Методы оптимизации в машинном обучении, ВМК МГУ, весна 2020

Практическое задание 4: Композитная оптимизация

Язык программирования: Python 3.

1 Алгоритмы

1.1 Композитная оптимизация

Задача композитной минимизации представляет собой задачу минимизации специального вида:

$$\min_{x \in Q} f(x) + h(x),$$

где $h : Q \to \mathbb{R}$ — *простая* выпуклая замкнутая функция, определенная на множестве Q в $\mathbb{R}^n$, $f : E \to \mathbb{R}$ — дифференцируемая функция с липшицевым градиентом, определенная на открытом множестве E в $\mathbb{R}^n$, содержащем Q.

Под простотой функции h подразумевается то, что для любого $\lambda > 0$ возможно эффективно вычислить проксимальное отображение

$$\mathrm{Prox}_{\lambda h}(x) := \operatorname*{argmin}_{y \in Q} \left\{ \lambda h(y) + \frac{1}{2}\|y - x\|^2 \right\}$$

для любого $x \in \mathbb{R}^n$. Например, для функции $\|\|_1 : \mathbb{R}^n \to \mathbb{R}$ соответствующее проксимальное отображение может быть вычислено по формуле

$$\mathrm{Prox}_{\lambda\|\|_1}(x) = (\mathrm{sign}(x_i)[|x_i| - \lambda]_+)_{1 \le i \le n}$$

для $x \in \mathbb{R}^n$, $\lambda > 0$, где $[t]_+ := \max\{t, 0\}$ — положительная срезка для $t \in \mathbb{R}$.

1.2 Субградиентный метод

Субградиентный метод — это метод решения безусловной негладкой выпуклой задачи оптимизации

$$\min_{x \in Q} \phi(x),$$

где $\phi : Q \to \mathbb{R}$ — выпуклая функция с ограниченными субградиентами, определённая на выпуклом замкнутом множестве Q в пространстве $\mathbb{R}^n$. Заметим, что задача композитной оптимизации формально может рассматриваться как общая задача негладкой выпуклой оптимизации с функцией $\phi = f + h$.

Итерация субградиентного метода заключается в шаге из текущей точки x_k в направлении (произвольного) анти-субградиента $\phi'(x_k)$, затем выполняется проекция на множество Q. При этом, поскольку для негладких задач норма субградиента $\|\phi'(x_k)\|$ не является информативной, субградиентный метод использует в качестве направления нормированный вектор $\phi'(x_k)/\|\phi'(x_k)\|$:

$$x_{k+1} = P_Q\left(x_k - \alpha_k \frac{\phi'(x_k)}{\|\phi'(x_k)\|}\right).$$

Для сходимости метода необходимо, чтобы длины шагов α_k убывали к нулю, но не слишком быстро:

$$\alpha_k > 0, \quad \alpha_k \to 0, \quad \sum_{k=0}^{\infty} \alpha_k = \infty.$$

Обычно длины шагов выбирают по правилу $\alpha_k = \alpha/\sqrt{k+1}$, где $\alpha > 0$ — некоторая константа.

Нужно отметить, что последовательность $(x_k)_{k=0}^{\infty}$, построенная субградиентным методом, может не быть (и, как правило, зачастую не является) релаксационной последовательностью для функции ϕ, т. е. неравенство $\phi(x_{k+1}) \leq \phi(x_k)$ может не выполняться. Поэтому в субградиентном методе в качестве результата работы после N итераций метода вместо точки x_N возвращается точка $y_N := \operatorname{argmin}\{\phi(x) : x \in \{x_0, \ldots, x_N\}\}$ (т. е. из всех пробных точек x_k, построенных методом, выбирается та, в которой значение функции оказалось наименьшим).[1]

1.2.1 Градиентный метод

Одним из самых простых методов решения задачи композитной минимизации является градиентный метод. Приведем схему этого метода, в которой используется одномерный поиск для динамической регулировки оценки константы Липшица градиента:

Алгоритм 1 Градиентный метод с адаптивным подбором длины шага

Вход: Выпуклая замкнутая функция $h : Q \to \mathbb{R}$, определенная на множестве Q в $\mathbb{R}^n$, дифференцируемая функция $f : E \to \mathbb{R}$, определенная на открытом множестве E в $\mathbb{R}^n$, содержащем Q, начальная точка $x_0 \in Q$, начальная оценка константы Липшица $L_0 > 0$ для $\nabla f|_Q$.

$L \leftarrow L_0$
for $k = 0, 1, \ldots$ **do**
  while True do
    $y \leftarrow \operatorname{argmin}_{x \in \mathbb{R}^n}\{f(x_k) + \langle \nabla f(x_k), x - x_k\rangle + \frac{L}{2}\|x - x_k\|^2 + h(x)\}$
    if $f(y) \leq f(x_k) + \langle \nabla f(x_k), y - x_k\rangle + \frac{L}{2}\|y - x_k\|^2$ **then**
      break
    end if
    $L \leftarrow 2L$
  end while
  $x_{k+1} \leftarrow y$
  $L \leftarrow \max\{L_0, L/2\}$
end for

Выход: Последняя вычисленная точка x_k

Несмотря на то, что на отдельных шагах этой схемы может выполняться достаточно много итераций одномерного поиска по подбору параметра L, можно показать, что *среднее* число итераций одномерного поиска за итерацию метода примерно равно *двум*. Параметр L_0, по сути дела, влияет лишь на число одномерных поисков на самых первых итерациях метода; его всегда можно выбрать равным единице ($L_0 = 1$) без принципиального ущерба для скорости сходимости метода, поскольку в итоговую оценку суммарной трудоемкости метода этот параметр входит под логарифмом.

Заметим, что формула для y может быть переписана в эквивалентной форме с помощью проксимального отображения:

$$y = \operatorname{Prox}_{h/L}\left(x_k - \frac{1}{L}\nabla f(x_k)\right).$$

1.2.2 Быстрый градиентный метод

Используя *технику ускорения Нестерова*, градиентный метод можно ускорить:

[1] Заметим, что в практической реализации метода для вычисления результата y_N сами точки $x_0, \ldots, x_N$ хранить в памяти не нужно.

Алгоритм 2 Быстрый градиентный метод для композитной минимизации

Вход: Выпуклая замкнутая функция $h : Q \to \mathbb{R}$, определенная на множестве Q в $\mathbb{R}^n$, дифференцируемая выпуклая функция $f : E \to \mathbb{R}$, определенная на открытом множестве E в $\mathbb{R}^n$, содержащем Q, начальная точка $x_0 \in Q$, начальная оценка константа Липшица $L_0 > 0$ для $\nabla f|_Q$.

$A_0 \leftarrow 0$
$v_0 \leftarrow x_0$.
for $k = 0, 1, \ldots$ **do**
  while True do
    $a_k \leftarrow \frac{1+\sqrt{1+4LA_k}}{2L}$
    $A_{k+1} \leftarrow A_k + a_k$
    $y_k \leftarrow \frac{A_k x_k + a_k v_k}{A_{k+1}}$
    $v_{k+1} \leftarrow \operatorname{argmin}_{x \in Q} \{\frac{1}{2}\|x - x_0\|^2 + \sum_{i=0}^{k} a_i [f(y_i) + \langle \nabla f(y_i), x - y_i \rangle + h(x)]\}$
    $x_{k+1} \leftarrow \frac{A_k x_k + a_k v_{k+1}}{A_{k+1}}$
    if $f(x_{k+1}) \leq f(y_k) + \langle \nabla f(y_k), x_{k+1} - y_k \rangle + \frac{L}{2}\|x_{k+1} - y_k\|^2$ **then**
      break
    end if
    $L \leftarrow 2L$
  end while
  $L \leftarrow L/2$
end for

Выход: Точка (одна из всех просмотренных x_k или y_k) с наименьшим значением целевой функции ϕ

Обратите внимание, что для вычисления точки v_{k+1} не нужно каждый раз суммировать по всем $0 \leq i \leq k$; вместо этого достаточно обновлять в итерациях взвешенную сумму градиентов $\sum_{i=0}^{k} a_i \nabla f(y_i)$.

В отличие от обычного градиентного метода, быстрый градиентный метод работает уже с несколькими последовательностями точек. При этом вдоль каждой из этих последовательностей целевая функция может уменьшаться не монотонно. Поэтому в качестве ответа $\bar{x}_k$ метода следует выдавать ту точку, в которой наблюдалось наименьшее (так называемое *рекордное*) значение целевой функции среди всех точек x_k, y_k, в которых выполнялись вычисления функции.

2 Задача Lasso

Модель Lasso является одной из стандартных моделей линейной регрессии. Имеется обучающая выборка $((a_i, b_i))_{i=1}^m$, где $a_i \in \mathbb{R}^n$ — вектор признаков i-го объекта, а $b_i \in \mathbb{R}$ — его регрессионное значение. Задача заключается в прогнозировании регрессионного значения b_{new} для нового объекта, представленного своим вектором признаков a_{new}.

В модели Lasso, как и в любой модели линейной регрессии, прогнозирование выполняется с помощью линейной комбинации компонент вектора a с некоторыми фиксированными коэффициентами $x \in \mathbb{R}^n$:

$$b(a) := \langle a, x \rangle.$$

Коэффициенты x являются параметрами модели и настраиваются с помощью решения следующей оптимизационной задачи:

$$\phi(x) := \frac{1}{2}\sum_{i=1}^{m}(\langle a_i, x \rangle - b_i)^2 + \lambda \sum_{j=1}^{n} |x_j| =: \frac{1}{2}\|Ax - b\|^2 + \lambda \|x\|_1 \to \min_{x \in \mathbb{R}^n}. \tag{1}$$

Здесь $\lambda > 0$ — коэффициент регуляризации (параметр модели). Особенностью Lasso является использование именно l^1-регуляризации (а не, например, l^2-регуляризации). Такая регуляризация позволяет получить разреженное решение. В разреженном решении x^* часть компонент равна нулю. (Можно

показать, что при $\lambda \geq \|A^T b\|_\infty$ все компоненты будут нулевыми). Нулевые веса соответствуют исключению соответствующих признаков из модели (признание их неинформативными).

В этом задании все рассматриваемые методы должны использовать критерий остановки по зазору двойственности (который Вы уже реализовывали в предыдущем домашнем задании).

3 Формулировка задания

1 Возьмите коды, прилагаемые к заданию (файлы `oracles.py`, `optimization.py`, и `presubmit_tests.py`). Эти файлы содержат прототипы функций, которые Вам нужно будет реализовать. Некоторые процедуры уже частично или полностью реализованы.

2 Реализуйте негладкий оракул для функции (1) (классы `LeastSquaresOracle` и `LassoNonsmoothOracle` в модуле `oracles`).

3 Реализуйте субградиентный метод (функция `subgradient_method` в модуле `optimization`).

4 Реализуйте композитный оракул для функции (1) (классы `L1RegOracle` и `LassoProxOracle` в модуле `oracles`).

5 Реализуйте градиентный метод для композитной минимизации (функция `proximal_gradient_method` в модуле `optimization`).

6 Реализуйте быстрый градиентный метод для композитной минимизации (функция `proximal_fast_gradient_method` в модуле `optimization`).

7 Проведите эксперименты, описанные ниже. Напишите отчет.

3.1 Эксперимент: Выбор длины шага в субградиентном методе

Исследуйте работу субградиентного метода в зависимости от выбора константы α_0 в формуле для длины шага. При этом для одной и той же задачи рассмотрите различные начальные точки x_0. Есть ли связь между «наилучшим» коэффициентом α_0 и начальной точкой x_0?

3.2 Эксперимент: Среднее число итераций одномерного поиска в градиентных методах

Для градиентного метода и быстрого градиентного метода постройте график зависимости суммарного числа итераций одномерного поиска от номера итерации метода. Действительно ли среднее число итераций линейного поиска примерно равно двум в обоих методах?

3.3 Эксперимент: Сравнение методов

Сравните три реализованных метода на задаче Lasso. При этом рассмотрите различные значения размерности пространства n, размера выборки m и коэффициента регуляризации λ.

Для сравнения методов постройте графики 1) гарантируемая точность по зазору двойственности против числа итераций и 2) гарантированная точность по зазору двойственности против реального времени работы. Для гарантированной точности по зазору двойственности используйте логарифмическую шкалу.

Данные (матрицу A и вектор b) для задачи Lasso можно сгенерировать случайно, либо взять реальные данные с сайта LIBSVM.

4 Оформление задания

Результатом выполнения задания являются

1. Файлы `optimization.py` и `oracles.py` с реализованными методами и оракулами.
2. Полные исходные коды для проведения экспериментов и рисования всех графиков. Все результаты должны быть воспроизводимыми. Если вы используете случайность — зафиксируйте `seed`.
3. Отчет в формате PDF о проведенных исследованиях.

Каждый проведенный эксперимент следует оформить как отдельный раздел в PDF документе (название раздела — название соответствующего эксперимента). Для каждого эксперимента необходимо сначала написать его описание: какие функции оптимизируются, каким образом генерируются данные, какие методы и с какими параметрами используются. Далее должны быть представлены результаты соответствующего эксперимента — графики, таблицы и т. д. Наконец, после результатов эксперимента должны быть написаны ваши выводы — какая зависимость наблюдается и почему.

Важно: Отчет не должен содержать никакого кода. Каждый график должен быть прокомментирован — что на нем изображено, какие выводы можно сделать из этого эксперимента. Обязательно должны быть подписаны оси. Если на графике нарисовано несколько кривых, то должна быть легенда. Сами линии следует рисовать достаточно толстыми, чтобы они были хорошо видимыми.

5 Проверка задания

Перед отправкой задания обязательно убедитесь, что ваша реализация проходит автоматические *предварительные* тесты `presubmit_tests.py`, выданные вместе с заданием. Для этого запустите команду

```
nosetests3 presubmit_tests.py
```

Важно: Файл с тестами может измениться. Перед отправкой обязательно убедитесь, что ваша версия `presubmit_tests.py` — последняя.

Важно: Решения, которые не будут проходить тесты `presubmit_tests.py`, будут автоматически оценены в 0 баллов. Проверяющий не будет разбираться, почему ваш код не работает и читать ваш отчет.

Оценка за задание будет складываться из двух частей:

1. Правильность и эффективность реализованного кода.
2. Качество отчета.

Правильность и эффективность реализованного кода будет оцениваться автоматически с помощью независимых тестов (отличных от предварительных тестов). Качество отчета будет оцениваться проверяющим. При этом оценка может быть субъективной и аппеляции не подлежит.

За реализацию модификаций алгоритмов и хорошие дополнительные эксперименты могут быть начислены дополнительные баллы. Начисление этих баллов является субъективным и безапелляционным.

Важно: Практическое задание выполняется самостоятельно. Если вы получили ценные советы (по реализации или проведению экспериментов) от другого студента, то об этом должно быть явно написано в отчёте. В противном случае «похожие» решения считаются плагиатом и все задействованные студенты (в том числе те, у кого списали) будут сурово наказаны.