МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра Вычислительной техники

Машинное обучение.

Лабораторная работа №2

Практика 3-4

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АПИМ-24  Выполнил:  Разуваев В. В. | Проверил:  Гаврилов А.В. |

**Содержание**

[Цель работы 3](#_Toc184590788)

[Задание 3](#_Toc184590789)

[Ход работы 3](#_Toc184590790)

[Задача 1 3](#_Toc184590791)

[Задача 2 5](#_Toc184590792)

[Заключение 10](#_Toc184590793)

# Цель работы

Освоить основные методы представление данных и получить базовые навыки работы с библиотекой NumPy.

# Задание

1. Написать функцию, которая заполняет матрицу с размерами (M,N) случайными числами распределенными по нормальному закону. Затем считает мат. ожидание и дисперсию для каждого из столбцов, а также строит для каждой строки стоит гистограмму значений (использовать функцию hist из модуля matplotlib.plot).

2. Реализовать нейронную сеть вычисляющую результат заданной логической операции: (a or b) and (b or c). Затем реализовать функции, которые будут симулировать работу построенной модели. Функции должны принимать тензор входных данных и список весов. Должно быть реализовано 2 функции:

- Функция, в которой все операции реализованы как поэлементные операции над тензорами

- Функция, в которой все операции реализованы с использованием операций над тензорами из NumPy.

Для проверки корректности работы функций необходимо:

- Инициализировать модель и получить из нее веса

- Прогнать датасет через не обученную модель и реализованные 2 функции. Сравнить результат.

- Обучить модель и получить веса после обучения

- Прогнать датасет через обученную модель и реализованные 2 функции. Сравнить результат

# Ход работы

## Задача 1

*Листинг 1 – Код для задачи 1 (матрица 5×10)*

import numpy as np

import matplotlib.pyplot as plt

def process\_matrix(file\_path, output\_path, M, N):

"""

Функция создает матрицу (M, N) со случайными числами, распределенными по нормальному закону,

считает мат. ожидание и дисперсию для каждого столбца, и строит гистограммы для каждой строки.

:param file\_path: путь к файлу для сохранения исходной матрицы.

:param output\_path: путь к файлу для сохранения результатов расчетов.

:param M: количество строк в матрице.

:param N: количество столбцов в матрице.

"""

# Генерация матрицы

matrix = np.random.randn(M, N) # Генерация случайных чисел по нормальному закону

# Сохранение матрицы в файл

np.savetxt(file\_path, matrix, fmt='%.5f', header='Generated matrix')

# Вычисление мат. ожидания и дисперсии для каждого столбца

mean\_values = np.mean(matrix, axis=0)

variance\_values = np.var(matrix, axis=0)

# Сохранение результатов в файл

with open(output\_path, 'w') as f:

f.write("Mat. expectation by columns:\n")

f.write(", ".join(map(str, mean\_values)) + "\n\n")

f.write("Column variance:\n")

f.write(", ".join(map(str, variance\_values)) + "\n")

# Построение гистограмм для каждой строки

for i, row in enumerate(matrix):

plt.figure(figsize=(6, 4))

plt.hist(row, bins=10, alpha=0.7, edgecolor='black')

plt.title(f'Гистограмма значений строки {i+1}')

plt.xlabel('Значение')

plt.ylabel('Частота')

plt.grid(True)

plt.savefig(f'histogram\_row\_{i+1}.png')

plt.close()

if \_\_name\_\_ == "\_\_main\_\_":

# Пример использования

file\_path = 'matrix1.txt'

output\_path = 'results1.txt'

M, N = 5, 10 # Размер матрицы

process\_matrix(file\_path, output\_path, M, N)

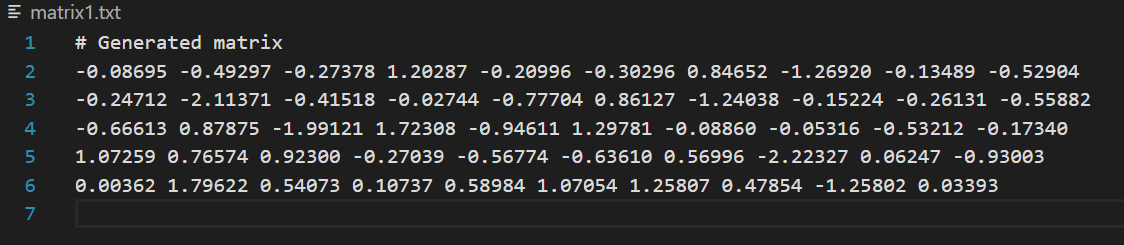


Рисунок 1 – Содержимое файла с исходной матрицей (matrix1.txt)

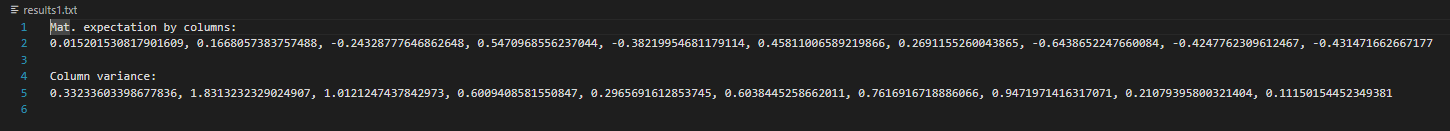
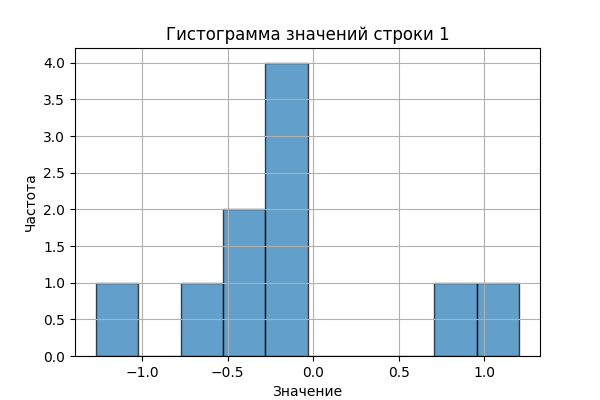
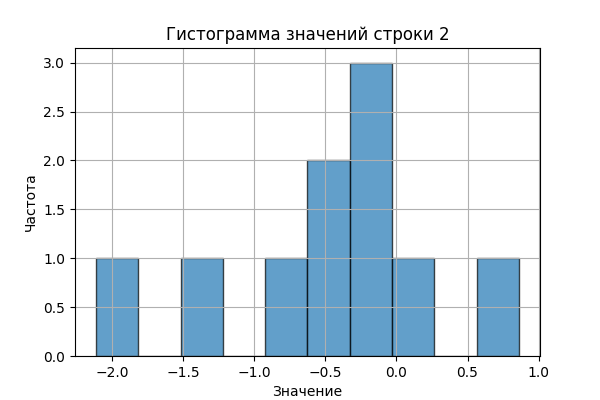
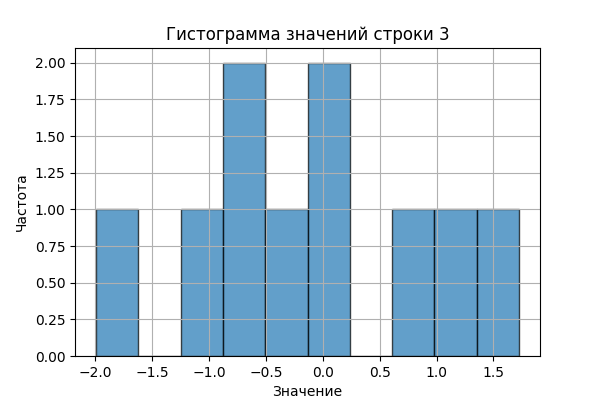
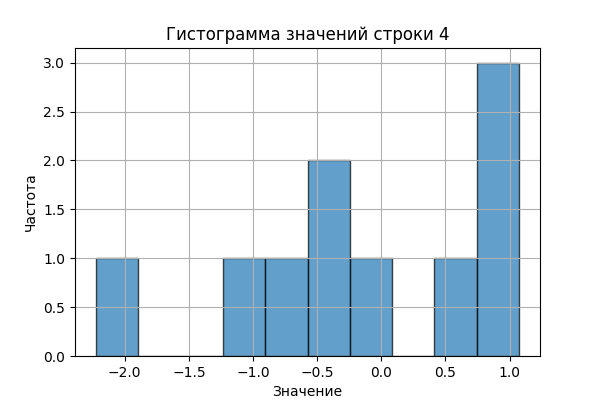


Рисунок 2 – Содержимое файла с результатами расчетов (results1.txt)









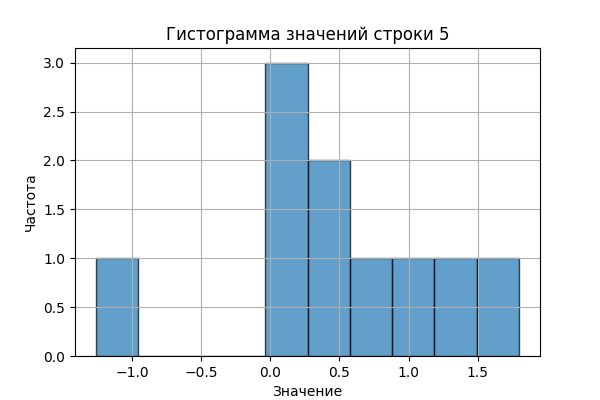


Рисунок 3-7 – Гистограммы

## Задача 2

*Листинг 2 – Код нейронной сети для вычислесления логической операции: (a or b) and (b or c).*

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

# Датасет для логической операции (a or b) and (b or c)

def generate\_dataset():

data = np.array([[int(x) for x in f"{i:03b}"] for i in range(8)]) # 3-битные входы

labels = np.array([((a or b) and (b or c)) for a, b, c in data], dtype=int)

return data, labels

# Создание модели

def create\_model():

model = Sequential([

Dense(10, activation='relu', input\_shape=(3,)),

Dense(1, activation='sigmoid')

])

model.compile(optimizer=Adam(learning\_rate=0.01), loss='binary\_crossentropy', metrics=['accuracy'])

return model

def logical\_operation\_elementwise(inputs, weights):

"""

Функция для вычисления результата логической операции с использованием поэлементных операций.

:param inputs: Входной тензор (np.array размера Nx3).

:param weights: Список весов модели, каждый элемент - список [weights, biases].

:return: Результаты логической операции (массив Nx1).

"""

# Первый слой

w1, b1 = weights[0] # Первый слой: веса и смещения

layer1 = np.maximum(0, np.dot(inputs, w1) + b1) # relu активация

# Второй слой

w2, b2 = weights[1] # Второй слой: веса и смещения

output = 1 / (1 + np.exp(-np.dot(layer1, w2) - b2)) # sigmoid активация

return output

def logical\_operation\_numpy(inputs, weights):

"""

Функция для вычисления результата логической операции с использованием NumPy операций.

:param inputs: Входной тензор (np.array размера Nx3).

:param weights: Список весов модели, каждый элемент - список [weights, biases].

:return: Результаты логической операции (массив Nx1).

"""

# Первый слой

w1, b1 = weights[0] # Первый слой: веса и смещения

layer1 = np.dot(inputs, w1) + b1

layer1[layer1 < 0] = 0 # relu активация

# Второй слой

w2, b2 = weights[1] # Второй слой: веса и смещения

output = 1 / (1 + np.exp(-np.dot(layer1, w2) - b2)) # sigmoid активация

return output

if \_\_name\_\_ == "\_\_main\_\_":

# Генерация данных

data, labels = generate\_dataset()

# Создание модели

model = create\_model()

# Получение весов

weights = [layer.get\_weights() for layer in model.layers]

# Проверка функций на не обученной модели

print("На не обученной модели:")

predictions\_model = model.predict(data)

predictions\_func1 = logical\_operation\_elementwise(data, weights)

predictions\_func2 = logical\_operation\_numpy(data, weights)

print("Модель:", predictions\_model.flatten())

print("Функция 1:", predictions\_func1.flatten())

print("Функция 2:", predictions\_func2.flatten())

# Обучение модели

model.fit(data, labels, epochs=100, verbose=0)

# Получение весов обученной модели

trained\_weights = [layer.get\_weights() for layer in model.layers]

# Проверка функций на обученной модели

print("\nНа обученной модели:")

predictions\_model\_trained = model.predict(data)

predictions\_func1\_trained = logical\_operation\_elementwise(data, trained\_weights)

predictions\_func2\_trained = logical\_operation\_numpy(data, trained\_weights)

print("Модель:", predictions\_model\_trained.flatten())

print("Функция 1:", predictions\_func1\_trained.flatten())

print("Функция 2:", predictions\_func2\_trained.flatten())

**Датасет**: Содержит все возможные комбинации трёх входных бит (000–111) и соответствующий результат операции (a or b) and (b or c).

**Модель:** Два слоя: первый слой с 10 нейронами (relu), второй с 1 нейроном (sigmoid).

**Функции:** Первая функция реализует логику поэлементно. Вторая функция использует операции над тензорами NumPy.

Выполняется проверка, совпадают ли результаты двух функций с результатами модели до и после обучения.

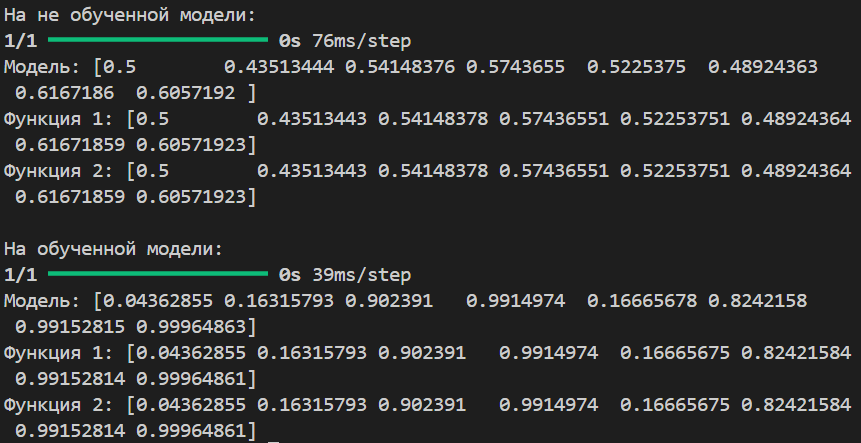


Рисунок 8 – Результат работы нейронной сети

**На не обученной модели:**

Результаты выглядят случайными (например, 0.5, 0.435, 0.541), потому что модель ещё не знает, как вычислять операцию.

Вероятности находятся ближе к 0.5, так как сигмоидная активация (последний слой) с рандомизированными весами часто даёт средние значения.

**На обученной модели:**

Результаты стали ближе к крайним значениям:

* 0.0436, 0.1631 — означает "ложь" (вероятность < 0.5).
* 0.9024, 0.9915, 0.9996 — означает "истина" (вероятность > 0.5).

Это подтверждает, что модель успешно обучилась и может корректно определять результат логической операции.

# Заключение

В ходе выполнения лабораторной работы была реализована нейронная сеть для вычислесления логических операций. Модель корректно обучается для решения задачи. Функции logical\_operation\_elementwise и logical\_operation\_numpy работают так же точно, как и модель, что подтверждается идентичными (с точностью до округления) результатами. Успешная проверка доказывает, что реализованные функции правильно воспроизводят работу нейронной сети. Также были освоены методы генерации, обработки и представления данных с помощью библиотеки NumPy.