Методы оптимизации

Лабораторная работа 4

Метод имитации отжига. Optuna

Авторы: Куприянов М3233, Долматова М3232, Шайдулин М3233

Ссылка на реализацию

Цели исследования

В заключительной 4 работе наша команда стремилась изучить основы методов стохастической оптимизации, в частности провести много экспериментов с методом имитации отжига и посмотреть, как он будет себя вести на примерах из первых работ.

Помимо этого, важная часть работы заключалась в рассмотрении возможностей готовых версий предложенных алгоритмов из различных библиотек в Python, подбор и настройка параметров вызываемых функций.

РАБОТА

Итак, первый пункт, опишем теорию, которая стоит за методом имитации отжига:

Начнем, пожалуй, с объяснения такого своеобразного названия: дело в том, что идея алгоритма основана на процессах, которые происходят при “отжиге“ металлов, в ходе этого действия металл нагревается до достаточно высокой температуры в результате чего начинает деформироваться кристаллическая решетка, а затем происходит постепенное охлаждение, в ходе которого “выбившиеся” атомы хотят попасть в более выгодное состояние с меньшей энергией (но с определенной долей вероятности могут на некоторое время и ухудшить свое положение из-за конкуренции, но в итоге это помогает достичь конечной цели). В итоге температура опускается до требуемой, а энергетический уровень оптимизирован.

Будем имитировать этот процесс:

Для этого вводим функцию энергетического состояния E; убывающую функцию состояния температуры T; Функцию изменения состояния S.

Тогда будем повторять следующие действия, пока температура не станет требуемой:

1. Случайно находим новое решение S’ в окрестности нынешнего S
2. Находим новое значение E(S’)
3. Находим изменение энергии: ΔE = E(S’)- E(S)
4. Если новое решение лучше текущего, то есть ΔE <=0, то (S’) принимаем как новое
5. Если хуже, тогда применяем его с вероятностью e^(- ΔE /T)

Температура понижается по заданному сначала закону охлаждения, например с константным шагом на каждой итерации.

Алгоритм закончится, когда T достигнет минимума или не улучшится решение за заданное число итераций.

Опишем плюсы и минусы алгоритма:

- На начальных этапах, когда T высокая, вероятность принятия плохих решений (с увеличением энергии ΔE) также высокая. Это позволяет алгоритму "прыгать" из одного локального минимума в другой, исследуя пространство преодолевая локальные минимумы.

- При постепенном снижение температуры уменьшается вероятность принятия худших решений. На более низких температурах алгоритм становится более жадным, он чаще принимает только улучшения. Это позволяет ему концентрироваться на поиске точного минимума в окрестностях текущего лучшего решения.

- При правильном выборе параметров охлаждения, то есть асимптотически медленном, например как $T=T0/\log(1+k)$, где k — номер итерации) алгоритм гарантированно сойдется к глобальному минимуму с вероятностью 1, хотя на практике это может потребовать очень много времени, но в теории есть гарантированная сходимость.

Минусы:

- Метод имитации отжига может быть очень медленным, особенно если требуется большое количество итераций для достижения хороших результатов. Также на это влияет медленное охлаждение и большое количество итераций на каждой температуре.

- Вероятность принятия худших решений уменьшается с понижением температуры, но могут возникнуть проблемы с подбором более оптимальной функции.

- для задач с высоко размерными пространствами или, где целевая функция имеет сложную форму, метод имитации отжига менее эффективен по сравнению с другими методами.

- Сложность парализации.

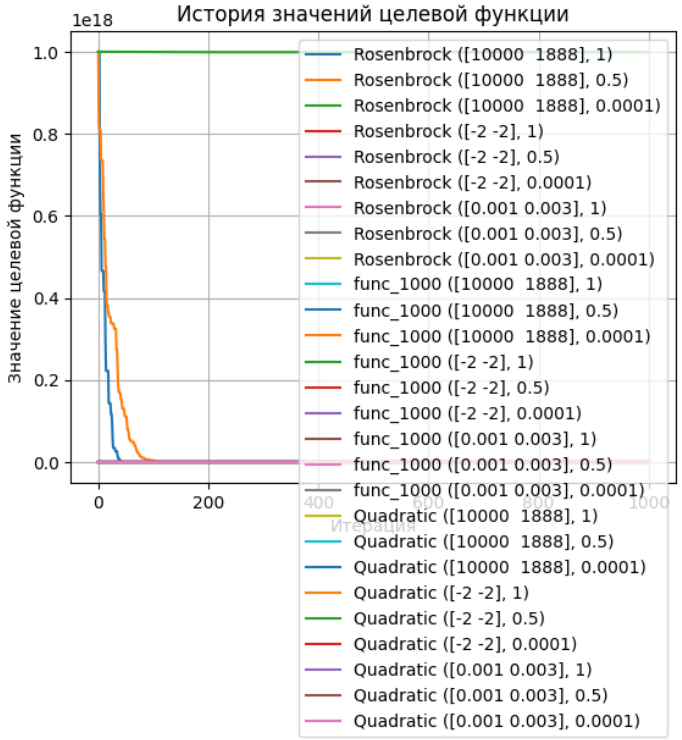
- Трудно подобрать параметры для оптимальной сходимости и времени.

- Зависимость от начального решения.

В нашей реализации выберем самую простую и часто используемую, то есть просто постепенное снижение температуры.

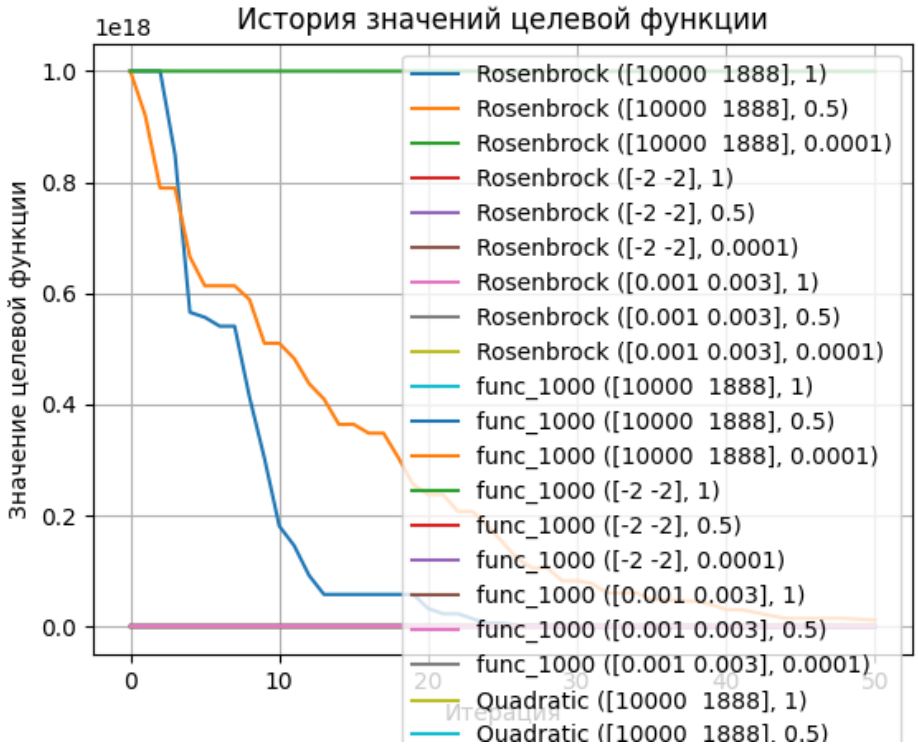
В качестве целевых функций мы взяли функции из работ 1 и 2. Также мы рассмотрели несколько точностей, которые бы влияли на шаг генерации новой точки в окрестности текущего решения, выведем результаты и график значений рассматриваемой функций в зависимости от начальной точки и точности:

Rosenbrock	[10000, 1888]	1.000000	[62.33, 3884.74]	3761.12	1000
Rosenbrock	[10000, 1888]	0.500000	[51.38, 2639.71]	2537.99	1000
Rosenbrock	[10000, 1888]	0.000100	[9997.37, 1887.85]	99891162279387888.00	1000
Rosenbrock	[-2, -2]	1.000000	[-7.41, 54.82]	71.60	1000
Rosenbrock	[-2, -2]	0.500000	[-6.67, 44.42]	58.88	1000
Rosenbrock	[-2, -2]	0.000100	[-1.54, -2.11]	2005.62	1000
Rosenbrock	[0.001, 0.003]	1.000000	[1.01, 1.04]	0.02	1000
Rosenbrock	[0.001, 0.003]	0.500000	[0.96, 0.91]	0.04	1000
Rosenbrock	[0.001, 0.003]	0.000100	[0.18, 0.03]	0.67	1000
func_1000	[10000, 1888]	1.000000	[0.63, 3023.46]	9141678.69	1000
func_1000	[10000, 1888]	0.500000	[0.32, 3302.41]	10906023.63	1000
func_1000	[10000, 1888]	0.000100	[9997.42, 1887.92]	99952052279.98	1000
func_1000	[-2, -2]	1.000000	[0.03, 0.94]	2.06	1000
func_1000	[-2, -2]	0.500000	[-0.00, -0.23]	0.07	1000
func_1000	[-2, -2]	0.000100	[-1.53, -1.17]	2339.72	1000
func_1000	[0.001, 0.003]	1.000000	[0.001, 0.003]	0.001	1000
func_1000	[0.001, 0.003]	0.500000	[0.001, 0.003]	0.001	1000
func_1000	[0.001, 0.003]	0.000100	[0.001, 0.003]	0.001	1000
Quadratic	[10000, 1888]	1.000000	[1.86, 3.08]	0.03	1000
Quadratic	[10000, 1888]	0.500000	[1.99, 3.01]	0.00	1000
Quadratic	[10000, 1888]	0.000100	[9998.36, 1888.11]	103480957.40	1000
Quadratic	[-2, -2]	1.000000	[1.99, 3.00]	0.00	1000
Quadratic	[-2, -2]	0.500000	[2.04, 3.05]	0.00	1000
Quadratic	[-2, -2]	0.000100	[-2.06, -0.98]	32.37	1000
Quadratic	[0.001, 0.003]	1.000000	[1.98, 2.94]	0.00	1000
Quadratic	[0.001, 0.003]	0.500000	[1.96, 3.00]	0.00	1000
Quadratic	[0.001, 0.003]	0.000100	[0.02, 0.04]	12.67	1000



Как мы видим результаты получаются гораздо точнее чем при использовании методов из лаб 1-2, что вполне ожидаемо, ведь теперь мы с меньшей вероятностью застреваем в локальных минимумах, но это зависит от подбора точности: как легко увидеть из таблицы или графика, важно выбрать ее не слишком большую и достаточно малую, чтобы не было застреваний и шаг менялся бы корректно, например, можно заметить, что при точности уже 1e-4 очень уж малым, поэтому значение функции остается почти неизменным, очень важно правильно подобрать, например, при малом числе итераций (100 или 50)

Значения целевой функции находится лучше при небольших точностях, и хоть значения получаются сильно менее точными, чем при большом числе итераций, получаем результаты примерно такие же или даже более эффективными, чем при методах из первых лабораторных:



Rosenbrock	[10000, 1888]	1.000000	[19.32, 2801.76]	589819679.52	50
Rosenbrock	[10000, 1888]	0.500000	[3333.99, 2643.95]	12349479560457916.00	50
Rosenbrock	[10000, 1888]	0.000100	[9998.98, 1887.70]	999556009338496128.00	50
Rosenbrock	[-2, -2]	1.000000	[-2, -2]	3609.00	50
Rosenbrock	[-2, -2]	0.500000	[-2, -2]	3609.00	50
Rosenbrock	[-2, -2]	0.000100	[-1.64, -1.99]	2206.68	50
Rosenbrock	[0.001, 0.003]	1.000000	[0.001, 0.003]	0.9989	50
Rosenbrock	[0.001, 0.003]	0.500000	[0.001, 0.003]	0.9989	50
Rosenbrock	[0.001, 0.003]	0.000100	[0.11, 0.004]	0.80	50
func_1000	[10000, 1888]	1.000000	[468.67, 811.30]	220310420.09	50
func_1000	[10000, 1888]	0.500000	[6004.33, 2633.08]	36058856066.02	50
func_1000	[10000, 1888]	0.000100	[9999.22, 1888.05]	99987883265.65	50
func_1000	[-2, -2]	1.000000	[-2, -2]	4004.00	50
func_1000	[-2, -2]	0.500000	[-2, -2]	4004.00	50
func_1000	[-2, -2]	0.000100	[-1.91, -2.27]	3659.00	50
func_1000	[0.001, 0.003]	1.000000	[0.001, 0.003]	0.001	50
func_1000	[0.001, 0.003]	0.500000	[0.001, 0.003]	0.001	50
func_1000	[0.001, 0.003]	0.000100	[0.001, 0.003]	0.001	50
Quadratic	[10000, 1888]	1.000000	[161.28, -64.91]	29980.83	50
Quadratic	[10000, 1888]	0.500000	[5165.23, 1129.90]	27928880.87	50
Quadratic	[10000, 1888]	0.000100	[9999.08, 1888.43]	103496387.60	50
Quadratic	[-2, -2]	1.000000	[-2, -2]	41.00	50
Quadratic	[-2, -2]	0.500000	[-2, -2]	41.00	50
Quadratic	[-2, -2]	0.000100	[-2.10, -1.60]	37.99	50
Quadratic	[0.001, 0.003]	1.000000	[0.001, 0.003]	12.98	50
Quadratic	[0.001, 0.003]	0.500000	[0.001, 0.003]	12.98	50
Quadratic	[0.001, 0.003]	0.000100	[-0.04, 0.09]	12.61	50

Что показывает, что этот метод как, пожалуй, лучший из рассмотренных относительно соотношения скорости/качества.

Прейдем к дополнительным пунктам:

В этой части работы мы использовали Optuna для автоматического подбора гиперпараметров модели линейной регрессии из 3 лабы для стохастического градиентного спуска (SGDRegressor), его мы взяли из библиотеки scikit-learn.

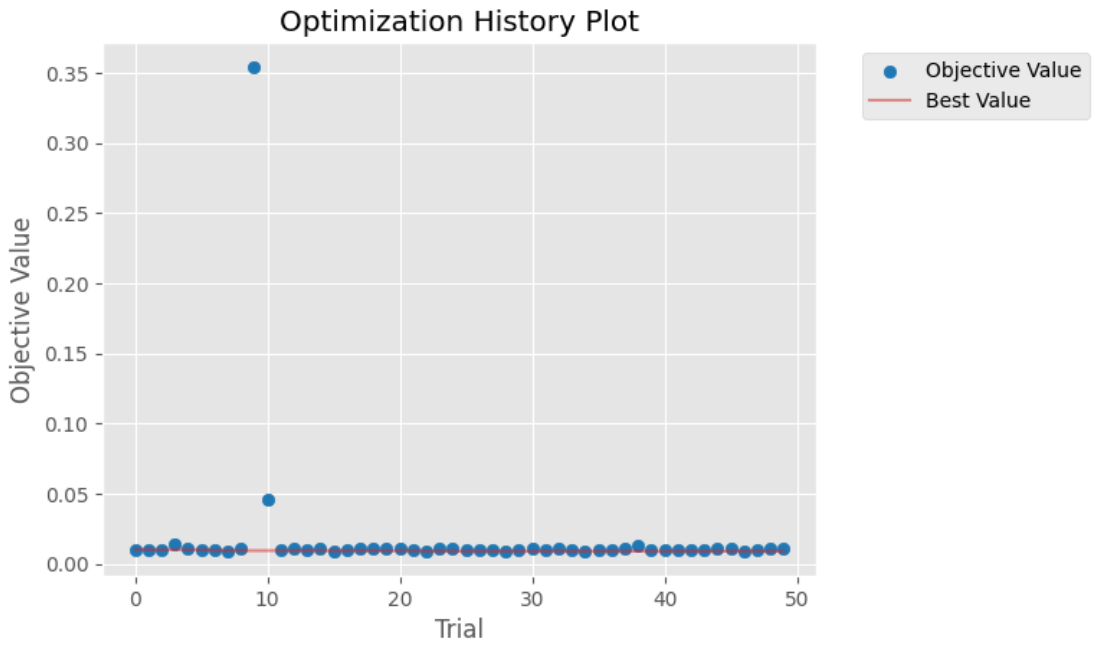
В данном коде мы находили набор гиперпараметров, который бы минимизировал среднеквадратичную ошибку (MSE)

Обучение происходило следующим образом: вначале данные делятся на батчи, и на каждом шаге SGD обновляет веса, используя только данные из одного батча. Это повторяется пока не будет сходимости.

В 3 лабе подбор гиперпараметров мог осуществляться вручную, из-за чего не учитывалась информацию из предыдущих испытаний, поэтому интересно было посмотреть на результаты, которые получатся при использовании Optuna.

Пару слов о результатах, как и ожидалось, параметры получены с большой точностью и во многом повторяют подтверждают наши выводы из третьей лабы, например, про оптимальный размер батча. Число используемых итераций кажется тоже вполне разумным и приемлемым для использования. В качестве единственного минуса можно отметить качество и структуру вывода в Optuna, которая на наш взгляд , сильно усложняет использование и анализ результатов.

Вот так выглядит график оптимизации:



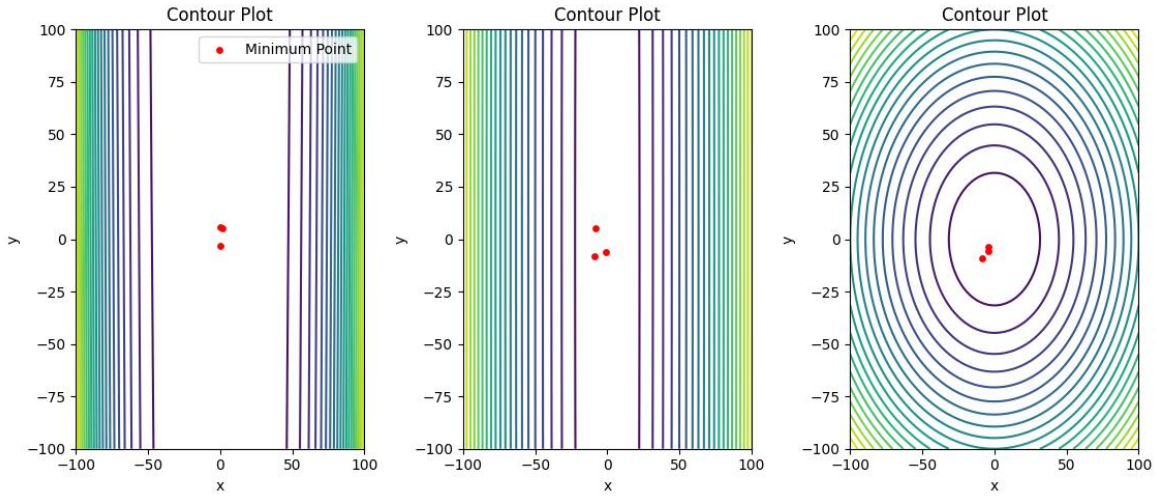
Схожую работу мы сделали и для метода Ньютона из второй лабораторной работы для подбора лучших значений точки:

В ней мы выбирали координаты точки из диапазона, которые затем подставляли я в метод Ньютона, trial.suggest_float('x0', -10, 10) и trial.suggest_float('y0', -10, 10) определяют диапазон значений для начальных координат.

Идея в том, чтобы потом optuna выбрала при каком запуске метода Ньютона лучший результат. Для каждой функции при 3 погрешностях создаётся объект study и запускается функция.

Также мы используем функцию objective, которая вызывается для каждой итерации trial. Оптимизация проводится 5 раз для каждой комбинации функции и допустимой погрешности (n_trials=5).

Лучшие параметры, найденные библиотекой для каждого запуска, сохраняются в список best_results и как видно из таблиц и графиков, Optuna хорошо работает и обеспечивает сходимость:



Не будем вставлять полный список для всех результатов, будем резюмировать итоги в виде одной небольшой таблицы:

Function	Best x_0 for each 5	Best y_0 for each 5
Rosenbrock	0.096881	-3.113041
Rosenbrock	1.389589	5.017208
Rosenbrock	-0.149228	5.917183
$1000x^2 + y^2$	-8.613924	-7.941655
$1000x^2 + y^2$	-0.812804	-6.165012
$1000x^2 + y^2$	-8.470941	5.052757
$x^2 + y^2$	-4.431272	-5.547114
$x^2 + y^2$	-8.771445	-9.227020
$x^2 + y^2$	-4.351485	-3.538662

Итоги

В заключение хотелось сказать, что мы изучили новый метод оптимизации функций, сравнили его результаты с тем, что получили из предыдущих методов, таким образом подтвердив результаты, которые ожидались из теоретического описания методов. Также наша команда разобрала возможности библиотеки Optuna для подбора оптимальных параметров различных методов из предыдущих лабораторных, что позволило по-новому взглянуть на ранее подобранные вручную параметры, а также на результаты, полученные при применении к ним функций, что отлично подытожило проделанные нами прошлые работы.