Неточные алгоритмы

22/04/2023

План лекции

Стандартная проблема

Есть задача оптимизации функции $f(\theta)$ - надо найти $argmin f(\theta)$, такое θ, что $f(\theta)$ минимально.

Мы не можем перебрать все возможные θ, и не имеем достаточно информации, чтобы оставить "перебираемое" множество.

θ может быть одним целым числом, перестановкой, маской, вектором вещественных чисел, графом, строкой - чем угодно.

Виды неточных алгоритмов

- С гарантиями (*approximation algorithms*)
- Без гарантий (*heuristics*)

Пример известной нам эвристики: $\alpha\beta$-отсечение

ϵ-приближения ответа

Для NP-полных задач очень часто пытаются придумать эвристики, которые будут получать *какой-то ответ*, не очень уж и далекий от верного.

Например, можно показать, что разные жадные алгоритмы для задачи о рюкзаке могут давать суммарную стоимость, всего в 1.5 раза ниже оптимальной.

MAX-3-SAT

- Возьмем $x_1 = x_2 = \ldots = 1$
- Возьмем $x_1 = x_2 = \ldots = 0$
- Хотя бы одно из решений выполнит половину скобок.

Существует и более умное $\frac{7}{8}$ приближение

Обмен скорости на точность

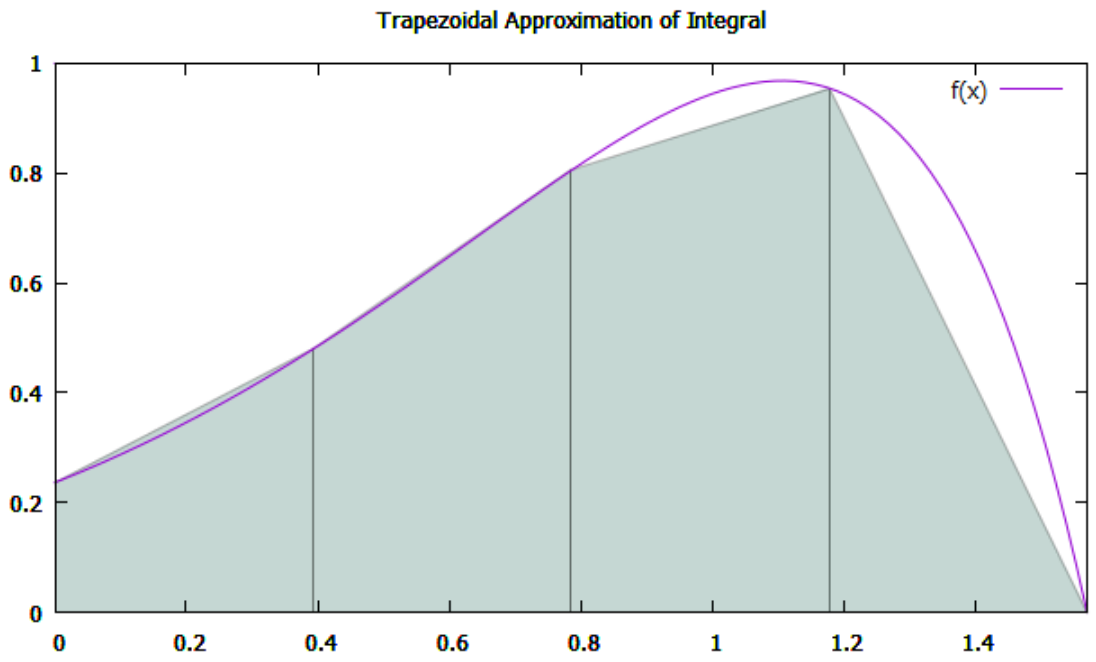
Для задачи о рюкзаке существует алгоритм с точностью $\left(1+\frac{1}{k}\right)$ и асимптотикой $O\left(kn^{k+1}\right)$. Таким образом, можно делать *размен* скорости на точность.

Численное интегрирование

Хотим уметь посчитать площадь какой-то фигуры.

Интегрирование - разбиение на тривиальные фигуры и подсчет приблизительной площади.

Мы знаем, что в пределе получается хороший ответ.



Метод Симпсона

Для интегрирования есть несколько сравнительно простых методов

- Метод прямоугольников
- Метод трапеций
- Метод Симпсона (приближение параболой)

Алгоритм

```
def calc(f, l, r, eps):
    if r - l < eps:
        return (r - l) / 6 * (f(l) + f(r) + 4 * f((r+l)/2))
    m = (l + r) / 2
    return calc(f, l, m, eps) + calc(f, m, r, eps)
```

Вопрос: Как обобщить на 2d (хотя бы метод прямоугольников)?

Метод Монте-Карло

- Пусть фигура жутко сложная, и вообще ничего непонятно
- Сделаем случайный классификатор: будем генерировать элемент пространства (точку) и проверять, что она находится в заданном множестве.
- Тогда, если мы знаем размер пространства, мы можем оценить, какую долю этого пространства занимало множество.

Эвристики

Стандартная история с эвристиками: взять какое-то базовое θ, а потом постепенно улучшать, чтобы в какой-то момент получить θ_{opt}.

Локальная оптимизация

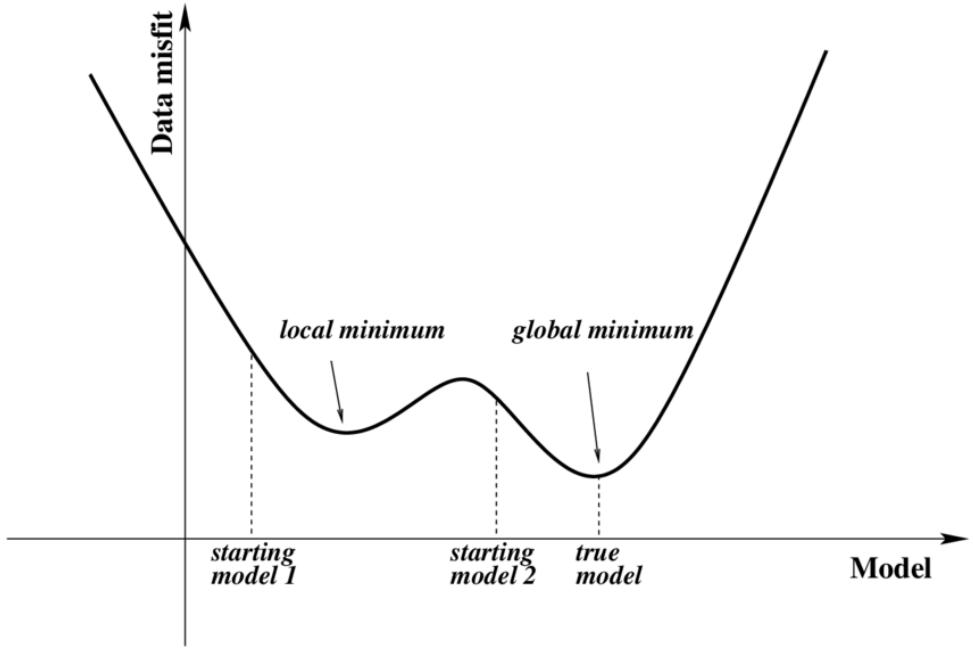
Берем θ_0. И учимся генерировать случайное $\theta^* = change(\theta)$ - небольшое изменение. Меняем θ на θ^*, если случилось улучшение.

```
while True:
    theta_new = change(theta)
    if f(theta_new) < f(theta):
        theta = theta_new
```

Через много итераций мы получим решение, которое уже не улучшить случайными изменениями. Можно верить, что это локальный минимум.

Глобальный минимум

Есть высокий риск "застрять" в локальном минимуме



Метод отжига

Будем делать локальную оптимизацию, но иногда разрешать себе делать "невыгодную" замену. Вероятность такой замены будем постепенно уменьшать.

```
T = 100
while True:
    theta_new = change(theta)
    if f(theta_new) <= f(theta) or exp((f(theta) - f(theta_new)) / T) > random(0, 1):
        theta = theta_new
    T *= 0.99
```

Генетический алгоритм

Идея заимствована из эволюции. Можно взять θ, и получить случайными мутациями набор $\theta_1, \ldots, \theta_n$. Дальше с помощью сравнений $f(\theta_1), \ldots, f(\theta_n)$ можно выяснить, какие мутации были самыми удачными, взять топ-N мутаций и попробовать для них следующий шаг эволюции.

Покоординатный метод

Если мы знаем вид функции f и можем посчитать ее частные производные по каждому числу в параметре θ, можно делать сдвиг вдоль частных производных:

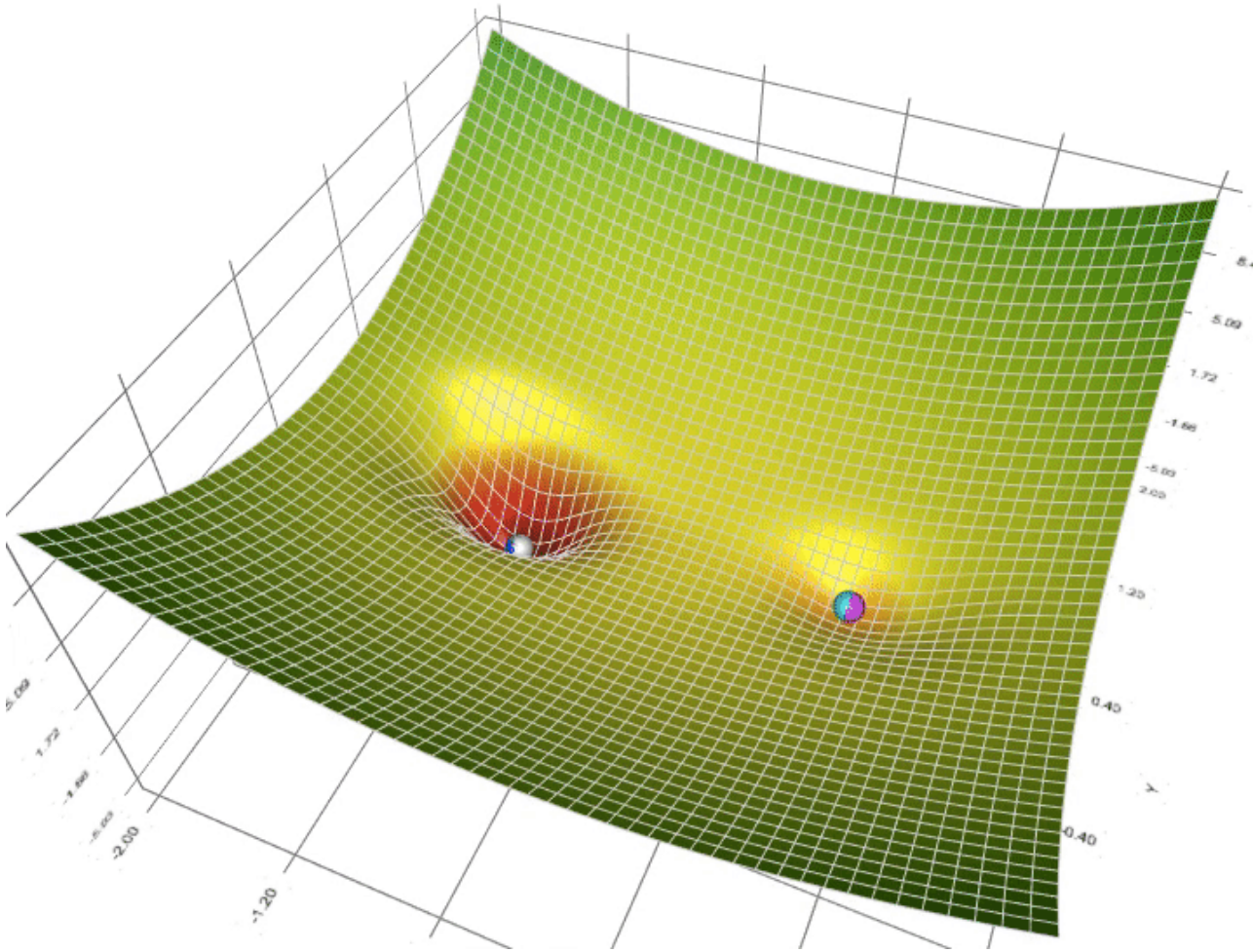
$$\theta^* = (\theta_1, \dots, \theta_i + \frac{\partial f}{\partial \theta_i}(\theta), \dots, \theta_n)$$

Градиентный спуск

Если мы знаем вид функции f и можем посчитать ее градиент в параметре θ, можно делать сдвиг вдоль антиградиента:

$$\theta^* = \theta - \nabla f(\theta)$$

Широко используется в нейронных сетях для оптимизации параметров.



Оффтоп + идея для пет-проекта - нейронные сети

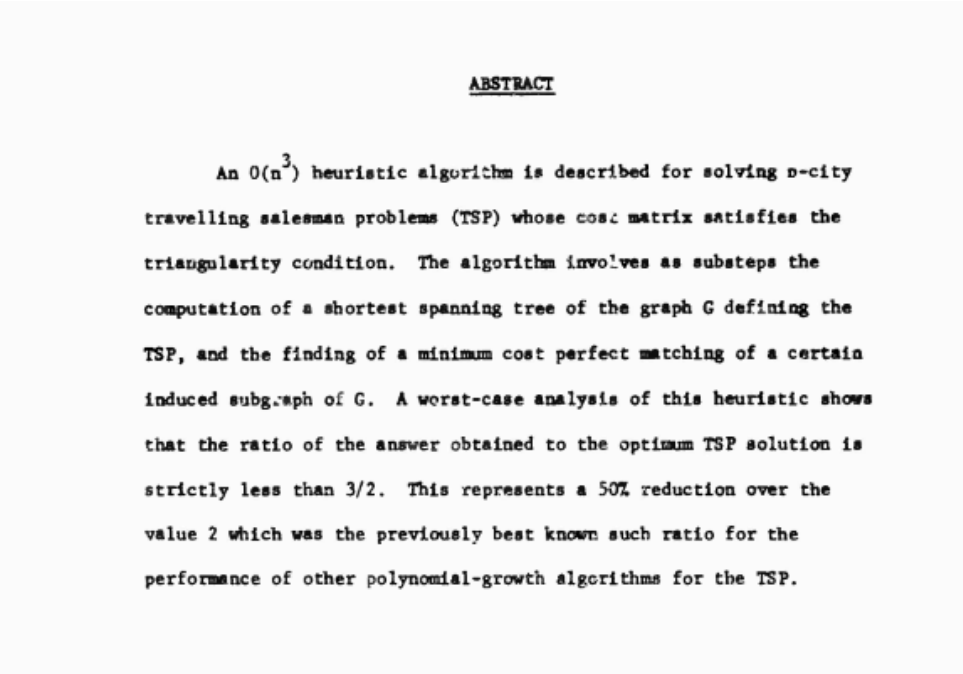
Нейронная сеть состоит из нескольких частей:

1. DAG вычислений (считает $g_\theta(x)$)
2. Функция ошибки (например, $f(\theta, x, y) = (g_\theta(x) - y)^2$)
3. Оптимизация f
   3.1. Генетический алгоритм: просто и наглядно, но не очень мощно
   3.2. Градиентный спуск: понадобится разобраться, как работает backpropagation, чтобы правильно искать градиент параметра

Если интересно 3.1, прикольно делать интерактивных агентов: собрать ИИ для игры, вот так это выглядит.

Пример статьи, которую можно взять за основу, если интересно 3.2.

~~Мем~~ байка



ABSTRACT

An $O(n^3)$ heuristic algorithm is described for solving n-city travelling salesman problems (TSP) whose cost matrix satisfies the triangularity condition. The algorithm involves as substeps the computation of a shortest spanning tree of the graph G defining the TSP, and the finding of a minimum cost perfect matching of a certain induced subgraph of G. A worst-case analysis of this heuristic shows that the ratio of the answer obtained to the optimum TSP solution is strictly less than 3/2. This represents a 50% reduction over the value 2 which was the previously best known such ratio for the performance of other polynomial-growth algorithms for the TSP.

A (Slightly) Improved Approximation Algorithm for Metric TSP

Anna R. Karlin*, Nathan Klein†, and Shayan Oveis Gharan‡

University of Washington

March 16, 2022

Abstract

For some $\epsilon > 10^{-36}$ we give a randomized $3/2 - \epsilon$ approximation algorithm for metric TSP.

Для дальнейшего изучения

- https://en.wikipedia.org/wiki/Karloff–Zwick_algorithm
- https://rtime.ciirc.cvut.cz/~hanzalek/KO/knapsack_e.pdf
- https://en.wikipedia.org/wiki/Genetic_algorithm
- https://ru.wikipedia.org/wiki/Стохастический_градиентный_спуск