Практическое задание №4

Вариант №6
(a and not b) or (c xor b)

Условие: 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.

Выполнение: Была составлена модель ИНС: на первом слое 8 нейронов, активирующая функция – RELU, параметр input_shape = 3 т.к. в тренировочных данных 3 столбца (a, b, c), на втором слое 32 нейрона, активирующая функция – RELU, в выходном слое 1 нейрон и активирующая функция sigmoid. В качестве оптимизатора используется adam, в качестве функции потери – бинарная кросс-энтропия, метрика – точность. Количество эпох установлено в 400.

```
model = Sequential()
model.add(Dense(8, activation='relu', input_shape=(3,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy])
```

Были написаны вспомогательные поэлементные функции:

`def naive_matrix_matrix_dot(x, y)` – функция, выполняющая скалярное произведение матриц.

`def naive_vector_dot(x, y)` - функция, выполняющая скалярное произведение векторов.

`def naive_relu(x)` – фукнция relu для матрицы x

`def sigmoid(x)` – функция sigmoid для матрицы x

Были реализованы функции, симулирующие работу построенной модели, используя формулу output = relu(dot(W, input) + b), где W – матрица весов слоя, b – вектор смещения слоя.

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

```
def naive_simulation(layers, input):
    output1 = naive_relu(naive_matrix_matrix_dot(input, layers[0].get_weights()[0]) + layers[0].get_weights()[1])
    output2 = naive_relu(naive_matrix_matrix_dot(output1, layers[1].get_weights()[0]) + layers[1].get_weights()[1])
    res = sigmoid(naive_matrix_matrix_dot(output2, layers[2].get_weights()[0]) + layers[2].get_weights()[1])
    return np.reshape(res, ((input.shape[0]), 1))
```

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

```
def np_simulation(layers, input):
    output1 = np.maximum(np.dot(input, layers[0].get_weights()[0]) + layers[0].get_weights()[1], 0)
    output2 = np.maximum(np.dot(output1, layers[1].get_weights()[0]) + layers[1].get_weights()[1], 0)
    res = sigmoid(np.dot(output2, layers[2].get_weights()[0]) + layers[2].get_weights()[1])
    return np.reshape(res, ((input.shape[0]), 1))
```

Прогоним датасет через обученную и необученную модель.

train_data:

0;0;0
0;0;1
0;1;0
0;1;1
1;0;0
1;0;1
1;1;0
1;1;1

labels:
0;
1;
1;
0;
1;
1;
1;
0;

Numpy untrained simulation:
[[0.5]
[0.49882147]
[0.50539292]
[0.50590487]
[0.48799276]
[0.49021686]
[0.50161931]
[0.48228239]]

Naive untrained simulation:
[[0.5]
[0.49882147]
[0.50539292]
[0.50590487]
[0.48799276]
[0.49021686]
[0.50161931]
[0.48228239]]

Untrained model predict:
[[0.5]
[0.49882147]
[0.5053929]

[0.50590485]
[0.48799276]
[0.49021685]
[0.50161934]
[0.4822824]]

Numpy trained simulation:
[[0.07421285]
[0.94602154]
[0.96585844]
[0.01427202]
[0.96686864]
[0.98050932]
[0.99950158]
[0.03236161]]

Naive untrained simulation:
[[0.07421285]
[0.94602154]
[0.96585844]
[0.01427202]
[0.96686864]
[0.98050932]
[0.99950158]
[0.03236161]]

Trained model predict:
[[0.07421285]
[0.94602156]
[0.96585846]
[0.014272]
[0.96686864]
[0.98050934]
[0.9995016]
[0.03236154]]

Из полученных данных можно сделать вывод, что отличия в результатах как до обучения, так и после минимальны. Значения, полученные с помощью функций naive_simulation и np_simulation полностью идентичны. Результаты, полученные с помощью model.predict() отличаются, начиная с 7 разряда после

запятой, это можно списать, например, на округление внутри функции model.predict().