Описание Математического Бота

Латыпов Владимир Витальевич

28 июля 2021 г.

Содержание

1. Формулировка задачи

Математический бот — это набор инструментов для работы с математическими сущностями через интерфейс привычных нам социальных сетей. В отличие от WolframAlpha, бот специализируется на оптимизации многомерных функций и численном решении уравнений с большим количеством переменных (проверено, что с этим бот справляется лучше).

Технически бот состоит из двух частей:

- Условный «front-end» — программа (написана на python), которая через vk_api выполняет коммуникативную функцию, то есть общается с пользователем и, когда требуется, вызывает основной блок.
- А именно — «back-end» интерфейс, который написан на C++ и компилируется в полноценное консольное приложение. Он и занимается оптимизацией функций, решениями уравнений и т.д.

Поэтому больший интерес, конечно, представляет back-end часть. Используется авторская библиотека для разбора математических выражений, умеющая в частности аналитически находить производные.

Для высококачественной оптимизации функции используется комбинация различных методов с помощью дерева оптимизации. Вот эти методы:

- Генетический алгоритм
- Алгоритм симуляции отжига
- Градиентный спуск
- Метод Ньютона

Последние два используют вычисление градиента и вторых производных, что реализуется с помощью автоматической генерации дерева вычислений для производных.

2. Обзор

Бот умеет:

- Оптимизировать функции

- Решать уравнения
- Строить графики

Третий режим добавлен для удобства — чтобы рассматривать функции (правда, пока только 2-х мерные), не выходя из диалога с ботом.

Во всех случаях вводится математическое выражение и, возможно, дополнительные параметры. Предусмотренно два варианта введения запросов боту:

- Через коммуникацию: от пользователя требуется читать сообщения бота, предложенные ответы, знакомясь с интерфейсом бота, и, отвечать на них (зачастую достаточно выбрать вариант из предложенных). Этот вариант подходит для людей, которые только знакомятся с возможностями бота.
- Quick Input Mode (QIM). Если пользователь точно знает, что ему нужно и что значат аргументы, для него целесообразно использовать эту возможность. Вся информация для запроса передаётся одним сообщением, содержащим корректрую команду QIM.

  Типичный запрос выглядит так (переносы строки не важны):

```
optimize x + y^2 - 1000.5
for x in [-10; 100],
y in (10.3e-100, 123)
| minorant 10
```

Подробнее об интерфейсе можно прочитать в Инструкции (ВСТАВИТЬ ССЫЛКУ!).

В отличие от других подобных инструментов, бот предоставляет дружелюбный интерфейс: если пользователь что-то некорректно указал, бот известит его об этом. Однако иногда бот проверяет характер: например, ему не очень нравится, когда много раз неправильно указывают аргумент или когда пишут с маленькой буквы.

3. Технические подробности

4. Описание алгоритма в общих чертах

4.1. Разбор выражений

Оптимизация: если какая-то ветка полностью вычислима заранее (без знания переменных), она считается сразу. Если +0 или *1, это тоже убирается.

4.2. Дерево алгоритмов оптимизации

4.2.1. Идея

Комбинируя разные алгоритмы, можно получить существенно более хороший результат, чем при запуске их по отдельности. Но важно понимать, в какой последовательности их лучше запускать.

Здесь нужно понимать, что, проектируя алгоритм оптимизации, приходится выбирать между скоростью сходимости и вероятностью попасть в локальный оптимум, не найдя то, что искали (конечно, это упрощённая картина, но полезно посмотреть под таким углом).

Умение избегать локальных оптимумов ради нахождения глобальных назовём глобальностью алгоритма.

Для понимания, как комбинировать алгоритмы, полезна идея: когда алгоритм с высокой глобальностью уже нашёл некую окрестность, в которой, по его мнению, находится оптимум, имеет смысл передать его результат в более локальный алгоритм, чтобы увеличить скорость и точность без потери этой окрестности.

Реализованные алгоритмы, которые могут быть полезны:

- Генетический алгоритм

- Алгоритм симуляции отжига
- Градиентный спуск
- Метод Ньютона

Алгоритмы расположены в порядке увеличения скорости сходимости и уменьшения глобальности. (Метод Ньютона можно назвать менее глобальным, чем градиентный спуск, так как он ищет не «ближайший» минимум, а любой «ближайший» экстремум (корень производной в данном случае))

Причём у некоторых алгоритмов есть разные гиперпараметры, которые также регулируют этот скорость сходимости. Проанализировав всё это, я пришёл к выводу, что целесообразно подключение алгоритмов друг к другу примерно так, как это описано в следующем разделе.

4.2.2. Схема дерева

На схеме используются такие обозначения алгоритмов:

- GA — Genetic Algorithm (генетический алгоритм)
- Annealing — Метод симуляции отжига
- GD — Gradient Descent (градиентный спуск)
- Newton — метод Ньютона

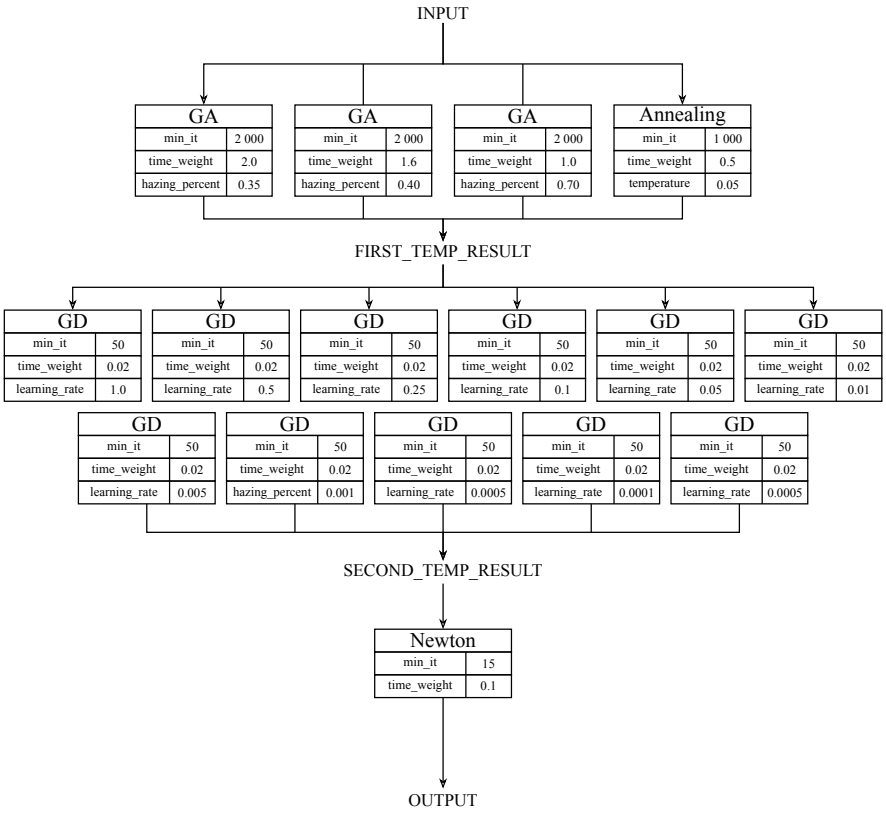

Рис. 1: Схема дерева оптимизации

(Первому слою не даётся никакого начального приближения)

Как видно из схемы, предполагается параллельное и последовательное соединение блоков.

При параллельном каждому child'у на вход даётся одно и то же начальное приближение (или его отсутствие, если первый слой), а выход параллельного блока — лучший из результатов его «child»-ов.

При параллельном — каждому следующему на вход даётся лучший из выходов всех предыдущих, а в качестве ответа также возвращается лучший из выходов child-ов.

В последовательном контейнере могут лежать как «голые» блоки, так и целые параллельные контейнеры. С параллельным блоком — аналогично.

Структура дерева задаётся через .json файл (пока что используется один и тот же файл для разных пользователей, но, возможно, в будущем имеет смысл изменять файл в зависимости, например, от наличия VIP-статуса у пользователя или от сложности его задачи).

Рис. 2: Пример конфигурационного файла

Также важная часть схемы — индивидуальные параметры блоков. Параметры { min_it, time_weight } есть у каждого блока вне зависимости от его типа.

Есть и типоспецифичные параметры. Например, hazing_percent для ГА и learning_rate для градиентного спуска. Как видно из схемы, на соответствующих слоях расположено несколько алгоритмов этих типов с разными значениями этих параметров. Такая конфигурация нужна потому, что для разных функций лучше будут работать вариации с разными параметрами.

4.2.3. Распределение вычислительного ресурса

В случае МатБота результаты профайлинга подтверждают предположение о том, что бо́льшая часть вычислительного времени ($\gg 95\%$) используется именно для подсчёта функции ошибки, а не для операций с геномами.

В условиях, когда выделенное количество вычислительных ресурсов может быть разным для разных людей и запросов, особенно остро встаёт вопрос о распределении этих вычислительных ресурсов.

Изначально backend-части бота на вход поступает число — условно «целевое количество итераций». По сути это выделенное для работы время, рассчитанное в количестве вычислений функции ошибки.

Это время было решено распределять пропорционально параметру time_weight (но количество итераций ограничено снизу порогом min_it). Но тут важно понимать, что это «время» — ещё не конечное значение итераций. Во-первых, в ГА непонятно, к чему применять слово «итерации» (см. ??). Во-вторых, градиентный спуск и метод Ньютона используют

5. Модификации в ГА

Подробное описание ГА (в том числе — моей версии) можно найти в описании робота-художника в соответствующей секции. Здесь сосредоточусь на том, что относится конкретно к боту.

В секции ?? описано, как вычислительный ресурс распределяется между разными алгоритмами. Однако на том моменте, когда стало известно количество раз, сколько можно вычислить функцию ошибки вов время исполнения ГА, вопрос ещё не закрыт.

Как известно, генетический алгоритм в каждую из E «эпох» вычисляет функцию ошибки для $population\_size$ геномов. То есть суммарное количество вычислений: $computational\_resource = E \times population\_size$.

Встаёт вопрос, какое соотношение значений E и $population\_size$ обеспечивает наилучшую работу алгоритма.

Я решил проверить это экспериментально. Изначальное предположение состоит в том, что разумно распределять ресурс так:

$$\begin{cases} E = computational\_resource^{epoch\_pow} \\ population\_size = \frac{computational\_resource}{E} = computational\_resource^{1-epoch\_pow} \end{cases} \tag{1}$$

Вопрос был в том, какое $epoch\_pow$ выбрать. Для решения вопроса я выбрал функцию с более или менее сложным рельефом и для разного количества итераций построил графики зависимости достигаемого fitness (значения целевой fitness-функции) от $epoch\_pow$.

Конечно, в целях помехоустойчивости для каждого $epoch\_pow$ делалось далеко не одно измерение. Важно оценивать не только минимальное, или только максимальное, или только некое среднее (иначе можно упустить важную информацию о недостатках или достоинствах того или иного $epoch\_pow$).

Поэтому для комплексной оценки я посчитал «нечёткие» верхнюю и нижнюю границы множества значений fitness-а для нескольких попыток запуска. Под нечёткой нижней границей понимается такое число, которое бы оценило «почти худший случай работы алгоритма». С верхней границей — аналогично (почти лучший).

Для их оценки я решил использовать вариацию среднего Колмогорова:

$$M(x_1, \ldots, x_n) = \varphi^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}\varphi(x_k)\right) \tag{2}$$

, где (в моём случае):

для нижней границы $\varphi(x) = x^{-1.5}$ (это повышает чувствительность к малым значениям), для верхней границы $\varphi(x) = x^{2.5}$ (это повышает чувствительность к большим значениям).

(альтернатива такой оценке — брать, например, результат алгоритмов для, например, 10-го и 90-го, процентилей, но этот метод хуже оценивает то, как именно распределены точки между ними и то, насколько сильно выбиваются верхние и нижние 10 процентов вверх и вниз по сравнению с остальными)

Результаты таковы:

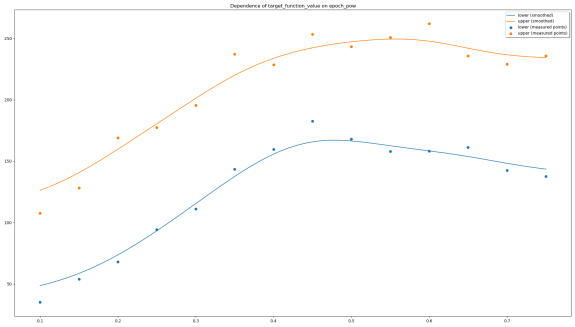

Рис. 3: График для 1'000 вычислений

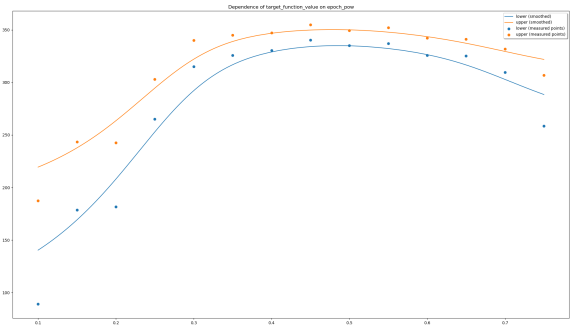

Рис. 4: График для 5'000 вычислений

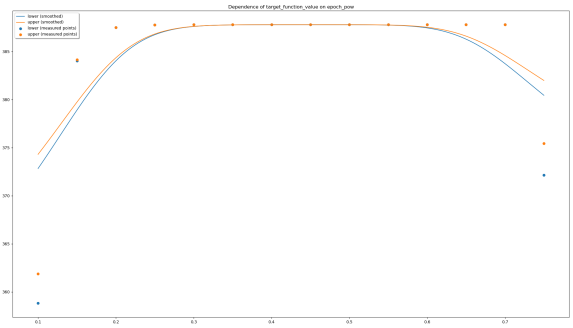

Рис. 5: График для 100'000 итераций

Проанализировав полученные данные, я пришёл к выводу, что лучше всего подходит значение $epoch\_pow \approx 0.45$

6. Дальнейшее развитие

— Добавить ещё алгоритмы. — Использовать аналитические методы, где возможно (например, для решения уравнений)