# Методы оптимизации

## 1. Оптимизация негладких функций

### Проблема оптимизации параметров алгоритма машинного обучения

На самом деле, использовать методы оптимизации функций, требующие существование градиента, получается не всегда. Даже если градиент у интересующей функции существует, часто оказывается, что вычислять его непрактично.

Например, существует проблема оптимизации параметров $\alpha_1, ..., \alpha_N$ некоторого алгоритма машинного обучения. Задача оптимизации состоит в том, чтобы подобрать эти параметры так, чтобы алгоритм давал наилучший результат. В частности, если качество работы алгоритма описывать функцией качества $Q(\alpha_1, ..., \alpha_N)$ от его параметров, задача оптимизации принимает вид:

$$Q(\alpha_1, ..., \alpha_N) \to \max_{\alpha_1,...,\alpha_N}.$$

Вычисление градиента в этом случае или невозможно в принципе, или крайне непрактично.

### Проблема локальных минимумов

Другая проблема градиентных методов — проблема локальных минимумов.

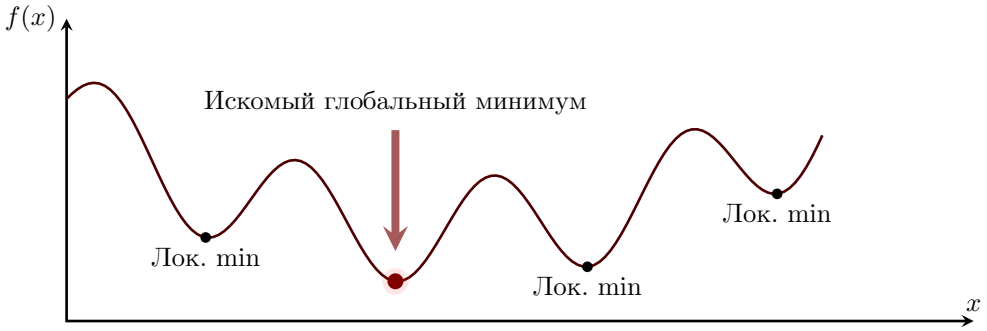


Рис. 1: К вопросу о проблеме локальных минимумов

Метод градиентного спуска, попав на дно локального минимума, где градиент также равен нулю, там и остается. Глобальный минимум так и остается не найденным. Решить эти проблемы позволяют методы случайного поиска. Общая идея этих методов заключается в намеренном введении элемента случайности.

### Пример: градиентный спуск без градиента

Если градиент по каким-либо причинам вычислить нельзя, задачу оптимизации можно попытаться решить с помощью модифицированного стохастического алгоритма градиентного спуска.

Пусть решается задача оптимизации $f(\vec{x}) \to \min_{\vec{x}}$, зафиксировано некоторое число $d$ — параметр метода. Используется следующий итерационный процесс:

1. Случайным образом выбирается вектор $\vec{u}$ (случайный вектор $\vec{u}$ равномерно распределен по сфере).
2. Вычисляется значение выражения, которое есть ни что иное, как численная оценка значения производной функции $f$ по направлению $\vec{u}$:
$$\frac{f(\vec{x}) - f(\vec{x} + d\vec{u})}{d}.$$
3. Сдвигаем точку в направлении $\vec{u}$ пропорционально вычисленной на предыдущем шаге величине.

Следует отметить, что ни на каком шаге градиент функции не вычисляется. Более того, направление смещения $\vec{u}$ выбирается случайным образом. Но так как величина смещения зависит от выражения функции в точке $\vec{x} + d\vec{u}$, в среднем происходит смещение по антиградиенту сглаженной функции. Причем, зафиксированное в начале число $d$, как раз имеет смысл «параметра сглаживания» при нахождении численной оценки производной.

## 2. Метод имитации отжига

Метод имитации отжига является одним из методов глобальной оптимизации, для работы которого не требуется гладкость функции. Является вариантом метода случайного поиска и известен как алгоритм Метрополиса.

Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Отжиг — вид термической обработки металлов, сплавов, заключающийся в нагреве до определённой температуры и последующем медленном охлаждении до комнатной температуры. Цель отжига — привести систему, которой является образец металла, в состояние с минимальной энергией. Атомы при отжиге могут как попасть в состояние с меньшей энергией, так и в состояние с большей. Причем вероятность попасть из текущего состояния в состоянии с большей энергией уменьшается с температурой. Постепенно температура уменьшается до комнатной и система попадает в состояние с минимальной энергией.

Для задач оптимизации имитация процесса может быть произведена следующим образом. Вводится параметр $T$, который имеет смысл температуры, и в начальный момент ему устанавливается значение $T_0$. Набор переменных, по которым происходит оптимизация, будет обозначаться как $x$.

В качестве начального состояния системы выбирается произвольная точка. Далее запускается итерационный процесс — на каждом шаге из множества соседних состояний случайно выбирается новое $x^*$. Если значение функции в этой точке меньше, чем значение в текущей точке, то эта точка выбирается в качестве нового состояния системы. В ином случае (т.е. если $f(x^*) > f(x)$) такой переход происходит с вероятностью $P$, зависящей от температуры $T$, текущего состояния $x$ и кандидата на новое состояние $x^*$ следующим образом:

$$P = e^{-\frac{f(x^*)-f(x)}{T}}.$$

Благодаря переходам в худшее состояние в методе имитации отжига удалось решить проблему локальных минимумов и не застревать в них. Постепенно, при уменьшении температуры, уменьшается и вероятность переходов в состояния с большим значением функции. Таким образом в конце имитации отжига в качестве $x$ оказывается искомый глобальный минимум.

Оказывается, что успешность этого алгоритма зависит от способа выбора кандидатов, другими словами, того, как именно происходит такой случайный выбор. Такие вопросы как, находит ли этот алгоритм минимум или насколько он эффективен для поиска состояния с меньшим значением функции $f$ (в случае если минимум не может быть найден), представляют отдельный интерес. Стоит также отметить, что практическую ценность могут иметь и алгоритмы, не гарантирующие достижения минимума.

Реализация метода имитации отжига доступна как функция `anneal` (англ. отжиг) в модуле `optimize` библиотеки SciPy.

## 3. Генетические алгоритмы

Генетические алгоритмы моделируют процесс естественного отбора в ходе эволюции и являются еще одним семейством методов оптимизации. Генетические алгоритмы включают в себя стадии генерации популяции, мутаций, скрещивания и отбора. Порядок стадий завит от конкретного алгоритма.

### Алгоритм дифференциальной эволюции

Для оптимизации функции $f(\vec{x})$ вещественных переменных $\vec{x} \in \mathbb{R}^n$ применяется алгоритм дифференциальной эволюции. Популяцией в алгоритме дифференциальной эволюции считается множество векторов из $\mathbb{R}^n$, причем каждая переменная этого пространства соответствует своему признаку. Параметрами этого алгоритма являются: размер популяции $N$, сила мутации $F \in [0, 2]$ и вероятность мутации $P$.

В качестве начальной популяции выбирается набор из $N$ случайных векторов. На каждой следующей итерации алгоритм генерирует новое поколение векторов, комбинируя векторы предыдущего поколения. А именно: для каждого вектора $x_i$ из текущего поколения:

**Стадия мутации** Случайно из популяции выбираются не равные $x_i$ векторы $v_1$, $v_2$, $v_3$. На основе этих векторов генерируется так называемый мутантный вектор:

$$v = v_1 + F \cdot (v_2 - v_3).$$

**Стадия скрещивания** Над мутантным вектором выполняется операция «скрещивания», в ходе которой каждая координата с вероятность $p$ замещается соответствующей координатой вектора $x_i$. Получившийся вектор называется пробным.

**Стадия отбора** Если пробный вектор оказывается лучше исходного $x_i$ (то есть значение исследуемой функции на пробном векторе меньше, чем на исходном), то в новом поколении он занимает его место.

Если сходимость не была достигнута, начинается новая итерация.

Следует также учитывать следующие замечания:

- Для решения задач оптимизации функций дискретных переменных достаточно переопределить только стадию мутации. Остальные шаги алгоритма остаются без изменений.
- Часто, чтобы увеличить эффективность алгоритма, создаются несколько независимых популяций. Для каждой такой популяции формируется свой начальный набор случайных векторов. В таком случае появляется возможность использовать генетические алгоритмы для решения задач глобальной оптимизации.
- Эффективность метода зависит от выбора операторов мутации и скрещивания для каждого конкретного типа задач.

Реализация метода доступна как функция `differentialevolution` в модуле `optimize` библиотеки SciPy.

# 4. Метод Нелдера-Мида

Метод Нелдера-Мида, или метод деформируемого многогранника, применяется для нахождения решения задачи оптимизации вещественных функций многих переменных

$$f(x) \to \min, \qquad x \in \mathbb{R}^n,$$

причем функция $f(x)$, вообще говоря, не является гладкой и может быть зашумленной. Другой особенностью метода является то, что на каждой итерации вычисляется значение функции $f(x)$ не более чем в трех точках. Это особенно важно в случае сложно вычислимой функции $f(x)$. Метод Нелдера-Мида является методом оптимизации по умолчанию в функции SciPy.optimize.minimize, прост в реализации и полезен на практике, но, с другой стороны, для него не существует теории сходимости — алгоритм может расходиться даже на гладких функциях.

Выпуклым множеством называется такое множество, содержащее вместе с любыми двумя точками соединяющий их отрезок. Выпуклой оболочкой множества называется его минимальное выпуклое надмножество. Симплексом (n-симплексом) называется выпуклая оболочка множества аффинно независимых $(n+1)$ точек (вершин симплекса).



Рис. 2: Отрезок (1-симплекс)

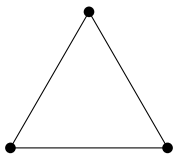

Рис. 3: Треугольник (2-симплекс)

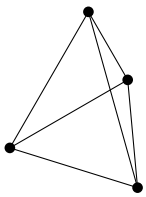

Рис. 4: Тетраэдр (3-симплекс)

## Описание алгоритма метода Нелдера-Мида

Основными параметрами метода являются $\alpha > 0$, $\beta > 0$ и $\gamma > 0$ — коэффициенты отражения, сжатия и растяжения соответственно. Пусть необходимо найти безусловный минимум функции $n$ переменных.

**1. (Подготовка)** Выбирается $n+1$ точка, образующие симплекс $n$-мерного пространства:

$$x_i = \left(x_i^{(1)}, x_i^{(2)}, ..., x_i^{(n)}\right), \qquad i = 1, ..., n+1.$$

В этих точках вычисляется значение функции $f(x)$:

$$f_1 = f(x_1), f_2 = f(x_2), ..., f_{n+1} = f(x_{n+1}).$$

**2. (Сортировка)** Среди вершин симплекса $\{x_i\}$ выбираются: точка $x_h$ с наибольшим (из значений на вершинах симплекса) значением функции $f_h$, точка $x_g$ со следующим по величине значением $f_g$ и точка $x_l$ с наименьшим значением функции $f_l$.

**3. (Центр тяжести)** Вычисляется центр тяжести всех точек, за исключением $x_h$ (вычислять значение функции в найденной точке не обязательно):

$$x_c = \frac{1}{n}\sum_{i \neq h} x_i.$$

**4. («Отражение»)** Точка $x_h$ отражается относительно точки $x_c$ с коэффициентом $\alpha$. В получившейся точке $x_r$ вычисляется значение функции:

$$x_r = (1+\alpha)x_c - \alpha x_h \qquad f_r = f(x_r).$$

**5. (Ветвление)** Этот шаг зависит от значения $f_r$ в сравнении с $f_l < f_g < f_h$.

1. Если $f_r < f_l$, то направление вероятно было выбрано удачное. Можно сделать попытку увеличить шаг. Производится растяжение, находится новая точка и значение функции в ней:

   $$x_e = (1-\gamma)x_c + \gamma x_r, \qquad f_e = f(x_e).$$

   Если $f_e < f_r$, то точке $x_h$ присваивается значение $x_e$, а иначе — значение $x_r$. После этого итерация заканчивается.
2. Если $f_l < f_r < f_g$, то выбор точки неплохой: точке $x_h$ присваивается значение $x_r$. После этого итерация заканчивается.
3. Если $f_g < f_r < f_h$, то точки $x_r$ и $x_h$ меняются местами (значения $f_r$ и $f_h$ тоже). После перейти на следующий шаг.
4. Если $f_h < f_r$, просто перейти на следующий шаг.

**5. («Сжатие»)** Строится точка $x_s$ и вычисляется значение функции в ней:

$$x_s = \beta x_h + (1-\beta)x_c, \qquad f_s = f(x_s).$$

**6. (Ветвление)** Если $f_s < f_h$, то точке $x_h$ присваивается значение $x_s$. После этого итерация заканчивается.

**7. («Глобальное сжатие»)** Иначе ($f_s > f_h$) имеет место ситуация, когда первоначальные точки оказались самыми удачными. Делается преобразование (сжатия к точке $x_i$):

$$x_i \leftarrow x_l + (x_i - x_l)/2, i \neq l.$$

**8. («Проверка сходимости»)** Последний шаг — проверка сходимости. Может выполняться по-разному. Если нужная точность не достигнута перейти к пункту 2.