Оглавление

Cost function

0:00

Нейронные сети - один из самых мощных алгоритмов обучения, которые мы имеем сегодня. В этом и в следующих нескольких видеороликах я хотел бы начать говорить об алгоритме обучения для подбора параметров нейронной сети с учетом набора тренировок. Как и при обсуждении большинства наших алгоритмов обучения, мы начнем с обсуждения функции стоимости для установки параметров сети.

0:22

Я собираюсь сосредоточиться на применении нейронных сетей к задачам классификации. Итак, предположим, что у нас есть сеть, показанная слева. И предположим, что у нас есть обучающий набор, такой как x I, y I пары примера обучения M.

0:38

Я собираюсь использовать верхний регистр L для обозначения общего количества слоев в этой сети. Итак, для сети, показанной слева, мы бы имели капитал L равный 4. Я собираюсь использовать S-индекс L для обозначения количества единиц, то есть числа нейронов. Не считая единицы смещения в их сети L. Так, например, у нас будет S-один, равный там, равный S три единицы, S два в моем примере - пять единиц. И выходной слой S four, который также равен S L, поскольку капитал L равен четырем. Выходной слой в моем примере под ним имеет четыре единицы.

1:17

Мы рассмотрим два типа проблем классификации. Первая - это двоичная классификация, где метки y равны 0 или 1. В этом случае у нас будет 1 единица вывода, поэтому этот блок нейронной сети сверху имеет 4 выходных блока, но если бы у нас была двоичная классификация, у нас было бы только одно который вычисляет h (x). И выход нейронной сети будет h (x) будет действительным числом. И в этом случае количество единиц вывода, S L, где L снова является индексом конечного слоя. Потому что это количество слоев, которые мы имеем в сети, поэтому количество единиц, которые мы имеем в выходном слое, будет равно 1. В этом случае, чтобы упростить запись позже, я также собираюсь установить K = 1, чтобы вы может думать о K, также обозначающем количество единиц в выходном слое. Второй тип проблемы классификации, который мы рассмотрим, будет проблемой многоклассовой классификации, где мы можем иметь K различных классов. Таким образом, наш ранний пример имел это представление для y, если у нас есть 4 класса, и в этом случае мы будем иметь единицы вывода капитала K и нашу гипотезу или выходные векторы, которые являются K-мерными. И количество выходных единиц будет равно K. И обычно мы будем иметь K больше или равно 3 в этом случае, потому что, если у нас есть две причины, то нам не нужно использовать один метод всех стихов. Мы используем один метод для всех стихов только в том случае, если мы имеем K больше или равно V классам, поэтому, имея только два класса, нам нужно будет использовать только одну верхнюю единицу. Теперь давайте определим функцию стоимости для нашей нейронной сети.

3:03

Функция стоимости, которую мы используем для нейронной сети, будет обобщением той, которую мы используем для логистической регрессии. Для логистической регрессии мы использовали для минимизации функции стоимости J (theta), которая составляла минус 1 / м этой функции стоимости, а затем плюс этот дополнительный член регуляризации здесь, где это была сумма от J = 1 до n, поскольку мы не упорядочивали член смещения тета0.

3:31

Для нейронной сети наша функция затрат будет обобщением этого. Где вместо того, чтобы иметь в основном только один, который является единицей вывода сжатия, мы можем вместо этого использовать K из них. Итак, вот наша функция стоимости. Теперь наша новая сеть выводит векторы в R K, где R может быть равным 1, если у нас есть проблема с двоичной классификацией. Я буду использовать обозначение h (x) i для обозначения i-го выхода. То есть h (x) является k-мерным вектором, и поэтому этот индекс i просто выбирает i-й элемент вектора, который выводится моей нейронной сетью.

4:08

Моя стоимость J (theta) теперь будет следующей. Is - 1 над M суммы аналогичного термина к тому, что мы имеем для логистической регрессии, за исключением того, что мы имеем сумму от K равно 1 до K. Это суммирование в основном представляет собой сумму по моему K-выводу. Единица. Поэтому, если у меня есть четыре выходных блока, то есть если в конечном слое моей нейронной сети есть четыре выходных блока, то это сумма от k равна от одной до четырех, в основном, функции стоимости алгоритма логистической регрессии, но суммируя эту функцию стоимости по каждому из мои четыре устройства вывода по очереди. И поэтому вы, в частности, замечаете, что это относится к Yk Hk, потому что мы в основном принимаем верхние юниты K и сравниваем это

со значением Yk, которое является одним из тех векторов, которые говорят, какую стоимость он должен быть. И, наконец, второй член здесь - термин регуляризации, аналогичный тому, что у нас было для логистической регрессии. Этот термин суммирования выглядит действительно сложным, но все, что он делает, это суммирование по этим терминам theta j i l для всех значений i j и l. За исключением того, что мы не суммируем суммы, соответствующие этим значениям смещения, как у нас для логистической прогрессии. Полностью мы не суммируем выражения, отвечающие на то, где i равно 0. Таким образом, потому что, когда мы вычисляем активацию нейрона, у нас есть такие термины. Theta i 0. Plus theta i1, x1 плюс и так далее. Там, где я предполагаю, что там есть два, это первый хит. И поэтому значения с нулем там, что соответствует чему-то, что умножается на x0 или a0. И поэтому это похоже на блок предвзятости и по аналогии с тем, что мы делали для логистической прогрессии, мы не будем суммировать эти термины в нашем терминах регуляризации, потому что мы не хотим их регуляризовать и строим их значения как ноль. Но это всего лишь одно возможное соглашение, и даже если вы должны были бы суммировать значение i, равное 0 до Sl, это будет работать примерно так же и не имеет большого значения. Но, может быть, эта конвенция о нерегулировании срока смещения несколько более распространена.

6:33

Итак, это функция стоимости, которую мы собираемся использовать для нашей нейронной сети. В следующем видео мы начнем говорить об алгоритме для оптимизации функции затрат.

Cost function

0:00

В предыдущем видео мы говорили о функции стоимости для нейронной сети. В этом видео, давайте начнем говорить об алгоритме, чтобы попытаться минимизировать функцию стоимости. В частности, мы поговорим об обратном алгоритме распространения.

0:13

Вот функция стоимости, которую мы записали в предыдущем видео. Мы хотели бы найти параметры theta, чтобы попытаться минимизировать j тета. Чтобы использовать либо градиентный спуск, либо один из алгоритмов предварительной оптимизации. Нам нужно сделать так, чтобы написать код, который принимает этот вход параметры theta и вычисляет j тета и эти частичные производные термины. Помните, что параметры в нейронной сети этих вещей, theta superscript l subscript ij, то есть действительное число и т. Д., Это частичные производные термины, которые нам нужно вычислить. Чтобы вычислить функцию стоимости j тэта, мы просто используем эту формулу здесь, и поэтому, что я хочу сделать для большей части этого видео, основное внимание уделено разговорам о том, как мы можем вычислить эти частичные производные термины. Начнем с того, что расскажем о случае, когда у нас есть только один пример обучения, поэтому представьте, если вы сделаете так, что весь наш набор тренировок содержит только один пример обучения, который представляет собой пару xy. Я не собираюсь писать x1y1, просто напишу это. Напишите один пример обучения как xy, и давайте рассмотрим последовательность вычислений, которые мы будем делать с этим примером обучения.

1:25

Первое, что мы делаем, это применить прямое распространение, чтобы вычислить, действительно ли гипотезы выходят с учетом ввода. Конкретно, названный a (1) является значением активации этого первого слоя, который был там. Итак, я собираюсь установить это на x, и тогда мы собираемся вычислить z (2), равный tota (1) a (1), a a (2) равно g, сигмоидальная функция активации, примененная к z (2) и это даст нам наши активации для первого среднего слоя. Это для второго уровня сети, и мы также добавляем эти условия смещения. Затем мы применяем еще 2 шага из этих четырех и распространение для вычисления a (3) и a (4), которое также является верхним из гипотез h of x. Таким образом, это наша векторизованная реализация прямого распространения и позволяет нам вычислять значения активации для всех нейронов в нашей нейронной сети.

2:27

Затем, чтобы вычислить производные, мы будем использовать алгоритм, называемый обратным распространением.

2:34

Интуиция алгоритма обратного распространения заключается в том, что для каждой ноты мы собираемся вычислить термин дельта-надстрочный индекс l индекс j, который каким-то образом будет представлять ошибку примечания j в слое l. Итак, напомним, что надстрочный индекс l, который активирует j единицы в слое l, и поэтому этот дельта-член в некотором смысле собирается захватить нашу ошибку в активации этого нейронного дуэта. Итак, как мы могли бы пожелать, чтобы активация этой заметки несколько отличалась. Конкретно, принимая пример нейронной сети, который у нас есть справа, который имеет четыре слоя. И поэтому капитал L равен 4. Для каждого блока вывода мы собираемся вычислить этот дельта-термин. Итак, дельта для единицы единицы в четвертом слое равна

3:23

просто активация этой единицы минус то, что было фактическим значением 0 в нашем примере обучения.

3:29

Итак, этот член здесь также может быть записан h из x индекса j, справа. Таким образом, этот дельта-терм - это просто разница между выводом гипотез и значением y в нашем обучающем наборе, тогда как y нижний индекс j является j элемента векторного значения y в нашем помеченном наборе тренировок.

3:56

И, кстати, если вы думаете о дельта a и y как о векторах, тогда вы также можете взять их и придумать его векторизованную реализацию, которая просто дельта 4 устанавливается как a4 минус y. Здесь, где каждый из этих дельта 4 a4 и y, каждый из них представляет собой вектор, размер которого равен числу выходных блоков в нашей сети.

4:25

Итак, мы теперь вычислили дельту 4 эры для нашей сети.

4:31

Далее мы вычисляем дельта-термины для более ранних слоев в нашей сети. Вот формула для вычисления delta 3: delta 3 равна tta 3 транспонированным временам delta 4. И это время точки, это операция умножения элемента у

4:47

что мы знаем из MATLAB. Таким образом, треугольник 3 переносит дельта 4, это вектор; g prime z3, который также является вектором, и поэтому точка-точка находится в умножении элемента у между этими двумя векторами.

5:01

Этот член g prime z3, который формально является фактически производной функции активации g, оцененной на входных значениях, заданных z3. Если вы знаете исчисление, вы можете попытаться разобраться в этом сами и увидеть, что вы можете упростить его до того же ответа, который я получаю. Но я просто скажу вам прагматично, что это значит. Что вы делаете, чтобы вычислить это g prime, эти производные члены - всего лишь a3 dot times1 минус A3, где A3 - вектор активаций. 1 - вектор единиц, а A3 - активация вектора значений активации для этого слоя. Затем вы применяете аналогичную формулу для вычисления дельта 2, где снова можно вычислить, используя аналогичную формулу.

5:48

Только теперь это похоже на a2, и я тогда докажу это здесь, но на самом деле вы можете это доказать, если знаете исчисление, что это выражение равно математически, производная функции g функции активации, которую я обозначая через g prime. И, наконец, вот и все, и нет термина delta1, потому что первый уровень соответствует входному слою, и это только функция, которую мы наблюдали в наших наборах обучения, так что у нее нет никакой ошибки, связанной с этим. Это не похоже, вы знаете, мы действительно не хотим пытаться изменить эти ценности. И поэтому мы имеем дельта-термины только для слоев 2, 3 и для этого примера.

6:30

Распространение названия происходит из того факта, что мы начинаем с вычисления дельта-термина для выходного слоя, а затем возвращаем слой и вычисляем дельта-термины для третьего скрытого слоя, а затем возвращаем еще один шаг для вычисления дельта 2 и так далее , мы как бы обратно распространяем ошибки с выходного уровня на уровень 3 на их, чтобы, следовательно, обратно обратно осложнение.

6:51

Наконец, вывод удивительно сложный, неожиданно задействованный, но если вы просто выполните несколько этапов шага вычисления, можно откровенно доказать вирусность каким-то сложным математическим доказательством. Можно доказать, что если вы игнорируете авторизацию, то частичные производные условия, которые вы хотите

7:12

точно определяются активациями и этими дельта-терминами. Это игнорирует лямбду или, альтернативно, регуляризацию

7:23

долгосрочная лямбда будет равна 0. Мы позже исправим эту деталь о терминах регуляризации, но, выполняя обратное распространение и вычисляя эти дельта-термины, вы можете, вы знаете, довольно быстро вычислить эти частичные производные термины для всех ваших параметров. Так что это много деталей. Давайте возьмем все и соединим все вместе, чтобы поговорить о том, как реализовать обратное распространение

7:46

для вычисления производных по вашим параметрам.

7:49

И для случая, когда у нас есть большой набор тренировок, а не только учебный набор одного примера, вот что мы делаем. Предположим, у нас есть обучающий набор из m примеров, как показано здесь. Первое, что мы собираемся сделать, это собирать эти дельта-индексы i j. Итак, этот треугольный символ? Это на самом деле главная греческая алфавитная дельта. Символом, который у нас был на предыдущем слайде, была дельта нижнего регистра. Таким образом, треугольник - это дельта капитала. Мы собираемся установить равным нулю для всех значений l i j. В конце концов, эта капитальная дельта l i j будет использоваться

8:26

для вычисления частного производного члена, частная производная относится к тете l i j of theta.

8:39

Итак, как мы увидим через секунду, эти дельта будут использоваться в качестве аккумуляторов, которые будут медленно добавлять вещи, чтобы вычислить эти частные производные.

8:49

Затем мы пройдем через наш тренировочный набор. Итак, мы скажем, что для i равно 1 до m, поэтому для i-й итерации мы будем работать с примером обучения xi, yi.

9:00

Итак, первое, что мы собираемся сделать, это установить a1, который является активацией входного слоя, установить, что он равен xi, является входом для нашего учебного примера i, а затем мы будем выполнять прямое распространение для вычисления активация для второго уровня, третий уровень и т. д. до последнего уровня, слой капитала L. Далее мы будем использовать выходной ярлык yi из этого конкретного примера, который мы рассматриваем, чтобы вычислить коэффициент ошибки для delta L для выхода там. Итак, дельта L - это то, что выдает гипотеза минус, чем была целевая метка?

9:41

И тогда мы будем использовать алгоритм обратного распространения для вычисления дельта L минус 1, delta L минус 2 и т. Д. До дельта 2, и еще раз есть дельта 1, потому что мы не связываем с этим входной слой.

9:57

И, наконец, мы собираемся использовать эти дельта-члены капитала, чтобы накопить эти частичные производные термины, которые мы записали на предыдущей строке.

10:06

И, кстати, если вы посмотрите на это выражение, это также можно процитировать. Конкретно, если вы думаете о дельта ij как матрице, индексируемой индексом ij.

10:19

Тогда, если delta L является матрицей, мы можем переписать это как delta L, обновляется как delta L plus

10:27

нижний регистр delta L плюс один раз aL транспонировать. Таким образом, это векторная реализация, которая автоматически обновляет все значения i и j. Наконец, после выполнения тела из четырех циклов мы выходим за пределы четырех циклов и вычисляем следующее. Мы вычисляем капитал D следующим образом, и у нас есть два отдельных случая, когда j равно нулю, а j не равно нулю.

10:56

Случай j равняется нулю, соответствует члену смещения, поэтому, когда j равно нулю, поэтому нам не хватает дополнительного члена регуляризации.

11:05

Наконец, хотя формальное доказательство довольно сложно, вы можете показать, что, как только вы вычислили эти термины D, это точно частная производная функции стоимости по каждому из ваших периметров, и поэтому вы можете использовать их в любом градиенте спуск или в одном из дополнительных разрешений

11:25

алгоритмы.

11:28

Итак, это алгоритм обратного распространения и как вы вычисляете производные от вашей функции затрат для нейронной сети. Я знаю, что это похоже на то, что это было много деталей, и это было сделано много шагов. Но как в заданиях программирования выписываются, так и позже в этом видео, мы дадим вам краткое изложение этого, чтобы мы могли объединить все части алгоритма, чтобы вы точно знали, что вам нужно реализовать, если вы хотите реализовать обратно для вычисления производных функции стоимости вашей нейронной сети по этим параметрам.

Backpropagation intuition

0:00

В предыдущем видео мы говорили о алгоритме backpropagation. Для многих людей, впервые увидев это, их первое впечатление часто говорит о том, что это действительно сложный алгоритм, и есть все эти разные шаги, и я не уверен, как они подходят друг другу. И это своего рода черный ящик всех этих сложных шагов. В случае, если вы чувствуете обратное распродажу, на самом деле все в порядке. Backpropagation, возможно, к сожалению, является менее математически чистым или менее математически простым алгоритмом по сравнению с линейной регрессией или логистической регрессией. И я действительно использовал backpropagation, вы знаете, довольно успешно в течение многих лет. И даже сегодня я по-прежнему не чувствую, что у меня очень хорошее чувство только, что он делает, или интуиция о том, что делает обратная пропаганда. Если для тех из вас, кто выполняет упражнения по программированию, это по крайней мере механически проведет вас через различные шаги, как реализовать обратную опору. Таким образом, вы сможете заставить его работать на себя. И то, что я хочу сделать в этом видео, выглядит немного больше на механических этапах backpropagation и попытаться дать вам немного больше интуиции о том, что механические шаги, которые делает задний щит, чтобы надеяться убедить вас в этом, вы знаете, это по крайней мере разумный алгоритм.

1:13

В случае, если даже после этого видео в случае, если обратное распространение по-прежнему кажется очень черным ящиком и вроде как слишком много сложных шагов и немного волшебным для вас, на самом деле все в порядке. И хотя я много лет использовал back prop, иногда это сложный алгоритм для понимания, но, надеюсь, это видео поможет немного. Чтобы лучше понять обратное распространение, давайте еще раз взглянем на то, что делает прямое распространение. Вот нейронная сеть с двумя входными единицами, которая не учитывает блок смещения, и две скрытые единицы в этом слое, а также две скрытые единицы в следующем слое. И затем, наконец, один выходной блок. Опять же, эти цифры два, два, два, не считая эти единицы смещения сверху. Чтобы проиллюстрировать прямое распространение, я собираюсь сделать эту сеть немного по-другому.

2:08

И, в частности, я собираюсь провести эту нейро-сеть с узлами, нарисованными как эти очень толстые эллипсисы, чтобы я мог писать в них текст. При выполнении прямого распространения у нас может быть какой-то конкретный пример. Скажем, например, x i comma y i. И это будет тот x i, который мы будем вводить во входной слой. Таким образом, это может быть x i 2 и x i 2 - значения, которым мы устанавливаем входной слой. И когда мы переводим вперед на первый скрытый слой здесь, мы делаем вычисление z (2) 1 и z (2) 2. Таким образом, это взвешенная сумма входов входных единиц. И затем мы применяем сигмоид логистической функции, а функция активации сигмоида применяется к значению z. Вот значения активации. Таким образом, мы получаем (2) 1 и a (2) 2. И тогда мы снова распространяем снова, чтобы получить здесь z (3) 1. Применим сигмоид логистической функции, активирующую функцию к ней, чтобы получить (3) 1.

И аналогично, так, пока мы не получим z (4) 1. Примените функцию активации. Это дает нам (4) 1, что является конечным выходным значением нейронной сети.

3:24

Давайте сожжем эту стрелку, чтобы дать себе больше места. И если вы посмотрите на то, что действительно делают эти вычисления, сосредоточив внимание на этом скрытом устройстве, скажем так. Мы должны добавить этот вес. Показанный в пурпуре есть мой вес тета (2) 1 0, индексирование не имеет значения. И вот здесь, который я выделяю красным цветом, это тета (2) 1 1, и этот вес здесь, который я рисую голубым, - это theta (2) 1 2. Итак, как мы вычисляем это значение, z (3) 1 is, z (3) 1 равно этому весу этого пурпурного веса, умноженному на это значение. Итак, это theta (2) 10 x 1. И затем плюс этот красный вес умножает это значение, так что это theta (2) 11 раз (2) 1. И, наконец, этот голубой вес умножает это значение, что, следовательно, плюс theta (2) 12 раз (2) 1. И так это распространение вперед. И получается, что, как мы увидим позже в этом видео, то, что делает backpropagation, делает процесс, очень похожий на это. За исключением того, что вместо вычислений, идущих слева направо от этой сети, вычисления с момента их движения справа налево от сети. И используя очень похожие вычисления, как это. И я скажу в двух слайдах именно то, что я имею в виду. Чтобы лучше понять, что делает backpropagation, давайте посмотрим на функцию стоимости. Это просто функция стоимости, которую мы имели, когда у нас есть только один блок вывода. Если у нас есть несколько единиц вывода, мы просто получаем суммирование, которое вы знаете по выходным единицам, индексированным k. Если у вас есть только один блок вывода, это функция стоимости. И мы распространяем распространение и обратное распространение по одному примеру за раз. Поэтому давайте просто сосредоточимся на одном примере x (i) y (i) и сосредоточимся на случае наличия одного выходного блока. Итак, y (i) здесь - просто действительное число. И будем игнорировать регуляризацию, поэтому лямбда равна 0. И этот последний термин, этот термин регуляризации, уходит. Теперь, если вы посмотрите внутри суммирования, вы обнаружите, что термин затрат связан с примером обучения, то есть стоимостью, связанной с примером обучения x (i), y (i). Это будет дано этим выражением. Таким образом, расходы на проживание с экзаменом i записываются следующим образом. И что эта функция стоимости делает это, она играет роль, подобную квадратной стрелке. Таким образом, вместо того, чтобы смотреть на это сложное выражение, если вы хотите, вы можете думать, что стоимость i составляет примерно квадратную разницу между тем, что выводится нейронной сетью, в сравнении с фактической стоимостью. Как и в случае логистических репрессий, мы фактически предпочитаем использовать немного более сложную функцию затрат, используя журнал. Но для целей интуиции, не стесняйтесь думать о функции стоимости как о том, как функция квадратов ошибок. И поэтому эта стоимость (i) измеряет, насколько хорошо работает сеть при правильном предсказывающем примере i. Насколько близок вывод фактической наблюдаемой метки y (i)? Теперь давайте посмотрим, что делает backpropagation. Одна полезная интуиция заключается в том, что backpropagation вычисляет эти дельта-надстрочные l индексы j. И мы можем думать об этом как о ошибке котировки значения активации, которое мы получили для единицы j в слое, в 1-м слое.

7:07

Более формально, для, и это возможно только для тех из вас, кто знаком с исчислением. Более формально то, что на самом деле являются дельта-терминами, - это частичная производная по z,

l, j, то есть эта взвешенная сумма входов, которые смешивали эти z-члены. Частичные производные по отношению к этим вещам функции стоимости.

7:27

Так конкретно, функция стоимости является функцией метки y и значения, это h из нейронной сети выходного значения x. И если бы мы могли зайти в нейронную сеть и немного изменить эти значения z l j, это повлияет на эти значения, которые выводит нейронная сеть. И это приведет к изменению функции затрат. И снова действительно, это только для тех из вас, кто является экспертом в Исчислении. Если вам комфортно с частными производными, то каковы эти дельта-термины, они оказываются частичной производной функции стоимости, в отношении этих промежуточных терминов, которые были сбиты с толку.

8:06

И поэтому они являются мерой того, насколько мы хотели бы изменить вес нейронной сети, чтобы повлиять на эти промежуточные значения вычисления. Чтобы повлиять на окончательный вывод нейронной сети h (x) и, следовательно, повлиять на общую стоимость. В случае, если это потеряло часть этой частичной производной интуиции, в случае, если это не имеет смысла. Не беспокойтесь об этом, мы можем обойтись без разговоров о частных производных. Но давайте рассмотрим более подробно о том, что делает backpropagation. Для выходного слоя первый дельта-треугольник, дельта (4) 1, как y (i), если мы делаем прямое распространение и обратное распространение в этом примере обучения i. Это означает, что y (i) минус a (4) 1. Так что это действительно ошибка, не так ли? Это разница между фактическим значением y минус того, что было предсказано, и поэтому мы собираемся вычислить дельта (4) 1 так. Затем мы сделаем это, распространим эти значения назад. Я объясню это через секунду и в конечном итоге вычислю дельта-термины для предыдущего слоя. Мы закончим дельта (3) 1. Дельта (3) 2. И тогда мы собираемся распространить это дальше назад, и закончим вычисление дельта (2) 1 и дельта (2) 2. Теперь расчет backpropagation очень похож на запуск алгоритма прямого распространения, но делает его обратно. Итак, вот что я имею в виду. Давайте посмотрим, как мы закончим с этим значением delta (2) 2. Итак, мы имеем дельта (2) 2. И, подобно распространению вперед, позвольте мне отметить пару весов. Итак, этот вес, который я собираюсь рисовать синим. Скажем, что вес - это тэта (2) 1 2, и это здесь, когда мы выделяем это красным цветом. Это будет, скажем, тета (2) из 2 2. Итак, если мы посмотрим, как вычисляется дельта (2) 2, как это вычисляется с помощью этой заметки. Оказывается, то, что мы собираемся делать, будет принимать это значение и умножать его на этот вес и добавлять его к этому значению, умноженному на этот вес. Таким образом, это действительно взвешенная сумма этих значений дельты, взвешенная по соответствующей силе края. Итак, полностью, позвольте мне заполнить это, эта дельта (2) 2 будет равна, Theta (2) 1 2 - это пурпурные точки времени delta (3) 1. Плюс, и то, что у меня было в красном, это тета (2) 2 раза дельта (3) 2. Таким образом, это действительно буквально эта красная волна, умножая это значение, плюс этот пурпурный вес, умноженный на это значение. И вот как мы заканчиваем это значение дельты. И как еще один пример, давайте посмотрим на это значение. Как мы получаем эту ценность? Ну, это аналогичный процесс. Если этот вес, который я собираюсь выделить зеленым цветом, если этот вес равен, скажем, дельта (3) 1 2. Тогда мы имеем, что дельта (3) 2 будет равна этой зеленой массе, тета (3) 12 раз дельта (4) 1. И кстати, до сих пор я писал значения дельта только для скрытых единиц, но исключая единицы смещения. В

зависимости от того, как вы определяете алгоритм backpropagation, или в зависимости от того, как вы его реализуете, вы знаете, что в конечном итоге вы можете реализовать то, что вычисляет значения дельта для этих единиц смещения. Единицы смещения всегда выводят значение плюс один, и они являются тем, чем они являются, и нет никакого способа изменить значение. И так, в зависимости от вашей реализации обратной поддержки, я обычно ее реализую. Я в конечном итоге вычисляю эти значения дельта, но мы просто отбрасываем их, мы их не используем. Потому что они не входят в состав вычислений, необходимых для вычисления производной. Поэтому, надеюсь, это дает вам немного лучшую интуицию о том, что делает пропозиция. В случае всего этого по-прежнему кажется своего рода волшебным, вроде черного ящика, в более позднем видео, при объединении видео, я постараюсь получить немного больше интуиции о том, что делает backpropagation. Но, к сожалению, это сложный алгоритм, который пытается визуализировать и понять, что он действительно делает. Но, к счастью, я был, я думаю, многие люди используют очень успешно в течение многих лет. И если вы реализуете алгоритм, вы можете иметь очень эффективный алгоритм обучения. Несмотря на то, что внутренняя работа именно того, как это работает, может быть сложнее визуализировать.

Implementation note unrolling parameters

0:00

В предыдущем видео мы говорили о том, как использовать обратное распространение

0:03

для вычисления производных функции затрат. В этом видео я хочу быстро рассказать вам об одной детали реализации разворачивания ваших параметров из матриц в векторы, которая нам нужна для того, чтобы использовать расширенные процедуры оптимизации.

0:20

В частности, предположим, что вы реализовали функцию cost, которая принимает входные данные, параметры theta, возвращает функцию cost и возвращает производные.

0:30

Затем вы можете передать это в расширенный алгоритм авторизации с помощью fminunc, и fminunc, кстати, не единственный. Существуют и другие усовершенствованные алгоритмы авторизации.

0:39

Но то, что все они делают, - это взять эти входные данные демонстративно функцию стоимости и некоторое начальное значение тета.

0:47

И оба, и эти подпрограммы предполагают, что тета и начальное значение тета, что это векторы параметров, возможно, Rn или Rn плюс 1. Но это векторы, и он также предполагает, что, вы знаете, ваша функция стоимости вернет в качестве второго возвращаемого значения этот градиент, который также является Rn и Rn плюс 1. Так что тоже вектор. Это хорошо работало, когда мы были с помощью логистической прогрессии, но теперь, когда мы с помощью нейронной

сети наши параметры уже не векторы, а вместо них эти матрицы, где для полноценной нейронной сети мы бы Параметр матрицы тета-1, тета-2, тета-3, что мы могли бы представлять в октаву, так как эти матрицы тета-1, тета-2, тета-3. И точно так же эти градиентные члены, которые должны были вернуться. Ну, в предыдущем видео мы показали, как вычислить эти градиентные матрицы, которые были капиталом D1, капиталом D2, капиталом D3, которые мы могли бы представить октаву в виде матриц D1, D2, D3.

1:48

В этом видео я хочу быстро рассказать вам об идее, как взять эти матрицы и развернуть их в векторы. Так что они в конечном итоге в формат, пригодный для передачи в качестве тета для выхода на градиент есть.

2:03

Конкретно, допустим, у нас есть нейронная сеть с одним входным слоем с десяти единиц, скрытый слой с десять единиц и один выходной слой с одним блоком, так что C1-это количество единиц в один слой и S2-число единиц в два слоя, и S3-число единиц в три слоя. В этом случае размерность ваших матриц тета и D будут заданы этими выражениями. Например, тета-один идет к матрице 10 на 11 и так далее.

2:34

Таким образом, если вы хотите преобразовать между этими матрицами. векторные иллюстрации. Что вы можете сделать, это принять ваши тета-1, тета-2, тета-3, и написать этот кусок кода, и это займет все элементы трех матриц тета и принять все элементы тета одна, все элементы тета-2, все элементы тета-3, и раскатать их и положить все элементы в большой длинный вектор.

2:58

Который thetaVec и аналогично

3:00

вторая команда возьмет все ваши матрицы D и развернет их в большой длинный вектор и назовет их DVec. И, наконец, если вы хотите вернуться от векторных представлений к матричным представлениям.

3:14

Что вы делаете, чтобы вернуться к тета один сказать, это взять thetaVec и вытащить первые 110 элементов. Таким образом, тета 1 имеет 110 элементов, потому что это матрица 10 на 11, так что вытаскивает первые 110 элементов, а затем вы можете использовать команду reshape, чтобы изменить их обратно в тета 1. И точно так же, чтобы вернуть тета 2, Вы вытаскиваете следующие 110 элементов и изменяете его. А для тета 3 Вы вытаскиваете последние одиннадцать элементов и запускаете reshape, чтобы вернуть тета 3.

3:48

Вот быстрая демонстрация октавы этого процесса. Для этого примера будем ставить тета 1 равной единицам 10 на 11, так что это матрица всех единиц. И просто, чтобы это было легче увидеть, давайте установим, что это 2 раза, 10 на 11, и давайте установим, что тета 3 равна 3 раза 1 из 1 на

11. Итак, это 3 отдельные матрицы: тета 1, тета 2, тета 3. Мы хотим поместить все это как вектор. ThetaVec равна тета-1, тета-2

4:28

тэта 3. Правильно, это двоеточие посередине и так

4:35

и теперь thetavec будет очень длинным вектором. Это 231 элемент.

4:42

Если я покажу его, то обнаружу, что это очень длинный вектор со всеми элементами первой матрицы, всеми элементами второй матрицы, затем всеми элементами третьей матрицы.

4:53

И если я хочу вернуть свои исходные матрицы, я могу изменить форму тетавека.

5:01

Давайте вытащим первые 110 элементов и изменим их форму на матрицу 10 на 11.

5:06

Это возвращает мне тета 1. И если я тогда вытащу следующие 110 элементов. Это индексы с 111 по 220. Я возвращаю все свои 2.

5:18

И если я пойду

5:20

от 221 до последнего элемента, который является элементом 231, и изменить форму 1 на 11, я возвращаюсь тета 3.

5:30

Чтобы сделать этот процесс действительно конкретным, вот как мы используем идею развертывания для реализации нашего алгоритма обучения.

5:38

Допустим, у вас есть некоторое начальное значение параметров theta 1, theta 2, theta 3. То, что мы собираемся сделать, это взять их и развернуть их в длинный вектор, который мы назовем начальным тета, чтобы передать fminunc в качестве начальной настройки параметров тета.

5:56

Другое дело, что нам нужно сделать, это реализовать функцию стоимости.

5:59

Вот моя реализация функции стоимости.

6:02

Стоимость функции будет давать нам ввод, thetaVec, который будет все мои параметры векторов, что в форме, которая была раскатали в векторе.

6:11

Итак, первое, что я собираюсь сделать, я собираюсь использовать thetaVec и я собираюсь воспользоваться изменить функции. Поэтому я вытащу элементы из thetaVec и использую reshape, чтобы вернуть мои исходные матрицы параметров, theta 1, theta 2, theta 3. Итак, это будут матрицы, которые я собираюсь получить. Таким образом, это дает мне более удобную форму, в которой можно использовать эти матрицы, чтобы я мог запускать прямое распространение и обратное распространение для вычисления моих производных и вычисления моей функции стоимости j теты.

6:39

И, наконец, я могу взять свои производные и развернуть их, чтобы сохранить элементы в том же порядке, что и при развертывании тет. Но я собираюсь развернуть D1, D2, D3, чтобы получить gradientVec, который теперь может вернуть моя функция стоимости. Он может возвращать вектор этих производных.

6:59

Итак, надеюсь, теперь у вас есть хорошее представление о том, как конвертировать назад и вперед между матричным представлением параметров по сравнению с векторным представлением параметров.

7:09

Преимущество матричного представления состоит в том, что, когда ваши параметры хранятся в виде матриц, это более удобно, когда вы делаете прямое и обратное распространение, и легче, когда ваши параметры хранятся в виде матриц, чтобы воспользоваться преимуществами векторизованных реализаций.

7:26

В то время как в отличие от векторного представления, когда у вас есть как thetaVec или DVec является то, что, когда вы используете передовые алгоритмы оптимизации. Эти алгоритмы, как правило, предполагают, что все ваши параметры развернуты в большой длинный вектор. И так с тем, что мы только что пережили, надеюсь, теперь вы можете быстро конвертировать между ними по мере необходимости.

Gradient checking

0:00

В прошлом видео мы говорили о том, как сделать прямого распространения и обратного распространения нейронной сети для вычисления производных. Но Back prop как алгоритм имеет много деталей и может быть немного сложнее реализовать. И одно неудачное свойство заключается в том, что есть много способов иметь тонкие ошибки в задней опоре. Так что, если вы запустите его с градиентным спуском или другим оптимизационным алгоритмом, это может выглядеть так, как будто он работает. И ваша функция затрат J от тета может в конечном итоге

уменьшается на каждой итерации градиентного спуска. Но это может оказаться правдой, даже если в вашей реализации Back prop может быть ошибка. Таким образом, похоже, что J из тета уменьшается, но вы можете просто закончить с нейронной сетью, которая имеет более высокий уровень ошибок, чем вы бы с реализацией без ошибок. И вы можете просто не знать, что была эта тонкая ошибка, которая давала вам худшую производительность. Итак, что мы можем с этим поделать? Есть идея, называемая градиентной проверкой, которая устраняет почти все эти проблемы. Итак, сегодня каждый раз, когда я реализую обратное распространение или подобный градиент [невнятно] в нейронной сети или любой другой достаточно сложной модели, я всегда реализую проверку градиента. И если вы это сделаете, это поможет Вам убедиться и получить высокую уверенность в том, что ваша реализация четырех опор и задней опоры или что-то на 100% правильно. И из того, что я видел, это в значительной степени устраняет все проблемы, связанные с какой-то ошибочной реализацией в качестве задней опоры. И в предыдущих видео я просил вас поверить в то, что формулы, которые я дал для вычисления дельт и vs и так далее, я попросил вас поверить, что они действительно вычисляют градиенты функции стоимости. Но как только вы реализуете численную проверку градиента, которая является темой этого видео, вы сможете полностью проверить для себя, что код, который Вы пишете, действительно вычисляет производную от перекрестной функции J.

1:52

Так вот идея, рассмотрим следующий пример. Предположим, что у меня есть функция J тета, и у меня есть некоторое значение тета, и для этого примера предположим, что тета - это просто действительное число. Предположим, что я хочу оценить производную этой функции в данный момент, и поэтому производная равна наклону касательной.

2:14

Вот как я собираюсь численно аппроксимировать производную, или, скорее, вот процедура численного аппроксимирования производной. Я собираюсь вычислить тета плюс Эпсилон, так что теперь мы переместим его вправо. Я вычислю тета минус Эпсилон и посмотрю на эти две точки, и соединю их прямой линией

2:43

И я соединю эти две точки прямой линией, и я использую наклон этой красной линии в качестве приближения к производной. То есть, истинная производная-это наклон синей линии вон там. Итак, вы знаете, кажется, что это было бы довольно хорошим приближением.

2:58

Математически, наклон этой красной линии-это вертикальная высота, деленная на эту горизонтальную ширину. Таким образом, эта точка сверху является J из (тета плюс Эпсилон). Эта точка здесь J (тета минус Эпсилон), так что эта вертикальная разница J (тета плюс Эпсилон) минус J тета минус Эпсилон и это горизонтальное расстояние составляет всего 2 Эпсилон.

3:23

Так что мое приближение будет что производная отношении тета-J от тета на эту величину тэта, что это примерно J от тета Эпсилон плюс минус J от тета-минус Эпсилон за 2 Эпсилон.

3:42

Обычно, я использую довольно небольшое значение для эпсилона, ожидаю, что Эпсилон будет возможно на заказе 10 к минус 4. Обычно существует большой диапазон различных значений для epsilon, которые работают нормально. На самом деле, если позволить эпсилону стать очень маленьким, то математически этот термин здесь, фактически математически, станет производным. Это становится точно наклоном функции в этой точке. Просто мы не хотим использовать Эпсилон, который слишком мал, потому что тогда вы можете столкнуться с численными проблемами. Поэтому я обычно использую Эпсилон от десяти до минус четырех. И кстати, некоторые из вас, возможно, видели альтернативная формула для встречи производной, эта формула.

4:21

Эта справа называется односторонней разницей, тогда как формула слева называется двусторонней разницей. Двусторонняя разница дает нам немного более точную оценку, поэтому я обычно использую это, а не эту одностороннюю оценку разницы.

4:35

Итак, конкретно, когда вы реализуете октаву, вы реализовали следующее, вы реализуете вызов для вычисления gradApprox, который будет нашей производной аппроксимации, как только здесь эта формула, J тета плюс Эпсилон минус J тета минус Эпсилон делится на 2 раза Эпсилон. И это даст вам численную оценку градиента в этой точке. И в этом примере кажется, что это довольно хорошая оценка.

5:01

Теперь на предыдущем слайде, мы рассмотрели случай, когда тета был свернутый номер. Теперь рассмотрим более общий случай, когда тета-параметр вектор, так скажем тета р-н. И это может быть раскатал версия параметры нейронной сети. Таким образом, тета-вектор, который имеет n элементов, тета 1 до тета n. Затем мы можем использовать аналогичную идею для аппроксимации всех членов частных производных. Конкретно частичная производная функции стоимости по первому параметру, тета-единица, которая может быть получена путем взятия J и увеличения тета-единицы. Итак, у Вас есть J тета один плюс Эпсилон и так далее. Минус J этого тета один минус Эпсилон и делим его на два эпсилона. Частичная производная по отношению ко второму параметру тета два, снова эта вещь, за исключением того, что вы возьмете J здесь вы увеличиваете тета два на Эпсилон, и здесь вы уменьшаете тета два на Эпсилон и так далее вплоть до производной. В отношении тета n даст вам увеличение и уменьшение тета и Эпсилон там.

6:09

Таким образом, эти уравнения дают вам способ численно аппроксимировать частичную производную от J по любому из ваших параметров тета i.

6:23

Полностью, то, что вы реализуете, поэтому следующее.

6:27

Мы реализуем следующее в октаве для численного вычисления производных. Скажем, для i = 1:n, где n-размерность нашего параметра вектора тета. Обычно я делаю это с развернутой версией параметра. Так что тета-это просто длинный список всех моих параметров в моей нейронной сети, скажем. Я thetaPlus = тэта, затем увеличить thetaPlus (я) элемент Эпсилон. И так это в основном thetaPlus равен тета за исключением thetaPlus (i), который теперь увеличивается на Эпсилон. Эпсилон, так тета плюс равен, пишем тета 1, тета 2 и так далее. Затем тета я Эпсилон добавил к нему, а затем мы спустимся к тета N. так это то, что тета плюс. И подобные эти две линии установить тета минус что-то подобное, за исключением того, что это вместо тета я плюс Эпсилон, это теперь становится тета я минус Эпсилон.

7:20

И тогда, наконец, Вы реализуете этот gradApprox (i), и это даст вам ваше приближение к частичной производной уважения тета i из J тета.

7:35

И то, как мы используем это в нашей реализации нейронной сети, заключается в том, что мы реализуем эти четыре цикла для вычисления верхней частичной производной функции стоимости по каждому параметру в этой сети, и затем мы можем взять градиент, который мы получили от backprop. Таким образом, DVec был производным, которое мы получили от backprop. Итак, backprop, backpropogation, был относительно эффективным способом вычисления производной или частичной производной. Функции стоимости по всем нашим параметрам. А что я обычно делаю это тогда, возьми мою численно вычисленные производные, что это gradApprox, что у нас здесь. И убедитесь, что это равно или приблизительно равно небольшим значениям числового округления, что это довольно близко. Так DVec, что я получил от сеть с обратным распространением ошибки. И если эти два способа вычисления производной дают мне один и тот же ответ или дают мне любые похожие ответы, вплоть до нескольких десятичных знаков, то я гораздо более уверен, что моя реализация backprop верна. И когда я подключаю эти векторы DVec к gradient assent или какому-то продвинутому алгоритму оптимизации, я могу быть гораздо более уверенным, что я правильно вычисляю производные, и поэтому, надеюсь, мой код будет работать правильно и хорошо оптимизирует J из тета.

8:57

Наконец, я хочу собрать все вместе и рассказать вам, как реализовать эту численную проверку градиента. Вот что я обычно делаю. Первое, что я делаю, это реализация обратного распространения для вычисления DVec. Итак, есть процедура, о которой мы говорили в предыдущем видео, чтобы вычислить DVec, который может быть нашей развернутой версией этих матриц. Итак, то, что я делаю, это реализовать численную проверку градиента для вычисления gradApprox. Вот что я описал ранее в этом видео и на предыдущем слайде.

9:24

Затем следует убедиться, что DVec и gradApprox дают одинаковые значения, вы знаете, скажем, до нескольких десятичных знаков.

9:32

И, наконец, и это важный шаг, прежде чем вы начнете использовать свой код для обучения, для серьезного обучения вашей сети, важно отключить проверку градиента и больше не вычислять эту вещь gradApprox, используя числовые производные формулы, о которых мы говорили ранее в этом видео.

9:50

Причина этого в том, что код проверки градиента числового кода, о котором мы говорили в этом видео, очень дорог с вычислительной точки зрения, это очень медленный способ попытаться аппроксимировать производную. В то время как, напротив, алгоритм обратного распространения, о котором мы говорили ранее, это то, о чем мы говорили ранее для вычислений. Знаешь, D1, D2, D3 для Dvec. Backprop гораздо более вычислительно эффективный способ вычисления производных.

10:17.

Поэтому, как только вы убедились, что ваша реализация back proposal верна, вы должны отключить проверку градиента и просто прекратить использовать это. Поэтому, чтобы повторить, вы должны обязательно отключить свой код проверки градиента перед запуском вашего алгоритма для многих итераций градиентного спуска или для многих итераций расширенных алгоритмов оптимизации, чтобы обучить свой классификатор. Конкретно, если бы вы запускали численную проверку градиента на каждой итерации градиентного спуска. Или, если бы вы были во внутреннем цикле вашей costFunction, то ваш код был бы очень медленным. Поскольку численный код проверки градиента намного медленнее, чем алгоритм обратного распространения, чем метод обратного распространения, где, вы помните, мы вычисляли дельту (4), дельту(3), дельту(2) и так далее. Это был алгоритм обратного распространения. Это гораздо более быстрый способ вычисления производных, чем проверка градиента. Поэтому, когда вы будете готовы, как только вы проверите правильность реализации обратной пропаганды, убедитесь, что вы выключили или отключили код проверки градиента во время тренировки вашего алгоритма, иначе ваш код может работать очень медленно.

11:20.

Так вот как вы берете сортов numericaly, и вот как вы можете проверить это реализация обратной пропаганды является правильным. Всякий раз,когда я применяю обратную пропаганду или аналогичный алгоритм распознавания градиента для сложного режима, I я всегда использую проверку градиента, и это действительно помогает мне убедиться, что мой код правильный.

Random initialization

0:00

В предыдущем видео мы собрали почти все части, необходимые для реализации и обучения в вашей сети. Есть еще одна идея, которой я хочу поделиться с вами - идея случайной инициализации.

0:13

Когда вы запускаете алгоритм градиентного спуска, или также расширенные алгоритмы оптимизации, нам нужно выбрать некоторое начальное значение для параметров тета. Таким

образом, для расширенного алгоритма оптимизации предполагается, что вы передадите ему некоторое начальное значение для параметров theta.

0:29

Теперь рассмотрим градиентный спуск. Для этого нам также нужно будет инициализировать theta к чему-то, а затем мы можем медленно предпринять шаги, чтобы спуститься вниз, используя градиентный спуск. Чтобы спуститься вниз, чтобы минимизировать функцию j тета. Так что же мы можем установить начальное значение тета? Можно ли задать начальное значение тета вектору всех нулей? В то время как это работало хорошо, когда мы использовали логистическую регрессию, инициализация всех ваших параметров до нуля фактически не работает, когда вы торгуете в своей собственной сети. Рассмотрим торговлю по нейронной сети follow, и предположим, что мы инициализируем все параметры сети до 0. И если вы это сделаете, тогда это будет означать, что при инициализации, этот синий вес, окрашенный в синий, будет равен этому весу, так что они оба равны 0. И этот вес, который я раскрашиваю в красный, равен этому весу, окрашенному в красный, а также этот вес, который я раскрашиваю в зеленый, будет равен значению этого веса. Это означает, что оба скрытых блока, A1 и A2, будут вычислять одну и ту же функцию входных данных. И, таким образом, вы в конечном итоге для каждого из ваших обучающих примеров, вы получаете 2 1 равно 2 2.

1:46

И более того, потому что я не собираюсь показывать это слишком подробно, но поскольку эти исходящие веса одинаковы, Вы также можете показать, что значения дельты также будут одинаковыми. Так что конкретно вы в конечном итоге с 1 Дельта 1, Дельта 2, Дельта равна 1 2 2, и если вы работаете через карту далее, что вы можете показать, что частные производные относительно параметров будут удовлетворять следующим, что Частная производная функции затрат с уважением к выходу из производных относительно этих двух синих волн в сети. Вы обнаружите, что эти две частные производные будут равны друг другу.

2:31

И это означает, что даже после одного большего обновления спуска, вы обновите, скажем, этот первый синий курс учился в разы больше этого, и вы обновите второй синий курс с некоторым темпом обучения в разы больше этого. И это означает, что даже после того, как один из них создал обновление спуска, эти две синие скорости, эти два параметра синего цвета будут в конечном итоге одинаковыми друг с другом. Поэтому есть некоторое ненулевое значение, но это значение будет равно этому значению. И точно так же, даже после одного обновления градиентного спуска, это значение будет равно этому значению. Там все еще будут ненулевые значения, только два красных значения равны друг другу. И точно так же два зеленых пути. Ну, они оба изменят значения, но в итоге оба будут иметь одинаковое значение. Таким образом, после каждого обновления параметры, соответствующие входам, входящим в каждый из двух скрытых блоков, идентичны. Это просто говорит о том, что два зеленых веса все еще одинаковы, два красных веса все еще одинаковы, два синих веса все еще одинаковы, и это означает, что даже после одной итерации, скажем, градиентного спуска и спуска. Вы обнаружите, что ваши две головные единицы все еще вычисляют точно такие же функции входов. У вас все еще есть a1(2) = a2(2). Итак, вы вернулись к этому делу. И по мере того, как вы продолжаете спускаться, голубые

волны, две синие волны, будут оставаться такими же, как и друг с другом. Две красные волны останутся такими же, как друг с другом, и две зеленые волны останутся такими же, как друг с другом.

3:56

Это значит, что ваша нейронная сеть действительно может вычислять очень интересные функции, верно? Представьте, что у вас не только два скрытых единиц, но представьте, что у вас много скрытых узлов. Тогда это говорит о том, что все ваши головные подразделения вычисляют одну и ту же функцию. Все ваши скрытые блоки вычисляют точно такую же функцию ввода. И это очень избыточное представление, потому что вы находите единицу логистической прогрессии. Он действительно должен видеть только одну функцию, потому что все они одинаковы. И это мешает вам и вашей сети делать что-то интересное.

4:31

Чтобы обойти эту проблему, мы инициализируем параметры нейронной сети случайным образом.

4:41

Конкретно, проблема была замечена на предыдущем слайде, это то, что называется проблемой симметричных путей, то есть пути одинаковы. Таким образом, эта случайная инициализация - это то, как мы выполняем нарушение симметрии. Итак, мы инициализируем каждое значение тета случайным числом между минус Эпсилон и Эпсилон. Это обозначение числа b между минус Эпсилон и плюс Эпсилон. Таким образом, мой вес для моих параметров будет случайным образом инициализирован между минус Эпсилон и плюс Эпсилон. То, как я пишу код, чтобы сделать это в октаве, я сказал, что Theta1 должен быть равен этому. Итак, этот Ранд 10 на 11, вот как вы вычисляете случайную матрицу 10 на 11. Все значения между 0 и 1, так как это сырые цифры, что принимать любые непрерывные значения между 0 и 1. Если взять число от нуля до единицы, умножить его на два раза INIT_EPSILON, затем на минус INIT_EPSILON, то получится число между минус epsilon и плюс epsilon.

5:45

И это приводит нас к тому, что этот Эпсилон не имеет ничего общего с эпсилоном, который мы использовали при проверке градиента. Поэтому при численной проверке градиента мы добавляли некоторые значения эпсилона и теты. Это ваша несвязанная ценность эпсилона. Мы просто хотели отметить init epsilon, чтобы отличить его от значения epsilon, которое мы использовали при проверке градиента. И точно так же, если вы хотите инициализировать theta2 в случайную матрицу 1 на 11, вы можете сделать это, используя этот фрагмент кода здесь.

6:16

Итак, подводя итог, чтобы создать нейронную сеть, вы должны случайным образом инициализировать волны до небольших значений, близких к нулю, между-epsilon и +epsilon say. А затем реализуйте обратное распространение, отлично справляйтесь с проверкой и используйте либо отличные алгоритмы спуска, либо расширенные алгоритмы оптимизации 1b, чтобы попытаться минимизировать j(тета) в зависимости от параметров тета, начиная с только случайно выбранного начального значения для параметров. И, делая нарушение симметрии, которое

является этим процессом, мы надеемся, что большой градиентный спуск или передовые алгоритмы оптимизации смогут найти хорошее значение тета.

Putting it together

0:00

Итак, мы взяли нас много видео, чтобы пройти алгоритм обучения нейронной сети.

0:05

В этом видео, что бы я хотел сделать, это попытаться поместите все части вместе, чтобы дать общее резюме или более широкий вид все части подходят друг другу и общего процесса для реализации алгоритма обучения нейронной сети.

0:21

При обучении нейронной сети первое, что вам нужно сделать выбирает некоторую сетевую архитектуру и по архитектуре я просто средняя схема соединения между нейронами. Итак, вы знаете, мы могли бы выбрать между скажем, нейронной сетью с тремя входными блоками и пять скрытых единиц и четыре выходных блока против одного из 3, 5 скрытых, 5 скрытый, 4 выхода и здесь 3, 5, 5, 5 единиц в каждом из трех скрытых слоев и четырех открытые блоки, и поэтому эти выбор количества скрытых единиц в каждом слое и сколько скрытых слоев, те являются выбором архитектуры. Итак, как вы делаете эти выборы?

0:59

Ну, во-первых, число входных единиц хорошо, что довольно хорошо определено. И как только вы решите исправить набор признаков x количество единиц ввода будет, как вы знаете, размер ваших возможностей x (i) будет определяться этим. И если вы делаете мультиклассирование классификации вывод этого будет определяется числом классов в вашей проблеме классификации. И просто напоминание, если у вас есть многоклассовая классификация, где у принимает значения say между

1:30

1 и 10, так что у вас есть десять возможных классов.

1:34

Тогда не забудьте правильно, выход у, поскольку они были векторами. Итак, вместо пункта 1 вы перекодировать его как вектор например, или для второй класс вы перекодируете его как вектор. Поэтому, если один из этих яблоки берут на себя пятый класс, вы знаете, у равно 5, тогда что вы показываете своим нервным сеть на самом деле не является ценностью от у равно 5, вместо этого здесь на верхнем слое, который есть десять единиц продукции, вы вместо этого будет вектор, который вы знаете

2:07

с одним в пятом положение и кучу нулей здесь. Таким образом, выбор номера блоков ввода и количества блоков вывода может быть несколько достаточно прост.

2:18

А что касается числа скрытых единиц и количество скрытых слоев, разумным дефолтом является использовать один скрытый слой и поэтому этот тип показана нейронная сеть слева Наиболее вероятно, что один скрытый слой.

2:34

Или если вы используете больше чем один скрытый слой, снова разумным дефолтом будет имеют такое же количество скрытых единиц в каждом отдельном слое. Итак, у нас есть два скрытые слои и каждый этих скрытых слоев имеют такое же число пять скрытых единиц, и здесь мы, знаете ли, три скрытых слоя и каждый из них имеет одинаковые число, то есть пять скрытых единиц.

2:57

Вместо этого сетевой архитектуры слева будет идеальным разумным дефолтом.

3:04

А что касается числа скрытых единиц - обычно, более скрытые единицы лучше; это просто, если у вас есть много скрытых единиц, это может стать более дорогостоящим, но очень часто, наличие более скрытых единиц - это хорошо.

3:17

И, как правило, количество скрытых единицы в каждом слое будут, возможно, сравнимый с размером x, сопоставимые с количество функций, или быть любым, где от такого же числа скрытых единиц входных функций для может быть, так, что три или четыре раза. Таким образом, количество скрытых единиц сравнимо. Вы знаете, несколько раз, или некоторые, что больше, чем число частотных функций ввода полезная вещь, чтобы сделать так, надеюсь, это даст вам один разумный набор вариантов по умолчанию для нейронной архитектуры и если вы будете следовать этим рекомендациям, вы вероятно, получит что-то, что работает ну, но в более поздний набор видеороликов, где Я буду говорить конкретно о советы по применению алгоритмов, я на самом деле скажите гораздо больше о том, как выбрать архитектуру нейронной сети. Или на самом деле много хочу скажите позже, чтобы сделать хороший выбор для количества скрытых единиц, количества скрытых слоев и т. д.

4:10

Далее, вот что мы необходимо реализовать, чтобы торговля нейронной сетью, есть на самом деле шесть шагов, которые я иметь; У меня есть четыре на этом слайд и еще два шага на следующем слайде. Первый шаг - установить нейронную сети и случайным образом инициализировать значения весов. И мы обычно инициализируем весов до малых значений вблизи нуля.

4:31

Затем мы реализуем прямое распространение так что мы можем ввести любая отличная нейронная сеть и вычислить h x, что выходной вектор значений y.

4:44

Затем мы также реализуем код для вычислите эту функцию стоимости j тэта.

4:49

И далее мы реализуем back-prop или обратное распространение

4:54

алгоритм, чтобы вычислить эти частичные производные термины, частичные производные j тэта по параметрам. Конкретно, чтобы реализовать заднюю опору. Обычно мы будем это делать с передним контуром над примерами обучения.

5:09

Некоторые из вас, возможно, слышали о продвинутый и, откровенно говоря, очень методы расширенной факторизации, в которых вы не имеют четыре петли над примеры m-обучения, что первый раз, когда вы выполняете back prop должны почти наверняка четыре цикл в вашем коде, где вы повторяете примеры, вы знаете, x1, y1, то так вы делаете вперед опору и задняя опора на первом пример, а затем в вторая итерация четырехтактный, вы распространяете вперед и обратное распространение во втором примере и т. д. Пока вы не закончите последний пример. Так что должно быть четыре цикла в вашей реализации обратной поддержки, по крайней мере, в первый раз, когда она была реализована. И тогда есть откровенно несколько сложных способов сделать это без четырехпетлевой, но Я определенно не рекомендую пытаясь сделать это гораздо больше сложную версию при первой попытке реализовать обратную опору.

5:59

Так что конкретно мы имеем четыре цикла по моим примерам m-training

6:03

и внутри четырехтактных мы собираюсь выполнить предпродажу и back prop, используя только этот пример.

6:09

И это означает, что мы возьмем x (i) и подать на мой входной слой, выполнить переднюю опору, выполнить заднюю опору

6:17

и это будет, если все эти активации и все эти дельта-термины для всех слоев всех моих единиц в нервном сеть затем все еще внутри этого четырехтактного, пусть я рисую фигурные скобки просто чтобы показать четыре петли, это в

6:34

октавный код, конечно, но это скорее последовательность Java код и четыре цикла охватывает все это. Мы собираемся вычислить эти дельта термины, которые являются формулой, которую мы дали ранее.

6:45

Кроме того, вы знаете, delta l plus один раз

6:48

a, l транспонировать код. И наконец, за пределами вычислив эти дельта термины, эти условия накопления, мы тогда будет кода, а затем позволяют вычислить эти частные производные термины. Правильно и эти частные производные условия должны приниматься учитывает также термин регуляризации лямбда. Итак, эти формулы были приведены в более раннем видео.

7:14

Итак, как вы это сделали? вы теперь, надеюсь, имеете код для вычислить эти частичные производные термины.

7:21

Следующий шаг пятый, что я do - это использовать градиент чтобы сравнить эти частичные которые были вычислены. Итак, я сравнивали версии, рассчитанные с использованием о братного распространения частные производные, рассчитанные с использованием численных

7:37

оценки с использованием численных оценок производных. Итак, я проверяю градиент, чтобы сделать что оба из них дают вам очень похожие значения.

7:45

Проведя проверку градиента, теперь это успокаивает нас, что наша реализация назад распространение является правильным и то очень важно, что мы отключим градиент, потому что градиент проверяющий код вычисляется очень медленно.

7:59

И, наконец, мы тогда использовать алгоритм оптимизации как градиентный спуск, или один из передовые методы оптимизации, такие как как LB GS, градиент воплощенные в fminunc или другие методы оптимизации. Мы используем их вместе с назад, так что назад пропаганда - это вещь который вычисляет эти частные производные для нас.

8:21

Итак, мы знаем, как вычислить функцию стоимости, мы знаем как вычислить частные производные, используя назад, поэтому мы может использовать один из этих методов оптимизации попытаться минимизировать j из тета в зависимости от параметров тета. И, кстати, для нейронных сетей, эта функция стоимости j тэта не выпуклый, или не является выпуклым, и поэтому это теоретически может быть восприимчивым к локальным минимумам, а в алгоритмы фактов, такие как градиентный спуск и методы предварительной оптимизации могут, в теории, застревают в местных

8:55

Оптима, но получается что на практике это обычно не огромная проблема и хотя мы не можем гарантировать что эти алгоритмы найдут глобальный оптимум, обычно такие алгоритмы, как градиентный спуск сделает очень хорошая работа, сводящая к минимуму функции стои мости j тета и получить очень хороший локальный минимум, даже если он не достигнет глобального оптимума. Наконец, градиентные спуски для нейронная сеть может показаться немного

волшебной. Итак, позвольте мне просто показать один больше фигуры, чтобы попытаться эта интуиция о том, что делает градиентный спуск для нейронной сети.

9:27

Это было похоже на что раньше я использовал объяснение градиентного спуска. Итак, у нас есть некоторая стоимость функции, и мы имеем ряд параметров в нашей нейронной сети. Правильно здесь я только что записал два значения параметра. В действительности, конечно, в нейронной сети, у нас с ними может быть множество параметров. Theta one, theta two - все это матрицы, не так ли? Таким образом, мы можем иметь очень большие размеры параметров, но из-за ограничения источника части, которые мы можем нарисовать. Я притворяюсь что у нас есть только два параметра в этой нейронной сети. Хотя очевидно, что на практике у нас намного больше.

9:59

Теперь эта функция стоимости j тета измеряет, насколько хорошо нейронная сеть соответствует данным обучения.

10:06

Итак, если вы примете точку как этот, здесь,

10:10

это точка, где j тета довольно низка, и это соответствует настройке параметров. Существует настройка параметров тета, где, как вы знаете, для большинства примеров обучения, выход

10:24

моей гипотезы, которая может быть довольно близким к y (i) и если это правда, это то, что заставляет мою функцию затрат быть довольно низкой.

10:32

Если, напротив, если бы вы были чтобы взять такое значение, точка, соответствующая этому, где для многих примеров обучения, выход моего нервного сеть далека от фактическое значение y (i) что наблюдалось в учебном наборе. Таким образом, на линии соответствуют тому, где гипотеза, где нейронные сеть выводит значения на учебном наборе, который вдали от y (i). Итак, это не хорошо подгонять тренировочный набор, тогда как такие моменты с низким значения функции стоимости соответствуют где j тета является низким и, следовательно, соответствует где происходит нейронная сеть подгонять мой тренировочный набор хорошо, потому что я имею в виду, что это то, что должно быть истинным для того, чтобы j тэта было малым.

11:15

Итак, какой градиентный спуск мы начнем с некоторого случайного начальная точка один там, и он будет постоянно идти вниз.

11:24

Итак, какое обратное распространение это вычисление направления градиента, и что градиентный спуск он делает небольшие шаги под гору пока, надеюсь, это не удастся, в этом случае, довольно хороший локальный оптимум.

11:37

Итак, когда вы реализуете назад градиент распространения и использования спуск или один из расширенные методы оптимизации, эта фотография объясняет, что делает алгоритм. Он пытается найти ценность параметров, где выходные значения в нейронном сеть тесно соответствует значения y (i) 's наблюдаемый в вашем учебном наборе. Итак, надеюсь, это даст вам лучшее понимание того, как много разных частей обучение нейронной сети сочетается.

12:07

В случае, даже после этого видео, в если вы все еще чувствуете себя там есть, как, много разных частей и не совсем ясно, что некоторые из них делают или как все из этих штук собрались вместе, на самом деле все в порядке.

12:18

Обучение нейронной сети и обратное распространение - сложный алгоритм.

12:23

И хотя я видел математика за спиной распространения в течение многих лет, и я использовал назад, я думаю, очень успешно, в течение многих лет, даже сегодня я все еще чувствую себя как я не всегда имеют отличную понять, что именно происходит в прошлом. И какой процесс оптимизации выглядит как минимизация j, если тета. Многое это намного сложнее алгоритм чувствовать, что у меня есть гораздо менее хорошая ручка точно, что это делает по сравнению, скажем, с линейной регрессией или логистической регрессией.

12:51

Которые были математически и концептуально гораздо более простые и более чистые алгоритмы.

12:56

Но так, если вы чувствуете Точно так же вы знаете, это на самом деле отлично хорошо, но если вы действительно реализуем обратное распространение, надеюсь вы обнаружите, что это является одним из самых мощных алгоритмов обучения, и если вы реализовать этот алгоритм, реализовать обратное распространение, реализовать одну из этих оптимизаций методов, вы обнаружите, что назад будет соответствовать очень сложным, мощным, нелинейным функции для ваших данных, и это один из самые эффективные алгоритмы обучения, которые мы имеем сегодня.

Autonomous driving

0:00

В этом видео я хотел бы показать вам забавный и исторически важный пример нейронных сетей, изучающих использование нейронной сети для автономного вождения. Это становится автомобилем, чтобы научиться водить себя.

0:14

Видео, которое я покажу минуту, было тем, что я получил от Дин Померло, который был коллегой, который работает в Университете Карнеги-Меллона на восточном побережье Соединенных Штатов. И в части видео вы видите такие визуализации. И я хочу рассказать, как выглядит визуализация перед началом видео.

0:32

Здесь внизу слева находится вид, видимый автомобилем того, что перед ним. И вот здесь вы видите дорогу, которая, может быть, немного идет влево, и затем немного поправимся.

0:44

И здесь наверху, эта первая горизонтальная полоса показывает направление, выбранное человеческим драйвером. И это место этой яркой белой полосы, которая показывает направление рулевого управления выбранный человеком-водителем, где вы знаете здесь далеко влево, соответствует рулевое управление левым, здесь соответствует рулевое управление вправо. И так это место, которое немного слева, немного слева от центра означает, что водитель-водитель в этом месте слегка поворачивался влево. И этот второй бот соответствует направлению рулевого управления, выбранному алгоритм обучения и снова местоположение такого рода белой полосы означает, что нейронная сеть была здесь выберите направление рулевого управления, которое немного влево. И на самом деле, прежде чем нейронная сеть начнет сначала наклоняться, вы видите, что сеть выводит серая полоса, как серый, как сплошная серая полоса в этом регионе и разновидность однородного серого пуха соответствует нейронной сети, которая была случайно инициализирована. И изначально не имея понятия, как управлять автомобилем. Или изначально не имея представления о том, в каком направлении управлять. И только после того, как он научился некоторое время, которое затем начнет выводиться, как сплошная белая полоса, только в небольшом часть области, соответствующая выбору конкретного направления рулевого управления. И это соответствует тому, когда нейронная сеть становится более уверенной при выборе полосы в одном конкретном месте, вместо того, чтобы выводить своего рода светло-серый пух, но вместо этого выводить белая полоса, которая более постоянно выбирает направление рулевого управления. >> ALVINN - это система искусственных нейронных сетей который учится управлять, наблюдая за приводом человека. ALVINN предназначен для управления NAVLAB 2, модифицированная армия Humvee, которая поставила датчики, компьютеры, и приводы для автономных навигационных экспериментов.

2:40

Первым шагом в настройке ALVINN является создание сети только здесь.

2:46

Во время обучения человек управляет автомобилем, а часы ALVINN.

2:55

Каждые две секунды ALVINN оцифровывает видеоизображение дороги вперед и записывает направление рулевого управления человека.

3:11

Это учебное изображение уменьшено с разрешением до 30 на 32 пикселя и в качестве входных данных для трехслойной сети ALVINN. Используя алгоритм обучения обратному распространению, ALVINN тренируется с выходом то же рулевое направление, что и человеческий драйвер для этого изображения.

3:33

Первоначально реакция управления сетью является случайной.

3:43

Примерно через две минуты обучения сеть учится точно подражать реакции рулевого управления человека.

4:02

Эта же процедура обучения повторяется для других типов дорог.

4:09

После того, как сети прошли обучение, оператор нажимает переключатель запуска и ALVINN начинает движение.

4:20

Двенадцать раз в секунду ALVINN оцифровывает изображение и подает его в свои нейронные сети.

4:33

Каждая сеть, работающая параллельно, создает направление рулевого управления, и мерой его «уверенности в своем» ответе.

4:48

Направление рулевого управления, из самой надежной сети, в этом сетевом обучении для одной дорожной полосы используется для управления транспортным средством.

5:07

Внезапно перед автомобилем появляется перекресток.

5:22

Когда транспортное средство приближается к пересечению, доверие сети одиночной полосы уменьшается.

5:37

Когда он пересекает перекресток, и впереди идет две полосы движения, уверенность сети двух полос движения возрастает.

5:51

Когда его «доверие возрастает, две полосы движения выбираются для управления. Безопасно направляя автомобиль в свою полосу на двухполосную дорогу.

6:05

>> Так что это было автономное вождение с использованием нейронной сети. Конечно, в последнее время более современные попытки сделать автономное вождение. В США и Европе мало проектов, и поэтому , которые дают более надежные управляющие контроллеры, чем это, но Я думаю, что это все еще довольно примечательно и довольно удивительно, как мгновенная нейронная сеть обученный с backpropagation, может научиться управлять автомобилем несколько хорошо.