Постановка задачи.

Необходимо реализовать нейронную сеть вычисляющую результат заданной логической операции. 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.

Вариант 2

(a or b) xor not(b and c)

Выполнение работы.

Был задан набор входных данных с соответствующими им выходными значениями для логической функции (a or b) xor not(b and c):

a	b	c	value
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0

1	0	1	0
1	1	0	0
1	1	1	1

Была задана модель сети, включающая два слоя по 12 нейронов с функцией активацией relu и выходным слоем из одного нейрона с функцией sigmoid.

Работу слоя с функцией relu можно представить в виде

$$output = relu(dot(W, input) + b)$$

Работу слоя с функцией sigmoid можно представить в виде

$$output = sigmoid(dot(W, input) + b)$$

Реализованы две функции для вычисления результата работы сети по указанным выше формулам. Одна из них использует для вычислений библиотечные функции NumPy:

```
def np_predict(data, weights):
    data = data.copy()

    functions = [relu, relu, sigmoid]
    for i in range(len(functions)):
        data = np.dot(data, np.asarray(weights[i][0]))
        data += np.asarray(weights[i][1])
        data = functions[i](data)
    return data
```

Другая использует реализованные вручную функцую, где операции представлены как поэлементные операции над тензорами:

```
def naive_predict(data, weights):
    data = data.copy()

    functions = [relu, relu, sigmoid]
    for i in range(len(functions)):
        data =
np.asarray([naive_matrix_vector_dot(np.asarray(weights[i][0]).transpose()
, line) for line in data])
        data = naive_add_matrix_and_vector(data,
np.asarray(weights[i][1]))
        data = np.asarray([functions[i](line) for line in data])
    return data
```

Значения weights извлекаются из созданной модели. Они содержат пару [W, b] для каждого слоя сети.

Входные данные прогоняются через модель, np_predict и naive_predict до обучения и после обучения.

Результаты прогона до обучения:

```
MODEL
[[0.5       ]
 [0.47883466]
 [0.46143335]
 [0.44313088]
 [0.5022234 ]
 [0.51229155]
 [0.4567747 ]
 [0.46261713]]
NP PREDICT
[[0.5       ]
 [0.47883465]
 [0.46143335]
 [0.44313088]
 [0.50222339]
 [0.51229154]
 [0.45677471]
 [0.46261712]]
NAIVE PREDICT
[[0.5       ]
 [0.47883465]
 [0.46143335]
 [0.44313088]
 [0.50222339]
 [0.51229154]
 [0.45677471]
 [0.46261712]]
```

Результаты прогона после обучения:

```
MODEL
[[0.7942213 ]
 [0.97650033]
 [0.23365697]
 [0.9717884 ]
 [0.00703892]
 [0.07872358]
 [0.05429202]
 [0.8949635 ]]
NP PREDICT
[[0.79422133]
 [0.97650033]
 [0.23365697]
 [0.97178836]
 [0.00703893]
 [0.07872357]
 [0.05429205]
 [0.89496347]]
NAIVE PREDICT
[[0.79422133]
 [0.97650033]
 [0.23365697]
 [0.97178836]
 [0.00703893]
 [0.07872357]
 [0.05429205]
 [0.89496347]]
```

Результаты работы моделей и функций незначительно отличаются в восьмом символе после запятой, но в целом они получились очень близкими.