ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

УДК 519.86

Отчет об исследовательском проекте на тему:

Распределенные алгоритмы в условиях схожести слагаемых

(промежуточный, этап 1)

Выполнил студент:

группы #БПМИ212, 3 курса Панкевич Сергей Александрович

Принял руководитель проекта:

Двинских Дарина Михайловна
Старший научный сотрудник
Факультет компьютерных наук НИУ ВШЭ

Москва 2024

Содержание

Аннотация

Алгоритмы распределенной оптимизации становятся все более актуальными, так как современные модели машинного обучения зачастую тренируются на массивах данных, которые невозможно разместить на одном вычислительном устройстве в силу их большого размера или приватности. В данной работе рассматривается случай централизованной оптимизации, где один сервер хранит параметры модели, данные разделены между несколькими устройствами и сервером, управляющим процессом оптимизации. Целью работы является обобщение оценок полученных в случае сильно выпуклых функций на более широкий класс функций в условиях схожести слагаемых.

Ключевые слова

Методы оптимизации, распределенная оптимизация, централизованная оптимизация, условие Поляка-Лоясиевича, схожесть слагаемых

1 Введение

1.1 Описание предметной области

Распределенная оптимизация возникает в случае, когда функционал зависит от данных, размещенных на различных носителях. Например, в случае поиска параметров некой модели машинного обучения оптимизируется эмпирическая функция потерь. В централизованном случае распределенной оптимизации данные распределены между сервером и другими компьютерами. Происходит итерационный процесс в ходе которого сервер поручает другим компьютерам выполнять некоторые вычисления, они направляют результаты обратно серверу и сервер выполняет шаги оптимизационного процесса, используя данные, размещенные на нем и полученные значения от других компьютеров

1.2 Постановка задачи

В данной работе работе рассматривается централизованная оптимизация в случае, когда слагаемые, из которых состоит оптимизируемый функционал, в какой-то мере схожи между собой. Целью работы является доказательство оценки на сходимость уже известных алгоритмов в терминах условия Поляка-Лоясиевича и исследование возможностей улучшения алгоритмов с учетом ограничений на функционал.

2 Формальная постановка задачи

Скажем, что всего есть m компьютеров и n элементов обучающей выборки на каждом, $N = nm$ – суммарное количество примеров. Обозначим за $z_i^{(j)}$ – элемент обучающей выборки под номером i на машине номер j. Будем обозначать за $x \in \mathbb{R}^d$ параметры. Обозначим за $l(x, z)$ – функцию потерь на элементе z, при векторе параметров x. Кроме того введем следуюшие обозначения:

$$f_i(x) = \frac{1}{n} \sum_{j=1}^{n} l(x, z_i^j) \tag{1}$$

$$F(x) = \frac{1}{m} \sum_{i=1}^{m} f_i(x) \tag{2}$$

То есть $F(x)$ представляет из себя эмпирический риск на всем массиве данных, а f_i – на

компьютере i. Тогда нашей целью будет являться решение следующей задачи оптимизации

$$\min_{x \in \mathbb{R}^d} \quad F(x) + \psi(x) \tag{3}$$

Где $\psi(x)$ представляет из себя некоторую функцию регуляризации. Для начала положим $\psi(x) = 0$.

3 Обзор литературы

3.1 Условие сильной выпуклости

Функция $f : \mathbb{R}^d \to \mathbb{R}$ называется сильно выпуклой на множестве S, если существует такая положительная константа μ, что

$$\mu I \preceq \nabla^2 f(x), \quad \forall\, x \in S \tag{4}$$

Это условие часто выполняется для изучаемых нами просто выпуклых функций из-за добавления L_2 регуляризации. В статье (Hendrikx et al., 2020)[1] с его учетом дается оценка скорости сходимости, которую мы приведем после описания еще одного важного условия.

3.2 Условие схожести слагаемых

Скажем, что функции $f(x)$ и $g(x)$ являются δ-схожими на некотором множестве S, если выполнено следующее.

$$\left\|\nabla^2 f(x) - \nabla^2 g(x)\right\| \leq \delta, \quad \forall\, x \in S, \tag{5}$$

Рассматриваемая нами функция F имеет вид суммы нескольких слагаемых. Заметим, что если данные на каждом компьютере получены случайным выбором без возвращения n элементов из обучающей выборки, то f_i весьма вероятно похожи между собой и хорошо приближают функцию F. А именно, в (Troop J., 2015)[2] с помощью матричного неравенства Хеффдинга показано, что с вероятностью не меньше $1 - p$ выполнено:

$$\left\|\nabla^2 f_i(x) - \nabla^2 F(x)\right\| \leq \sqrt{\frac{32 A_l^2 \log(d/p)}{n}}, \tag{6}$$

Для некоторого числа A_l такого, что $A_l \geq \|\nabla^2 \ell(x, z_i)\|\ \forall\, x \in S$.

Использовать δ-схожесть для уменьшения числа итераций алгоритма в случае сильно выпуклых функций можно, если воспользоваться зеркальным спуском с дивергенцией Брэгмана $D_\phi(x, x_t)$, определяемой следующим образом

$$D_\phi(x, y) \triangleq \phi(x) - \phi(y) - \nabla\phi(y)^\top (x - y). \tag{7}$$

Скажем, что сервер имеет номер 1 среди всех машин, тогда введем следующую $\phi(x)$

$$\phi(x) = f_1(x) + \frac{\delta}{2}|x\|^2. \tag{8}$$

Теперь можно модифицировать предыдущий алгоритм.

Algorithm 1 Зеркальный спуск в условиях схожести слагаемых

receive (η, x_1, K) as input
for $t = 1, 2 \ldots, K$ **do**
  send x_t to all computers
  for $i = 1, 2 \ldots, m$ **do**
    receive x_t
    compute $\nabla f_i(x_t)$
    send $\nabla f_i(x_t)$
  end for
  receive $\nabla f_1(x_t), \nabla f_2(x_t), \ldots, \nabla f_m(x_t)$
  compute $\nabla F(x_k) = \frac{1}{m}\sum_{i=1}^m \nabla f_i(x_t)$
  $x_{t+1} = \text{argmin}_{x\in\mathbb{R}^d}\left\{\nabla F(x_t)^\top x + \frac{1}{\eta} D_\phi(x, x_t)\right\}$.
end for
return x_K

Для него выполняется оценка (Hendrikx et al., 2020)[1]:

$$D_\phi(x*, x_{k+1}) \leq \left(1 - \frac{\mu}{\mu + 2\delta}\right)^k D_\phi(x*, x_1) \tag{9}$$

Где $x^* = \text{argmin}\, F(x)$.

4 Формальная постановка задачи

$$r = \sum_{i=1}^{n} f_i$$

$$r = p + q$$

$$q = f_1$$

$$p = \frac{1}{n} \sum [f_i - f_1]$$

5 Методы решения задачи

5.1 Примитивы

Для начала определим два примитива, которыми будут пользоваться последующие алгоритмы

Algorithm 2 Подсчет $\nabla r(x)$

1: **Input**: $x \in \mathbb{R}^d$
2: send x to all computers
3: **for** $i = 1, 2 \ldots, n$ **do**
4: receive x
5: compute $\nabla f_i(x)$
6: send $\nabla f_i(x)$
7: **end for**
8: receive $\nabla f_1(x_t), \nabla f_2(x_t), \ldots, \nabla f_n(x_t)$
9: compute $\nabla r(x) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x_t)$
10: **return** $\nabla r(x)$

Приведем подробное описание принципа работы, в дальнейшем в аналогичном контексте используется такая же схема взаимодействия сервера и других компьютеров

Сервер получает на вход значение x. В строчке 2 он отсылает его всем компьютерам. После чего все компьютеры параллельно исполняют строчки 4, 5 и 6. В строчке 8 сервер получает резульаты вычислений всех компьютеров. В конце он вычисляет $\nabla r(x)$, используя полученные результаты, и возвращает результат.

Данное действие требует 1 раунд коммуникаций и n локальных вызовов оракула градиента.

Этот алгоритм использует схему вычислений, аналогичную предыдущей, и также требует 1 раунд коммуникаций и n локальных вызовов оракула градиента.

5.2 Централизованная версия градиентного спуска

Приведем пример работы градиентного спуска в условиях централизованной оптимизации.

Где $\eta = \frac{1}{L}$

Algorithm 3 Подсчет $\nabla q(x)$

send x to all computers
for $i = 2, 3 \ldots, n$ **do**
  receive x
  compute $\nabla f_i(x)$
  send $\nabla f_i(x)$
end for
receive $\nabla f_2(x_t), \nabla f_2(x_t), \ldots, \nabla f_n(x_t)$
compute $\nabla f_1(x)$
compute $\nabla q(x) = \frac{1}{n} \sum_{i=1}^{n} [\nabla f_i(x) - \nabla f_1(x)]$
return $\nabla q(x)$

Algorithm 4 Централизованный градиентный спуск

Input: $x^0 \in \mathbb{R}^d$
Parameters: η, K
for $t = 0, 1 \ldots, K-1$ **do**
  $x_{t+1} = x_t - \eta \nabla r(x_t)$
end for
return x_K

5.3 Централизованная версия ускоренного градиентного спуска

Algorithm 5 Централизованный ускоренный градиентный спуск

Input: $x^0 \in \mathbb{R}^d$
Parameters: $\eta, \kappa, \mu, L, \gamma$
$y_0 = x_0$
for $t = 0, 1 \ldots, K-1$ **do**
  $y_{t+1} = x_t - \eta \nabla r(x_t)$
  $x_{t+1} = (1+\gamma) y_{t+1} - \gamma y_t$
end for
return x_K

Где $\eta = \frac{1}{\eta}$, $\kappa = \frac{L}{\mu}$, $\gamma = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$

Данные алгоритмы эквивалентны градиентному и ускоренному градиентному спуску в классическом нераспределенном случае по производимым вычислениям. Мы будем их использовать в качестве базовых решений для сравнения эффективности по числу коммуникаций и числу вызовов оракула градиента f_i.

5.4 Accelerated Extragradient

В статье [???] приведен алгоритм, который является асимптотически оптимальным как по числу локальных вызовов оракула градиента, так и по числу коммуникаций.

Algorithm 6 Accelerated Extragradient

1: **Input**: $x^0 \in \mathbb{R}^d$
2: **Parameters**: K, η, θ, α
3: $x_f^0 = x^0$
4: **for** $t = 0, 1 \ldots, K-1$ **do**
5: $\quad x_g^k = \tau x^k + (1-\tau)x_f^k$
6: $\quad x_f^{k+1} \approx \operatorname{argmin}_{x \in \mathbb{R}^d} \left[A_\theta^k(x) \coloneqq p(x_g^k) + \langle \nabla p(x_g^k), x - x_g^k \rangle + \frac{1}{2\theta}\|x - x_g^k\|^2 + q(x) \right]$
7: $\quad x^{t+1} = x^t + \eta\alpha(x_f^{t+1} - x^t) - \eta \nabla r(x_f^{t+1})$
8: **end for**
9: **return** x^K

В данном случае в строчке 6 имеется в виду приближенное решение задачи, для которого соблюдается условие

$$\|\nabla A_\theta^k(x_f^{k+1})\|^2 \leq \frac{L_p^2}{3}\|x_g^k - \operatorname*{argmin}_{x \in \mathbb{R}^d} A_\theta^k(x)\|^2$$

5.4.1 Алгоритм решения

5.5 Modified Accelerated Extragradient

Заметим, что условие [???] непрактично, так как невозможно применять итерационный метод для поиска решения и прекращать итерации, как только оно стало выполнено. Это из-за того что в выражении справа необходимо знать $\operatorname{argmin}_{x \in \mathbb{R}^d} A_\theta^k(x)$. Поэтому в представленном в статье [???] алгоритме необходимо делать фиксированное число итераций оптимизационного процесса такое, что условие [] гарантированно выполнено.

В нашем методе будем искать решение такое, что выполняется другое условие:

$$\|\nabla A_\theta^k(x_f^{k+1})\|^2 \leq L_p^2\|x_g^k - x_f^{k+1}\|^2$$

Искать эту точку будем, минимизируя A_θ^k, с использованием ускоренного градиентного спуска, начиная из точки x_g.

Algorithm 7 Subproblem solver 2

1: **Input**: $x_g \in \mathbb{R}^d$
2: **Parameters**: $\eta, \kappa, \mu, L, \gamma$
3: $y_0 = x_0$
4: **for** $t = 0, 1 \ldots, K-1$ **do**
5: $\quad y_{t+1} = x_t - \eta \nabla r(x_t)$
6: $\quad x_{t+1} = (1+\gamma)y_{t+1} - \gamma y_t$
7: **end for**
8: **return** x_K

5.6

В качестве начального решения будем использовать централизованную верси

6 Теоретический анализ

6.1 Оценка на количество итераций для решения подзадачи

7 Условие схожести слагаемых

оценить Lq через дельту $\delta = O(1/\sqrt{m})$

8 Экспериментальное исследование метода

Будем рассматривать следующую задачу минимизации эмпирического риска (Ridge-регрессию)

$$F(w) = \frac{\lambda}{2}||w||_2^2 + \frac{1}{2N}\sum_{i=1}^{n}(w^t x_i - y_i)^2 \to_w \min$$

Где x_i – вектор параметров объекта i, w – параметры модели, λ – коэффициент регуляризации. Эквивалентно ее можно переписать в матричном виде следующим образом:

$$\frac{1}{2N}||Xw - y||_2^2 + \frac{\lambda}{2}||w||_2^2 \to_w \min$$

Где X – матрица высоты N и ширины d, в строке i которой находится x_i^t. Будем ее называть матрицей дизайна.

Будем делать два типа экспериментов: на синтетических данных и реальных. В обоих случаях распределенность будет симулироваться. Мы будем производить все вычисления на одном устройстве, но при соверешении действий, эквивалентных коммуникации сервера и компьютеров, будем инкрементировать счетчик числа коммуникаций. Аналогично при подсчете градиента функции, относящихся к слагаемым на одной из машин, будем инкрементировать счетчик числа локальных вызовов.

8.1 Эксперимент на синтетических данных

Будем рассматривать сеть из n машин, первая из которых является сервером. На каждой машине будут располагаться m примеров.

- Сгенерируем матрицу дизайна для сервера X_1, где $X_{1ij} \sim^{i.i.d} \mathcal{N}(0, L)$ с использованием **numpy**. Константа L позволяет регулировать константу липшица градиента функции потерь по параметрам. Чем больше L, тем в среднем выше константа липшицевости.
- После чего сгенерируем вектор w, $w_i \sim^{i.i.d} \mathcal{N}(0, 1)$.
- Теперь положим $y = X_1 w + \varepsilon$, где $\varepsilon_i \sim^{i.i.d} \mathcal{N}(0, 1)$

  Это соответствует классическим статистическим предположениям в задаче линейной регрессии.
- Сгенерируем матрицы дизайна других машин ($k \geq 2$) следующим образом $X_k = X_1 + \delta P_k$, где $P_{kij} \sim^{i.i.d} \mathcal{N}(0, 1)$. Константа δ позволяет регулировать схожесть данных на сервере и других машинах. Чем она меньше, тем в среднем ближе будут гессианы функций на разных машинах.

Частично для приближения к условиям статьи [] были выбраны следующие константы

- $\lambda = 0.1$
- $n = 25$
- $m = 100$
- $d = 50$
- $\delta = 0.1$
- $L = 2$

8.2 Эксперимент на реальных данных

В качестве реальных данных для эксперимента использовался набор данных, в котором собрана информация о погоде.

В качестве признаков выбирались столбцы

- “Wind Speed (km/h)’
- “Humidity’
- “Wind Bearing (degrees)’
- “Visibility (km)’

- “Loud Cover’
- “Pressure (millibars)’
- “Temperature (C)’

В качестве предсказываемой величины “Apparent Temperature (C)’.

Кроме того к признакам добавлен единичный свободный член. Признаки отнормированы так, что имеют нулевое среднее и единичную выборочную дисперсию. Объекты распределены между 25 машинами в случайном порядке.

8.3 Оценка констант δ, L_p, L_q, μ

Для работы некоторых методов необходимо знание констант гладкости и сильной выпуклости. Кроме того их знание полезно для отслеживания зависимостей качества различных методов от свойств данных.

$$f_i(w) = \frac{1}{m}||X_i w - y||_2^2 + \frac{\lambda}{2}||w||_2^2$$

$$\nabla_w f_i = \frac{1}{m} X_i^T (X_i w - y) + \lambda w$$

$$\nabla_w^2 f_i = \frac{1}{m} X_i^T X_i + \lambda I_d$$

8.3.1 Оценка L_q

$$q(w) = f_1(w)$$

$$\nabla_w^2 q = \frac{1}{m} X_i^T X_i + \lambda I_d$$

$$L_q = \frac{1}{m} \lambda_{\max}(X_i^T X_i) + \lambda$$

8.3.2 Оценка L_p

$$p(w) = \frac{1}{n} \sum_{i=1}^{n} [f_i(w) - f_1(w)]$$

$$\nabla_w^2 p = \frac{1}{n} \sum_{i=1}^{n} [\nabla_w^2 f_i - \nabla_w^2 f_1] =$$

$$= \frac{1}{n}\sum_{i=1}^{n}[\frac{1}{m}X_i^T X_i + \lambda I_d - \frac{1}{m}X_1^T X_1 - \lambda I_d] =$$

$$= \frac{1}{n}\sum_{i=1}^{n}[\frac{1}{m}X_i^T X_i] - \frac{1}{m}X_1^T X_1 =$$

$$L_p = \lambda_{\max}(\frac{1}{n}\sum_{i=1}^{n}[\frac{1}{m}X_i^T X_i] - \frac{1}{m}X_1^T X_1)$$

Где $\lambda_{\max}$ – наибольшее собственное значение матрицы, которое можно найти с помощью функции **eigvalsh** из пакета **scipy**, которая подходит для работы с симметричными вещественными матрицами.

8.3.3 Оценка δ

Отметим, что в нашем случае у всех функций гессиан является константным, поэтому

$$\delta = \max_i ||\nabla_w^2 f_1 - \nabla_w^2 f_i||_2 = \frac{1}{m}\max_i ||X_1^T X_1 - X_i^T X_i||$$

8.3.4 Оценка μ

Каждая функция $\frac{1}{m}||X_i w - y||_2^2$ является выпуклой, $\frac{\lambda}{2}||w||_2^2$ является λ-сильно выпуклой, поэтому каждая функция f_i и $r = \sum \frac{1}{n} f_i$ являются μ-сильно выпуклой, где $\mu = \lambda$.

8.4 Оценка сходимости методов

Для оценки сходимости каждого метода на итерации T будем вычислять во сколько раз квадрат расстояния до оптимума меньше, то есть

$$\frac{||w^T - w_*||_2^2}{||w^0 - w_*||},$$

где

$$w_* = \operatorname{argmin} \frac{1}{2N}||Xw - y||_2^2 + \frac{\lambda}{2}||w||_2^2$$

Во всех методах будем полагать $w_0 = 0$. Для задачи (todo) сущетсвует точное аналитическое решение:

$$w_* = (X^T X + N\lambda I_d)^{-1} X^T y,$$

которое мы будем использовать в экспериментах.

Отметим, что зачастую аналитическое решение непрактично из-за высокой временной сложности обращения матриц и вычислительной неустойчивости, поэтому на практике зачастую используются именно итерационные методы, подобные методам, описываемым в данной работе. Кроме того для широкого класса задач, не существует явного аналитического выражения решения, поэтому такие методы

Список литературы

[1] Hendrikx et al. “Statistically preconditioned accelerated gradient method for distributed optimization”. B: *International conference on machine learning* (2020).

[2] Troop J. “User-friendly tail bounds for sums of random matrices”. B: *Foundations of computational mathematic* (2012).