Методы стохастической оптимизации

Усов Федор

Халиков Даниил

МОП 162
05 октября 2018

Оптимизация и стохастика

- Цель оптимизации – найти параметры, на которых достигается экстремум целевой функции
- Стохастическая оптимизация - класс алгоритмов оптимизации, использующая случайность в процессе поиска оптимума.
- Применяется, когда целевая функция либо слишком сложная, либо мы хотим повысить быстродействие алгоритма оптимизации.

Плюсы

- Универсальность
- Преимущество в скорости вычислений
- Можно свести сложную модель к более простой

Минусы

- Флуктуации
- Необходимость для каждой модели что-то придумывать

Разные стохастические методы

- Стохастический градиентный спуск (**SGD**)
- Полустохастический градиентный спуск (**S2GD**)
- Стохастический средний градиент (**SAG**)
- Алгоритм имитации отжига
- Генетический алгоритм

Стохастический градиентный спуск (SGD)

- Оцениваем значение градиента, вычисляя градиент только на одном элементе обучения:

$$L(x) = \sum_{i=1}^{N} L_i(x) \to \min$$

- Выбираем i случайно, равномерно для каждого шага k:

$$x_{k+1} = x_k - \eta_k \nabla L_i(x_k)$$

- Для сходимости стохастического градиентного спуска необходимы условия Роббинса-Монро:

$$\sum_{k}^{\infty} \eta_k = \infty, \sum_{k}^{\infty} \eta_k^2 < \infty$$

Стохастический градиентный спуск (SGD) Особенности

- Медленная сходимость в отличие от полного
- Скорость вычисления не зависит от числа сэмплов
- Простота реализации

Полустохастический градиентный спуск (S2GD)

- Градиент не изменяется кардинально
- Значит можем использовать старую информацию
- Поэтому вычислим градиент
- Сделаем несколько шагов используя это направление с помощью стохастического
- Функция должна иметь липшицев непрерывный градиент с константой L > 0:

$$\forall x, z \in \mathbb{R}^d, f(x) \leq f(x) + \langle f'(x), z - x) + \frac{L}{2} \parallel z - x \parallel^2$$

- Функция должна быть μ-сильно выпуклой:

$$\forall x, z \in \mathbb{R}^d, f(x) \geq f(x) + \langle f'(x), z - x) + \frac{\mu}{2} \parallel z - x \parallel^2$$

Полустохастический градиентный спуск (S2GD)

- Можем приблизить значение f(y), если y близок к x, как:

$$\nabla f(y) = \nabla f(y) - \nabla f(x) + \nabla f(x)$$
$$\nabla f(y) \approx \nabla f_i(y) - \nabla f_i(x) + \nabla f(x)$$

- Во внешнем цикле вычисляем полный градиент
- Во внутреннем используем стохастический спуск со случайным числом семплов

Полустохастический градиентный спуск (S2GD)

Algorithm 1 Semi-Stochastic Gradient Descent (S2GD)

parameters: m = max # of stochastic steps per epoch, h = stepsize, ν = lower bound on μ

for $j = 0, 1, 2, \ldots$ **do**

$g_j \leftarrow \frac{1}{n}\sum_{i=1}^{n} f_i'(x_j)$

$y_{j,0} \leftarrow x_j$

Let $t_j \leftarrow t$ with probability $(1-\nu h)^{m-t}/\beta$ for $t = 1, 2, \ldots, m$

for $t = 0$ to $t_j - 1$ **do**

Pick $i \in \{1, 2, \ldots, n\}$, uniformly at random

$y_{j,t+1} \leftarrow y_{j,t} - h(g_j + f_i'(y_{j,t}) - f_i'(x_j))$

end for

$x_{j+1} \leftarrow y_{j,t_j}$

end for

Полустохастический градиентный спуск Особенности

- Использует идеи обоих градиентных спусков
- Работает существенно быстрее полного
- Сходится лучше стохастического

Стохастический средний градиент (**SAG**)

- Главная идея – усреднять градиент по сэмплам
- На каждом шаге будем случайно выбирать объект, по которому честно посчитаем градиент
- Остальные – с прошлых шагов

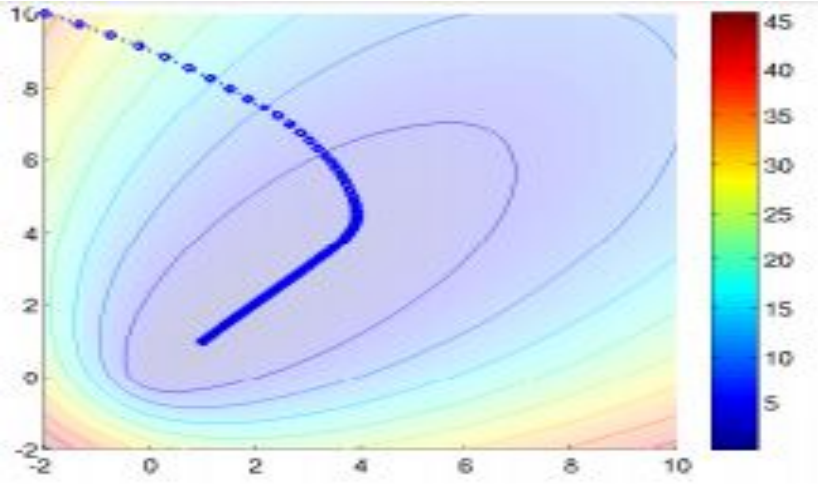
$$x_{k+1} = x_k - \frac{\mu}{n}\left(\sum_{i=1}^{n} y_i^{k+1}\right)$$

$$y_i^{k+1} = \begin{cases} \nabla f_i(x_k), & \text{if i = j} \\ y_i^k, & \text{otherwise} \end{cases} \quad (where\ j = random[1, n])$$

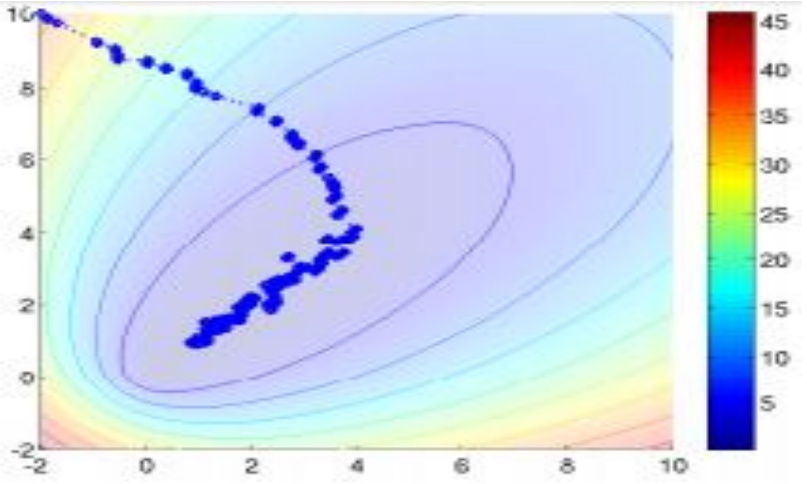
Стохастический средний градиент (**SAG**) Особенности

- Обновляем только градиент одного сэмпла, остальные запоминая с предыдущих шагов
- Более быстрая сходимость чем у стохастического
- Простота реализации

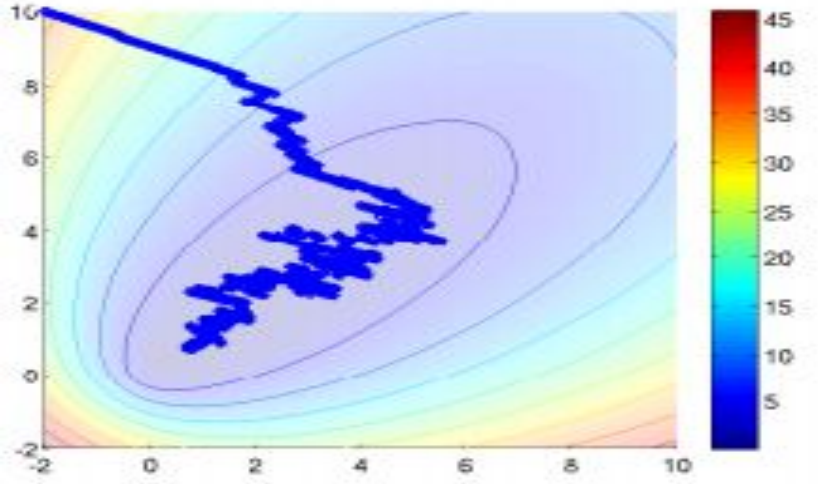
Иллюстрации работы методов, где N = 1000, w - двумерный



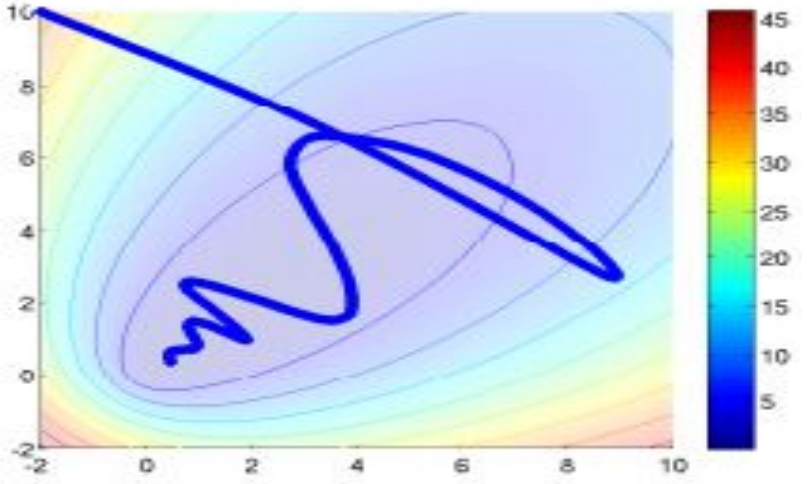
(a) Gradient descent



(b) Stochastic gradient descent



(c) Semi-stochastic gradient descent



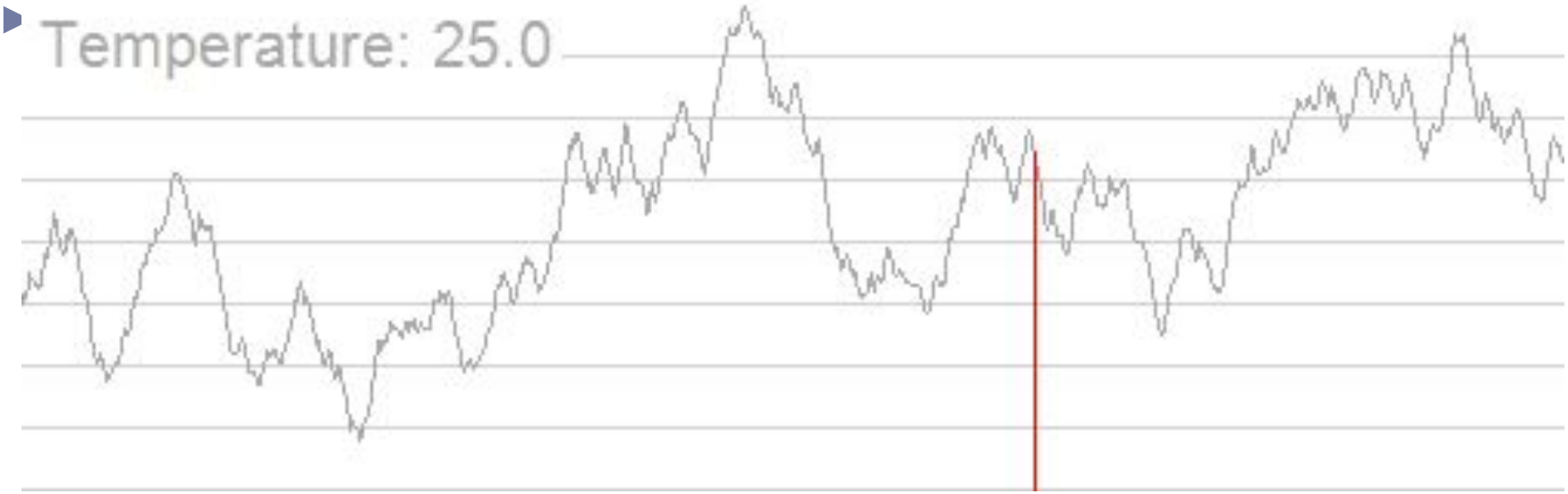
(d) Stochastic average gradient

Алгоритм имитации отжига [:|||||:]

- Главная идея – использовать физический процесс кристаллизации вещества
- Есть температура T, функция перехода из старого состояния в новое – S и функция энергии - E.
- С каждой итерацией уменьшаем температуру, пока она не станет меньше граничного значения.
- Если энергия меньше, перейдем в новое состояние
- Если больше, перейдем с определенной вероятностью $e^{-\frac{\Delta E}{t_i}}$
- Например :

Алгоритм имитации отжига [:|||||:] Особенности

- Простота реализации
- Хорошо работает в разных оптимизационных задачах
- . .



Генетический алгоритм

- Главная идея – использовать эволюционный отбор
- Генерация начальной популяции
- Повторяем:
- Скрещивание
- Случайные мутации
- Отбор
- Пока не будут получены приемлемые результаты

Генетический алгоритм

- Скрещивание происходит как обмен случайными хромосомами(значения признаков)
- Некоторая доля особей получает случайные мутации
- Отбираем случайных особей, с вероятностью зависимой от их приспособленности

Генетический алгоритм
Особенности

- При больших размерностях задачи оценка функции приспособленности требует больших затрат
- Плохо масштабируются под сложность задачи
- Неясное условие остановки
- Чтобы алгоритм работал хорошо, надо тщательно подстраивать модель
- Может хорошо работать

Дальнейшие улучшения для стохастического градиентного спуска

- Адаптивный коэффициент обучения
- Momentum
- Ускоренный градиент Нестерова
- Adagrad
- RMSprop
- Adadelta
- Adam
- AdaMax

Адаптивный коэффициент обучения

- При маленьких коэффициентах обучения будем долго идти к оптимальному значению
- При больших – можем перескочить точку оптимума
- Есть разные стратегии изменения коэффициента:
  - Уменьшать с каждой итерацией (например 1/k)
  - Сначала повышать, потом уменьшать
  - В зависимости от ошибки

Momentum

- SGD может колебаться при большом изменении в одном из измерений
- Поэтому будем учитывать предыдущие градиенты
- Добавим импульс движения p:

$$p_{k+1} = \gamma p_k + \eta \nabla L_i(x_k), \gamma < 1$$
$$x_{k+1} = x_k - p_{k+1}$$

- В итоге мы увеличиваем скорость движения в измерениях, где она слабо меняется, и уменьшаем, где она изменяется в разных направлениях

Ускоренный градиент Нестерова

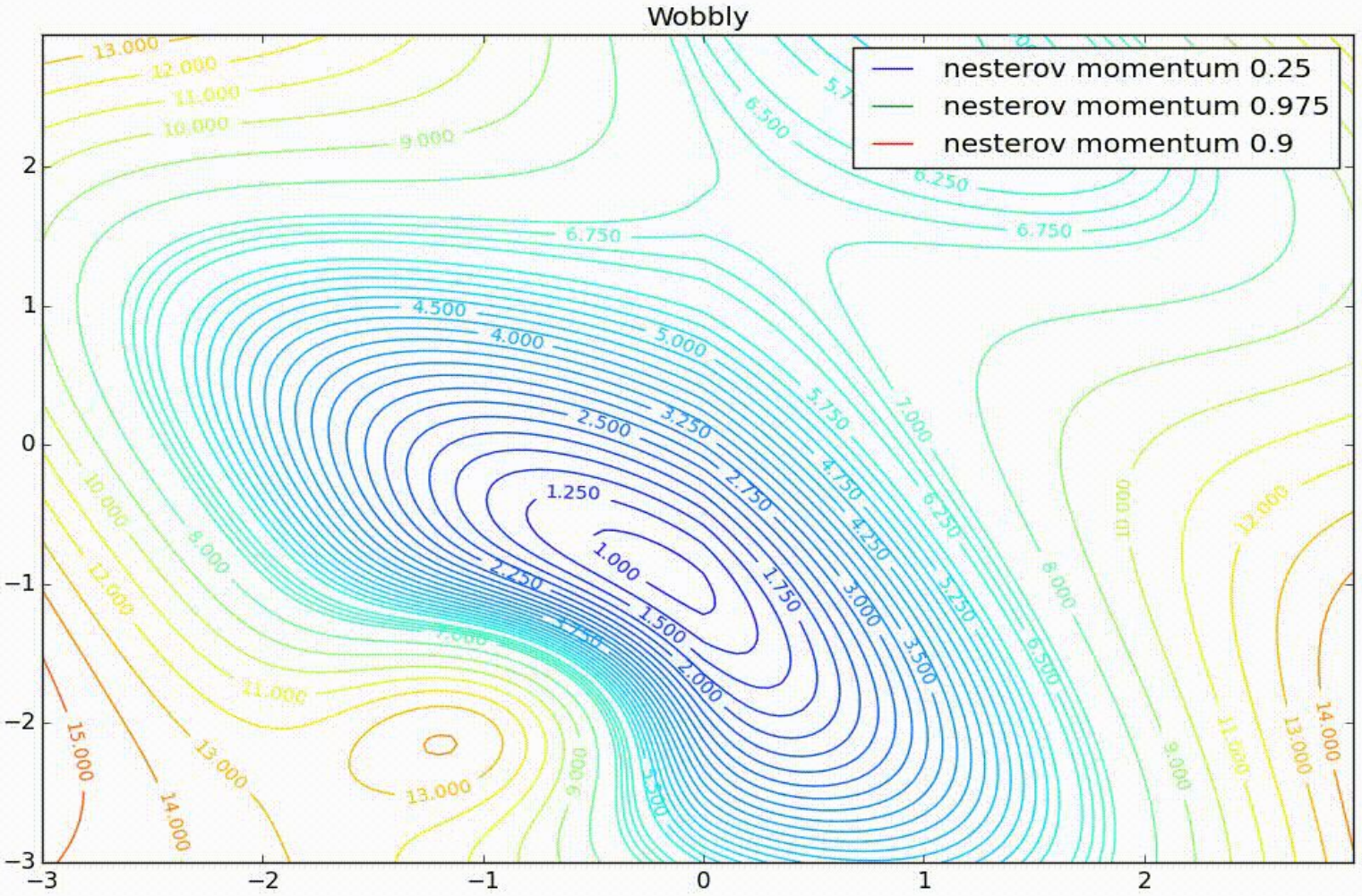
- Используем идею momentum'а, только будем считать градиент в примерном месте, котором окажемся

$$p_{k+1} = \gamma p_k + \eta \nabla L_i(x_k - \gamma p_k), \gamma < 1$$

$$x_{k+1} = x_k - p_{k+1}$$

- Позволяет избежать "столкновения" с крутой поверхностью
- γ - нужно подбирать для конкретной задачи, так как при больших может сильно колебаться, а при маленьких похож на обычный SGD

Ускоренный градиент Нестерова для разных коэффициентов



AdaGrad

- Главная идея – для каждого параметра адаптивно подбираем длину шага, используя прошлые изменения параметра
- Позволяет работать со sparse data, где важные признаки встречаются редко
- Будем считать сумму квадратов изменений:

$$G^j_{k+1} = G^j_k + \left(\nabla L^j(w_k)\right)^2$$

- Теперь используя информацию считаем отклонение

$$w^j_{k+1} = w^j_k - \frac{\eta}{\sqrt{G^j_{k+1} + \epsilon}}\left(\nabla L^j(w_k)\right)$$

AdaGrad

- Не надо точно подбирать коэффициент обучения
- Если признак часто обновляется, то его скорость обновления замедлится
- Семейство алгоритмов – можно изменять формулу

RMSprop

- Недостаток Adagrad, что G_t может бесконечно накапливаться
- Исправим это используя momentum:

$$G_{k+1}^j = \gamma G_k^j + (1-\gamma)\left(\nabla L^j(w_k)\right)^2$$
$$w_{k+1}^j = w_k^j - \frac{\eta}{\sqrt{G_{k+1}^j + \epsilon}}\left(\nabla L^j(w_k)\right)$$

- В знаменателе – RMS, отсюда название RMSprop

$$RMS[\nabla L(w_k)] = \sqrt{G_{k+1} + \epsilon}$$

- Если задаем начальное нулевое значение, то RMS будет долго накапливаться

Adadelta

- Также как и RMSprop, только добавим в числитель стабилизирующий член:

$$w_{k+1} = w_k - \frac{RMS[\Delta w]_{k-1}}{RMS[\nabla L(w_k)]} \nabla L(w_k)$$

$$P_k(\Delta w) = \gamma P_{k-1}(\Delta w) + (1-\gamma)\Delta w_k^2$$

$$RMS[\Delta w]_t = \sqrt{P_t(\Delta w) + \epsilon}$$

- Без большого начального стабилизирующего члена получим поведение обратное Adagard и RMSprop.

Adam

- Сочетает идею momentum и слабого обновления весов для типичных признаков.

$$p_{k+1} = \gamma_1 p_k + (1 - \gamma_1)\nabla L_i(x_k), \gamma_1 < 1$$

$$g_{k+1} = \gamma_2 g_k + (1 - \gamma_2)\nabla L_i^2(x_k), \gamma_2 < 1$$

- Аналогично RMSprop при малых начальных значениях они будут долго накапливаться

Adam

- Искусственно увеличим их :

$$\hat{p_k} = \frac{p_k}{1 - \gamma_1^k} \quad \hat{g_k} = \frac{g_k}{1 - \gamma_2^k}$$

- Правило обновления:

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{\hat{g_k}} + \epsilon} \hat{p_k}$$

AdaMax

- Рассмотрим момент в l_p :

[redacted]

- Для больших p работает плохо
- Для бесконечности – хорошо

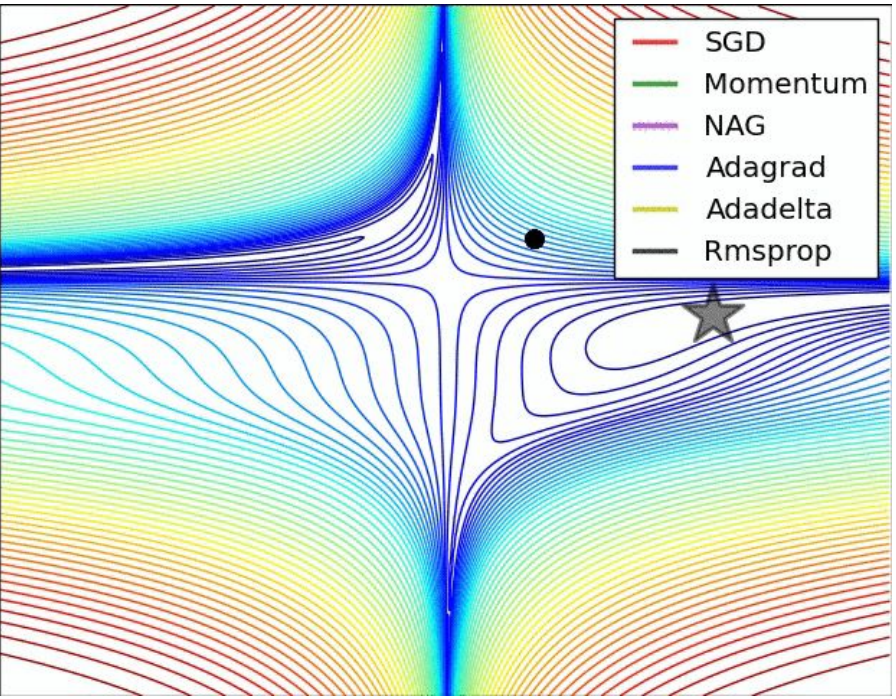
$$g_{k+1} = \gamma_2^{\infty} v_k + (1 - \gamma_2^{\infty}) |\nabla L_i(x_k)|^{\infty} =$$
$$max(\gamma_2 \cdot v_k, |\nabla L_i(x_k)|)$$

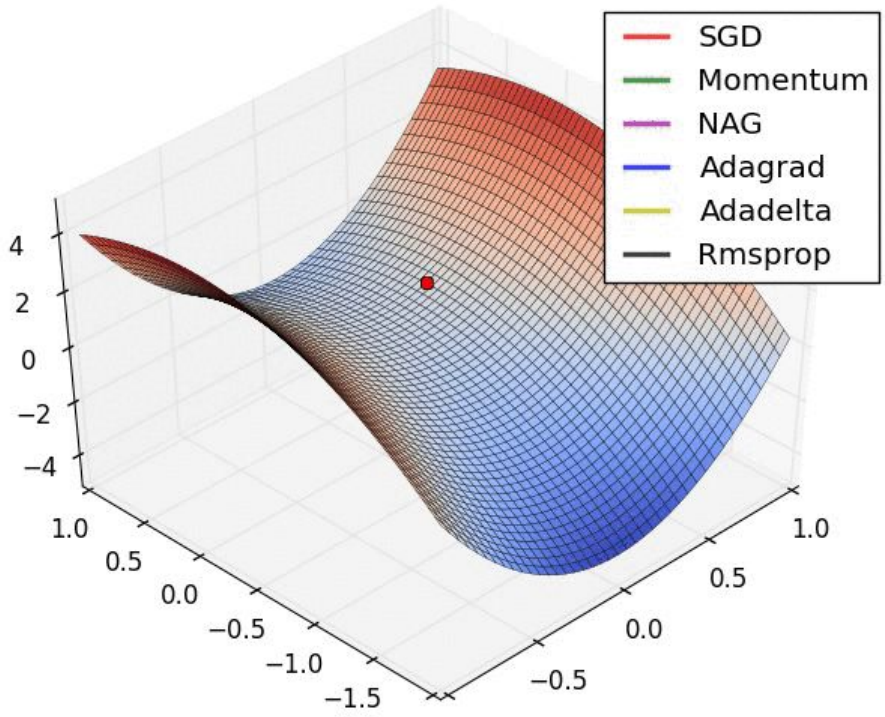
- Правило обновления:

$$w_{k+1} = w_k - \frac{\eta}{g_k} \hat{m}_k$$

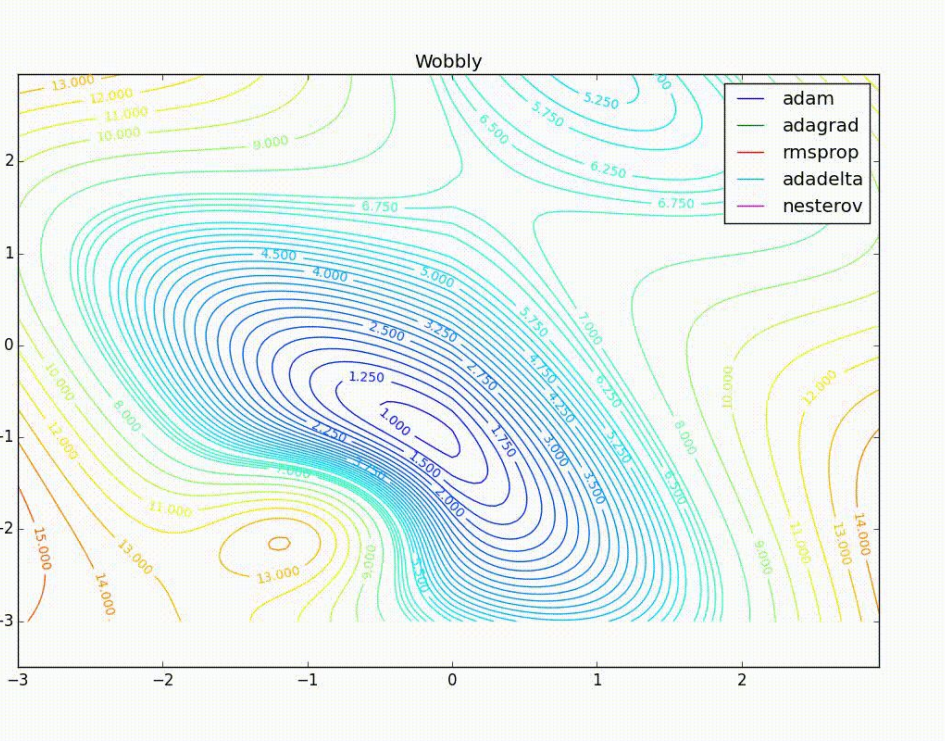
Не требуется смещение относительно нуля

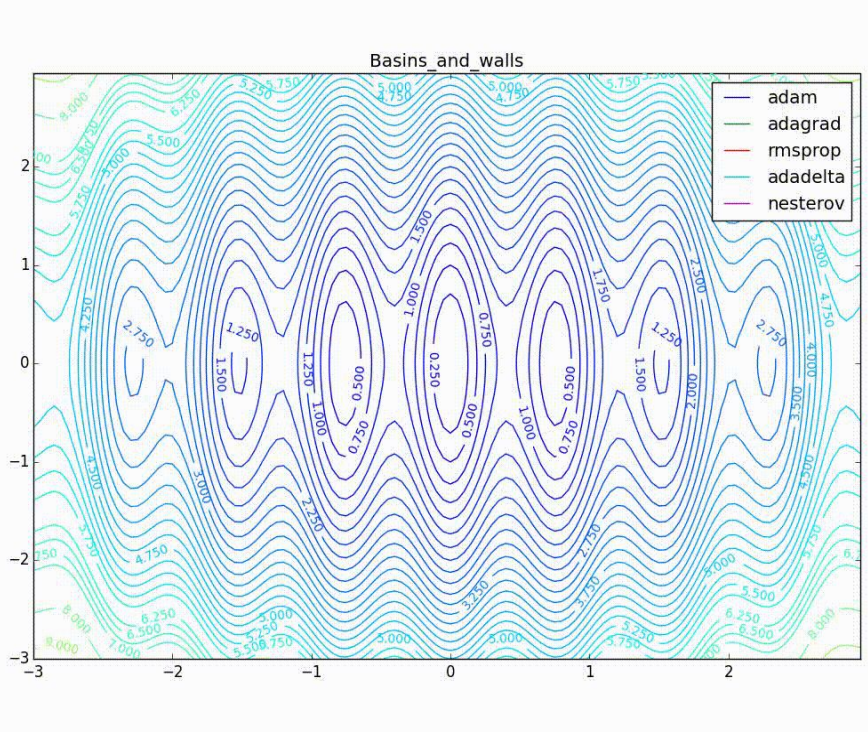
Примеры


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

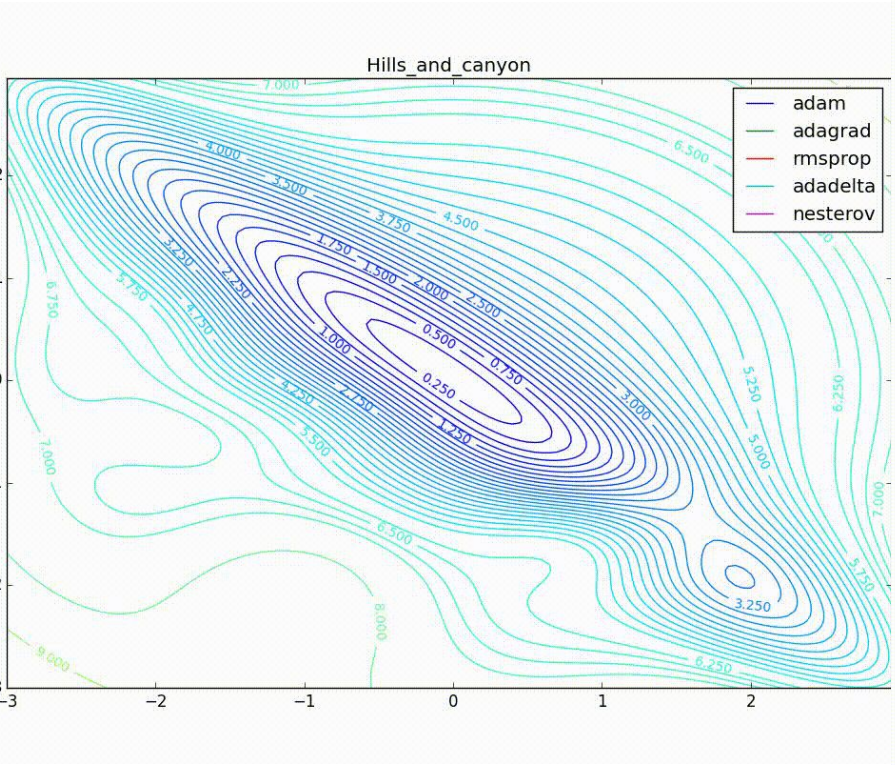

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

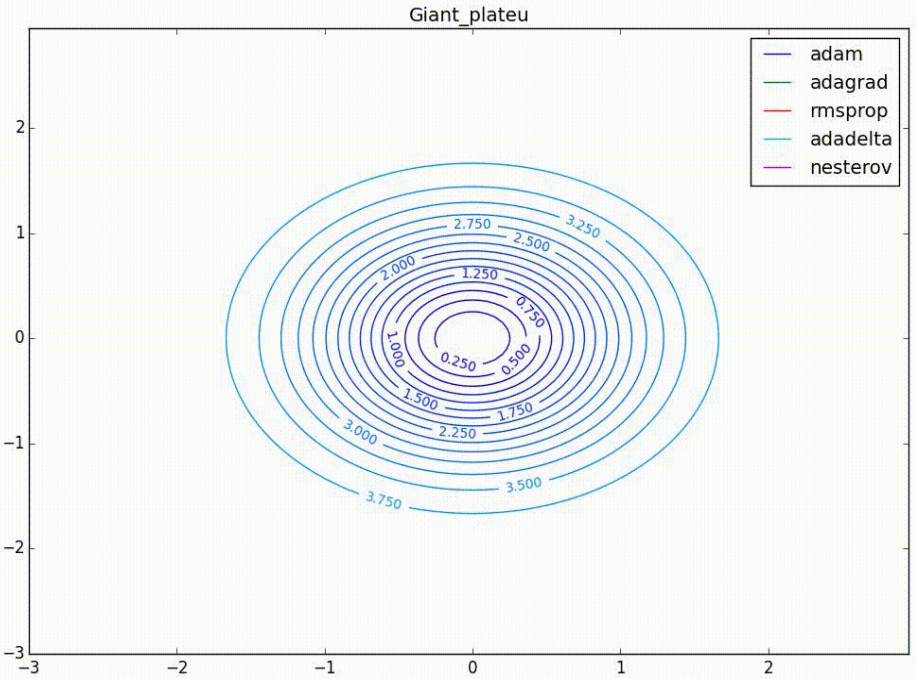
Примеры


Wobbly
adam
adagrad
rmsprop
adadelta
nesterov


Basins_and_walls
adam
adagrad
rmsprop
adadelta
nesterov

Примеры


Hills_and_canyon
adam
adagrad
rmsprop
adadelta
nesterov


Giant_plateu
adam
adagrad
rmsprop
adadelta
nesterov

Выводы

- Стохастика применяется повсеместно для избегания затратных компьютерных вычислений
- Каждый из методов имеет свои преимущества и недостатки
- С помощью дополнительных улучшений методов можно достичь хорошей сходимости для разного вида задач

Использованные материалы

- https://habr.com/post/318970/
- http://ruder.io/optimizing-gradient-descent/index.html#adagrad
- https://sgugger.github.io/the-1cycle-policy.html
- https://perso.telecom-paristech.fr/rgower/pdf/optimization_1-expanded.pdf
- https://pdfs.semanticscholar.org/presentation/00cb/cdc152439da3a7374a7b1f212a09cd00aa9e.pdf
- https://arxiv.org/pdf/1312.1666.pdf
- https://arxiv.org/pdf/1809.01225.pdf