Оптимизация: Проксимальный метод, ADMM

Феоктистов Дмитрий

7 марта 2024

Дисклеймер. Изложение в тексте не является математически строгим, цель автора – познакомить студентов с методами, показать интуицию, стоящую за ними, а также объяснить, что учить оптимизацию полезно и интересно. Для поулучения более правильного изложения рекомендуется ознакомиться с [11, 2, 10, 12].

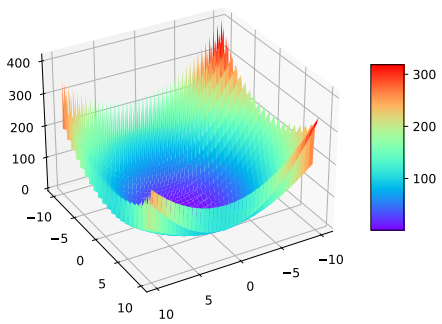
А зачем вообще изучать оптимизацию машин лернеру?

Итак, какую задачу мы решаем в машинном обучении?

$$\min_{\theta} f(\theta)$$

Подозрительно похоже на задачу, с которой работает оптимизация. Но въедливый читатель может возразить: все адекватные теоретические гарантии в оптимизации получены для выпуклых функций, а в машинном обучении уже лет 10 как нейронные сети, которые почти наверное не являются выпуклыми. Что ж, замечание действительно верное, но на него есть ответы:

1. Сойтись к локальному минимуму или стационарной точке лучше, чем не сойтись никуда.
2. В окрестности локального минимума функция становится выпуклой, и там мы сможем быстро сойтись.
3. Иногда невыпуклая функция является в некотором смысле «зашумленной» версией выпуклой или похожей на выпуклую, например:



4. Некоторые серьезные задачи можно решить чисто оптимизационными методами, где все будет выпукло и красиво.

5. На практике тупой перенос методов, разработанных для выпуклых задач, на нейронки и другие модели машинного обучения работает очень даже хорошо.

Градиентный спуск: откуда и для чего?

Какой самый популярный метод оптимизации мы знаем? Конечно же, градиентный спуск. Но откуда он вообще взялся? Неожиданно, из диффуров... Рассмотрим следующую динамическую систему:

$$\frac{dx}{dt} = -\nabla f(x)$$

И покажем, что значение функции $V(x) = f(x)$ убывает на траекториях системы (для тех, кто не забыл диффуры, такая функция называется функцией Ляпунова). Действительно:

$$\frac{dV(x(t))}{dt} = \langle \nabla f(x(t)), \frac{dx(t)}{dt} \rangle = \langle \nabla f(x(t)), -\nabla f(x(t)) \rangle = -\|\nabla f(x)\|_2^2 \leq 0$$

Хммм, а что если посчитать производную динамики конечно разностным способом, то это свойство должно сохраниться. Давайте сделаем это! Что же тогда выйдет:

$$\frac{x^{k+1} - x^k}{\alpha_k} = -\nabla f(x^k)$$

$$\Rightarrow x^{k+1} = x^k - \alpha_k \nabla f(x^k)$$

Вау, да это же градиентный спуск! Что мы можем сразу сказать про него? А то, что из выражения для $\frac{dV(x(t))}{dt}$ следует, что он точно сойдется к некоторой стационарной точке (так как динамика закончится, когда $\nabla f(x) = 0$), а в случае выпуклых функций к глобальному минимуму, что уже круто. Но мы то хотим не только сходимость, мы то хотим сходимость, как можно быстрее. Итак, обозначим за f^* оптимальное значение функции, тогда известно, что в классе выпуклых L-гладких функций (выпуклых функций с липшицевым градиентом, т.е. $\|\nabla f(y) - \nabla f(x)\|_2 \leq L\|x - y\|_2$) и при хорошем градиентном шаге верно:

$$f(x^k) - f^* = O(\frac{1}{k})$$

Замечание. Так как я не настолько хорош как Светило, то душить вас доказательством оценочек я не буду, но всем интересующимся советую заглянуть в учебник.

Замечание. Эта скорость сходимости является далеко не самой лучшей, на практике часто используются другие методы, о которых вам расскажут на следующих занятиях.

Как можно догадаться, L-гладкость это довольно обременительное свойство, действительно, SVM, регрессия с L_1 регуляризацией не являются гладкими, поэтому хочется уметь что-то говорить и в случае, когда это замечательное свойство отсутствует. Оказывается, в этом случае помогает понятие субградиента. Субградиентом функции $f(x)$ в точке x называется такой вектор a, что:

$$\forall y : f(y) \geq f(x) + \langle a, y - x \rangle$$

Нас больше всего интересуют следующие моменты в этом определении: для выпуклой функции субградиент существует в любой точке, если существует градиент, то он является субградиентом. И если заменить в градиентом спуске градиент на субградиент и немного модифицировать выбор шага, то получится субградиентный метод, который сходится со скоростью $O(\frac{1}{\sqrt{k}})$, что значительно медленнее.

Замечание. Некорректно говорить «субградиентный спуск», так как нет гарантий, что при каждом шаге алгоритма, значение функционала будет уменьшаться.

То есть, как только мы хотим получить разреженную модель, добавив L_1 регуляризацию даже к гладкой функции, так сразу нужно сильно дольше ждать сходимости. Это очень грустно, и возникает вопрос:

Как жить, если нет гладкости?

Вернемся к истокам, а именно к разностной схеме. Как вы узнаете из курса численных методов, полученная разностная схема является явной, но есть и неявные схемы, при которых значение нужной функции берется буквально в следующей точке. Давайте посмотрим, что можно получить, использовав такой подозрительный прием:

$$\frac{x^{k+1} - x^k}{\alpha_k} = -\nabla f(x^{k+1})$$

И заметим, что если рассмотреть функцию $g(u) = \frac{1}{2\alpha_k}\|u - x_k\|_2^2$, то оказывается, что:

$$\frac{x^{k+1} - x^k}{\alpha_k} = \nabla g(x^{k+1})$$

А значит, что наша неявная разностная схема превращается в равенство:

$$\nabla g(x^{k+1}) + \nabla f(x^{k+1}) = 0$$

А это есть ни что иное как условие оптимальности для функции $f(u) + g(u)$ в точке x^{k+1}. А в силу того, что $f(u)$ и $g(u)$ выпуклы, то x^{k+1} есть просто минимум суммы этих функций. Давайте это запишем:

$$x^{k+1} = \arg\min_u \{f(u) + \frac{1}{2\alpha_k}\|u - x_k\|_2^2\}$$

И теперь мы готовы ввести понятие проксимального оператора:

$$prox_{f,\alpha}(x) = \arg\min_u \{f(u) + \frac{\alpha}{2}\|u - x_k\|_2^2\}$$

Как можно догадаться, проксимальный метод:

$$x^{k+1} = prox_{f,\frac{1}{\alpha_k}}(x^k)$$

Сходится с той же скоростью, что и градиентный спуск, т.е. $O(\frac{1}{k})$. Успех? Еще нет. Мы получили, что новая точка получается путем решения какой-то другой оптимизационной задачи, и что-то подсказывает, что аналитически не всегда выйдет посчитать этот оператор.

Без паники! Давайте вспомним, как более точно выглядит наша задача оптимизации, в том числе при использовании L_1 регуляризации:

$$\min_\theta f(\theta) + r(\theta)$$

Где часто первая функция хорошая, гладкая, а вторая тоже хорошая, но не гладкая. Мы знаем, что для первой классно работает градиентный спуск, а для второй должен неплохо работать проксимальный метод. Хммм, а давайте поженим два метода и применим к каждой функции то, что работает для нее хорошо. Тогда мы получим проксимальный градиентный метод:

$$x^{k+1} = prox_{r,\frac{1}{\alpha_k}}(x^k - \alpha_k \nabla f(x_k))$$

И этот метод так же будет сходиться со скоростью $O(\frac{1}{k})$! Но мы сильно расширили класс задач, к которому он применим: теперь мы можем оптимизировать гладкие функции, к которым применен негладкий регуляризатор с явно вычислимым проксимальным оператором. Давайте разберем примеры использования этого метода.

Рассмотрим задачу оптимизации на множестве:

$$\min_x f(x) \;\; s.t. \;\; x \in X$$

И внесем ограничение в функционал через индикатор, тогда получим задачу:

$$\min_x f(x) + I_X(x)$$

Хотим применить к ней проксимальный градиентный метод. Для этого достаточно посчитать проксимальный оператор для $I_X(x)$:

$$prox_{I_X,\alpha}(x) = \arg\min_u \{I_X(u) + \frac{\alpha}{2}\|u-x\|_2^2\} = \arg\min_{u \in X} \{\frac{\alpha}{2}\|u-x\|_2^2\} = \arg\min_{u \in X} \{\|u-x\|_2^2\} = Pr_X(x)$$

То есть мы получили, что проксимальный градиентный метод обобщает метод проекции градиента, а также бонусом получили и скорость сходимости последнего.

Ну и теперь перейдем к тому, о чем мы говорили все занятие - к L_1 регуляризации. А выводить ее мы будем для задачи линейной регрессии. Спойлер: полученный метод является основой для всех современных солверов подобных задач. Рассмотрим задачу:

$$\min_w \|Xw - y\|_2^2 + \lambda\|w\|_1$$

Найдем проксимальный оператор для второго слагаемого:

$$prox_{\lambda\|\cdot\|_1,\alpha}(x) = \arg\min_u \{\lambda\|u\|_1 + \frac{\alpha}{2}\|u-x\|_2^2\} = \arg\min_u \{\|u\|_1 + \frac{\alpha}{2\lambda}\|u-x\|_2^2\} = \arg\min_u \{\sum_{i=1}^{d} |u_i| + \frac{\alpha}{2\lambda}(u_i - x_i)^2\}$$

И заметим, что каждое слагаемое зависит только от одной координаты u. Тогда мы получаем d одномерных задач вида:

$$\min_{u_i} |u_i| + \frac{\alpha}{2\lambda}(u_i - x_i)^2$$

Откуда легко получается решение:

$$u_i = \begin{cases} x_i - \frac{\lambda}{\alpha}, \; x_i > \frac{\lambda}{\alpha}, \\ 0, \; |x_i| \le \frac{\lambda}{\alpha}, \\ x_i + \frac{\lambda}{\alpha}, \; x_i < -\frac{\lambda}{\alpha}, \end{cases}$$

У проксимального оператора для первой нормы есть специальное название – soft thresholding или shrinkage оператор (обозначается как Sr_β, где β – параметр проксимального оператора). А полученный метод решения регрессионной задачи называется Iterative Shrinkage-Thresholding Algorithm (ISTA):

$$w^{k+1} = Sr_{\frac{\lambda}{\alpha}}(w^k - \alpha_k X^T(Xw - y))$$

Замечание. На практике часто используется метод FISTA: Fast Iterative Shrinkage-Thresholding Algorithm, который получается из ISTA добавлением ускорения. А что такое ускорение и как его добавлять вы узнаете в следующий раз.

Замечание. Конечно, мы не стали бы тратить целую пару, если бы за нее только научились бы быстрее решать задачи с L_1 регуляризацией. Во-первых, это не единственная полезная регуляризация, ведущая к уменьшению количества параметров. Можно также накладывать регуляризацию на ядерную норму матрицы, что будет вести к малоранговости. А чтобы записать проксимальный оператор для нее достаточно воспользоваться следующим свойством:

$$X = U\Sigma V^T, f(X) = g(\sigma) \;\Rightarrow\; prox_{f,\alpha}(X) = Udiag(prox_{g,\alpha}(\sigma))V^T$$

Во-вторых, проксимальные операторы, а тем более soft thresholding используются при решении большого количества задач, и даже позволяют получить новый взгляд на обучение нейронных сетей [5, 6, 1]. В-третьих, проксимальные методы будут иметь ключевое место в ADMM – очень популярном методе, который мы посмотрим после лирического отступления.

Лирическое отступление

Помимо технических результатов, мы получили сегодня несколько важных философских наблюдений. Во-первых, крайне важно учитывать структуру задачи и модифицировать методы под нее. Если бы мы просто запускали субградиентный спуск, то мы бы остались со скоростью $O(\frac{1}{\sqrt{k}})$. Если бы мы остановились просто на проксимальном методе, то мы бы не получили возможности применять его на практике, так как он считается явно довольно редко. Но стоило нам учесть, что функционал у нас строгого вида, так сразу получили ускорение. Небольшая затравка на будущее: вообще говоря, в машинном обучении у нас функционал имеет еще более строгий вид:

$$\min_\theta \sum_{i=1}^{n} f_i(\theta) + r(\theta)$$

Что делать с последним слагаемым мы узнали, а вот как учесть, что у нас вид суммы, вы узнаете в следующий раз. Во-вторых, мы поняли, что теоретический анализ довольно важен, так как как раз таки он и позволяет учесть структуру. В теоремах о сходимости методов всегда содержится инструкция по его применению: для каких функций он сходится, из каких точек, с какой скоростью. Эта идея очень хорошо описана в параграфе 6 главы 5 книги «Введение в оптимизацию» Б.Т. Поляка [12], всем советую ее прочитать тем более, что она занимает всего 4 страницы. Но, конечно, как показывает практика, вариант запрогать и запустить метод или заставить компьютер найти метод оптимизации самостоятельно ([4]) тоже работает.

Дедовский метод, решающий кучу прикладных задач

Сейчас мы разберем Alternating Direction Method of Multipliers (ADMM). Наверное, это самый популярный метод оптимизации после различных вариаций градиентного спуска, он и по сей день используется при решении различных задач, в том числе и в машинном обучении [9, 3, 8, 7]. Пусть у нас имеется следующая задача:

$$\begin{cases} f(x) + g(u) \to \min_{x,u} \\ s.t. \;\; Ax + Bu = c \end{cases}$$

Из практики мы знаем, что регуляризация часто делает задачу более стабильной, давайте добавим такую регуляризацию, что наша исходная задача не изменится:

$$\begin{cases} f(x) + g(u) + \frac{r}{2}\|Ax + Bu - c\|_2^2 \to \min_{x,u} \\ s.t. \;\; Ax + Bu - c = 0 \end{cases}$$

Действительно, пока у нас выполнено ограничение, то регуляризатор ни на что не влияет, а как только мы их нарушаем, то он начинает штрафовать, и мы в теории должны получить более стабильную траекторию и более быструю сходимость (так оно и есть). Звучит довольно разумно. И давайте вспомним, что условия оптимальности в случае задачи с ограничениями у нас записываются через Лагранжиан, давайте выпишем его:

$$L_r(x, u, \lambda) = f(x) + g(u) + \lambda^T(Ax + Bu - c) + \frac{r}{2}\|Ax + Bu - c\|_2^2$$

И давайте заметим, что:

$$\lambda^T(Ax + Bu - c) + \frac{r}{2}\|Ax + Bu - c\|_2^2 = \frac{1}{2r}\|\lambda + r(Ax + Bu - c)\|_2^2 - \frac{1}{2r}\|\lambda\|_2^2$$

Убедиться в справедливости этого утверждения можно просто раскрыв скобки. Тогда наш Лагранжиан начинает выглядеть так:

$$L_r(x, u, \lambda) = f(x) + g(u) + \frac{1}{2r}\|\lambda + r(Ax + Bu - c)\|_2^2 - \frac{1}{2r}\|\lambda\|_2^2$$

А что мы хотим делать с Лагранжианом? Правильно! Минимизировать по прямым переменным, и максимизировать по двойственным. Давайте по прямым делать явную минимизацию, а по двойственным градиентный подъем. Эта идея приводит к следующей схеме, которая и называется ADMM:

$$\begin{cases} x^{k+1} = \arg\min_x \{f(x) + \frac{1}{2r}\|\lambda^k + r(Ax + Bu^k - c)\|_2^2\} \\ u^{k+1} = \arg\min_u \{g(u) + \frac{1}{2r}\|\lambda^k + r(Ax^{k+1} + Bu - c)\|_2^2\} \\ \lambda^{k+1} = \lambda^k + r(Ax^{k+1} + Bu^{k+1} - c) \end{cases}$$

Замечание. Почему бы не делать просто градиентный спуск по прямым переменным и градиентный подъем по двойственным? Во-первых, такая схема может расходиться. Во-вторых, вся магия метода заключается в том, что часто внутренние задачи решаются аналитически.

Замечание. Почему Alternating Direction Method of Multipliers? Alternating Direction из-за того, что все группы переменных обновляются независимо и попеременно. А Multipliers говорит об использовании множителей Лагранжа.

Какие же есть преимущества у данного метода? Во-первых, как показывает практика, использование информации о двойственных переменных часто ускоряет сходимость. Во-вторых, часто внутренние задачи решаются аналитически, причем довольно эффективно. А это значит, что ADMM превращается в просто итерационный подсчет по формулам, не требующий информации о градиентах и пр, что делает его итерации очень быстрыми. В-третьих, мы снова учли структур задачи: при пересчете x мы используем только $f(x)$, при пересчете u только $g(u)$, а это значит, что если эти функции дорого вычислять, то нам не придется вызывать обе функции при обновлении каждого блока переменных. Это наблюдение также наталкивает на мысль, что иногда мы можем себе позволить решать внутренние задачи численно. В-четвертых, несмотря на довольно специфичный вид решаемой задачи, он включает в себя большое количество задач, что делает этот метод одним из самых популярных методов оптимизации. Часто переменные u являются вспомогательными и вводятся искусственно, чтобы свести задачу к нужному виду, этот прием мы рассмотрим далее.

Замечание. Возникает вопрос, как выбирать r. Как можно видеть из третьей строчки, r является градиентном шагом при подъеме в пространстве двойственных переменных. А уж как выбирать размеры шага мы хорошо знаем.

Теперь рассмотрим идеологию, которую продвигает этот метод. Вот есть у нас какая-то задача оптимизации, мы должны ее переписать в виде, который допускает ADMM, после чего все минимумы нужно посчитать аналитически, а затем просто запрогать эту схему, чтобы стало хорошо. Давайте посмотрим, как это можно делать на примере уже хорошо знакомых по сегодняшней паре задач.

Для начала разберем оптимизацию на простых множествах:

$$\min_x f(x) \;\; s.t. x \;\; \in X$$

$$\min_x f(x) + I_X(x)$$

$$\begin{cases} f(x) + I_X(z) \to \min_{x,z} \\ s.t. \;\; x = z \end{cases}$$

И путем несложных математических выкладок получим следующую схему:

$$\begin{cases} x^{k+1} = prox_{f,\frac{r}{2}}(z^k - \frac{\lambda^k}{r}) \\ z^{k+1} = Pr_X(x^{k+1} + \frac{\lambda^k}{r}) \\ \lambda^{k+1} = \lambda^k + r(x^{k+1} - z^{k+1}) \end{cases}$$

Теперь разберем задачу регрессии с L_1 регуляризацией:

$$\min_x \frac{1}{2}\|Xw - y\|_2^2 + \gamma\|w\|_1$$

$$\begin{cases} \frac{1}{2}\|Xw - y\|_2^2 + \gamma\|z\|_1 \to \min_{w,z} \\ s.t. \;\; w = z \end{cases}$$

И путем несложных математических выкладок получим следующую схему:

$$\begin{cases} w^{k+1} = (rI + X^T X)^{-1}(X^T y + rz^k - \lambda^k) \\ z^{k+1} = Sr_{\frac{r}{\gamma}}(w^{k+1} + \frac{\lambda^k}{r}) \\ \lambda^{k+1} = \lambda^k + r(w^{k+1} - z^{k+1}) \end{cases}$$

Также в домашке вы сможете дополнительно поупражняться в записи ADMM для прикольной задачи. А на этом все!

Список литературы

[1] Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. arXiv preprint arXiv:1810.00861, 2018.

[2] Stephen P Boyd. Convex optimization.

[3] Ting-An Chen, De-Nian Yang, and Ming-Syan Chen. Alignq: alignment quantization with admm-based correlation preservation. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 12538–12547, 2022.

[4] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. Advances in Neural Information Processing Systems, 36, 2024.

[5] Thomas Frerix, Thomas Möllenhoff, Michael Moeller, and Daniel Cremers. Proximal backpropagation. arXiv preprint arXiv:1706.04638, 2017.

[6] Tim Tsz-Kit Lau, Jinshan Zeng, Baoyuan Wu, and Yuan Yao. A proximal block coordinate descent algorithm for deep neural network training. arXiv preprint arXiv:1803.09082, 2018.

[7] Hongjia Li, Ning Liu, Xiaolong Ma, Sheng Lin, Shaokai Ye, Tianyun Zhang, Xue Lin, Wenyao Xu, and Yanzhi Wang. Admm-based weight pruning for real-time deep learning acceleration on mobile devices. In Proceedings of the 2019 on Great Lakes Symposium on VLSI, pages 501–506, 2019.

[8] Fei Wen, Rendong Ying, Peilin Liu, and Robert C Qiu. Robust pca using generalized nonconvex regularization. IEEE Transactions on Circuits and Systems for Video Technology, 30(6):1497–1510, 2019.

[9] Shaokai Ye, Xiaoyu Feng, Tianyun Zhang, Xiaolong Ma, Sheng Lin, Zhengang Li, Kaidi Xu, Wujie Wen, Sijia Liu, Jian Tang, et al. Progressive dnn compression: A key to achieve ultra-high weight pruning and quantization rates using admm. arXiv preprint arXiv:1903.09769, 2019.

[10] А.В. Гасников. Универсальный градиентный спуск.

[11] Ю.Е. Нестеров. Методы выпуклой оптимизации.

[12] Б.Т. Поляк. Введение в оптимизацию.