Лабораторная работа 4

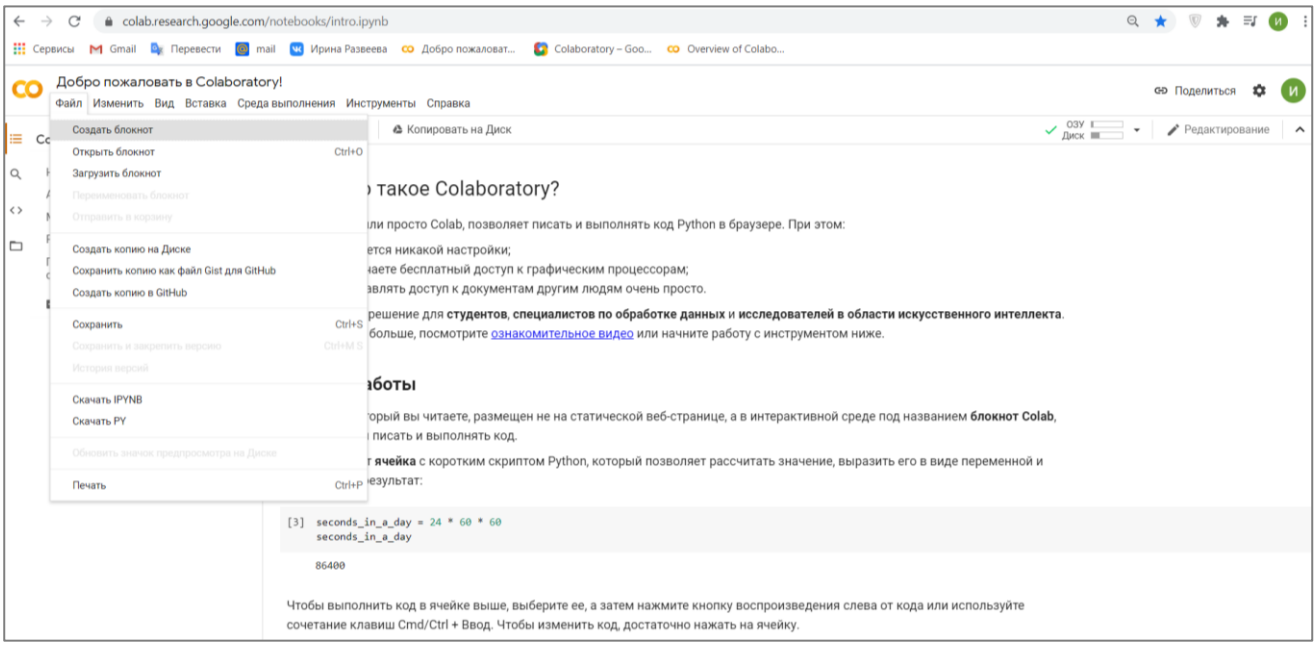
Построение нейросети. Часть 1

Цель работы: построить нейронную сеть на Python в Google Colaboratory.

1. Откройте платформу по ссылке:

https://colab.research.google.com/notebooks/intro.ipynb

2. Перейдя во вкладку Файл выбрать «Создать блокнот».



3. Перед нами пустая рабочая область.

Документ, который вы читаете, размещен не на статической веб-странице, а в интерактивной среде под названием блокнот Colab, позволяющей писать и выполнять код.

Оригинальная модель, предназначенная для моделирования того, как человеческий мозг обрабатывает визуальные данные и учится распознавать объекты, была предложена Фрэнк Розенблаттв 1950-х гг.

Перцептрон принимает один или несколько двоичных входов x1, x2,…, xn и создает двоичный выход:

Чтобы вычислить результат, вы должны:

- иметь веса w1, w2,…, wn, выражающие важность соответствующего ввода
- двоичный выход (0 или 1) определяется тем, является ли взвешенная сумма ∑ wj xj большей или меньшей некоторого порога

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \text{threshold} \\ 1 & \text{otherwise} \end{cases}$$

Создадим собственную нейронную сеть.

	Input			Output
Example 1	0	0	1	0
Example 2	1	1	1	1
Example 3	1	0	1	1
Example 4	0	1	1	0
New situation	1	0	0	?

Первые четыре примера назовем тренировочной выборкой, где Input — входный сигнал, Output — выходной сигнал.

Необходимо определить закономерность. Что должно стоять на месте «?»

Вероятно, вы заметили, что выходной сигнал всегда равен самой левой входной колонке. Таким образом ответ будет 1.

Процесс обучения нейронной сети

Как же должно происходить обучение нашего нейрона, чтобы он смог ответить правильно? Мы добавим каждому входу вес, который может быть

положительным или отрицательным числом. Вход с большим положительным или большим отрицательным весом сильно повлияет на выход нейрона. Прежде чем мы начнем, установим каждый вес случайным числом. Затем начнем обучение:

1. Берем входные данные из примера обучающего набора, корректируем их по весам и передаем по специальной формуле для расчета выхода нейрона.
2. Вычисляем ошибку, которая является разницей между выходом нейрона и желаемым выходом в примере обучающего набора.
3. В зависимости от направления ошибки слегка отрегулируем вес.
4. Повторите этот процесс 10 000 раз.

В конце концов вес нейрона достигнет оптимального значения для тренировочного набора. Если мы позволим нейрону «подумать» в новой ситуации, которая сходна с той, что была в обучении, он должен сделать хороший прогноз.

Формула для расчета выхода нейрона

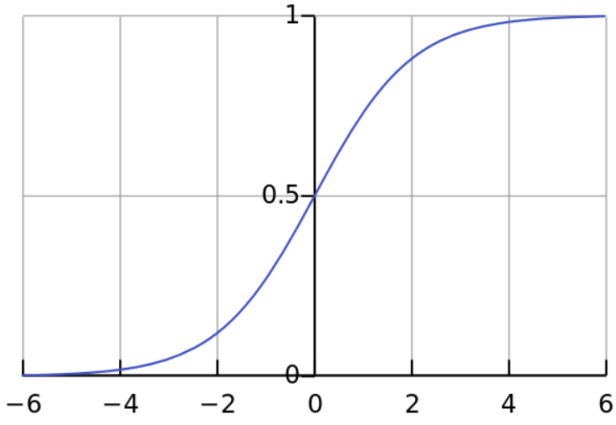
Вам может быть интересно, какова специальная формула для расчета выхода нейрона? Сначала мы берем взвешенную сумму входов нейрона, которая:

$$\sum weight_i \cdot input_i = weight1 \cdot input1 + weight2 \cdot input2 + weight3 \cdot input3$$

Затем мы нормализуем это, поэтому результат будет между 0 и 1. Для этого мы используем математически удобную функцию, называемую функцией Sigmoid:

$$\frac{1}{1+e^{-x}}$$

Если график нанесен на график, функция Sigmoid рисует S-образную кривую.



Подставляя первое уравнение во второе, получим окончательную формулу для выхода нейрона:

$$\text{Output of neuron} = \frac{1}{1 + e^{-(\sum weight_i input_i)}}$$

Формула для корректировки веса

Во время тренировочного цикла мы корректируем веса. Но насколько мы корректируем вес? Мы можем использовать формулу «Взвешенная по ошибке» формула:

$$\text{Adjust weights by} = error \cdot input \cdot SigmoidCurveGradient(output)$$

Почему эта формула? Во-первых, мы хотим сделать корректировку пропорционально величине ошибки. Во-вторых, мы умножаем на входное значение, которое равно 0 или 1. Если входное значение равно 0, вес не корректируется. Наконец, мы умножаем на градиент сигмовидной кривой. Чтобы понять последнее, примите во внимание, что:

1. Мы использовали сигмовидную кривую для расчета выхода нейрона.
2. Если выходной сигнал представляет собой большое положительное или отрицательное число, это означает, что нейрон так или иначе был достаточно уверен.
3. Из графика Сигмоиды мы можем видеть, что при больших числах кривая Сигмоида имеет небольшой градиент.

Если нейрон уверен, что существующий вес правильный, он не хочет сильно его корректировать. Умножение на градиент сигмовидной кривой делает именно это.

Градиент Сигмоды получается, если посчитать взятием производной:

$$\text{SigmoidCurveGradient(output)} = output \cdot (1 - output)$$

Вычитая второе уравнение из первого получаем итоговую формулу:

$$\text{Adjust weights by} = error \cdot input \cdot output \cdot (1 - output)$$

Существуют также другие формулы, которые позволяют нейрону учиться быстрее, но приведенная имеет значительное преимущество: она простая.

Написание Python кода

Хоть мы и не будем использовать библиотеки с нейронными сетями, мы импортируем 4 метода из математической библиотеки numpy. А именно:

exp — экспоненцирование

array — создание матрицы

dot — перемножения матриц

random — генерация случайных чисел

Например, мы можем использовать array() для представления обучающего множества, показанного ранее.

```
from numpy import exp, array, random, dot
training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
training_set_outputs = array([[0, 1, 1, 0]]).T
```

«.T» — функция транспонирования матриц. Итак, теперь мы готовы для более красивой версии исходного кода. Заметьте, что на каждой итерации мы обрабатываем всю тренировочную выборку одновременно.

Запустим в Google Colaboratory следующий код:

```
    # Подключаем необходимые библиотеки
from numpy import exp, array, random, dot


class NeuralNetwork():
    def __init__(self):
        # Заполните генератор случайных чисел, чтобы он генерировал одни и те
# же числа при каждом запуске программы
        random.seed(1)

        # Мы моделируем один нейрон с 3 входными подключениями и 1 выходным
        # Мы назначаем случайные веса матрице 3 x 1 со значениями в диапазоне
# от -1 до 1 и средним значением 0.

        self.synaptic_weights = 2 * random.random((3, 1)) - 1

    # Сигмовидная функция, описывающая S-образную кривую.
    # Мы передаем взвешенную сумму входных данных через эту функцию, чтобы
# нормализовать их от 0 до 1.
    def __sigmoid(self, x):
        return 1 / (1 + exp(-x))

    # Производная сигмовидной функции.
    # Это градиент сигмовидной кривой.
    # Это показывает, насколько мы уверены в существующем весе.
    def __sigmoid_derivative(self, x):
```

```
        return x * (1 - x)

    # Мы обучаем нейронную сеть методом проб и ошибок.
    # Каждый раз регулируя синаптические веса.
    def train(self, training_set_inputs, training_set_outputs, number_of_train
ing_iterations):
        for iteration in range(number_of_training_iterations):
            # Передайем обучающий набор через нашу нейронную сеть (отдельный
#нейрон).
            output = self.think(training_set_inputs)

            # Вычислите ошибку (разницу между желаемым и прогнозируемым #
выходными данными).
            error = training_set_outputs - output

            # Умножьте ошибку на вход и снова на градиент сигмовидной кривой.
            # Это означает, что менее уверенные веса корректируются больше.
# Это означает, что входные данные, которые равны нулю, не вызывают изменения
# весов.
            adjustment = dot(training_set_inputs.T, error * self.__sigmoid_der
ivative(output))

            # Отрегулируйте веса.
            self.synaptic_weights += adjustment


    def think(self, inputs):
        # Передайте входные данные через нашу нейронную сеть (наш единственный
 # нейрон).
        return self.__sigmoid(dot(inputs, self.synaptic_weights))


if __name__ == "__main__":

    #Инициализируйте нейронную сеть с одним нейроном.
    neural_network = NeuralNetwork()

    print ("Случайные начальные синаптические веса:")
    print (neural_network.synaptic_weights)

    # Обучающий набор. У нас есть 4 примера, каждый из которых состоит из 3
# входных значений и 1 выходного значения.
    training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
    training_set_outputs = array([[0, 1, 1, 0]]).T

    # Обучите нейронную сеть с помощью обучающего набора.
    # Количество итераций - 10 000 раз
    neural_network.train(training_set_inputs, training_set_outputs, 10000)

    print ("Новые синаптические веса после тренировки: ")
    print (neural_network.synaptic_weights)

    # Протестируйте нейронную сеть в новой ситуации.
```

```
print ("Учитывая новую ситуацию [1, 0, 0] -> ?: ")
print (neural_network.think(array([1, 0, 0])))
```

```
Случайные начальные синаптические веса:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Новые синаптические веса после тренировки:
[[ 9.67299303]
 [-0.2078435 ]
 [-4.62963669]]
Учитывая новую ситуацию [1, 0, 0] -> ?:
[0.99993704]
```

Самостоятельная работа:

1. Какие функции активации вы знаете?

2. Попробуйте использовать другую функцию активации. Что означает полученный результат?

3. В строке

neural_network.train(training_set_inputs, training_set_outputs, 10000)

замените количество итераций на 1000, на 100.

Что означает полученный результат?