Magzyan1

mrGCrusader, micromolecul, Matveiiy

January 2025

1 Градиентный спуск

Градиентный спуск - важный алгоритм, применияемый в множестве сфер computer-science. Этот метод приминяется для дифференцируемых, унимодальных функций, действующих в пространство $\mathbb{R}$. Он работает следующим образом:
1) Выбираем точку, для старта нашего алгоритма
2) Вычисляем градиент функции в этой точке(мы можем так сделать, так как функция дифференцируема)
3) Идём по направлению антигриента на величину равную $\|h(m) * \nabla f\|$, где h - величина, на которую мы переходим на m шаге(learning rate scheduling), он может зависеть от нескольких параметров.
4) Проверяем выполнение критерия остонова, он может быть разным, и зависеть от нескольких параметров. Если критерий выполнен, то мы останавливаем наш алгоритм(вероятно мы нашли приближённую к минимуму функцию), если не выполнен, то переходим к шагу 2.

2 Критерии останова

Наша команда реализовала несколько критериев останова от самых простых, до самых сложных, вот некоторые из них:
1) MaxIterations - критерий, выполняющийся, когда мы проделали шаг 2 n раз. Этот критерий довольно прост в реализации, однако при использовании этого критерия есть риск перевычисления.
2) Convergence - критерий, выполняющийся, когда разница в значениях функции на n и $n-1$ шаге меньше заданного нами параметра ϵ. Этот критерий также прост в реализации, однако алгоритм может не выполнятся бесконечное количество времени(например если передали плохую функцию без минимума).
3) GradientNormComparative - критерий, выполняющийся, когда норма градиента меньше заданного нами значения $\epsilon * \|\nabla f(x_0)\|$ Этот критерий также может быть бесконечно неверен, если нам передали функцию без минимального значения.
Также мы реализовали класс Combine, способный комбинировать критерии останова. Комбинация критериев - важная механика, помогающая регулировать безопастность и эффективность алгоритма.

3 Шаг спуска

Также важным этапом реализации являлся выбор learning rate scheduling. Вот некоторые варианты из тех, которые мы реализовали:
1) Constant - шаг спуска равен заданной нами константе, этот шаг может привести к некоторым ошибкам, связанным с точностью, обычно используют монотонно убывающие функции, для

большей точности вычислений, однако это также повышает и объём вычислений, для нахождения ответа.
Далее представленны методы, которые просто являются монотонно убывающими функциями от шага k:
2) TimeBasedDecay - шаг спуска равен: $h(k) = \frac{h_0}{(1+\lambda*k)}$ - шаг с ассимтотикой $o(k^{-1})$
3) ExponentialDecay - шаг спуска равен: $h(k) = h_0 * exp(-\lambda * k)$ - шаг с ассимтотикой $o(exp(-\lambda * k))$

4 Наискорейший градиентый спуск

Наискорейший градиентный спуск - обычный градиентный спуск с особым шагом спуска, который предполагает движение до тех пор, пока мы не достигнем минимума функции f на этом направлении. Мы реализовали это с помощью различных реализаций одномерного спуска: Фибоначи, Армихо, Гольштейна, используя scipy.optimize.

5 Функции

Наш градиентный спуск рассчитан на дифференцируемые функции, однако для алгоритма на менее приятных функциях, мы немного модифицировали алгориитм вычисления градиента - метод get_modifier_value работает следующим образом: пользователь задаёт верхнюю границу, при этом если градиент больше заданной верхней границы, то мы считаем, что мы находимся в точке разрыва, тогда поведение следующее: мы выбераем рандомное направление, и идём в его сторону.
Также стоит отметить некоторые сложности, с которыми мы столкнулись, при построении примеров, например при построении функции Розенброка проблема заключалась в том, что значения функции слишком большие, мы не смогли обработать их, поэтому наша программа выдаёт ошибку.
Теперь перейдём к примерам функций, иллюстрирующих работу спуска:

Рис. 1)$y = x^2 + y^2$

Scheduler	**StopCriteria**	**Iterations**	**Result**
iter200	const	200	5.187225613695227e-19
iter200	time	200	0.02506265664203748
conv	time	249552	1.766190362115678e-08
conv	exp	357	1.651473508156676
iter200	exp	200	1.651473514629834
armijo	conv	115	8.958978971875905e-11

Как можно видеть из таблицы на простой функции константный шаг работает хорошо, градиент плавно затухает и все хорошо получается. TimeBasedDecay затухает, но медленно сходится всё-таки, поэтому требуется много итераций. ExponentialDecay слишком быстро затухает и плохо находит min.

Рис. 2)$y = x^2 + y - 1$

Эта функция не имеет точки min. Поэтому Convergence и GradientNorm работают на ней очень долго, и необходимо ограничить количество итераций. Поэтому найденное мин. значение зависело от количества итераций.

Интересно было бы рассмотреть шумную функцию, имеющую следующую формулу:

Рис. 3)$y = x^2 + y^2 + random$

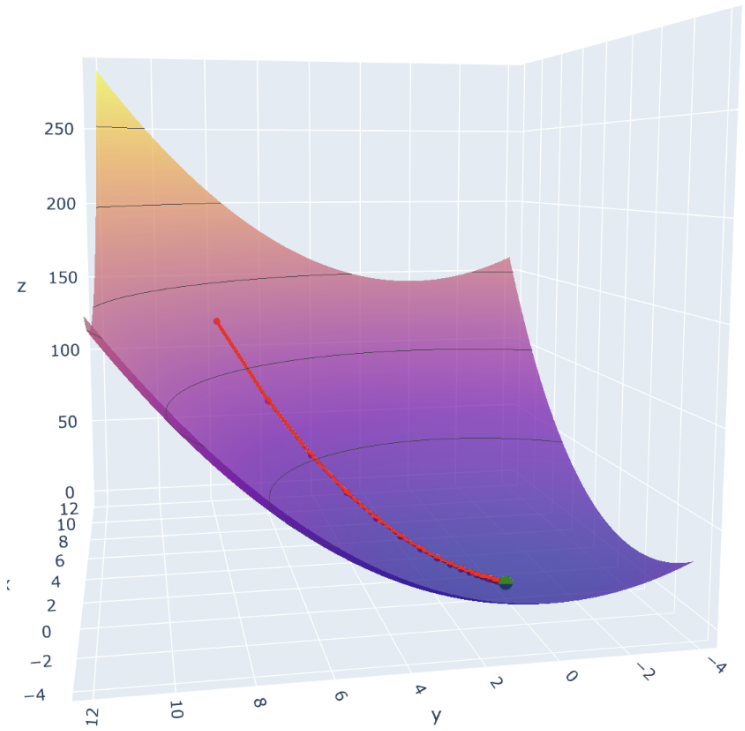

Рис. 1: Пример графика

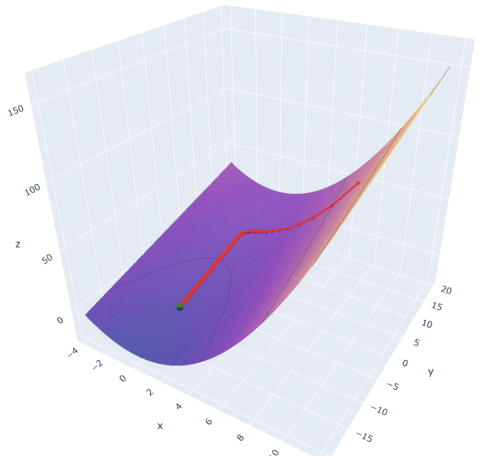

Рис. 2: Пример графика

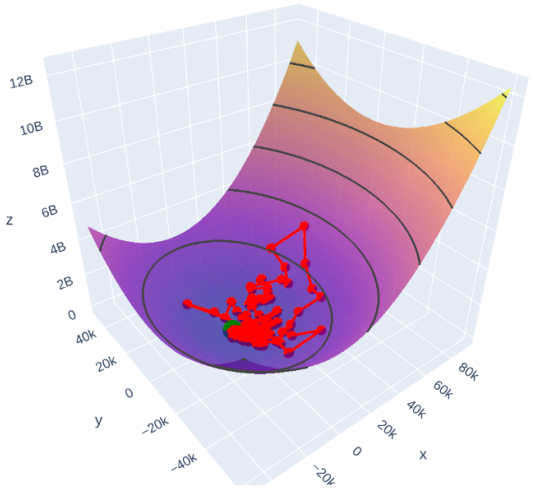

Рис. 3: Шумная функция

Интересно, что функция работает корректно именно при TimeBasedDecay lrs, при этом Constant lrs и ExponentialDecay lrs показыавют себя плохо, это объясняется тем, что Constant сходится слишком медленно(если быть точным, то вообще не сходится), а ExponentialDecay сходится слишком быстро. Также критерий останова Convergence также работает некорректно, так же не обязан работать корректно, так как разность значений каждый раз зависит от случайнойсти, что может привести либо к примерно бесконечному времени работы, либо к досрочному завершению Теперь давайте посмотрим на другую интересную(не является гладкой) функцию:

Рис. 4) $y = 100 * \sqrt{|y - 0.01 * x^2|} + 0.01 * |x + 10|$

Лучше всего работает с lrs Constant, потому что функция имеет несколько локальных минимумов, и засчёт большего шага больше возможности обойти много таких точек, в то время как TimeBasedDecay и ExponentialDecay быстро уменьшают шаг. Также стоит заметить, что если критерий останова - Convergence, то есть вероятность бесконечного количества итераций.

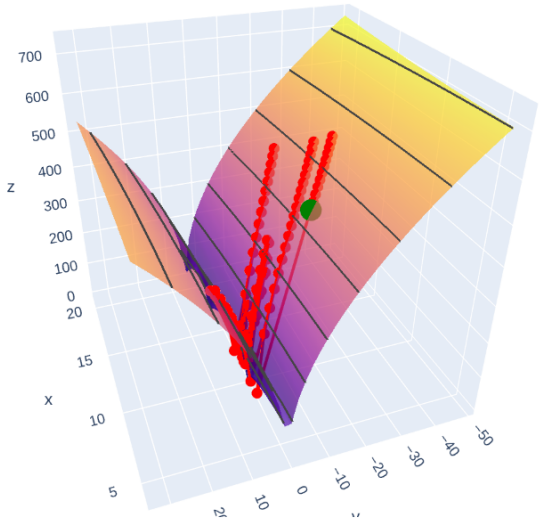

Рис. 4: Плохая функция

6 Вывод

Наша команда написала гибкий алгоритм градиентного спуска, в который можно передавать функцию критерия останова, learning_rate_scheduling, и многие другие параметры. Также в процессе тестирования мы выяснили, что для разных функций подходят разные функции шага, и разные критерии останова. Мы реализовали функции lrs такие как: константный шаг, различные виды шагов с затуханием (экспоненциальное, полиномиальное и др.). Также, разработали методы одномерного спуска, которые работают на унимодальных функциях(фибоначчи и золотого сечение), и общие правила выбора шага (Армихо и Гольдштейна) - для наискорешего спуска.