МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №4

по дисциплине «Искусственные нейронные сети»

Тема: Операции с тензорами в библиотеке Keras

Студент гр. 8383 _____ Федоров И.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Задача.

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.

Вариант 7: (a or b) and (a xor not b)

Данные берутся из файлов *train.csv* и *labels.csv* с помощью функции Numpy *np.genfromtxt()*. Файлы выглядят следующим образом:

train.csv	*labels.csv*
0;0 0;1 1;0 1;1	0; 0; 0; 1;

Была создана следующая модель сети:

```
model = Sequential()
model.add(Dense(8, activation='relu', input_shape=(2,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])
```

Была реализована функция, симулирующая данную модель, в которой все операции реализованы с использованием операций над тензорами из NumPy:

```
def np_sim_model(layers, input):
    out_1 = np.maximum(np.dot(input, layers[0].get_weights()[0])
+ layers[0].get_weights()[1], 0)
    out_2 = np.maximum(np.dot(out_1, layers[1].get_weights()[0])
+ layers[1].get_weights()[1], 0)
    res = sigmoid(np.dot(out_2, layers[2].get_weights()[0]) +
layers[2].get_weights()[1])
    return  1 - res
```

Были реализованы вспомогательные функции поэлементной операции *relu* для матрицы (тензора второго ранга) и функции сигмоида:

```
def naive_relu(x):
    assert len(x.shape) == 2
    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] = max(x[i, j], 0)
    return x

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

Также были реализованы функция для скалярного произведения двух векторов, а также для скалярного произведения двух матриц:

```
def naive_vector_dot(x, y):
    assert len(x.shape) == 1      # убедиться что x вектор
    assert len(y.shape) == 1      # убедиться что y вектор
    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_dot(x, y):
    assert len(x.shape) == 2       # убедиться что x матрица
    assert len(y.shape) == 2       # убедиться что y матрица
    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]):
            row_x = x[i, :]
            column_y = y[:, j]
            z[i, j] = naive_vector_dot(row_x, column_y)
    return z
```

Была реализована функция, симулирующая работу модели, в которой все операции реализованы как поэлементные операции над тензорами:

```
def naive_sim_model(layers, input):
    out_1 = naive_relu(naive_matrix_dot(input,
layers[0].get_weights()[0]) + layers[0].get_weights()[1])
    out_2 = naive_relu(naive_matrix_dot(out_1,
layers[1].get_weights()[0]) + layers[1].get_weights()[1])
    res = sigmoid(naive_matrix_dot(out_2,
layers[2].get_weights()[0]) + layers[2].get_weights()[1])
    return 1 - res
```

В модели используется слой Dense, который реализует следующую операцию:

```
output = activation(dot(input, kernel) + bias)
```

где *activation* - функция активации, переданная в слой (в данном случае используется relu и sigmoid), *kernel* - матрица весов слоя, *bias* - вектор смещения, созданный слоем.

Был проведен эксперимент с необученной моделью и 2 функциями, получившие на вход слои с весами до обучения. Результаты приведены ниже:

```
Untrained model:
[[0.5 ]
[0.55096054]
[0.52629894]
[0.5239351 ]]
Numpy fun
[[0.5 ]
[0.55096054]
[0.5262989 ]
[0.52393504]]
Naive fun
[[0.5 ]
[0.55096054]
[0.5262989 ]
[0.52393504]]
```

Можно заметить, что результаты функций и модели совпадают с достаточно высокой точностью.

Был проведен эксперимент с обученной моделью и 2 функциями, получившие на вход слои с весами после обучения. Результаты ниже:

```
Trained model:
[[3.8862476e-04]
[1.7948569e-04]
[5.4018148e-03]
[9.9822670e-01]]
Numpy fun
[[3.88624982e-04]
[1.79485831e-04]
[5.40181737e-03]
[9.98226678e-01]]
Naive fun
[[3.88624982e-04]
[1.79485831e-04]
[5.40181737e-03]
[9.98226678e-01]]
```

Видно, что аналогично предыдущему эксперименты, результаты одинаковы с высокой точностью, также можно заметить, что модель и функции правильно оценивают результат.

Дополнение:

$a \oplus b = \bar{a} \square b + a \square \bar{b}$

$a \oplus \bar{b} = \bar{a} \square \bar{b} + a \square b$

$(a\,or\,b)\ and\ (a\ xor\ not\ b) = (a\ or\ b)\ and\ ((\bar{a}\ and\ \bar{b})\ or\ (a\ and\ b)) =$

$= (a\ or\ b)\ and\ (\bar{a}\ and\ \bar{b})\ or\ (a\ or\ b)\ and\ (a\ and\ b) = \bar{a}ab + \bar{a}\bar{b}b + ab + ab = a\ and\ b$

Можно было взять более простую модель.