В этой серии статей вы изучите как проектировать и обучать нейронную сеть, которая распознает рукописные цифры. В процессе мы достигнем точности в 99% и изучим эвристики, которые профессионалы используют для эффективного обучения своих моделей.

Мы будем использовать набор данных MNIST (коллекция из 60к размеченных цифр). Мы решим эту проблему за менее чем 100 строк кода на Python/TensorFlow.

Что мы изучим:

1. Что такое нейронная сеть и как обучить её
2. Как построить однослойную нейронную сеть, используя TensorFlow
3. Как добавлять слои в нейронную сеть
4. Советы по обучению: переобучение, отсев, распад скорости обучения …
5. Как устранять неисправности в глубоких нейронных сетях
6. Как строить сверточные нейронные сети

Что нам для этого нужно:

1. python 2 или 3 (рекомендуется 3 версия)
2. TensorFlow
3. Matplotlib(библиотека для визуализаций на python)

Чтобы установить все необходимое программное обеспечение следуйте инструкциям в файле (приложение 1)

Теория

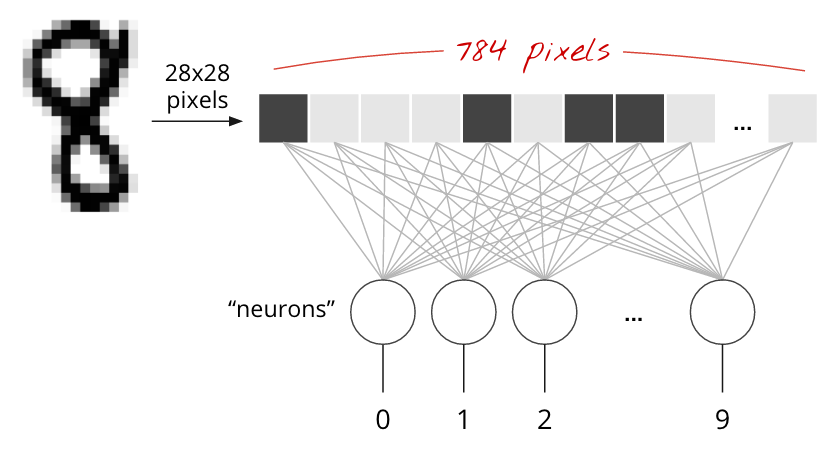
Обучение нейронной сети

Наша нейронная сеть берет рукописные цифры и классифицирует их, то есть распознает их как 0, 1, 2 и так далее до 9. Она делает это на основе внутренних переменных (весов и смещений), которые должны иметь правильные значения для хорошей классификации. Мы получаем «правильные значения» в процессе обучения (мы изучим этот процесс позже). Прямо сейчас вам необходимо понимать, что весь процесс обучения выглядит примерно так:

Примеры из размеченного набора => обновления весов и смещений => улучшение классификации (цикл)

Однослойная нейронная сеть

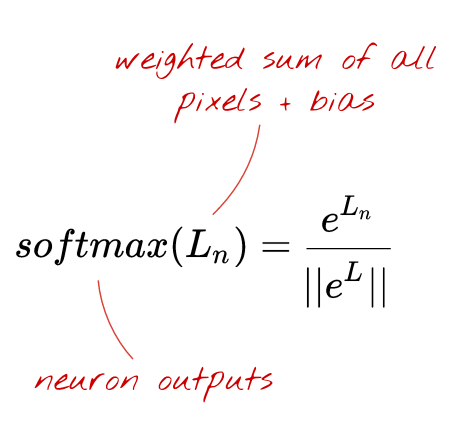
Рукописные цифры из набора данных MNIST представляют из себя черно-белые изображения размером 28\*28 пикселей. Простейшим подходом для их классификации является использование 28\*28 = 784 пикселей в качестве входов для однослойной нейронной сети.



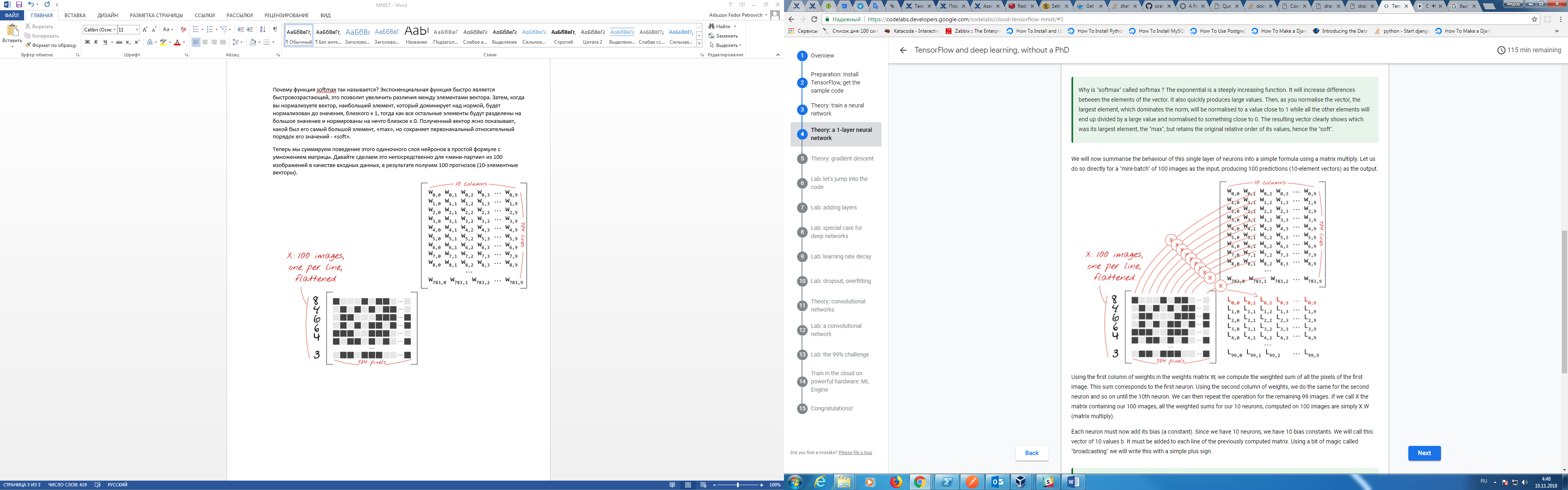
Каждый «нейрон» в нейронной сети делает взвешенную сумму всех своих входов, добавляет константу, называемую «смещением», а затем передает результат через некоторую нелинейную функцию активации.

Здесь мы создаем 1-слойную нейронную сеть с 10 выходными нейронами, так как мы хотим классифицировать цифры на 10 классов (от 0 до 9).

Для задач классификации лучше всего использовать функцию активации softmax.

Почему функция softmax так называется? Экспоненциальная функция быстро является быстровозрастающей, это позволит увеличить различия между элементами вектора. Затем, когда вы нормализуете вектор, наибольший элемент, который доминирует над нормой, будет нормализован до значения, близкого к 1, тогда как все остальные элементы будут разделены на большое значение и нормированы на нечто близкое к 0. Полученный вектор ясно показывает, какой был его самый большой элемент, «max», но сохраняет первоначальный относительный порядок его значений - «soft».

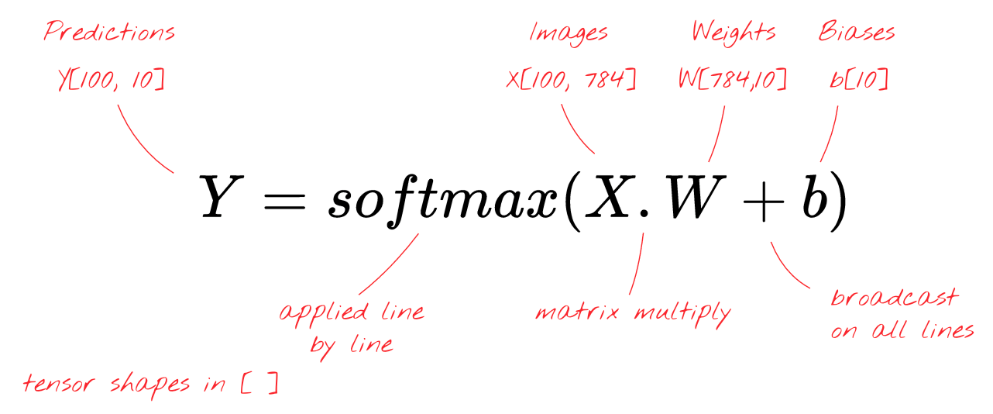
Теперь мы суммируем поведение этого одиночного слоя нейронов в простой формуле с умножением матрицы. Давайте сделаем это непосредственно для «мини-партии» из 100 изображений в качестве входных данных, в результате получим 100 прогнозов (10-элементные векторы).



Используя первый столбец весов в матрице весов W, мы вычисляем взвешенную сумму всех пикселей первого изображения. Эта сумма соответствует первому нейрону. Используя второй столбец весов, мы делаем то же самое для второго нейрона и так далее до 10-го нейрона. Затем мы можем повторить операцию для остальных 99 изображений. Если мы назовем X матрицей, содержащей наши 100 изображений, все взвешенные суммы для наших 10 нейронов, рассчитанные на 100 изображений, просто X.W (перемножение матриц).

Каждый нейрон должен теперь добавить свой смещение (константу). Так как у нас 10 нейронов, мы имеем 10 констант смещения. Мы будем называть этот вектор из 10 значений b. Он должен быть добавлен к каждой строке ранее вычисленной матрицы.

Наконец, мы применяем функцию активации softmax и получаем формулу, описывающую 1-слойную нейронную сеть, применяемую к 100 изображениям:



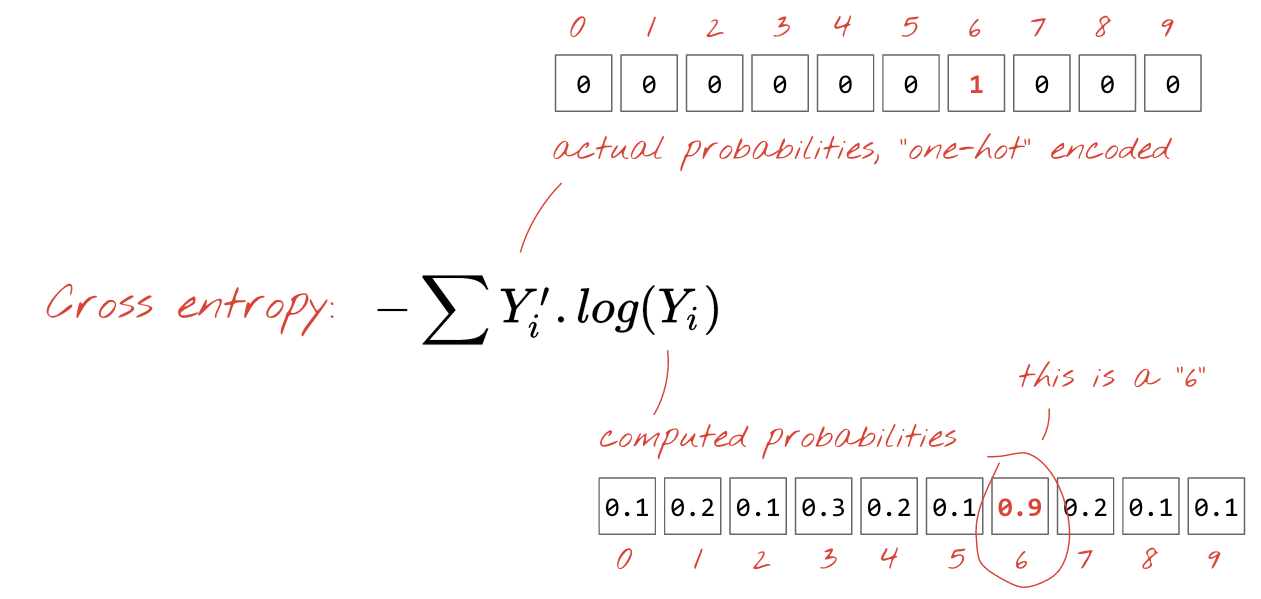
Кстати, что такое «тензор»?

«Тензор» подобен матрице, но с произвольным числом измерений. Одномерным тензором является вектор. Двухмерным тензором является матрица. И тогда у вас могут быть тензоры с размерами 3, 4, 5 или более.

Градиентный спуск

Теперь, когда наша нейронная сеть производит прогнозы на входных изображениях, нам нужно измерить, насколько они хороши, то есть расстояние между тем, что сеть говорит нам и что мы знаем. Помните, что у нас есть истинные метки для всех изображений в этом наборе данных.

Любое расстояние будет работать, даже обычное эвклидово расстояние, но для задач классификации расстояние, называемое «кросс-энтропией», более эффективно.



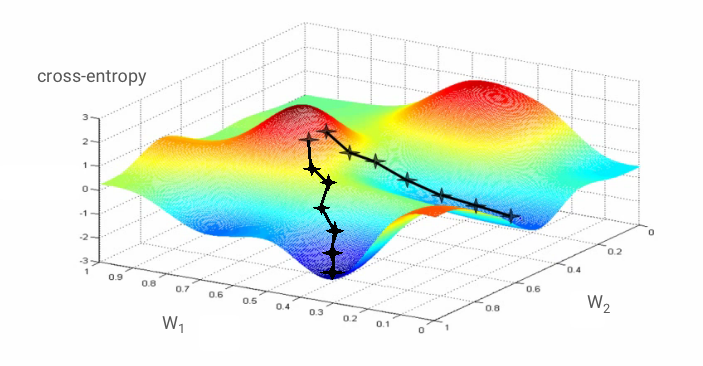
«One-hot» кодирование означает, что вы представляете метку «6», используя вектор из 10 значений с нулями, но 6-е значение равно 1. Это удобно, потому что формат очень похож на то, как наша нейронная сеть выводит предсказания (вектор из 10 значений).

«Обучение» нейронной сети на самом деле означает использование обучающих изображений и меток для корректировки весов и смещений, чтобы минимизировать функцию потери кросс-энтропии. Вот как это работает.

Кросс-энтропия - это функция весов, смещений, пикселей обучающего изображения и его известной метки.

Если вычислить частные производные кросс-энтропии относительно всех весов и всех смещений, мы получим «градиент», рассчитанный для данного изображения, метки и приведенного значения весов и смещений. Помните, что у нас есть 7850 весов и смещений, поэтому вычисление градиента потребует большого количества вычислений. К счастью, TensorFlow сделает это за нас.

Математическое свойство градиента состоит в том, что он указывает переменную, по которой мы будем расти быстрее всего. Поскольку мы хотим идти туда, где кросс-энтропия низкая, мы идем в противоположном направлении. Мы обновляем веса и смещения на части градиента и делаем то же самое снова, используя следующую партию обучающих изображений. Надеясь, что это приведет нас к дну ямы, где кросс-энтропия минимальна.



На этом рисунке кросс-энтропия представлена как функция от 2 весов. На самом деле их гораздо больше. Алгоритм градиентного спуска следует по пути наискорейшего спуска на локальный минимум. Размеченные изображения изменяются на каждой итерации, так что мы сходимся к локальному минимуму, который работает для всех изображений.