Задача 8

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:

1.Функция, в которой все операции реализованы как поэлементные операции над тензорами

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

Для проверки корректности работы функций необходимо:

1.Инициализировать модель и получить из нее веса.

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

3.Обучить модель и получить веса после обучения

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

Логическая операция: (a and c and b) xor (a or not b)

Выполнение

В переменную data были считанны данные из файла data.txt. Затем был вычислен ожидаемый результат логической операции на этих данных:

```
logical_result = np.array([int(logical_operation(row[0], row[1], row[2]))
                           for row in data])
print("Real result:")
print(logical_result)
```

Real result:

[1 1 0 0 1 1 1 0]

Затем была создана модель обучения из трех слоев:

```
model = Sequential()
model.add(Dense(8, activation='relu', input_dim=3))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', met-
rics=['accuracy'])
```

После были написаны функции, симулирующие работу модели: функции elements_operation() и numpe_operations(). В первой функции операции реализованы как поэлементные операции с тензорами, во второй – все реализованы с использованием операций над тензорами из numpy.

Первая функция:

```
def elements_operations(data, layers):
    for i in range(len(layers) - 1):
        data = relu(naive_matrix_matrix_dot(data,
layers[i].get_weights()[0]) + layers[i].get_weights()[1])

    data = sigmoid(naive_matrix_matrix_dot(data, layers[len(lay-
ers)-1].get_weights()[0]) + layers[len(layers)-1].get_weights()
[1])
    return data
```

Вторая функция:

```
def numpy_operations(data, layers):
    for i in range(len(layers) - 1):
        data = np.maximum(np.dot(data, layers[i].get_weights()[0])
+ layers[i].get_weights()[1], 0)

    data = sigmoid(np.dot(data, layers[len(layers)-
1].get_weights()[0]) + layers[len(layers)-1].get_weights()[1])
    return data
```

Для работы первой функции были реализованы дополнительные функции для работы с матрицами:

```
def relu(x):
    return np.maximum(x, 0)


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def naive_vector_dot(x, y):
```

```
    assert len(x.shape) == 1
    assert len(y.shape) == 1
    assert x.shape[0] == y.shape[0]
    z = 0.
    for i in range(x.shape[0]):
        z += x[i] * y[i]
    return z


def naive_matrix_matrix_dot(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 2
    assert x.shape[1] == y.shape[0]
    z = np.zeros((x.shape[0], y.shape[1]))
    for i in range(x.shape[0]):
        for j in range(y.shape[1]):
            z[i, j] += naive_vector_dot(x[i, :], y[:, j])
    return z
```

Перед тем, как обучить модель, датасет был прогнан через необученную:

```
Elementwise function:
[[0.5       ]
 [0.51407476]
 [0.55743346]
 [0.57106068]
 [0.56908976]
 [0.57209977]
 [0.62628556]
 [0.64200762]]
Numpy function:
[[0.5       ]
 [0.51407476]
 [0.55743346]
 [0.57106068]
 [0.56908976]
 [0.57209977]
 [0.62628556]
 [0.64200762]]
```

Обе функции вернули одинаковые значения.

Затем датасет был прогнан через обученную модель:

```
Elementwise function:
[[0.99106555]
 [0.99422757]
 [0.00876975]
 [0.0021821 ]
 [0.99987522]
 [0.99924228]
 [0.99100112]
 [0.00683157]]
Numpy function:
[[0.99106555]
 [0.99422757]
 [0.00876975]
 [0.0021821 ]
 [0.99987522]
 [0.99924228]
 [0.99100112]
 [0.00683157]]
```

Значения так же одинаковы для обеих функций. Так же эти значения соответствуют действительному значению вектора результата операции ([1 1 0 0 1 1 1 0])