Оптимизация: Модификации градиентного спуска

Оганов Александр

18 февраля 2025

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.

Мотивация

Как и на прошлой паре, мы решаем следующую задачу:

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

Однако теперь обратимся к различным модификациям градиентного спуска, которые либо его ускоряют (моментум, ускорение Нестерова), либо позволяют решать новые задачи (стохастический градиентный спуск). Цель конспекта — дать понимание откуда появлялись различные приемы и какая мотивация стоит за их использованием.

Градиентный спуск

Вспомним как выглядит итерация градиентного спуска:

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

Напомним основные детали, которые послужат мотивацией для следующих выкладок.

Обозначим за f^* оптимальное значение функции, тогда если f — L-гладкая функция[1] и правильно выбрать α_k, то будет верно:

$$\text{Если } f(x^k) - f^* \leq \varepsilon, \text{ то } k = O(\frac{1}{\varepsilon}). \tag{1}$$

Важно заметить, что функционал будет убывать каждую итерацию, что является очень приятным свойством. Однако нам бы хотелось найти метод, который сходился бы за меньшее число итераций и был бы асимптотически наилучшим, если такой метод вообще есть...

Многошаговые методы

В градиентном спуске на каждом шаге никак не используется информация с прошлых итераций, а что если начать как-нибудь ее учитывать[2]?

[1] Выпуклая функции с Липшицевым градиентом, т.е. $\|\nabla f(y) - \nabla f(x)\|_2 \leq L\|x - y\|_2$

[2] Чем больше фичей, тем лучше!

Так как на каждой итерации мы считаем градиент, разумно было бы не забывать о нем, а как-то переиспользовать, например так:

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

Если $\beta = 0$[3], то мы получим любимый нами градиентный спуск, то есть мы обобщили метод, что уже делает нашу идею интересней. Однако не совсем понятно какой смысл несет такая запись, и как наше желание использовать прошлые значения градиента согласуется со здравым смыслом.

Для начала заметим, что для градиентного спуска верно $\nabla f(x^{k-1}) = \frac{x^{k-1} - x^k}{\alpha_{k-1}}$, то есть мы можем переписать наш метод следующим образом:

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k) + \mu_k (x^k - x^{k-1}), \quad \mu_k := -\frac{\beta}{\alpha_{k-1}}.$$

Для упрощения изложения будем считать $\mu_k := \mu$.Чем-то полученное выражение напоминает разностную схему. Если чуть-чуть поиграться с коэффициентами и вспомнить численные методы, то можно увидеть дискретизацию следующего дифференциального уравнения:

$$\mu \frac{d^2x}{dt^2} = -\nabla f(x^k) - p\frac{dx}{dt}.$$

Полученное уравнение описывает движение тела в потенциальном поле f при наличии силы трения. Так как тело постоянно тратит энергию на силу трения, то в конце концов тело окажется в точке минимума потенциала f. Однако это лишь физический смысл полученного метода, который никак не доказывает сходимость. Условия сходимости и более подробное описание метода можно найти [Поляк(1983)].

Из приведенных выше рассуждений мы получили метод «тяжелого шарика», итерации которого обычно записывают так:

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

Такое название произошло из приведенный выше физической аналогии (движение тяжелого шарика при наличии силы трения), кроме того слагаемое $\mu(x^k - x^{k-1})$ часто называют инерцией. Заметим, что величина инерции зависит от параметра μ, который необходимо выбирать для каждой модели, часто выбирают $\mu \in [0, 1)$.

К сожалению, у данного метода нет разумных[4] теоретических гарантий сходимости, а еще метод не обеспечивает монотонной минимизации функционала. Возникает вопрос: а зачем нам метод, который теоретически хуже градиентного спуска, так еще и требует дополнительной памяти?

Несмотря на такое количество минусов, на практике метод сходится быстрее градиентного спуска, но почему? На практике траектории получаются более «гладкие», что может ускорить сходимость Рис. 1 и Рис. 2a.

Рассмотрим как выглядят итерации метода начиная с первой:

$$x^1 - x^0 = -\alpha_0 \nabla f(x^0), \tag{2}$$
$$x^2 - x^1 = -\alpha_1 \nabla f(x^1) + \mu(x^1 - x^0), \tag{3}$$
$$x^3 - x^2 = -\alpha_2 \nabla f(x^2) + \mu(x^2 - x^1), \tag{4}$$
$$x^{k+1} - x^k = -\alpha_k \nabla f(x^k) + \mu(x^k - x^{k-1}). \tag{5}$$

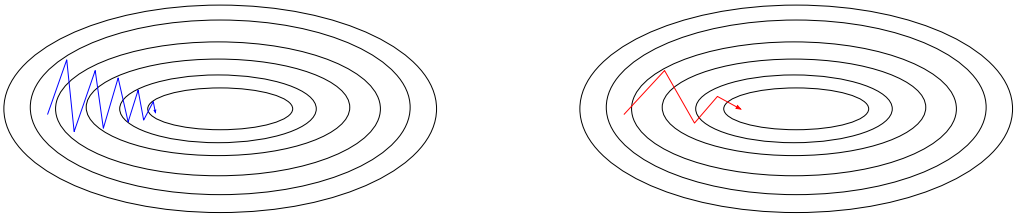
[3]Для простоты считаем, что β не зависит от k. При желании можно рассмотреть общий случай

[4]Оценки, конечно, существуют, но их тяжело связать с реальным миром

Будем считать, что $\alpha_k := \alpha$, тогда подставляя левую часть 2 в правую часть 3, после чего аналогично подставляя 3 в 4 и т.д., мы получим:

$$
\begin{aligned}
x^2 - x^1 &= -\alpha(\nabla f(x^1) + \mu \nabla f(x^0)), \\
x^3 - x^2 &= -\alpha(\nabla f(x^2) + \mu \nabla f(x^1) + \mu^2 \nabla f(x^0)), \\
x^{k+1} &= x^k - \alpha \sum_{i=0}^{k} \mu^{k-i} \nabla f(x^i).
\end{aligned}
$$

Тем самым мы показали, что добавление инерции соответствует экспоненциальному сглаживанию градиентов всех прошлых значений [Wikipedia contributors(2024a)].



(a) Градиентный спуск

(b) Градиентный спуск с инерцией

Рис. 1: Примеры траекторий получаемых при оптимизации

Ускоренный метод Нестерова

Выше развивалась идея использования уже посчитанных градиентов, при этом мы всегда считали их в точках $f(x^k)$, что достаточно разумно. К сожалению, мы не получили никаких теоретических гарантий, хотя бы для выпуклых функций, но не стоит отчаиваться. Далее мы рассмотрим метод Нестерова, который часто называется «Ускоренным методом Нестерова». Для начала приведем сам алгоритм, далее постараемся понять, почему он записан именно так[5].

В методе Нестерова мы считаем градиент по следующей схеме:

$$
\begin{aligned}
x^{k+1} &= y^k - \alpha_k \nabla f(y^k), \\
y^{k+1} &= x^{k+1} + \mu_k (x^{k+1} - x^k).
\end{aligned} \tag{6}
$$

Если внимательно посмотреть на алгоритм получения y^{k+1}, можно заметить слагаемое отвечающее за инерцию (ускорение). Для наглядности подставим y^k в формулу для x^{k+1} и получим:

$$
x^{k+1} = x^k - \alpha_k \nabla f(x^k + \mu_{k-1}(x^k - x^{k-1})) + \mu_{k-1}(x^k - x^{k-1}). \tag{7}
$$

Традиционно метод записывают в виде 6. При записи 7, можно увидеть как метод Нестерова продолжает идею метода тяжелого шарика, но теперь инерция учитывается и при подсчете градиента. Благодаря теории известно:

[5]Спойлер! Метод получается из теоретических оценок, путем подбора параметров для лучшей сходимости, тем самым гарантируя оптимальность метода. Чем-то напоминает метод Лупанова...

Если f — L-гладкая функция и правильно выбрать α_k, μ_k, то будет верна оценка[6]:

$$\text{Если } f(x^k) - f^* \leq \varepsilon, \text{ то } k = O(\frac{1}{\sqrt{\varepsilon}}).$$

Сравним полученную оценку на число итераций с оценкой для классического градиентного спуска 1. Мы видим, что линейная зависимость числа итераций k от величины $\frac{1}{\varepsilon}$ (обратной требуемой точности), заменилась на $\frac{1}{\sqrt{\varepsilon}}$. То есть при улучшении точности в 100 раз метод градиентного спуска будет работать в 100 раз дольше, а метод Нестерова только в 10 раз. Причем для сильно выпуклых функций в случае метода Нестерова уже будет справедлива намного более сильная логарифмическая оценка [Bubeck(2014)], но наша цель не рассмотреть как можно больше оценок.

К сожалению, понять откуда именно появилась идея, добавить ускорение в подсчет градиента без доказательства сходимости, невозможно, а его вывод займет почти всю пару. Выше нам удавалось интерпретировать работу алгоритма через дифференциальные уравнения и физику, но в случае метода Нестерова интерпретация не так очевидна [Ahn and Sra(2022), Su et al.(2015)Su, Boyd, and Candes].

На практике часто выбирают $\alpha_k := \alpha,\ \mu_k = \mu$, кроме того даже при выборе константного шага, как и и для градиентного спуска можно получить оценку на скорость сходимости.

Важно заметить, что лишь добавив инерцию в подсчет градиента, мы смогли:

- получить теорию, чего не удалось в методе тяжелого шарика;
- при одинаковых предположениях на функцию, получить лучше скорость сходимости, чем для градиентного спуска

Самым приятным является следующий факт: «Ускоренный метод Нестерова оптимален в классе методов, где следующая точка получается линейной комбинацией предыдущих точек и градиентов в них». Доказывать мы это, конечно, не будем, но для желающих стоит обратиться к [Нестеров(2010)]. К сожалению, ничего хорошего не бывает бесплатным, и нам требуется хранить x^k для следующей итерации. Хранения копии параметров кажется незначительным при маленькой размерности пространства, но это стоит учитывать, особенно при работе с нейронными сетями, где размерность сильно выше.

FISTA, ускорение все что вам нужно

На прошлой паре мы познакомились с алгоритмом ISTA (Iterative Shrinkage-Thresholding Algorithm), какую задачу он решает? Напомним, что итерация алгоритма выглядит так:

$$w^{k+1} = Sr(w^k - \alpha_k X^T(Xw^k - y)).$$

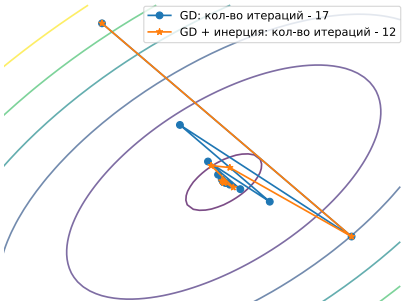
При этом скорость сходимости алгоритма совпадает со скоростью сходимости градиентного спуска, то есть $k = O(\frac{1}{\varepsilon})$. Что произойдет если мы добавим ускорение по аналогии с методом Нестерова? Мы получим примерно итерацию следующего вида:

$$\begin{aligned} w^{k+1} &= Sr(v^k - \alpha_k X^T(Xv^k - y)), \\ v^{k+1} &= w^{k+1} + \mu_k(w^{k+1} - w^k). \end{aligned}$$

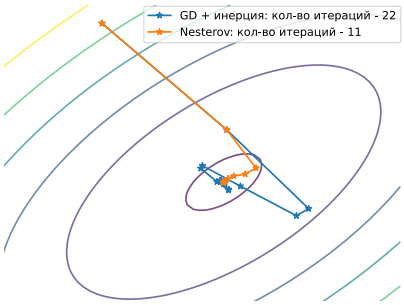
Если правильно выбрать μ_k, то согласно статье [Beck and Teboulle(2009)] мы получим скорость сходимости $k = O(\frac{1}{\sqrt{\varepsilon}})$. Авторами предложена следующая последовательность μ_k[7]:

[6] Доказательство можно найти в [Нестеров(2010)]

[7] Вообще говоря есть и другие последовательности, например, $\mu_k = \frac{k}{k+3}$, но мы повторим последовательность из статьи



(a) Сравнение градиентного спуска и метода тяжелого шарика, $\alpha = 0.4, \mu = 0.2$



(b) Сравнение градиентного спуска и метода Нестерова, $\alpha = 0.2, \mu = 0.5$

Рис. 2: Примеры траекторий получаемых при оптимизации функции $f(x, y) = \frac{((x-2)+(y-1))^2}{4} + ((x-2)-(y-1))^2$, при $(x_0, y_0) = (0, 4.5)$

$$
\begin{aligned}
t_0 &= 1, \\
t_{k+1} &= \frac{1 + \sqrt{1 + 4t_k^2}}{2}, \\
\mu_k &:= \frac{t_k - 1}{t_{k+1}}.
\end{aligned}
$$

Идейно мы[8] проделали следующие шаги:

1. получили ускорение Нестерова для градиентного спуска и исследовали как оно ускоряет сходимость
2. вспомнили про проксимальные методы и их связь с градиентным спуском
3. попробовали использовать ускорение Нестерова по аналогии для другой задачи
4. подобрали μ_k и получили схожие результаты с градиентным спуском

Стохастический градиентный спуск

Для любителей обучать машины и нейронные сети рассмотренные выше методы, часто являются непозволительной роскошью. Почему? Размеры датасетов, сложность моделей — все это не дает нам посчитать $\nabla f(x)$, а тем более $\nabla^2 f(x)$, где x — датасет, а данные функции нам крайне важны для оптимизации. Внимательный читатель заметит, что минимизация функции на датасете (выборке) ничего не говорит об ее значениях на всем пространстве (генеральной совокупности), так что даже $f(x)$ также не вычислимая, где x — генеральная совокупность. Еще более внимательный читатель скажет: «вообще говоря модель еще не должна переобучаться». Проблемы, которые заметили внимательные читатели, мы оставим машинообучателям, а сами постараемся разобраться с основами и проблемами стохастической оптимизации.

Сформулируем все что сказано выше более математически. $\mathcal{D}$ — датасет, а решаем мы следующую задачу:

[8] Мы и авторы статьи

$$\min_{\theta}[f(\theta) := \mathbb{E}_{\xi\sim\mathcal{D}} f(\theta, \xi)]. \tag{8}$$

Мы предполагаем, что датасет достаточно хорошо приближает генеральную совокупность, то есть минимизация функции потерь на датасете действительно решает задачу машинного обучения.

Если мы решаем задачу линейной регрессии без использования смещения и в качестве функции потерь MSE, то запись 8 будет знакома в следующем виде:

$$\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} (\theta^T x_i - y_i)^2, \quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^{|\mathcal{D}|}.$$

Заметим, что в случае когда у нас есть весь датасет[9] (например задача линейной регрессии), то мы аппроксимируем $\mathbb{E}_{\xi\sim\mathcal{D}} f(\theta, \xi)$ — методом Монте-Карло. Методы Монте-Карло хорошо развиты и в дальнейшем мы используем некоторые известные в них приемы.

Мы рассматриваем стохастическую оптимизацию первого порядка, поэтому важно понять, как работать с $\nabla f(\theta)$. Разумным является следующее предположение:

$$\nabla f(\theta) = \mathbb{E}_{\xi\sim\mathcal{D}} \nabla_\theta f(\theta, \xi). \tag{9}$$

Вдохновившись градиентным спуском, попробуем написать итерацию его стохастической версии:

$$\begin{aligned} &\xi^k \sim \mathcal{D}, \\ &\theta^{k+1} = \theta^k - \alpha_k \nabla f(\theta^k, \xi^k). \end{aligned}$$

Мы получили стохастический градиентный спуск (SGD), но прежде чем его использовать, хотелось бы понять его слабые стороны и применимость.

Для начала заметим, что θ^k случайная величина (так как $\nabla f(\theta^k, \xi^k)$ случайная величина), поэтому нас будет интересовать сходимость в смысле математического ожидания (мы не будем вдаваться в подробности и определения σ-алгебр, для желающих советуем обратиться к [Поляк(1983)]) Если $f(\cdot, \cdot)$ — выпуклая, то выбирая правильно α_k, будет верна следующая оценка:

$$\hat{\theta} = \frac{\sum_{i=0}^{k} \alpha_i \theta^i}{\sum_{i=0}^{k} \alpha_i},$$

$$\mathbb{E}_{\xi\sim\mathcal{D}}[\|\nabla f(\theta, \xi) - \nabla f(\theta)\|_2^2] \leq \sigma^2, \tag{10}$$

$$\mathbb{E} f(\hat{\theta}) - f^* \leq O(\frac{1}{k}) + \text{const}(\alpha_i) * \sigma^2. \tag{11}$$

Обозначение $\text{const}(\alpha_i)$ означает, что константа зависит от всех значений α_i. Для начала отметим, что оценки можно улучшить если выбирать шаг чуть хитрее или же добавлять разные

[9]Постановка задачи, когда известен весь датасет называется — офлайн, когда датасет всегда пополняется или меняется — онлайн. Стохастическая оптимизация решает задачу в обеих постановках. Для мотивации мы использовали офлайн задачу, так как она ближе для большинства читателей

требования на функцию (L-гладкость, μ-выпуклость). Однако зависимость от σ не исчезнет, соответственно весь дальнейший анализ не поменяется.

Выражение 10 отвечает за ограниченность дисперсии градиента, а если быть точным за равномерную ограниченность. Но как влияет дисперсия на сходимость? Например, в окрестности оптимума мы знаем, что градиент около нуля, но при высокой дисперсии мы будем менять направление градиента случайно, что будет мешать сходимости.

Обратим внимание на 11, слагаемое $O(\frac{1}{k})$ совпадает с оценкой для обычного градиентного спуска. При этом в случае SGD сходимости не будет, так как всегда присутствует слагаемое зависящее от σ[10]. Не важно сколько итераций мы будем делать, метод будет продолжать блуждать в окрестности минимума без сходимости.

Можно ли это исправить? Да! Есть 2 пути: выбрать меньше шаг (следует из доказательства и зависимости const от α_i), попробовать уменьшить дисперсию градиента.

Мы подробнее остановимся на методах из второй категории, которые называются методами редукции дисперсии (variance reduction).

Методы редукции дисперсии

На самом деле, с задачей уменьшения дисперсии мы уже сталкивались! Где именно? Правильно, при изучении ансамблей алгоритмов одной из мотиваций было уменьшение дисперсии базовой модели. Повторим эту идею для оценки $\nabla f(\theta)$, то есть в итерации градиентного шага произведем следующую замену:

$$\nabla f(\theta^k, \xi^k) \to \frac{1}{|S^k|} \sum_{i \in S^k} f(\theta^k, \xi_i).$$

При такой записи все элементы S^k выбираются независимо, где $S^k \subset \{1, \dots, |\mathcal{D}|\}$. Использование оценки не по одному сэмплу, а по подмножеству называют батчированием, при этом выбранное подмножество — батчем. Обычно все S^k выбираются одного размера (менять размер батча можно, но на практике трудно исполнимо). Проверить, что дисперсия уменьшиться в $|S^k|$ раз предоставляется читателю, так как все выкладки похожи на уже знакомые вам приемы из классического машинного обучения.

Для начала вспомним немного теории вероятности. Если ξ, η две случайные величины, определенные на одном вероятностном пространстве, то верно следующее:

$$\mathbb{D}[\xi - \eta] = \mathbb{D}[\xi] + \mathbb{D}[\eta] - 2\text{Cov}(\xi, \eta).$$

Чем нам может быть полезно это утверждение? Если ξ, η сильно скоррелированы (корреляция равна 1), то мы знаем, что $\text{Cov}(\xi, \eta) = \sqrt{\mathbb{D}[\xi]\mathbb{D}[\eta]}$. Получается:

$$\mathbb{D}[\xi - \eta] = (\sqrt{\mathbb{D}[\xi]} - \sqrt{\mathbb{D}[\eta]})^2.$$

Если дисперсии ξ, η равны, то получаем, что дисперсия разности равна 0! Ура! Если у нас есть сильно скоррелированные ξ, η с примерно похожими дисперсиями, то $\mathbb{D}[\xi - \eta]$ будет меньше, чем $D[\xi]$ и $\mathbb{D}[\eta]$, посмотрим, как этот случай применим к нашей задаче.

Мы хотим уменьшить $\mathbb{D}_{\xi \sim \mathcal{D}}[\nabla f(\theta, \xi)]$.

Рассмотрим $\mathbb{D}[\nabla f(\theta, \xi) - \nabla f(\hat{\theta}, \xi)]$, по выкладкам выше дисперсия тем меньше, чем более скоррелированы $\nabla f(\theta, \xi)$, $\nabla f(\hat{\theta}, \xi)$ и чем сильнее похожи их дисперсии. Если бы взяли $\theta := \theta^k$ и $\hat{\theta} := \theta^{k-1}$ (последовательные значения при градиентном спуске), то мы могли бы надеяться на

[10] К сожалению, мы живем не в идеальном мире, где дисперсии оценок нулевые...

выполнение этих требований. Почему? На соседних итерациях точка подсчета градиента изменяется не сильно, поэтому и значения градиентов будут примерно похожи. Данный факт, дает надежду, что мы получим сильно скоррелированные случайные величины с очень похожими дисперсиями[11]. Конечно, рассуждения выше не являются математическим доказательством, поэтому предлагаем любопытному читателю обратиться к секции 3 [Johnson and Zhang(2013)].

Было бы здорово в процедуре SGD сделать следующую замену:

$$\nabla f(\theta^k, \xi^k) \to \nabla f(\theta^k, \xi^k) - \nabla f(\theta^{k-1}, \xi^k). \tag{12}$$

Благодаря такой замене мы смогли уменьшить дисперсию, но появилась новая проблема. Для начала посмотрим по какому градиенту мы ходим в среднем: $\mathbb{E}_\xi[\nabla f(\theta^k, \xi^k) - \nabla f(\theta^{k-1}, \xi^k)] = \nabla f(\theta^k) - \nabla f(\theta^{k-1})$, то есть мы используем смещенную оценку на стохастический градиент. Как мы можем поменять математическое ожидание (сделать оценку не смещенную), но при этом не повлиять на дисперсию? Чисто математически это легко исправить, достаточно добавить константу равную $\nabla f(\theta^{k-1})$ на каждой итерации цикла и получим замену:

$$\nabla f(x^k, \xi^k) \to \nabla f(\theta^k, \xi^k) - \nabla f(\theta^{k-1}, \xi^k) + \nabla f(\theta^{k-1}). \tag{13}$$

Выше мы писали, что считать честный градиент каждую итерацию невозможно, а выбрали $\hat{\theta} = \theta^{k-1}$ ради сильной скоррелированности с θ^k. Если выбрать $\hat{\theta}$ чуть умнее, например обновлять его каждые S итераций, то полный градиент придется считать только каждые S итераций (что очень хорошо), но при этом увеличится дисперсия оценки (что достаточно плохо).

Отметим два особых случая:

- если $S = 0$, то получили стандартный градиентный спуск
- если $S = |\mathcal{D}|$, то в среднем число подсчетов градиента асимптотически не отличается от стохастического градиентного спуска, но при этом мы уменьшили дисперсию оценки градиента

Полученный метод называют SVRG [Johnson and Zhang(2013)] (Stochastic variance reduction gradient), который в виде псевдокода записать в алгоритме 1.

Algorithm 1 SVRG

Require: количество эпох K, период обновления S, $\alpha^{k,s}$, θ^0
  for $k = \overline{0, K-1}$ **do**
    Обновить $\hat{\theta}^k = \theta^{k-1,S}$
    Посчитать и сохранить $G^k = \nabla f(\hat{\theta}^k)$
    for $s = \overline{0, S-1}$ **do**
      Сгенерировать независимо ξ_s
      $g^{k,s} = \nabla f(\theta^{k,s}, \xi_s) - \nabla f(\hat{\theta}^k, \xi_s) + G^k$
      $\theta^{k,s+1} = \theta^{k,s} - \alpha^{k,s} g^{k,s}$
    end for
  end for

Одной из слабых сторон полученного метода является наличие вложенного цикла. На практике мы часто стремимся избавиться от вложенного цикла, особенного когда он нужен лишь для обновления $\hat{\theta}$ каждые K итераций. В данном алгоритме можно реализовать счетчик и решить проблему технически, а можно подумать над модификацией алгоритма и обновлять $\hat{\theta}$ с

[11] За одну итерацию мы не сильно приблизимся к минимуму, поэтому масштаб стохастических градиентов не поменяется, как и среднее направление, которое указывает в сторону минимума

некоторой вероятностью. После такого обновления получим алгоритм L-SVRG (Loopless SVRG) [Kovalev et al.(2019)Kovalev, Horváth, and Richtárik], итерация которого записана ниже.

$$g^k = \nabla f(\theta^k, \xi_k) - \nabla f(\hat{\theta}^k, \xi_k) + \nabla f(\hat{\theta}^k) \tag{14}$$

$$\theta^{k+1} = \theta^k - \alpha^k g^k \tag{15}$$

$$\hat{\theta}^{k+1} = \begin{cases} \theta^k, & \text{с вероятностью } p \\ \hat{\theta}^k, & \text{с вероятностью } 1-p \end{cases} \tag{16}$$

Прием со случайным обновлением часто встречается в алгоритмах сэмплирования, что в некотором смысле связывает подходы стохастической оптимизации и методы Монте-Карло для сэмплирования[12]. Бывают случаи, когда методы стохастической оптимизации служат хорошей мотивацией для методов сэмплирования [Zou et al.(2018)Zou, Xu, and Gu].

Результаты

Как вы уже заметили, методы оптимизации имеют строгую теорию и сложные, для кого-то, красивые теоремы. На практике, особенно в машинном обучении и нейронных сетях, минимизируются функции, для которых и близко нет строгой теории. Сегодня мы разобрали разные приемы, для каждого есть оценки сходимости и теоремы, и при этом мы всегда можем использовать эти приемы в комбинации и даже без доказанных теорем. Да, у нас не всегда будут гарантии, что наш алгоритм сходится, но всегда можно попробовать. Мы надеемся, что пример с алгоритмами ISTA и FISTA является хорошей демонстрацией того, как можно применять известные приемы, полученные из разных задач, к новым проблемам. Вы всегда можете использовать батчирование вместе с ускорением в SGD[13] или добавить к ускорению редукцию дисперсии. Чем больше базовых приемов вам известно, тем более творческим и тонким является выбор метода обучения.

Список литературы

[Поляк(1983)] Бю Т. Поляк. Введение в оптимизацию. Наука, 1983. https://cmcagu.ru/docs/books/Polyak-optimizationintro.pdf.

[Wikipedia contributors(2024a)] Wikipedia contributors. Exponential smoothing — Wikipedia, the free encyclopedia, 2024a. URL https://en.wikipedia.org/w/index.php?title=Exponential_smoothing&oldid=1194746887. [Online; accessed 13-March-2024].

[Нестеров(2010)] Ю.В. Нестеров. Введение в выпуклую оптимизацию. МЦНМО, 2010. https://old.mipt.ru/dcam/upload/abb/nesterovfinal-arpgzk47dcy.pdf.

[Bubeck(2014)] Sebastien Bubeck. Nesterov's accelerated gradient descent for smooth and strongly convex optimization, Mar 2014. URL https://web.archive.org/web/20210121055037/https://blogs.princeton.edu/imabandit/2014/03/06/nesterovs-accelerated-gradient-descent-for-smooth-and-strongly-convex-optimization/.

[Ahn and Sra(2022)] Kwangjun Ahn and Suvrit Sra. Understanding nesterov's acceleration via proximal point method, 2022.

[Su et al.(2015)Su, Boyd, and Candes] Weijie Su, Stephen Boyd, and Emmanuel J. Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights, 2015.

[12] Правда в методах сэмплирования встречаются более сложные условия [Wikipedia contributors(2024b)]

[13] Когда вы обучаете нейронную сеть с torch.optim.SGD(momentum > 0) то вы это и делаете

[Beck and Teboulle(2009)] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM Journal on Imaging Sciences, 2(1):183–202, 2009. doi: 10.1137/080716542. URL https://doi.org/10.1137/080716542.

[Johnson and Zhang(2013)] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/ac1dd209cbcc5e5d1c6e28598e8cbbe8-Paper.pdf.

[Kovalev et al.(2019)Kovalev, Horváth, and Richtárik] Dmitry Kovalev, Samuel Horváth, and Peter Richtárik. Don't jump through hoops and remove those loops: SVRG and katyusha are better without the outer loop. CoRR, abs/1901.08689, 2019. URL http://arxiv.org/abs/1901.08689.

[Wikipedia contributors(2024b)] Wikipedia contributors. Metropolis–hastings algorithm — Wikipedia, the free encyclopedia, 2024b. URL https://en.wikipedia.org/w/index.php?title=Metropolis%E2%80%93Hastings_algorithm&oldid=1206304394. [Online; accessed 13-March-2024].

[Zou et al.(2018)Zou, Xu, and Gu] Difan Zou, Pan Xu, and Quanquan Gu. Stochastic variance-reduced Hamilton Monte Carlo methods. In Jennifer Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 6028–6037. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/zou18a.html.