Лекция 11
Графы вычислений и элементы глубинного обучения

Е. А. Соколов
ФКН ВШЭ

25 декабря 2016 г.

Многие методы, которые мы изучали ранее, тем или иным способом используют информацию о градиентах функции потерь. Так, в линейных моделях параметры настраиваются с помощью градиентного спуска, а в градиентном бустинге каждый базовый алгоритм аппроксимирует градиент функции потерь. При этом бустинг во многом является эвристическим алгоритмом, в котором используется большое количество допущений. К тому же даже градиентный бустинг имеет ограниченную силу, и не всегда может извлечь максимальную пользу из больших объёмов данных. Возникает вопрос — можно ли усилить линейные модели так, чтобы их выразительная способность существенно повысилась, и при этом сохранилась возможность обучать параметры напрямую с помощью градиентных методов оптимизации? Такие модели существуют и называются *нейронными сетями*. Это название сложилось исторически, поскольку раньше существовала уверенность в сходстве нейронов в головном мозге и архитектуры нейронных сетей — правда, позже выяснилось, что это не так. По этой причине правильнее использовать термин *графы вычислений* (computation graphs), и именно его мы будем стараться употреблять далее.

Введём для начала основные определения и обсудим общий подход к обучению графов вычислений, а затем обсудим ряд их важных особенностей.

1 Графы вычислений

Будем считать, что объекты принадлежат пространству $\mathbb{X} = \mathbb{R}^d$, а ответы — пространству $\mathbb{Y}$. Пусть задан некоторый объект $x \in \mathbb{X}$. Вычислим для него значение функции $f_1(x, w_1)$, где $w_1 \in \mathbb{R}^{m_1}$ — параметры, и обозначим результат вычисления этой функции через $v_1(x) \in \mathbb{R}^{d_1}$. Затем вычислим функцию $v_2(x) = f_2(v_1(x), w_2)$, где $w_2 \in \mathbb{R}^{m_2}$, $v_2(x) \in \mathbb{R}^{d_2}$. Будем считать, что всего в этой цепочке n функций, и в последнюю очередь мы вычисляем $v_n(x) = f_n(v_{n-1}(x), w_n)$, где $w_n \in \mathbb{R}^{m_n}$, $v_n(x) \in \mathbb{R}^{d_n}$, причём выходы функции $v_n(x)$ являются прогнозами. Иными словами, мы определили модель $a(x)$, результат работы которой определяется путём последовательного вычисления *слоёв* $f_1, \ldots, f_n$.

Существует большое количество типов слоёв. Один из популярных вариантов — *полносвязный слой* (fully connected layer), у которого каждый выход вычисляется

как нелинейная функция от скалярного произведения входного вектора на веса. Например, если i-й слой является полносвязным, то j-я компонента его выхода будет вычисляться как

$$f_{ij}(v, W, w_0) = g\left(\sum_{k=1}^{d_{i-1}} w_{jk} v_k + w_{0j}\right), \quad j = 1, \ldots, d_i,$$

причём параметры состоят из матрицы $W \in \mathbb{R}^{d_{i-1} \times d_i}$ и вектора $w_0 \in \mathbb{R}^{d_i}$. Заметим, что полносвязный слой легко записать в векторном виде:

$$f_i(v, W, w_0) = g(Wv + w_0).$$

Функция $g(z)$ называется *функцией активации* и может быть, например, сигмоидой $\frac{1}{1+\exp(-z)}$. Также часто используется функция ReLU (rectified linear unit):

$$g(z) = \max(0, z).$$

Полносвязный слой никак не учитывает структуру данных, в нём каждый выход зависит от каждого входа. Иногда можно сделать слои более приспособленными к конкретной задаче. Так, в обработке изображений применяются *свёрточные* слои, а в анализе текстов *рекуррентные* слои. Также слои не должны выстраиваться в строго последовательную структуру, они вместе со связями должны лишь представлять собой граф без ориентированных циклов, истоком которого являются признаки объекта, а стоком — прогнозы. Тем не менее, для простоты обозначений и изложения мы будем рассматривать только графы с последовательным расположением слоёв. О примерах слоёв мы поговорим позже, а пока обсудим обучение графов вычислений в общем случае.

2 Обучение и обратное распространение ошибки

Итак, мы обсудили, что граф вычислений должен быть ациклическим, а каждая вершина (или слой) f_i в нём может иметь параметры w_i, которые необходимо обучить. Как и во всех других методах машинного обучения, будем обучать параметры путём минимизации функционала ошибки:

$$Q(w) = \sum_{i=1}^{\ell} L(y_i, a(x_i; w)) \to \min_w,$$

где w — вектор, состоящий из параметров всех слоёв. Если все слои f_i являются дифференцируемыми по входам v и по параметрам w_i, то частная производная функционала по любому набору параметру $\frac{\partial Q}{\partial w_i}$ существует, но при этом её может быть очень тяжело вычислить аналитически, поскольку функционал является глубокой суперпозицией слоёв.

Тем не менее, задача вычисления производных по параметрам может быть решена с помощью правила дифференцирования сложных функций. Допустим, мы хотим найти производную $\partial Q / \partial w_i$. Выразим её через производные по более поздним слоям графа:

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial f_{i+1}} \frac{\partial f_{i+1}}{\partial f_i} \frac{\partial f_i}{\partial w_i}.$$

Производные $\partial f_{i+1}/\partial f_i$ и $\partial f_i/\partial w_i$ можно посчитать аналитически исходя из вида слоя f_i, а производная $\partial Q/\partial f_{i+1}$ снова может быть выражена через более поздние слои графа аналогичным способом. Таким образом, если каждый слой может вычислять производные по входу и по параметрам, то можно рекуррентно вычислить производные функционала по всем параметрам графа. Сначала вычисляются производные по параметрам самого последнего слоя, затем по параметрам предпоследнего и так далее до самого первого слоя. Данная процедура называется *обратным распространением ошибки* (backprop).

Получается, что при использовании градиентных методов каждый их шаг будет состоять из трёх действий:

1. Прямой проход, при котором вычисляются значения всех слоёв f_i и прогнозы $a(x)$;
2. Обратный проход, при котором вычисляются все производные $\partial Q/\partial w_i$;
3. Обновление параметров на основе вычисленных производных.

Отсюда можно сделать вывод, что если для каждого слоя реализованы операции прямого и обратного проходов, то можно достаточно легко вычислять производные в сколь угодно больших графах вычислений. Такой подход широко используется и называется *автоматическим дифференцированием* — по сути, пользователь может самостоятельно собирать графы из заранее реализованных слоёв, и не беспокоиться о процедурах обучения.

3 Аппроксимация с помощью графов вычислений

Линейные модели имеют большое количество преимуществ, но при этом могут аппроксимировать лишь линейные зависимости между целевой переменной и признаками. Удалось ли нам устранить этот недостаток, перейдя к нелинейным графам вычислений? Ответ на этот вопрос даёт теорема Цыбенко [1], согласно которой непрерывную функцию $F(x)$, заданную на компактном множестве, можно с любой точностью ε приблизить графом вычислений, состоящим из двух полносвязных слоёв, если первый из них использует сигмоидную функцию активации:

$$\left| F(x) - \sum_{j=1}^{n_1} \alpha_j \sigma(w_j^T x + w_{0j}) \right| < \varepsilon.$$

Получается, что даже двух слоёв достаточно, чтобы восстановить любую непрерывную зависимость. Разумеется, для этого понадобится очень большой размер первого слоя n_1, при котором на практике невозможно будет избежать переобучения; по этой причине популярностью пользуются глубинные модели, состоящие из большего количества слоёв, которые при этом имеют меньше параметров.

4 Свёрточные сети

В компьютерном зрении достаточно давно анализ изображений основан на применении *фильтров* — специальных преобразований, позволяющих обнаружить

участки, обладающие определёнными свойствами. Примером может служить лапласовский фильтр, детектирующий края объектов на изображении. С помощью фильтров можно выделить большое количество признаков, на которых далее можно обучать любую модель — например, градиентный бустинг.

Позже возникла идея обучения фильтров под конкретную задачу, которая, в свою очередь, привела к появлению свёрточных слоёв и свёрточных нейронных сетей (convolutional neural networks). Обсудим устройство свёрточного слоя подробнее.

Пусть дано изображение, представляющее собой функцию $F(a, b)$, заданную на сетке размера $p \times q$. Если изображение является многоканальным (например, имеет несколько цветовых каналов), то дальнейшие действия проводятся для каждого канала в отдельности. Фильтром называется функция $G(a, b)$, заданная на небольшой сетке $(-s, \times, s) \times (-t, \times, t)$. Так, фильтр Лапласа имеет размер 3×3 и задаётся как

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Чтобы применить фильтр к изображению в точке (i, j), необходимо вычислить их свёртку:

$$f(i, j) = \sum_{k=-s}^{s} \sum_{l=-t}^{t} F(i + k, j + l)G(k, l).$$

Вычислив свёртку в каждой точке, мы получим новое изображение $f(i, j)$, которое и будет результатом работы свёрточного слоя. Коэффициенты фильтра $G(a, b)$ являются параметрами и подбираются в процессе обучения графа вычислений [1].

Заметим, что мы могли бы в каждой точке (i, j) использовать фильтр с собственными коэффициентами, но это привело бы к появлению очень большого количества параметров. Поэтому чаще всего для всего изображения используется один и тот же фильтр — это является своеобразной регуляризацией и соответствует предположению о том, что если фильтр полезен в одной части изображения, то он должен иметь смысл и во всех остальных частях.

Transfer learning. Свёрточные сети показывают очень высокое качество в задачах классификации изображений, но для достижения такого качества необходимо использовать графы с большим количеством слоёв, для обучения которых требуются огромные выборки и большие вычислительные ресурсы. При этом очень часто на практике возникают задачи, в которых имеется лишь несколько тысяч изображений, на которых невозможно обучить адекватный граф вычислений. В таких случаях, как правило, берут граф, обученный на большом наборе данных (например, VGG-19, обученный на ImageNet), берут выходы одного из последних слоёв, и работают с ними как с признаками изображений, на которых обучаются стандартные алгоритмы машинного обучения. Такой подход с использованием одной модели для разных задач носит название *transfer learning*.

[1] Подробнее о свёрточных сетях можно почитать в лекциях cs231n: http://cs231n.github.io/convolutional-networks/

5 Рекуррентные сети

Свёрточные слои хорошо подходят для анализа изображений, поскольку учитывают их пространственную структуру. Текстовые данные представляют собой последовательный набор токенов (то есть символов, букв или других базовых элементов), и для работы с ними применяются рекуррентные сети (recurrent neural networks), позволяющие обрабатывать такие последовательности.

Мы не будем вдаваться в детали реализации рекуррентных слоёв, а попытаемся лишь описать общую их идею. В рекуррентных слоях дополнительно появляется *время* — в каждый следующий момент времени t на вход подаётся очередной токен x_t из входной последовательности. Также рекуррентный слой хранит скрытое состояние h_t, которое позволяет «помнить» токены, которые поступали на вход ранее. В момент времени t происходит обновление скрытого состояния

$$h_t = g_1(W_{xh}x_t + W_{hh}h_{t-1}),$$

после чего вычисляется выход

$$f_t = g_2(W_{hy}h_t).$$

Матрицы W_{xh}, W_{hh} и W_{hy} являются параметрами и настраиваются в процессе обучения.

С помощью рекуррентных сетей удаётся решать задачи машинного перевода и распознавания речи, генерировать новые тексты в определённом стиле, а также строить вопросно-ответные системы.

6 Глубинное обучение

Нейронные сети впервые стали использоваться в 1950-х годах, причём они были разработаны именно как компьютерные модели нейронов. Их результаты имели большой резонанс, и, разумеется, на них были возложены надежды по моделированию работы человеческого мозга.

Позже оказалось, что принципы работы нейронов несколько сложнее линейной комбинации входов — так, модель Ходжкина-Хаксли использует дифференциальные уравнения для описания соответствующих биологических процессов. Тем не менее, термин «нейронные сети» закрепился и до сих пор широко используется.

Уже в 1980-е годы были известны архитектуры и с полносвязными, и со свёрточными, и с рекуррентными слоями, а также были разработаны процедуры их обучения. Тем не менее, популярность нейронных сетей была низкой, и вместо них на практике использовались другие подходы. Дело в том, что нейронные сети обладают огромной выразительной мощностью, но для эксплуатации этой мощности необходимы большие объёмы обучающих данных — без них нейросеть либо переобучится, либо окажется слишком слабой из-за регуляризации. Популярность к нейросетям пришла уже в 21-м веке, когда во многих областях появились большие наборы данных, на них удалось обучить сложные нейронные сети, и за счёт этого были получены резкие увеличения качества по сравнению с более слабыми моделями. Такие сети имели большое количество слоёв, из-за чего их стали называть *глубинными сетями*, а саму

часть машинного обучения, посвящённую нейронным сетям — *глубинным обучением* (deep learning).

Благодаря нескольким прорывам глубинные сети стали активно развиваться, появились новые градиентные методы обучения и разнообразные подходы к регуляризации, а благодаря увеличению вычислительных мощностей стало возможно обучать всё более сложные архитектуры. Также следует отметить, что методы стохастической оптимизации (SGD и другие) нашли очень широкое применение в глубинном обучении — без них невозможно настроить большой граф вычислений на больших выборках.

Список литературы

[1] *Cybenko, G. V.* (1989). Approximation by Superpositions of a Sigmoidal function. // Mathematics of Control Signals and Systems, vol. 2 no. 4 pp. 303-314.