2021

Лекция 4. Оптимизация

МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ

ПАПУЛИН С.Ю. (papulin.study@yandex.ru)

Дополнительные материалы доступны по ссылке: https://github.com/mlmethods

Содержание

1. Оптимизация

Оптимизация – это минимизация или максимизация некоторой функции с учетом ограничений на её переменные:

$$x^* = \underset{x}{\operatorname{argmin}} f(x) \text{ при условии } \begin{matrix} c_i(x) = 0, i \in \xi \\ c_i(x) \geq 0, i \in \chi \end{matrix},$$

где x – вектор переменных (неизвестных или параметров); f – целевая функция; c_i – множество функций ограничения равенства и неравенства; ξ и χ – индексы для ограничений с равенством и неравенством, соответственно.

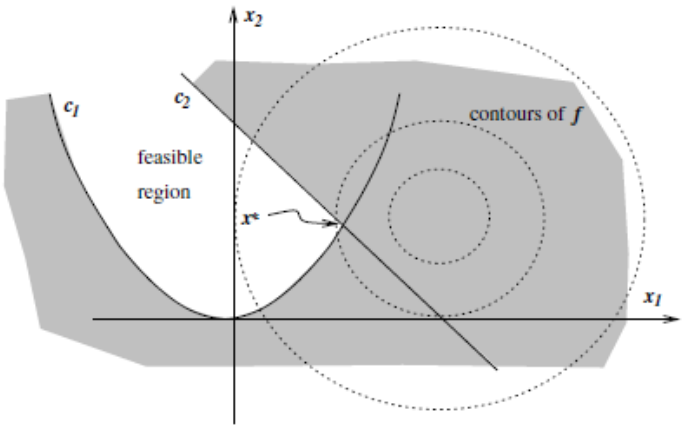


Рисунок – Геометрическое представление задачи оптимизации [1]

Дискретное и непрерывная оптимизация

Дискретная оптимизация оперирует целочисленными, бинарными или более абстрактными переменные. Например, есть задача по определению количества электростанций, которые необходимо построить в течение следующих 5 лет. Определяющей особенностью дискретной оптимизации является то, что x берется из конечного множества (но часто очень большого)

Задачи с непрерывной оптимизацией, как правило, проще решить, так как гладкость функций позволяет использовать значения целевой функции и ограничений в определённой точке x. Таким образом мы можем получить информацию о поведении функции во всех точках близкой к x.

При дискетных задачах, наоборот, поведение целевой функции и ограничений может меняться значительно, когда мы движемся от одной возможной точки к другой, даже если эти две точки близко расположены.

Условная и безусловная оптимизация

В задачах безусловной оптимизации отсутствуют ограничения. Кроме того, задача безусловной оптимизации может появиться за счет переформулирования задачи условной оптимизации. В этом случае ограничение заменяется на дополнительное слагаемое в целевой функции, которое выступает в качестве штрафа.

Задачи условной оптимизации возникает из моделей, в которых ограничения играют важную роль, например, бюджетные ограничения в экономике. Данные ограничения могут быть простыми границами для различных переменных (например, $0 \leq x_1 \leq 100$), линейными (например, $\sum_{i=1}^{n} x_i = 1$) или нелинейными неравенствами, которые представляют сложные отношения между переменными.

Виды задач оптимизации:

- Линейное программирование (linear programming): когда целевая функция и ограничения являются линейными по x
- Нелинейное программирование (nonlinear programming): когда по крайней мере одно ограничение или целевая функция является нелинейной функцией.

Локальная и глобальная оптимизация

Многие алгоритмы задач нелинейной оптимизации ищут только локальное решение, точку, в которой целевая функция меньше, чем во всех возможных рядом лежащих точках. Они не всегда находят глобальное решение, то есть точку, которая дает наименьшее значение целевой функции среди всех возможных точек.

Для задач выпуклого программирования (convex programming) и линейного программирования локальное решение также является и глобальным.

В общем случае нелинейные задачи (условные и безусловные) могут иметь локальное решение, которое не является глобальным.

Алгоритмы оптимизации

Алгоритмы оптимизации являются итеративными. Они начинаются с исходного предположения о значении x и генерируют последовательность улучшенных оценок (итерации) до остановки, в лучшем случае в точке решения. Подход, определяющий движение от одной итерации к другой, отличается в зависимости от алгоритма. Большинство подходов используют значения целевой функции и функций ограничения, первые и вторые производные этих функций, если возможно. Некоторые алгоритмы аккумулируют информацию от предыдущих итераций, другие используют только локальную информацию, полученную от текущей точки.

2. Безусловная оптимизация

В безусловной оптимизации мы минимизируем целевую функцию, которая зависит от действительных переменных, при отсутствии ограничений во всех значениях этих переменных:

$$x^* = \operatorname{argmin}_x f(x),$$

где $x^*, x \in \mathbb{R}^p$ – вектор действительных значений, $p \geq 1$; x^* – значение, при котором целевая функция имеет минимальное значение; $f: \mathbb{R}^p \to \mathbb{R}$ – гладкая функция (непрерывно дифференцируемая функция).

Обычно у нас нет глобальное представления о целевой функции. Всё, что мы знаем, это значения функции и возможно некоторые её производные на множестве точек $x_0, x_1, x_2, \ldots$.

Алгоритмы оптимизации используют данную информацию или её часть для определения надежного решения. Предпочтение отдается тем алгоритмам, которые используют меньше вычислительных ресурсов.

При использовании алгоритмов оптимизации наша цель – это найти точку, в которой целевая функция имеет минимально возможное значение, то есть глобальный минимум. Глобальным минимум, как правило, трудно найти, так как наши знания о функции локальны. Большинство алгоритмов способны отыскать локальный минимум, то есть точку, в окрестности которой функция принимает наименьшее значения.

Иногда у нас есть дополнительная информация о глобальном поведении функции, которая может помочь определить глобальный минимум. Важным частным случаем является выпуклые функции, для которых локальный минимум соответствует глобальному.

Как найти минимум целевой функции? Простым перебор возможных значений – задача недостижимо сложная. Если наша функция гладкая, то есть более эффективный способ найти локальный минимум. Когда целевая функция дважды непрерывно дифференцируема, мы можем сказать, что x^* есть точка локального минимума посредством градиента $\nabla f(x^*)$ и гессиана $\nabla^2 f(x^*)$. Для исследования поведения целевой функции используется теорема Тейлора.

Алгоритмы безусловной оптимизации

Все алгоритмы безусловной оптимизации требуют, чтобы была задана начальная точка x_0. Если есть исходные знания о характере целевой функции, то можно выбрать x_0, которая давала бы неплохую оценку решения. В противном случае начальная точка должна быть выбрана алгоритмом либо посредством особого подхода, либо случайным образом.

Начиная с x_0 алгоритмы оптимизации генерируют последовательность итераций $\{x_k\}_{k=1}^{\infty}$, которые останавливаются либо когда не происходит существенных улучшений, либо когда считаем, что точка решения аппроксимирована с достаточной точностью по заранее заданным критериям остановки.

Для того, чтобы принять решения куда двигаться от одной итерации x_k к другой, алгоритмы используют информацию о функции f в точке x_k и некоторые алгоритмы ещё и ранее полученные

значения $x_0, x_1, \ldots, x_{k-1}$. Они используют эту информацию, чтобы найти новую итерацию x_{k+1} с меньшим значением целевой функции в этой точке, чем x_k

Существует два основных подхода для перемещения из текущей точки x_k к новой итерации x_{k+1}:

- Линейный поиск
- Регион доверия

Линейный поиск

В линейном поиске алгоритм выбирает направление p и ищет вдоль этого направления, начиная с текущего положения x_k, новое значение итерации, которое бы давало меньшее значением целевой функции.

Расстояние вдоль выбранного направлением может быть приблизительно найдено за счет решения минимизации одномерной функции от α_k на каждой итерации:

$$\alpha_k^* = \underset{\alpha_k}{\operatorname{argmin}} f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)$$

Минимизация обозначенной функции требует дополнительных вычислений и в общем случае нет необходимости в точном определении минимального значения. Вместо этого алгоритмы линейного поиска генерируют ограниченное количество значений шага до тех пор, пока не будет получено удовлетворяющее приближение α_k^*. В полученной новой точке выбирается новое направление и новый шаг, и процесс повторяется.

Регион доверия

В алгоритмах с использованием региона доверия полученная информация о целевой функции применяется для построения модели функции m_k, чье поведение в окрестности текущей точки x_k похоже на поведение действительной целевой функции f. Так как модель m_k может давать плохую аппроксимацию f, когда x далеко от x_k, мы ограничивает поиск некоторым регионом вокруг x_k. Другими словами, мы решаем следующую задачу

$$\boldsymbol{p}_k^* = \underset{p_k}{\operatorname{argmin}} m_k(\boldsymbol{x}_k + \boldsymbol{p}_k),$$

где $\boldsymbol{x}_k + \boldsymbol{p}_k$ лежит в регионе доверия.

Если кандидат на решение не дает достаточного уменьшения для f, мы полагаем, что регион доверия слишком большой и уменьшаем его, заново решаем задачу минимизации. Как правило, регион доверия задается радиусом $\|p\|_2 \leq \Delta$ и $\Delta > 0$.

Модель m_k обычно задается квадратичной функцией

$$m_k(\boldsymbol{x}_k + \boldsymbol{p}_k) = f_k + \boldsymbol{p}_k^T \nabla f_k + \frac{1}{2} \boldsymbol{p}_k^T B_k \boldsymbol{p}_k,$$

где f_k – скалярное значение целевой функции в точке x_k; ∇f_k – вектор частных производных; B_k – гессиан $\nabla^2 f_k$ или его аппроксимация.

Таким образом, два подхода отличаются последовательностью того, как они выбирают направление и расстояние для движения. Линейный поиск начинается с фиксированного направления $\boldsymbol{p}_k$ и затем определяется подходящее расстояние, размер шага α_k. При использовании региона доверия, сначала выбираем максимальное расстояние (радиус региона доверия Δ_k) и затем ищем направление и шаг, которые дают наилучший результат с учетом ограничений по расстоянию. Если это шаг является неудовлетворительным, то уменьшаем Δ_k и пробуем снова.

Примеры методов, которые могут использовать как линейный поиска, так и регион доверия:

- Метод наискорейшего спуска (the steepest descent method)
- Метод Ньютона (Newtons method)
- Квазиньютоновские методы (quasi-Newton methods)

3. Линейный поиск (Line Search)

Общая форма

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \Delta_x,$$

где $\boldsymbol{x}$ – вектор $[x_1 \quad \cdots \quad x_p]^T$; Δ_x – некоторое смещение.

На каждой итерации метода линейного поиска вычисляется направление поиска $\boldsymbol{p}$ и затем определяется на сколько смещаться вдоль этого направления. Предыдущее выражение можно переписать в следующей форме

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \cdot \boldsymbol{p},$$

где α – некоторая скалярное значение, размер шага, $\alpha > 0$; $\boldsymbol{p}$ – вектор, определяющий направление.

Успешность метода линейного поиска зависит от правильного выбора как направления, так и размера шага.

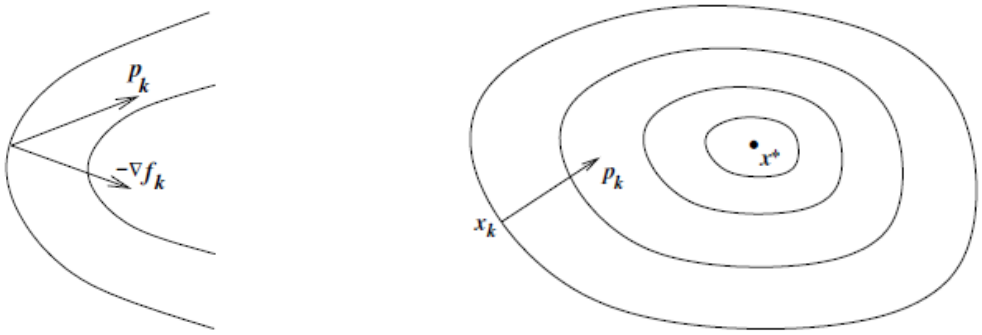


Рисунок – Направление спуска [1]: а) вектор направления поиска $\boldsymbol{p}$ и вектор обратного градиента $-\nabla f(\boldsymbol{x})$; б) спуск вдоль направления обратного градиента

Большинство алгоритмов линейного поиска используют направление спуска в качестве $\boldsymbol{p}$, то есть

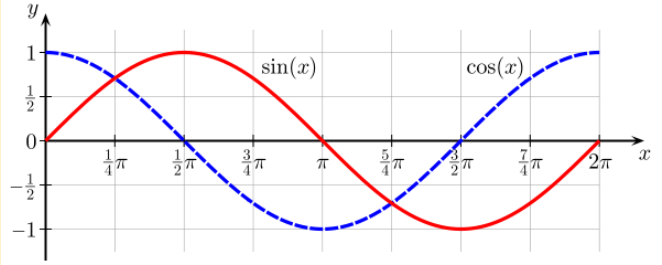
$$\boldsymbol{p}^T \nabla f(\boldsymbol{x}) < 0,$$

где

$$\nabla f = \left[\frac{\partial f}{\partial x_1} \quad \cdots \quad \frac{\partial f}{\partial x_p}\right]^T$$

Пояснение

$$a^T b = \|a\|\|b\| \cos\varphi$$



Данное свойство гарантирует, что целевая функция f может быть уменьшена вдоль этого направления. Неравенство говорит о том, что в общем случае направление спуска должно быть меньше $\pi/2$ по отношению к обратному направлению градиента.

3.1. Метод градиентного спуска / Метод наискорейшего спуска

Метод наискорейшего спуска – это метод линейного поиска с направлением $\boldsymbol{p} = -\nabla f(\boldsymbol{x})$ на каждом шаге:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \cdot \nabla f(\boldsymbol{x}).$$

Алгоритм 1. Градиентный спуск

1	initialize($\boldsymbol{x}$)	Начальная точка
2	while stopCriteria is False:	Количество итераций, проверка сходимости
3	$\alpha_k \leftarrow \alpha(k)$	Шаг
4	$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha_k \nabla f(\boldsymbol{x})$	Обновление текущего положения
5	return $\boldsymbol{x}$	

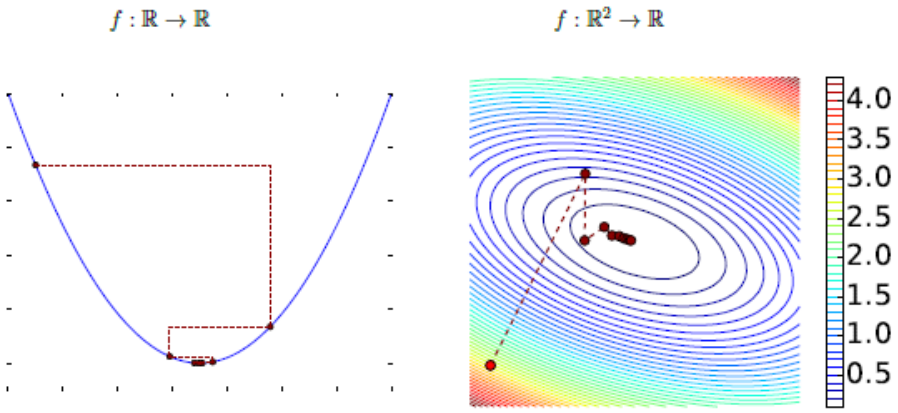


Рисунок – Пример градиентного спуска для целевой функции с одной и двумя переменными [3]

3.2. Выбор значения параметра шага

При расчете величины шага α_k мы стоим перед компромиссом: с одной стороны мы хотим выбрать α_k такое, чтобы получить существенное уменьшение f, и в то же время мы не хотим тратить слишком много времени на выбор α_k.

В идеале нам необходимо найти глобальный минимум функции

$$\phi(\alpha_k) = f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k),$$

где $\boldsymbol{p}_k$ – вектор, определяющий направление движения; $\alpha_k > 0$.

В общем случае эта задача очень затратная. Чтобы найти даже локальный минимум ϕ с приемлемой точностью, может потребоваться слишком много вычислений целевой функции f и возможно градиента ∇f.

Как правило, чтобы адекватно уменьшить целевую функцию минимальной ценой, используется «неточный» линейный поиск при определении размера шага. В этом случае алгоритмы последовательно перебирают различные значения α_k, и останавливаются, когда удовлетворены определенные условия.

Градиентный спуск с постоянным шагом

$$\alpha_k = c$$

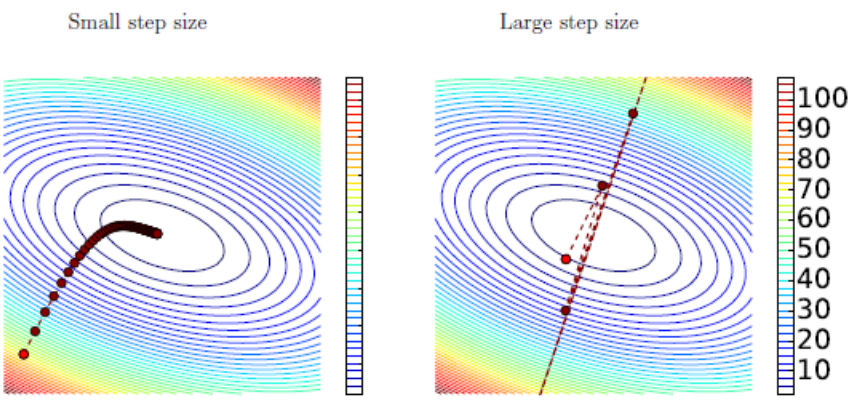


Рисунок – Пример градиентного спуска с разным размером шага [3]

Итеративно уменьшающийся шаг

$$\alpha_k = \frac{1}{k}$$

или

$$\alpha_k = \frac{1}{\sqrt{k}}$$

Минимизация целевой функции по направлению градиента

Для определения размера шага можно произвести ещё одну оптимизацию, но уже по

$$\alpha_k^* \leftarrow \underset{\alpha_k}{\operatorname{argmin}}\, h(\alpha_k) = \underset{\alpha_k}{\operatorname{argmin}}\, f\left(\boldsymbol{x}_k - \alpha_k \cdot \nabla f(\boldsymbol{x}_k)\right)$$

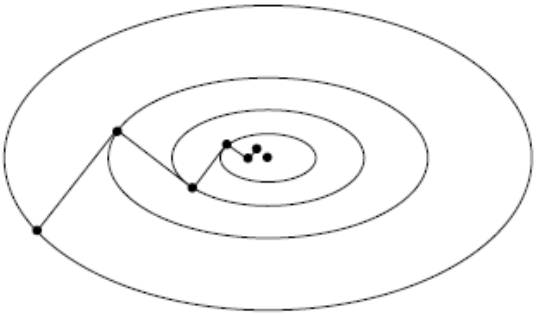


Рисунок – Шаг наискорейшего спуска [1]

$$\nabla_\alpha f\left(\boldsymbol{x}_k - \alpha_k \cdot \nabla f(\boldsymbol{x}_k)\right) = 0$$

$$\frac{d}{d\alpha_k} f\left(\boldsymbol{x}_k - \alpha_k \cdot \nabla f(\boldsymbol{x}_k)\right) = 0 \rightarrow \alpha_k$$

Пример

Исходная квадратичная функция:

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T Q\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$$

На k-ой итерации имеем

$$f\left(\boldsymbol{x}_k - \alpha_k \cdot \nabla f(\boldsymbol{x}_k)\right) = \frac{1}{2}\left(\boldsymbol{x}_k - \alpha_k \cdot \nabla f(\boldsymbol{x}_k)\right)^T Q\left(\boldsymbol{x}_k - \alpha_k \cdot \nabla f(\boldsymbol{x}_k)\right) - b^T\left(\boldsymbol{x}_k - \alpha_k \cdot \nabla f(\boldsymbol{x}_k)\right)$$

Продифференцируем по α_k и приравняем к нулю

$$\frac{d}{d\alpha_k} f\left(\boldsymbol{x}_k - \alpha_k \cdot \nabla f(\boldsymbol{x}_k)\right) = 0$$

В итоге получим шаг α_k на k-ой итерации равным

$$\alpha_k = \frac{\nabla f(\boldsymbol{x}_k)^T \nabla f(\boldsymbol{x}_k)}{\nabla f(\boldsymbol{x}_k)^T Q \nabla f(\boldsymbol{x}_k)}$$

Соответственно приращение будет

$$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \underbrace{\frac{\nabla f(\boldsymbol{x}_k)^T \nabla f(\boldsymbol{x}_k)}{\nabla f(\boldsymbol{x}_k)^T Q \nabla f(\boldsymbol{x}_k)}}_{\alpha_k} \nabla f(\boldsymbol{x}_k)$$

Обратный линейный поиск (Backtracking line search)

Популярное условие неточного линейного поиска (inexact linear search) гласит, что α_k должно прежде всего давать достаточное уменьшение целевой функции f, определяемое следующим неравенством:

$$\underbrace{f(x_k + \alpha \cdot p_k)}_{\phi(\alpha)} \leq \underbrace{f(x_k) + c\alpha\nabla f(x_k)^T p_k}_{l(\alpha)},$$

где c – некоторая константа, $c \in (0,1)$.

Другими словами, сокращение f должно быть пропорционально и размеру шага α и производной по направлению $\nabla f(x_k)^T p_k$.

Данное неравенство иногда называют правило Armijo.

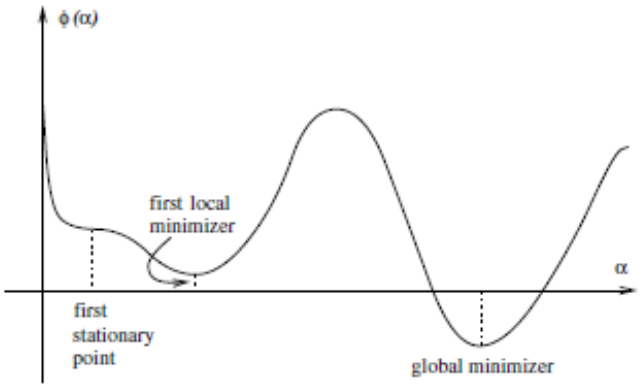


Рисунок – Пример функции вдоль выбранного направления [1]

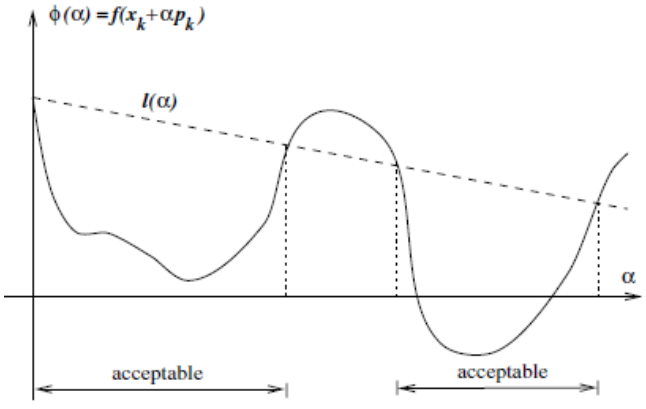


Рисунок – Достаточное условия уменьшения значения целевой функции [1]

Алгоритм 2. Обратный линейный поиск шага для k-ой итерации

1	$\text{choose}(\alpha_0 > 0, \rho \in (0,1), c \in (0,1))$	Выбор начальных значений
2	$\alpha \leftarrow \alpha_0$	
3	while $f(x_k + \alpha \cdot p_k) > f(x_k) + c\alpha \nabla f(x_k)^T p_k$:	Проверка условия
4	$\alpha \leftarrow \rho\alpha$	Обновление шага
5	return α	

В данном случае начальный размер шага α_0 выбирает равным 1 при использовании метода Ньютона и квазиньютоновских методов, но может иметь и другое значение, например, для наискорейшего спуска. Приемлемость размера шага будет определено после конечного числа попыток, так как α_k в конечном счете станет достаточно малым значением, при котором будет удовлетворено достаточное условие уменьшения целевой функции. На практике коэффициент сжатия (уменьшения) ρ может варьироваться на каждой итерации линейного поиска, указав, например, диапазон $0 < \rho_{low} < \rho_{high} < 1$.

Рассмотренный поход к остановке линейного поиска хорошо подходит для методов Ньютона, но менее подходит для квазиньютоновских методов и методам сопряженного градиента (conjugate gradient).

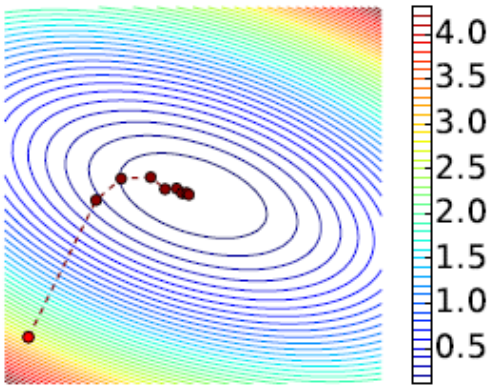


Рисунок – Пример градиентного спуска с использованием обратного линейного поиска [3]

3.3. Критерии остановки алгоритмов поиска минимального значения целевой функции

1. Количество итераций
2. Изменение по x

$$\left\|x^{(k+1)} - x^{(k)}\right\|_2 \leq \epsilon$$

3. Относительное изменение по x

$$\frac{\left\|x^{(k+1)} - x^{(k)}\right\|}{\left\|x^{(k)}\right\|} \leq \epsilon$$

4. Изменение целевой функции

$$\left\|f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right)\right\| \leq \epsilon$$

5. Градиента целевой функции

$$\left\|\nabla f\left(x^{(k)}\right)\right\| \leq \epsilon$$

3.4. Стохастический градиентный спуск

Стохастический градиентный спуск применяется для функций вида

$$f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x).$$

При большом количестве элементов исходного множества (обучающего множества) вычисление всего градиента f может быть вычислительно невыполнимым. Стохастический градиентный спуск позволяет обойти данную проблему, используя градиент отдельных компонент f_i.

Для правильной работы алгоритма наблюдения должны быть независимы и одинаково распределённые (i.i.d.), чтобы в среднем результат стремился в минимум. Кроме того, такого рода случайность дает больше шансов избежать «застревания» в локальный минимуме.

Также в качестве достоинства можно выделить то, что данный подход может быть использован при онлайн обучения, например, когда необходимо оптимизировать функцию, которая зависит от большого набора данных, но при этом в каждый момент времени есть доступ только к части данных.

Вариант 1. Градиент одного компонента

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f_i(\boldsymbol{x}),$$

где α – размер шага, $\alpha > 0$.

Алгоритм 3. Стохастический градиентный спуск. Вариант 1

1	initialize(x)	Начальное значение
2	while stopCriteria is False:	Количество эпох, проверка сходимости
3	shuffle(D)	Тасовка
4	for i in $1..m$:	
5	$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f_i(\boldsymbol{x})$	Обновление значения
6	return x	

Вариант 2. Мини-пакеты

Стохастический градиентный спуск с мини-пакетами по сравнению с предыдущим вариантом использует сумму из b градиентов компонент f_i, случайным образом выбранных из всего множества:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \cdot \sum_{i \in \mathcal{B}} \nabla f_i(\boldsymbol{x})$$

где $\mathcal{B}$ - множество индексов компонентов f размером b, такое что $b \ll m$.

Алгоритм 4. Стохастический градиентный спуск. Вариант 2

initialize(x)	Начальное значение
while stopCriteria is False:	Количество эпох, проверка сходимости
shuffle(D)	Тасовка (выборка без возврата)
batches $\leftarrow$ split(D, b)	Формирование наборов индексов компонент размером b
for $\mathcal{B}$ in batches:	
$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \cdot \sum_{i \in \mathcal{B}} \nabla f_i(\boldsymbol{x})$	Обновление значения
return x	

Если $\mathcal{B}$ сгенерировано таким образом, что каждый индекс имеет одинаковую вероятность p принадлежать данному множеству, то это несмещенная оценка ∇f:

$$\mathbb{E}\left[\sum_{i=1}^{m}(1_{i\in\mathcal{B}})\nabla f_i(x)\right]=\sum_{i=1}^{m}\mathbb{E}[(1_{i\in\mathcal{B}})\nabla f_i(x)]=\sum_{i=1}^{m}p(i\in\mathcal{B})\nabla f_i(x)=mp\nabla f(x)$$

Интуитивно, направление стохастического градиентного спуска в среднем соответствует направлению наискорейшего спуска. Однако вариация в оценке может привести к расхождению, если размер шага не уменьшается.

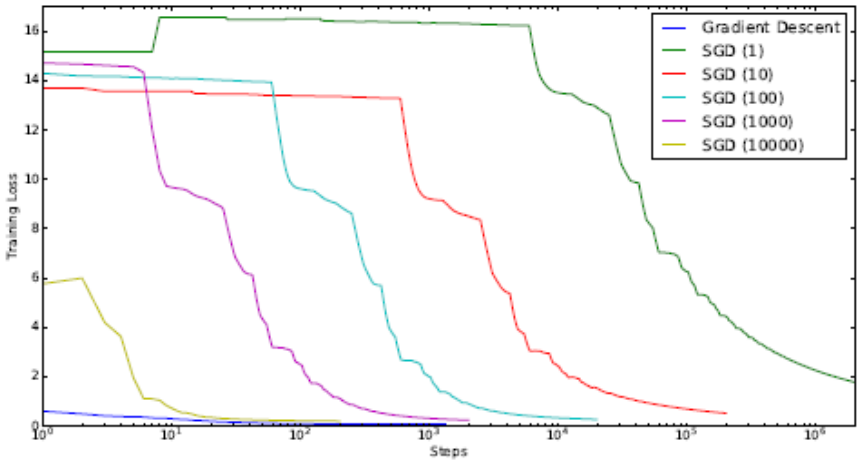


Figure 13: Convergence of stochastic gradient descent when fitting a logistic-regression model on the MNIST data set. The plot shows the value of the cost function on the training set for different batch sizes. Note that the updates for smaller batch sizes are much faster so the horizontal axis does not reflect running time.

Одним из преимуществ использования мини-пакетов это то, что можно использовать матричные операции для оптимизации процесса вычисления.

3.5. Метод Ньютона

Аппроксимация Тейлора второго порядка функции f в окрестности точки $\boldsymbol{x}$ есть

$$\hat{f}(\boldsymbol{x}+\boldsymbol{v})=f(\boldsymbol{x})+\nabla f(\boldsymbol{x})^T\boldsymbol{v}+\frac{1}{2}\boldsymbol{v}^T\nabla^2 f(\boldsymbol{x})\boldsymbol{v}$$

Аппроксимация Тейлора

Ряд Тейлора для дифференцируемой функции с одной вещественной переменной в окрестности точки x имеет вид

$$f(x+v)=f(x)+f'(x)v+f''(x)\frac{v^2}{2!}+\cdots+f^{(k)}(x)\frac{v^k}{k!}+\cdots$$

Если ограничиться некоторой степенью k, то получим полином, которой аппроксимирует функцию f в окрестности точки x. В частности, для малого значения v имеем

$$f(x+v)\approx f(x)+f'(x)v+f''(x)\frac{v^2}{2!}+\cdots+f^{(k)}(x)\frac{v^k}{k!}$$

Чем выше степень, тем точнее аппроксимация. Это называется аппроксимация Тейлора k-го порядка функции f в окрестности точки x.

Данная аппроксимирующая функция является выпуклой квадратичной по $\boldsymbol{v}$. Минимальное значение примет, когда градиент равен нулю:

$$\nabla_v \hat{f}(\boldsymbol{x}+\boldsymbol{v}) = \nabla f(\boldsymbol{x}) + \nabla^2 f(\boldsymbol{x})\boldsymbol{v} = \boldsymbol{0}$$

Получим, что смещение $\boldsymbol{v}$, при котором аппроксимация примет минимальное значение есть

$$\boldsymbol{v} = -\frac{\nabla f(\boldsymbol{x})}{\nabla^2 f(\boldsymbol{x})} = -\nabla^2 f(\boldsymbol{x})^{-1}\nabla f(\boldsymbol{x})$$

Запишем смещение как $\Delta x_{\mathrm{nt}} = \boldsymbol{v}$.

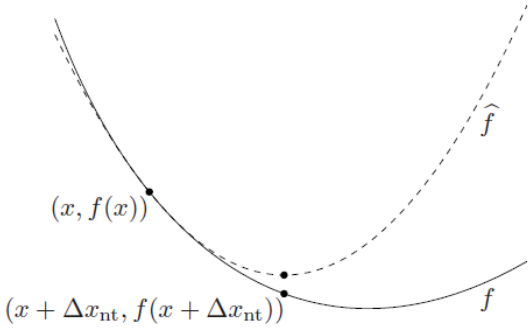


Рисунок – Функции $f(x)$ и её аппроксимация второго порядка $\hat{f}(x)$ [2]

Таким образом, шаг в методе Ньютона есть

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha\Delta x_{\mathrm{nt}},$$

где $0 < \alpha \leq 1$.

Или в полном виде

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha\nabla^2 f(\boldsymbol{x})^{-1}\nabla f(\boldsymbol{x})$$

Градиентный спуск и аппроксимация Тейлора

При поиске минимального значения нам необходимо двигаться таким образом, чтобы $f(\boldsymbol{x}+\boldsymbol{v}) < f(\boldsymbol{x})$. Аппроксимация Тейлора первого порядка функции f в окрестности точки x есть

$$f(\boldsymbol{x}+\boldsymbol{v}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T\boldsymbol{v}.$$

Чтобы получить максимальное улучшение, нам необходимо двигаться в обратном направлении градиента $\nabla f(\boldsymbol{x})$

$$f\big(\boldsymbol{x} - \alpha\nabla f(\boldsymbol{x})\big) = f(\boldsymbol{x}) - \alpha\nabla f(\boldsymbol{x})^T\nabla f(\boldsymbol{x}) < f(\boldsymbol{x}),$$

где

$$\nabla f(\boldsymbol{x})^T\nabla f(\boldsymbol{x}) > 0$$

4. Градиентный спуск для линейной регрессии

Задачи оптимизации для линейной регрессии:

$$\widehat{\boldsymbol{\theta}} = \text{argmin}_{\theta} L(\boldsymbol{\theta}),$$

где $L(\boldsymbol{\theta})$ – функция потерь:

$$L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\|\boldsymbol{y} - X\boldsymbol{\theta}\|_2^2 = \frac{1}{n}(\boldsymbol{y} - X\boldsymbol{\theta})^T(\boldsymbol{y} - X\boldsymbol{\theta}) = \frac{1}{n}(\boldsymbol{\theta}^T X^T X - 2\boldsymbol{\theta}X^T\boldsymbol{y} + \boldsymbol{y}^T)$$

$$\boldsymbol{\theta}, \boldsymbol{x}_i \in \mathbb{R}^p$$

$$y \in \mathbb{R}$$

Обучающее множество:

$$D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$$

Градиент $L(\boldsymbol{\theta})$:

$$\nabla L(\boldsymbol{\theta}) = \frac{1}{n}(2X^T X\boldsymbol{\theta} - 2X^T\boldsymbol{y})$$

Формула градиентного спуска (в векторном виде)

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \nabla L(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + 2\alpha\frac{1}{n}X^T(\boldsymbol{y} - X\boldsymbol{\theta}) = \boldsymbol{\theta} + 2\alpha\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta}\right)\boldsymbol{x}_i$$

Обновление параметра θ_j (для одного элемента вектора $\boldsymbol{\theta}$):

$$\theta_j \leftarrow \theta_j + 2\alpha\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta}_j\right)x_{ij}$$

Алгоритм 5. Градиентный спуск для линейной регрессии

1	initialize(θ)	Начальная позиция
2	while stopCriteria is False:	Количество итераций, проверка сходимости
3	$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + 2\alpha\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta}\right)\boldsymbol{x}_i$	Обновление текущего положения
4	return $\widehat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$	

5. Стохастический градиентный спуск для линейной регрессии

Функция потерь:

$$L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} l_i(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta})^2$$

Градиент компоненты $l_i(\boldsymbol{\theta})$:

$$\nabla l_i(\boldsymbol{\theta}) = -2 \cdot \left(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta}\right)\boldsymbol{x}_i$$

Обновление значений параметров в стохастическом градиентном спуске

Вариант 1.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla l_i(\boldsymbol{\theta}) = \boldsymbol{\theta} + 2\alpha\left(y_i - \boldsymbol{x}_i^T \boldsymbol{\theta}^{(k)}\right)\boldsymbol{x}_i$$

Вариант 2.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \sum_{i \in \mathcal{B}} \nabla l_i(\boldsymbol{\theta}) = \boldsymbol{\theta} + 2\alpha \sum_{i \in \mathcal{B}} \left(y_i - \boldsymbol{x}_i^T \boldsymbol{\theta}\right)\boldsymbol{x}_i$$

Алгоритм 6. Стохастический градиентный спуск для линейной регрессии

initialize(θ)	Начальное значение
while stopCriteria is False:	Количество эпох, проверка сходимости
shuffle(D)	Тасовка (выборка без возврата)
batches $\leftarrow$ split(D, b)	Формирование наборов индексов компонент размером b
for $\mathcal{B}$ in batches:	
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + 2\alpha \sum_{i \in \mathcal{B}} \left(y_i - \boldsymbol{x}_i^T \boldsymbol{\theta}\right)\boldsymbol{x}_i$	Обновление значения
return $\widehat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$	

Масштабирование признаков

При использовании рассмотренных алгоритмов важным является масштабирование признаков. В случае с разным масштабом схождение может занять значительно больше времени. Поэтому, как правило, признаки стандартизую перед запуском алгоритма. Ниже приведен пример обозначенной проблемы:

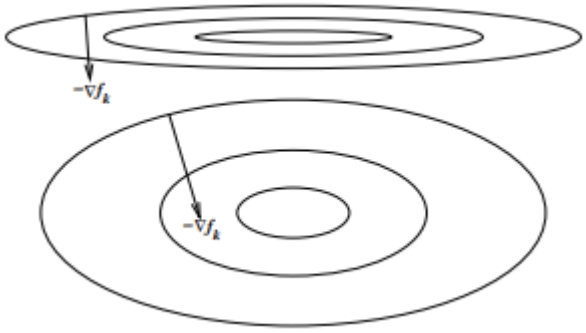


Рисунок – Масштабирование аргументов целевой функции

Сравнение методов оценки параметров линейной регрессии

Метод	Большое количество наблюдений	Большое количество признаков*	Количество параметров	Необходимость масштабирования признаков
МНК	+	–	0	–
МНК (SVD)	+	–	0	–
ГС	–	+	2	+
Стохастический ГС	+	+	≥2	+
Стохастический ГС (мини-пакеты)	+	+	≥2	+

* – более 100000

Список литературы

1. Nocedal, Jorge and Wright, Stephen J.. Numerical optimization. 2. ed. New York, NY: Springer, 2006.
2. Boyd, Stephen and Vandenberghe, Lieven. Convex Optimization. : Cambridge University Press, 2004.
3. Lecture Notes // Optimization-based Data Analysis. URL: https://math.nyu.edu/~cfgranda/pages/OBDA_spring16/material/optimization_algorithms.pdf
4. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron