

# 1 Введение.

Что вообще такое оптимизация? Поиск некоторого оптимального значения. Обычно это задачи на тему максимизации/минимизации значения некоторой функции (и поиска аргумента, при котором оно достигается).

Встречается это дело практически везде. В машинном обучении мы ищем параметр модели, когда она лучшим образом что-то предсказывает. Или мы хотим минимизировать невязку в системе линейных уравнений. Или мы проектируем какое-то устройство, и нам нужно сделать так, чтобы оно работало оптимально. Отсюда есть много слабо связанных методов и разделов, которые созданы для разных задач.

О'кей, вот хотим мы найти

$$\operatorname{argmin}_{x \in X} f(x)$$

Тут бывают разные  $X$  и разные  $f$ . Самый большой раздел — когда  $X$  это  $\mathbb{R}^n$ . Тут существует много разных задач (тот же поиск коэффициентов нейросети) и много методов. Методы делятся на 0, 1 и 2 порядка в зависимости от гладкости  $f$ .

0. Нулевого порядка (differentiation-free) не опираются на гладкость никак и не используют производную (например, покоординатный спуск, симплекс-метод). При этом тут всё равно обычно полагают непрерывность  $f$ , иначе жить совсем грустно.
1. Методы первого порядка используют градиент. Самый известный метод — градиентный спуск, когда мы идём в направлении антиградиента. И отсюда метод стохастического градиентного спуска (когда мы не вычисляем градиент точно) и некоторые другие вариации. И вдобавок бывают методы сопряжённых градиентов, сопряжённых направлений, сопряжённых невязок. Многие, кстати, требуют от функции ещё каких-то условий.
2. Методы 2 порядка полагают, что функция дифференцируется два раза и либо явно вычисляют вторые производные (тогда это какие-то ньютоновские методы), либо используют просто их наличие, а явно не вычисляют (квазиньютоновские методы).

Вдобавок к этому есть какие-нибудь методы, основанные на случайности (метод Монте-Карло, например). И есть ещё методы, которые в одномерном случае работают (и многие многомерные методы имеют одномерные методы как подзадачу). Что у нас есть в одномерном случае?

- Дихотомия. Если предполагается, что функция имеет один минимум, то модно пилить отрезок пополам и искать. Или не пополам, а в отношении золотого сечения или ещё как-то.
- Полиномиальные методы. Можно считать, что наша функция близка к многочлену и искать минимумы этого многочлена.

Всё вышеперечисленное можно сочетать: сначала построить приближение многочленом, если там какой-то кринж, то воспользоваться другим методом.

Ещё стоит заметить, что у нас есть локальные минимумы, и иногда нам хватит локального, а не глобального. И методы у нас обычно ищут локальный метод, потому что искать глобальный минимум сложно и очень немногие методы могут это сделать (и ещё и от функции чего-то требуют).

Ещё есть отдельный подкласс задач, когда  $X = \mathbb{N}$  или  $X = \mathbb{Z}$  или большое конечное множество или вообще какие-то не-числа (обычно дискретной природы). Такого сорта задачи называются задачами целочисленной оптимизации. Конкретно с целыми числами — задачами целочисленного программирования. Тут методы совершенно особые, и мы их в этом курсе затрагивать не будем.

Бывают более сложные варианты, когда у нас  $X = \mathbb{R} \times \mathbb{N}$ , например. Это задачи смешанного программирования, и с ними вообще всё плохо жить.

И ещё есть задачи, когда  $X$  — это некоторое подмножество  $\mathbb{R}^n$ . Такие задачи — задачи с ограничениями. К ним, кстати, обычно применяется другая нотация:

$$\begin{aligned} \operatorname{argmin}_{x \in \mathbb{R}^n} f(x) \\ c_i(x) = 0 \\ c_j(x) \geq 0 \end{aligned}$$

То есть обычно ограничения — это равенство или неравенство. Это тоже достаточно большой и сложный раздел со своими методами. Иногда задача сводится к задаче без ограничений, иногда приходится решать как есть своими особыми методами. Например, есть линейное программирование, когда у нас

$$f(x) = c_0 + c_1x_1 + c_2x_2 \cdots + c_nx_n \quad A_1x = b_1, A_2x \geq b_2$$

Это много где применимая задача, к ней много что сводится. А решается она, например, симплекс-методом.

Примером задачи, которая сводится к задаче линейного программирования, — транспортная задача. У нас есть какие-то фабрики и пункты продажи товара. Возникает вопрос с какой фабрики в каком количестве что куда вести. Каждая фабрика производит фиксированное число в единицу времени, у каждого магазина есть спрос (и мы не хотим создавать дефицит) и есть стоимость доставки на единицу товара из каждой фабрики в каждый магазин.

Ещё есть задачи квадратичного программирования, когда

$$f(x) = x^T Ax \quad A > 0$$

Положительная определённость матрицы хороша тем, что минимум у этой функции будет единственным.

Ещё выпуклое программирование бывает, когда  $f$  выпуклая функция и следовательно имеет единственный минимум.

## 2 Решение задачи без ограничений.

Тут есть два класса методов — линейный поиск и доверительные регионы. Что интересно, доверительные регионы показывают себя лучше (но и пишутся сложнее).

- Линейный поиск каким-то берёт точку и исходя из каких-то факторов (тот же антиградиент, например) идём от этой точки в определённом направлении. На какой шаг идём — тоже зависит от метода. Например, это может быть какой-то константный шаг (как в обычном градиентном спуске). Но это может быть плохо. При слишком большом шаге мы сильно проиграем, а если и не так, то, может, нам придётся долго ждать. Кстати, да, обычно стоит цель не только найти минимум, но и сделать это быстро. «Быстро», кстати, измеряется обычно не в секундах, а в количестве раз, сколько мы вычислим функцию и/или её градиент.
- В случае доверительного региона мы строим некоторую модель для нашей функции, ищем минимум модели, уточняет модель, повторяем. Что такое модель? Какая-то функция. Самая простая — квадратичная (в некоторой окрестности локального минимума функция достаточно близка к квадратичной). При этом, чем лучше наша функция соответствует модели, тем больше мы можем расширить область для следующей модели.

В доверительные регионы также входит задача нахождения гиперпараметров. У нас есть какие-то параметры модели (шаг спуска, например). И тут внутри нашей задачи оптимизации возникает ещё одна задача оптимизации, и тут всё совсем грустно, потому что не хочется очень много раз вычислять функцию. И тут обычно берут несколько комбинаций параметров и либо равномерно раскидывают по пространству параметров, либо случайно (второе лучше). Но ещё есть байесовский метод, когда мы итеративно уточняем модель по схеме, схожей с методом доверительных регионов.

Ну и небольшой офф-топ с просто интересным алгоритмом (генетическим алгоритмом), который выглядит так: берём несколько случайных точек (интерпретируем их как популяцию), выбираем несколько лучше, особым образом скрещиваем их, добавляем немного случайности (мутации) и так создаём новое поколение. И этот алгоритм тоже довольно хорошо работает для довольно многих задач (и, кстати, не требует дифференцирования). Более того, он может работать даже не на числах, а хоть даже на строках.