ', w1 =', '%5.2f' % w[1], ', w2 =',  
 '%5.2f' % w[2])  
 print('----------------')  
  
#Функция прямого прохода  
def forward\_pass(x):  
 global n\_y  
 n\_y[0] = np.tanh(np.dot(n\_w[0], x)) # вычисляем скалярное произведение весов на входные значения и применяем гиперболическую функцию активации для 1-го нейрона   
 n\_y[1] = np.tanh(np.dot(n\_w[1], x)) # вычисляем скалярное произведение весов на выходные значения из прошлого нейрона и применяем гиперболическую функцию активации для 2-го нейрона   
 n2\_inputs = np.array([1.0, n\_y[0], n\_y[1]])   
 z2 = np.dot(n\_w[2], n2\_inputs) # вычисляем скалярное произведение весов на выходные значения из 1-го нейрона и 2-го нейрона   
 n\_y[2] = 1.0 / (1.0 + np.exp(-z2)) #применяем сигмоидальную функцию активации  
  
#Функция обратного прохода  
def backward\_pass(y\_truth):  
 global n\_error  
 error\_prime = -(y\_truth - n\_y[2]) #вычисляем глобальный коэффициент ошибки  
 derivative = n\_y[2] \* (1.0 - n\_y[2]) #находим производную функцию от сигмоиды  
 n\_error[2] = error\_prime \* derivative #вычисляем коэффициент ошибки для выходного нейрона  
 derivative = 1.0 - n\_y[0]\*\*2 #находим производную функцию от гиперболического тангенса  
 n\_error[0] = n\_w[2][1] \* n\_error[2] \* derivative #вычисляем коэффициент ошибки для 1-го нейрона  
 derivative = 1.0 - n\_y[1]\*\*2 #находим производную функцию от гиперболического тангенса  
 n\_error[1] = n\_w[2][2] \* n\_error[2] \* derivative #вычисляем коэффициент ошибки для 2-го нейрона  
  
#функция обновления весов  
def adjust\_weights(x):  
 global n\_w  
 n\_w[0] -= (x \* LEARNING\_RATE \* n\_error[0]) #обновляем веса 1-го нейрона  
 n\_w[1] -= (x \* LEARNING\_RATE \* n\_error[1]) #обновляем веса 2-го нейрона  
 n2\_inputs = np.array([1.0, n\_y[0], n\_y[1]])  
 n\_w[2] -= (n2\_inputs \* LEARNING\_RATE \* n\_error[2]) #обновляем веса выходного нейрона из 2-ух предыдущих

Листинг 3 – цикл обучения нейросети

all\_correct = False  
while not all\_correct: #начало обучения нейросети  
 all\_correct = True  
 np.random.shuffle(index\_list) # Randomize order  
 for i in index\_list: # Train on all examples  
 forward\_pass(x\_train[i]) # прямой проход  
 backward\_pass(y\_train[i]) # обратный проход  
 adjust\_weights(x\_train[i]) #обновление весов  
 show\_learning() # отображение обновлённых весов  
 for i in range(len(x\_train)): # Проверка сходимости нейросети  
 forward\_pass(x\_train[i]) #прямой проход  
 print('x1 =', '%4.1f' % x\_train[i][1], ', x2 =',  
 '%4.1f' % x\_train[i][2], ', y =',  
 '%.4f' % int(n\_y[2])) #отображение таблицы истинности  
 if(((y\_train[i] < 0.5) and (n\_y[2] >= 0.5))  
 or ((y\_train[i] >= 0.5) and (n\_y[2] < 0.5))): #условия сходимости (если выход у нейросети и настоящий ответ находится в одном диапазоне, то обучение прекращается, в ином случаем продолжается)  
 all\_correct = False