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

ОТЧЕТ

по практической работе № 4

по дисциплине «Искусственные нейронные сети»

Студент гр. 8383 _____ Костарев К.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

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:

1. Инициализировать модель и получить из нее веса (Как получить веса слоя, Как получить список слоев модели)
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.

Вариант 3.

(a and b) or c

Выполнение работы.

В данном случае была выбрана модель с двумя полносвязными слоями с функцией активации Relu по 8 и 16 нейронов соответственно. Параметр на входном слое input_dim = 3. Выходной слой содержит единственный нейрон с функцией активации сигмоида.

```
model = models.Sequential()
model.add(layers.Dense(8, activation='relu', input_dim=3))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

В качестве функции оптимизации выбрана RMSProp, функции потерь – бинарная кроссэнтропия, метрика – точность.

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

Обучение проводится в течение 100 эпох.

```
model.fit(x, y, epochs=100, batch_size=1, verbose=0)
```

Реализованы функции для симуляции модели naïve_model и numpy_model.

```
def numpy_model(W, B, input):
    x = input.copy()
    for i in range(len(W)):
        x = np.dot(x, W[i])
        x += B[i]
        x = numpy_relu(x) if i != range(len(W))[-1] else numpy_sigmoid(x)
    return x


def naive_model(W, B, input):
    x = input.copy()
    for i in range(len(W)):
        x = np.array([naive_matrix_dot(v, W[i]) for v in x])
        x = np.array([naive_add(v, B[i]) for v in x])
        x = [numpy_relu(v) for v in x] if i != range(len(W))[-1] else
[numpy_sigmoid(v) for v in x]
    return np.array(x)
```

Демонстрация работы.

X =

[[0 0 0]

[0 0 1]

[0 1 0]

[0 1 1]

[1 0 0]

[1 0 1]

[1 1 0]

[1 1 1]],

Y = [0 1 0 1 0 1 1 1]

- Необученная модель:

Predict

[[0.5]

[0.47172162]

[0.47084087]

[0.4946049]

[0.4969492]

[0.42986822]

[0.4177384]

[0.4372345]]

NumPy

[[0.5]

[0.47172163]

[0.47084087]

[0.49460488]

[0.4969492]

[0.42986821]

[0.41773841]

[0.43723449]]

Regular

[[0.5]

[0.47172163]

[0.47084087]

[0.49460488]

[0.4969492]

[0.42986821]

[0.41773841]

[0.43723449]]

- Обученная модель:

Predict

[[0.42488667]

[0.8482004]

[0.55823934]

[0.9086031]

[0.46283665]

[0.9298555]

[0.71736455]

[0.94773346]]

NumPy

[[0.42488665]

[0.84820035]

[0.55823937]

[0.90860309]

[0.46283664]

[0.9298555]

[0.71736455]

[0.94773338]]

Regular

[[0.42488665]

[0.84820035]

[0.55823937]

[0.90860309]

[0.46283664]

[0.9298555]

[0.71736455]

[0.94773338]]