Po-op

project on optimization

Кубраков Н.С, Лемихов А.А
Суханова С.Я, Угнивенко В.А

1 Аннотация.

Статья "A Delay-tolerant Proximal-Gradient Algorithm for Distributed Learning" , посвященная асинхронному методу решения стандартной задачи крупномасштабного машинного обучения, вдохновила нас на проведение собственного независимого исследования, являющегося логическим продолжением предложенной нам публикации. На первом шаге мы провели сравнение гибкого алгоритма асинхронной оптимизации, описанного в работе, с алгоритмом синхронной оптимизации для решения негладкой задачи обучения. Дальнейшие изыскания посвещены ответу на вопрос: насколько хорошо или плохо работают (и работают ли вообще) асинхронные версии других алгоритмов по сравнению с их синхронными аналогами, причем были проведены испытания как медленных методов (градиентный спуск), так и быстрых (ускоренный метод Нестерова).

2 Постановки задач и алгоритмы их решения.

2.1 Стандартная задача крупномасштабного обучения.

2.1.1 Постановка задачи:

Мы рассматриваем общую задачу машинного обучения, имеющую вид следующей негладкой задачи оптимизации:

$$\min_{x \in R^n} \frac{1}{n} \sum_{i=1}^{n} l_i(x) + r(x),$$

где n - размер обучающего набора, l_i - гладкие выпуклые эмпирические функции потерь, а r - негладкий выпуклый регуляризатор. Этот шаблон моделирует широкий спектр проблем, возникающих в машинном обучении и обработке сигналов. Исследуем её частный случай:

$$\frac{1}{n} \cdot \sum_{j=1}^{n} \log(1 + \exp(-y_j z_j^T x))) + \lambda_1 \|x\|_1 + \frac{\lambda_2}{2} \|x\|_2^2 \ \to \min_x,$$

с гиперпараметром λ_2, принимающим классическое значение $\lambda_2 = \dfrac{1}{n}$. Мы используем общедоступный набор данных, представленный в Таблице 1, для получения констант y_j и z_j.

Dataset	n	d	L	λ_1
Covtype	581	54	21,9	10^6

Таблица 1: Характеристики наборов данных, используемых в наших экспериментах; n, d, L и λ_1 обозначают соответственно размер обучающего набора, число признаков, константу Липшица и значение гиперпараметра, соответствующего регуляризации l_1.

2.1.2 Асинхронный алгоритм решения:

Master :	Slave i :

Инициализируем $\overline{x} = \overline{x}^0,\ k = 0$
while не сошлись do
  when агент заканчивает итерацию:
  получаем приращение Δ от него
  $\overline{x} \leftarrow \overline{x} + \Delta$
  Отправляем в ответ $\overline{x}$
  $k \leftarrow k + 1$
end
Прерываем всех агентов
Output $x = \mathrm{prox}_{\gamma r}(\overline{x})$
	Инициализируем $x = x_i^0 = \overline{x}^0$
	while не прерван мастером do
	  Получаем последний $\overline{x}$
	  Берем x из предыдущей итерации
	  Выбираем число повторений p
	  Инициализируем $\Delta = 0$
	  for q = 1 to p do
	  $z \leftarrow \mathrm{prox}_{\gamma r}(\overline{x} + \Delta)$
	  $x^+ \leftarrow z - \gamma \frac{1}{n_i} \sum_{j \in S_i} \nabla l_j(z)$
	  $\Delta \leftarrow \Delta + \pi_i (x^+ - x)$
	  $x \leftarrow x^+$
	  end
	  Отправляем приращение Δ мастеру
	end

Введём обозначения, употребляемые в алгоритме:

$$\pi_i = \frac{n_i}{n}; \gamma = \frac{1}{L}$$

$$\mathrm{prox}_{\gamma r} = \arg\min_z \{ r(z) + \frac{1}{2\gamma} ||z - x||^2 \}$$

2.1.3 Синхронный алгоритм решения:

Master :	Slave i :

Инициализируем $\overline{x} = \overline{x}^0,\ k = 0$
while не сошлись do
  for i := 1 to i = cores do
    when агент заканчивает итерацию:
    получаем приращение от него
    $\overline{x} \leftarrow \overline{x} + \Delta$
  Отправляем в ответ $\overline{x}$ всем агентам
  $k \leftarrow k + 1$
end
Прерываем всех агентов
Output $x = \mathrm{prox}_{\gamma r}(\overline{x})$
	Инициализируем $x = x_i^0 = \overline{x}^0$
	while не прерван мастером do
	  Получаем последний $\overline{x}$
	  Берем x из предыдущей итерации
	  Выбираем число повторений p
	  Инициализируем $\Delta = 0$
	  for q = 1 to p do
	  $z \leftarrow \mathrm{prox}_{\gamma r}(\overline{x} + \Delta)$
	  $x^+ \leftarrow z - \gamma \frac{1}{n_i} \sum_{j \in S_i} \nabla l_j(z)$
	  $\Delta \leftarrow \Delta + \pi_i (x^+ - x)$
	  $x \leftarrow x^+$
	  end
	  Отправляем приращение Δ мастеру
	end

2.2 Задача квадратичного программирования.

2.2.1 Постановка задачи:

$$\frac{1}{2} ||Ax - b||_2^2 \ \rightarrow \min,$$

с достаточно обусловленой матрицей A большой размерности $n = 50$, обладающей явным диагональным преобладанием.

2.2.2 Алгоритм решения асинхронным градиентным спуском:

Master :	Slave i :

Инициализируем $\bar{x} = \bar{x}^0,\ k = 0$
while не сошлись do
  when агент заканчивает итерацию:
  получаем приращение от него
  $\bar{x} \leftarrow \bar{x} + \Delta$
  Отправляем в ответ $\bar{x}$
  $k \leftarrow k + 1$
end
Прерываем всех агентов
Output: $\bar{x}$
	Инициализируем $x = x_i^0 = \bar{x}^0$
	while не прерван мастером do
	  Получаем последний $\bar{x}$
	  Берем $x = \bar{x}$
	  Выбираем число повторений p
	  Инициализируем $\Delta = 0$
	  for $q = 1$ to p do
	  $\Delta \leftarrow \Delta + \mathrm{partgrad}_i(x)$
	  $x \leftarrow x + \mathrm{partgrad}_i(x)$
	  end
	  Отправляем приращение Δ мастеру
	end

Где partgrad$_i(x)$ - часть градиента целевой функции в точке x, вычисляемая i процессом. Фактически, компонентны обычного градиента с $\frac{(i-1)\cdot n}{\mathrm{cores}}$ по $\frac{i \cdot n}{\mathrm{cores}}$, а остальные позиции суть нули.

2.2.3 Алгоритм решения синхронным градиентным спуском:

Master :	Slave i :

Инициализируем $\bar{x} = \bar{x}^0,\ k = 0$
while не сошлись do
  for i := 1 to i = cores do
    when агент заканчивает итерацию:
    получаем приращение от него
    $\bar{x} \leftarrow \bar{x} + \Delta$
  Отправляем в ответ $\bar{x}$ всем агентам
  $k \leftarrow k + 1$
end
Прерываем всех агентов
Output: $\bar{x}$
	Инициализируем $x = x_i^0 = \bar{x}^0$
	while не прерван мастером do
	  Получаем последний $\bar{x}$
	  Берем $x = \bar{x}$
	  Выбираем число повторений p
	  Инициализируем $\Delta = 0$
	  for $q = 1$ to p do
	  $\Delta \leftarrow \Delta + \mathrm{partgrad}_i(x)$
	  $x \leftarrow x - \frac{1}{L}\mathrm{partgrad}_i(x)$
	  end
	  Отправляем приращение Δ мастеру
	end

2.2.4 Алгоритм решения синхронным методом Нестерова:

Master :	Slave i :

Инициализируем $\bar{x} = \bar{x}^0,\ k = 0$
while не сошлись do
  for i := 1 to i = cores do
    when агент заканчивает итерацию:
    получаем приращение от него
    $\bar{x} \leftarrow \bar{x} + \Delta$
  Отправляем в ответ $\bar{x}$ всем агентам
  $k \leftarrow k + 1$
end
Прерываем всех агентов
Output: $\bar{x}$
	Инициализируем $x = x_i^0 = \bar{x}^0$
	while не прерван мастером do
	  Получаем последний $\bar{x}$
	  Берем $y_k = \bar{x}$
	  Выбираем число повторений p
	  Инициализируем $\Delta = 0$
	  $\mathtt{for}\ q = 1\ \mathtt{to}\ p\ \mathtt{do}$
	    $x_{k+1} \leftarrow y_k - \frac{1}{L} \cdot \mathtt{partgrad}(y_k)$
	    $y_{k+1} \leftarrow x_{k+1} + \frac{k}{(k+3) \cdot \mathtt{cores}} \cdot (x_{k+1} - x_k)$
	    $\Delta \leftarrow \Delta y_{k+1} - y_k$
	    $y_k \leftarrow y_{k+1}$
	    $x_k \leftarrow x_{k+1}$
	  $k \leftarrow k + 1$
	  end
	  Отправляем приращение Δ мастеру
	end

Здесь $\mathtt{cores}$ - количество агентов.

2.2.5 Алгоритм решения асинхронным методом Нестерова:

Master :	Slave i :

Инициализируем $\bar{x} = \bar{x}^0,\ k = 0$
while не сошлись do
    when агент заканчивает итерацию:
    получаем приращение от него
    $\bar{x} \leftarrow \bar{x} + \Delta$
    Отправляем в ответ $\bar{x}$
  $k \leftarrow k + 1$
end
Прерываем всех агентов
Output: $\bar{x}$
	Инициализируем $x = x_i^0 = \bar{x}^0$
	while не прерван мастером do
	  Получаем последний $\bar{x}$
	  Берем $y_k = \bar{x}$
	  Выбираем число повторений p
	  Инициализируем $\Delta = 0$
	  $\mathtt{for}\ q = 1\ \mathtt{to}\ p\ \mathtt{do}$
	    $x_{k+1} \leftarrow y_k - \frac{1}{L} \cdot \mathtt{partgrad}(y_k)$
	    $y_{k+1} \leftarrow x_{k+1} + \frac{k}{(k+3) \cdot \mathtt{cores}} \cdot (x_{k+1} - x_k)$
	    $\Delta \leftarrow \Delta y_{k+1} - y_k$
	    $y_k \leftarrow y_{k+1}$
	    $x_k \leftarrow x_{k+1}$
	  $k \leftarrow k + 1$
	  end
	  Отправляем приращение Δ мастеру
	end

3 Численные эксперименты.

3.1 Стандартная задача крупномасштабного обучения.

Эксперимент проводился для описанной выше задачи 2.1, на части датасета Covtype ($\frac{1}{1000}$ от общего числа данных). Критерием останова алгоритма выбрали сходимость по аргументу с точностью $3 \cdot 10^{-5}$ к известному решению. За неимением возможности испытать метод на большом датасете и при участии большого числа машин было решено внести случайную задержку в пересылку сообщений между мастером и агентами, чтобы увеличить роль внутренних вычислений, по сравнению с вычислениями, осуществляемыми при непрерывном обмене сообщениями с мастером. Графики зависимости времени схождения у решению с заданной точностью от числа процессоров ($\mathtt{cores}$) и числа внутренних итераций внутри каждого агента (p) представлены на рисунках 1 и 2.

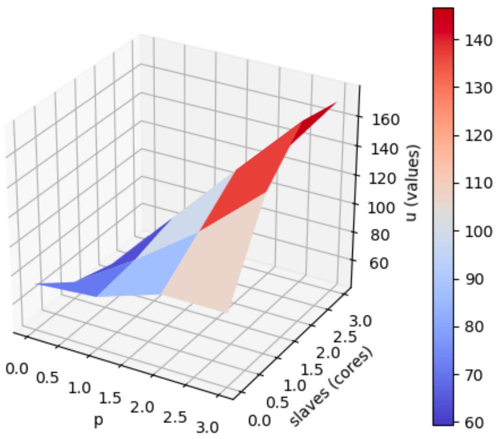

Рис. 1 Синхронный вариант.

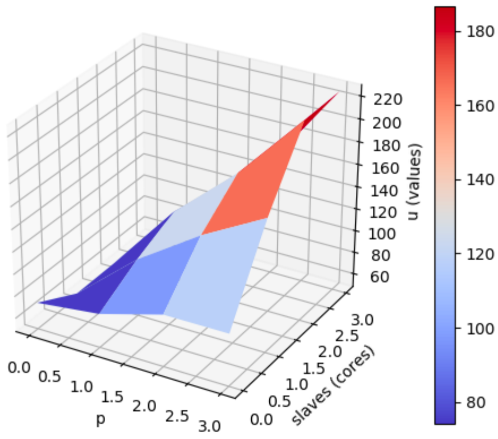

Рис. 2 Асинхронный вариант.

Первый важный вывод - асинхронный метод действительно сходится. За счет отсутсвия необходимости агентам ждать другие процессы, асинхронный метод, казалось бы, должен был выграть у синхронного, но факт того, что у агентов не самое актуальное значение x_k сказывается на скорости сходимости (количестве итераций) в отрицательную сторону и в итоге асинхронный метод проигрывает на 10%. Стоит отметить, что для обоих методов не наблюдается практически никакого ускорения от числа агентов, а даже наоборот заметно замедление. Все дело в специфике проводимого эксперимента. Если бы мы использовали набор данных целиком и в распоряжении у нас было бы больше 8 агентов, нам удалось бы так же увидеть положительную динамику при увеличении числа агентов. Этот же эксперимент в целом задавался вопросом, сходится ли асинхронный метод и если сходится, то насколько хорошо и ответ на вопрос был получен.

3.2 Задача квадратичного програмирования. Градиентный спуск.

Для проведения эксперимента заранее была получена матрица с диагональным приобладанием, а также такой вектор b, чтобы мы достоверно знали, какой вектор x является решением задачи. Стоит отметь принципиально отличную структуру графиков: поскольку время итерации много меньше чем в предыдущем методе, при больших числах агентов имеет место уменьшение времени с увеличением p.

Для синхронного метода постоен график зависимости времени исполнения от $cores$ и p на рис.3. На гафике видно что нет смысла использовать очень большие p, но уже небольшие значения позволяют справиться с задержками. Можно сделать вывод что метод очень устойчив, поскольку минимум на рис.3 очень протяженный. На рис.4 представленна более подробная структура той же зависимости при небольших p. Так же стоит отметить что эксперименты с ассинхронным методом очень нестабильны и имеются необусловленные пики на рис.6. Стоит отметить, что нет смысла применять ассинхонный метод к этой задаче и для этого метода, поскольку кроме получения нестабильности, происходит снижение скорости. Сравнение двух методов приведено на рис.5 при малых значения p.

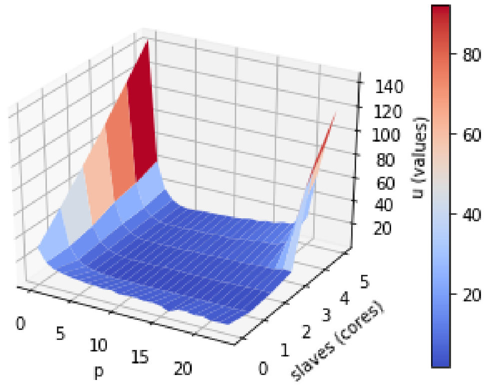

Рис. 3 Синхронный вариант на больших p.

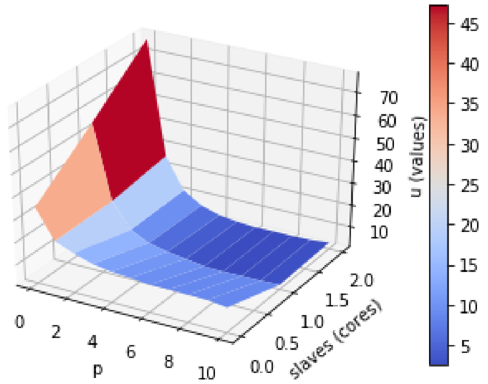

Рис. 4 Синхронный вариант на маленьких p.

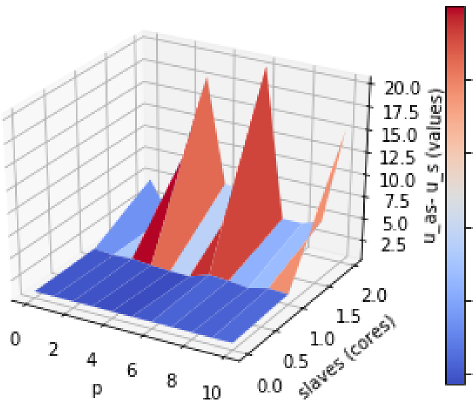

Рис. 5 Во сколько раз асинхронный метод медленнее синхронного.

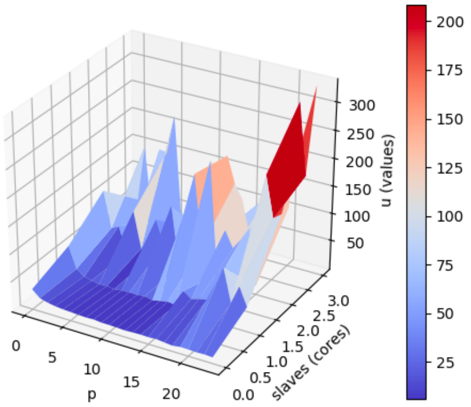

Рис. 6 Асинхронный метод на маленьких p.

3.3 Задача квадратичного програмирования. Ускоренный градиентный спуск.

Эксперимент проводился для той же задачи и матрицы из прошлого пункта. Еще до проведения эксперимента мы ожидали, что синхронный ускоренный метод будет работать, о том, будет ли вообще сходиться асинхронный ускоренный метод нам было не известно. Эксперимент показал, что этот метод действительно сходится, правда в разы медленнее синхронного. Зато сравнение времени схождения с одной и той же точностью этого метода и медленного асинхронного из прошлого пункта демонстрирует доминирование ускоренного метода в 5 и более раз.

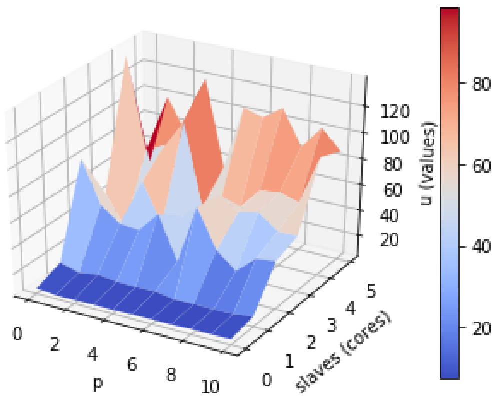

Рис. 7 Асинхронный метод.

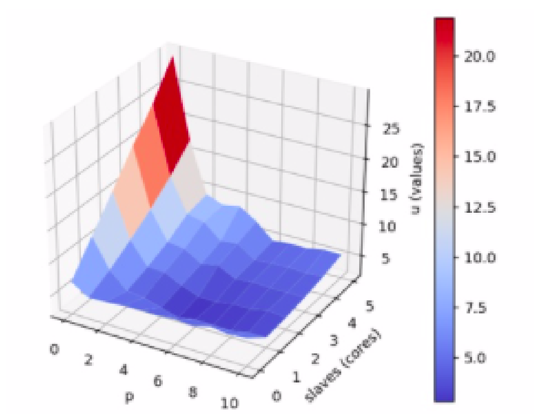

Рис. 8 Синхронный метод.

4 Заключение.

Подведем итоги. Было выяснено, что увеличение количества внутренних итераций положительно сказывается на времени работы метода, как обычного градиентного, так и и синхронного ускоренного. Асинхронный ускоренный метод не только сходится, но и делает это значительно быстрее обычного градиентного метода. Но для решаемых нами задач асинхронные методы использовать оказывается не резонно (сама задача не достаточно трудная). Были построены инструменты, позволяющие фактически проводить подобные испытания для любых методов при любой заранее заданной точности, для любых интервалов по количеству ядер и внутренних итераций.

Реализация алгоритмов - `https://github.com/lemikhovalex/Optimization-3.1v2`

5 Выполненная работа.

Члены команды/задачи:	Никита	Александр	Светлана	Виталий
Реализация метода из статьи	+	+		
Реализация градиентного метода		+	+	
Реализация метода Нестерова		+		+
Тест программ, получение экспериментальных данных	+	+		
Построение графиков			+	+
Анализ результатов	+		+	+
Контроль ресурсов и команды	+			
Электронное оформление проекта			+	+
Researcher	+	+	+	+

Список литературы

[1] Konstantin Mishchenko, Franck Iutzeler, Je ro^me Malic,3 Massih-Reza Amin. A Delay-tolerant Proximal-Gradient Algorithm for Distributed Learning.

[2] `https://nbviewer.jupyter.org/github/amkatrutsa/MIPT-Opt/blob/master/AccGrad/AccGrad.ipynb`

Материалы подготовлены для сдачи проекта по курсу "Методы оптимизации" .
Долгопрудный, Россия, 2019.