Практическое задание №4, вариант 5(1)

Ефимовой Марии, 8382

Задание:

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 (Как получить веса слоя, Как получить список слоев модели) 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.

Ход работы:

В качестве модели была выбрана ИНС со следующей структурой:

В качестве оптимизатора выбран “adam”, функция потерь: бинарная кросс-энтропия, метрика: точность (accuracy).

```
def get_model(self):
    model = Sequential()
    model.add(Dense(16, activation='relu', input_shape=(3,)))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

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

Для симуляции работы нейронной сети на основе весов были реализованы две функции:

- def sim_by_el(self): - функция, в которой все операции реализованы как поэлементные операции над тензорами.
- def sim_by_numpy(self)- функция, в которой все операции реализованы с использованием операций над тензорами из NumPy.

А также функции:

- sigmoid для реализации функции sigmoid
- relu для реализации функции relu

Изначально имеем заполненную матрицу, в функции def sim_by_el(self) проходимся по двумерному массиву и записываем в переменную s += data[i][k] * weights[l][0][k][j], а далее результат с помощью relu и sigmoid: res[i][j] = self.relu(s + weights[l][1][j]), res[i][j] = self.sigmoid(s + weights[l][1][j]).
В функции def sim_by_numpy(self) изначально записываем результат в res = self.data.copy(), далее

применяем такие же формулы, как и в первой функции: res = self.relu(np.dot(res, weights[i][0]) + weights[i][1]), res = self.sigmoid(np.dot(res, weights[-1][0]) + weights[-1][1]).

Изначальные данные:

```
def __init__(self):
    self.data = np.array([[0, 0, 0],
                          [1, 1, 0],
                          [0, 0, 1],
                          [0, 1, 1],
                          [0, 1, 0],
                          [1, 0, 1],
                           [1, 0, 0]])
    self.model = self.get_model()
```

Операция варианта 1:

```
def operation(self, x):
    return (x[0] ^ x[1]) and (not(x[1] ^ x[2]))
```

Выходные данные (200 эпох):

```
Model res:
[[0.02517006]
 [0.01505628]
 [0.02449915]
 [0.85220504]
 [0.00416464]
 [0.02102679]
 [0.9512093 ]]
Operations with elements res:
[[0.02517008]
 [0.01505629]
 [0.02449914]
 [0.85220505]
 [0.00416464]
 [0.02102679]
 [0.95120927]]
Using numpy res:
[[0.02517008]
 [0.01505629]
 [0.02449914]
 [0.85220505]
 [0.00416464]
 [0.02102679]
 [0.95120927]]
Real res:
 [0, 0, 0, True, False, False, True]
```