

Стохастический градиентный спуск. Адаптивные методы

МЕТОДЫ ВЫПУКЛОЙ ОПТИМИЗАЦИИ

НЕДЕЛЯ 13

Даня Меркулов



Даня Меркулов, Петр Остроухов

Оптимизация для всех! ЦУ



Задача с конечной суммой

Задача с конечной суммой

Рассмотрим задачу минимизации среднего значения функции на конечной выборке:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n f_i(x)$$

Задача с конечной суммой

Рассмотрим задачу минимизации среднего значения функции на конечной выборке:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n f_i(x)$$

Шаг градиентного спуска для этой задачи:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(x) \quad (\text{GD})$$

Задача с конечной суммой

Рассмотрим задачу минимизации среднего значения функции на конечной выборке:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n f_i(x)$$

Шаг градиентного спуска для этой задачи:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(x) \quad (\text{GD})$$

- Сходимость с постоянным α или линейным поиском.

Задача с конечной суммой

Рассмотрим задачу минимизации среднего значения функции на конечной выборке:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n f_i(x)$$

Шаг градиентного спуска для этой задачи:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(x) \tag{GD}$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n . Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$. Для FineWeb $n \approx 15 \cdot 10^{12}$ токенов.

Задача с конечной суммой

Рассмотрим задачу минимизации среднего значения функции на конечной выборке:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n f_i(x)$$

Шаг градиентного спуска для этой задачи:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(x) \tag{GD}$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n . Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$. Для FineWeb $n \approx 15 \cdot 10^{12}$ токенов.

Задача с конечной суммой

Рассмотрим задачу минимизации среднего значения функции на конечной выборке:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n f_i(x)$$

Шаг градиентного спуска для этой задачи:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(x) \quad (\text{GD})$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n . Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$. Для FineWeb $n \approx 15 \cdot 10^{12}$ токенов.

Перейдем от вычисления полного градиента к его несмещенной оценке. На каждой итерации будем выбирать индекс i_k случайно и равномерно:

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k) \quad (\text{SGD})$$

При $p(i_k = i) = \frac{1}{n}$ стохастический градиент является несмещенной оценкой полного градиента:

$$\mathbb{E}[\nabla f_{i_k}(x)] = \sum_{i=1}^n p(i_k = i) \nabla f_i(x) = \sum_{i=1}^n \frac{1}{n} \nabla f_i(x) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x) = \nabla f(x)$$

Это означает, что математическое ожидание стохастического градиента совпадает с истинным градиентом $f(x)$.

Результаты для градиентного спуска

Стохастическая итерация в n раз дешевле, но сколько шагов потребуется для достижения заданной точности?

Результаты для градиентного спуска

Стохастическая итерация в n раз дешевле, но сколько шагов потребуется для достижения заданной точности?

Если ∇f липшицев, справедливы оценки:

Условие	GD	SGD
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

Результаты для градиентного спуска

Стохастическая итерация в n раз дешевле, но сколько шагов потребуется для достижения заданной точности?

Если ∇f липшицев, справедливы оценки:

Условие	GD	SGD
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- SGD имеет низкую стоимость итерации, но низкую скорость сходимости.

Результаты для градиентного спуска

Стохастическая итерация в n раз дешевле, но сколько шагов потребуется для достижения заданной точности?

Если ∇f липшицев, справедливы оценки:

Условие	GD	SGD
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- SGD имеет низкую стоимость итерации, но низкую скорость сходимости.
 - Сублинейная скорость даже в сильно выпуклом случае.

Результаты для градиентного спуска

Стохастическая итерация в n раз дешевле, но сколько шагов потребуется для достижения заданной точности?

Если ∇f липшицев, справедливы оценки:

Условие	GD	SGD
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- SGD имеет низкую стоимость итерации, но низкую скорость сходимости.
 - Сублинейная скорость даже в сильно выпуклом случае.
 - Оценки скорости не могут быть улучшены при стандартных предположениях.

Результаты для градиентного спуска

Стохастическая итерация в n раз дешевле, но сколько шагов потребуется для достижения заданной точности?

Если ∇f липшицев, справедливы оценки:

Условие	GD	SGD
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- SGD имеет низкую стоимость итерации, но низкую скорость сходимости.
 - Сублинейная скорость даже в сильно выпуклом случае.
 - Оценки скорости не могут быть улучшены при стандартных предположениях.
 - Оракул возвращает несмешенную аппроксимацию градиента с ограниченной дисперсией.

Результаты для градиентного спуска

Стохастическая итерация в n раз дешевле, но сколько шагов потребуется для достижения заданной точности?

Если ∇f липшицев, справедливы оценки:

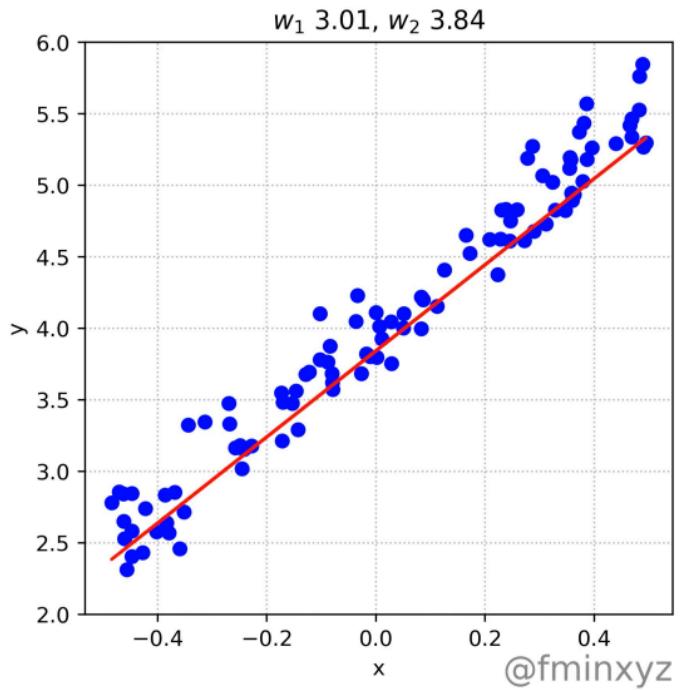
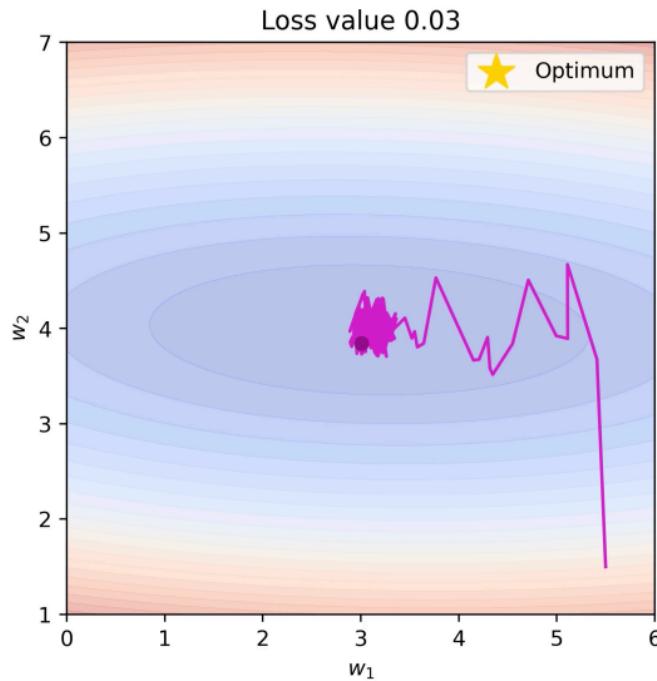
Условие	GD	SGD
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- SGD имеет низкую стоимость итерации, но низкую скорость сходимости.
 - Сублинейная скорость даже в сильно выпуклом случае.
 - Оценки скорости не могут быть улучшены при стандартных предположениях.
 - Оракул возвращает несмешенную аппроксимацию градиента с ограниченной дисперсией.
- Методы с ускорением и квазиньютоновские методы не улучшают асимптотическую скорость в стохастическом случае, влияя лишь на константы (узкое место — дисперсия, а не число обусловленности).

Стохастический градиентный спуск (SGD)

Типичное поведение

Stochastic Gradient Descent. Batch = 2



Гладкий PL-случай с постоянным шагом

- i** Пусть f — L -гладкая функция, удовлетворяющая условию Поляка-Лоясиевича (PL) с константой $\mu > 0$, а дисперсия стохастического градиента ограничена: $\mathbb{E}[\|\nabla f_i(x_k)\|^2] \leq \sigma^2$. Тогда стохастический градиентный спуск с постоянным шагом $\alpha < \frac{1}{2\mu}$ гарантирует

$$\mathbb{E}[f(x_k) - f^*] \leq (1 - 2\alpha\mu)^k [f(x_0) - f^*] + \frac{L\sigma^2\alpha}{4\mu}.$$

Гладкий выпуклый случай

Вспомогательные обозначения

Для (возможно) неконстантной последовательности шагов $(\alpha_t)_{t \geq 0}$ определим *взвешенное среднее*

$$\bar{x}_k \stackrel{\text{def}}{=} \frac{1}{\sum_{t=0}^{k-1} \alpha_t} \sum_{t=0}^{k-1} \alpha_t x_t, \quad k \geq 1.$$

Везде ниже $f^* \equiv \min_x f(x)$ и $x^* \in \arg \min_x f(x)$.

Гладкий выпуклый случай с постоянным шагом

- i** Пусть f — выпуклая функция (не обязательно гладкая), а дисперсия стохастического градиента ограничена $\mathbb{E}[\|\nabla f_{i_k}(x_k)\|^2] \leq \sigma^2 \forall k$. Если SGD использует постоянный шаг $\alpha_t \equiv \alpha > 0$, то для любого $k \geq 1$

$$\mathbb{E}[f(\bar{x}_k) - f^*] \leq \frac{\|x_0 - x^*\|^2}{2\alpha k} + \frac{\alpha \sigma^2}{2}$$

где $\bar{x}_k = \frac{1}{k} \sum_{t=0}^{k-1} x_t$.

При выборе постоянного $\alpha = \frac{\|x_0 - x^*\|}{\sigma\sqrt{k}}$ (зависящего от k) имеем

$$\mathbb{E}[f(\bar{x}_k) - f^*] \leq \frac{\|x_0 - x^*\|\sigma}{\sqrt{k}} = \mathcal{O}\left(\frac{1}{\sqrt{k}}\right).$$

Гладкий выпуклый случай с убывающим шагом

$$\alpha_k = \frac{\alpha_0}{\sqrt{k+1}}, \quad 0 < \alpha_0 \leq \frac{1}{4L}$$

i При тех же предположениях, но с убывающим шагом $\alpha_k = \frac{\alpha_0}{\sqrt{k+1}}$

$$\boxed{\mathbb{E}[f(\bar{x}_k) - f^*] \leq \frac{5\|x_0 - x^*\|^2}{4\alpha_0\sqrt{k}} + 5\alpha_0\sigma^2 \frac{\log(k+1)}{\sqrt{k}}} = \mathcal{O}\left(\frac{\log k}{\sqrt{k}}\right).$$



Мини-батч SGD

Мини-батч SGD

Детерминированный метод использует все n градиентов:

$$\nabla f(x_k) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k).$$

Мини-батч SGD

Детерминированный метод использует все n градиентов:

$$\nabla f(x_k) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k).$$

Стохастический метод аппроксимирует это, используя только один элемент:

$$\nabla f_{ik}(x_k) \approx \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k).$$

Мини-батч SGD

Детерминированный метод использует все n градиентов:

$$\nabla f(x_k) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k).$$

Стохастический метод аппроксимирует это, используя только один элемент:

$$\nabla f_{ik}(x_k) \approx \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k).$$

Распространённый вариант — использовать выборку элементов B_k («**мини-батч**»):

$$\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \approx \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k),$$

особенно полезно для векторизации и распараллеливания.

Мини-батч SGD

Детерминированный метод использует все n градиентов:

$$\nabla f(x_k) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k).$$

Стохастический метод аппроксимирует это, используя только один элемент:

$$\nabla f_{ik}(x_k) \approx \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k).$$

Распространённый вариант — использовать выборку элементов B_k («**мини-батч**»):

$$\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \approx \frac{1}{n} \sum_{i=1}^n \nabla f_i(x_k),$$

особенно полезно для векторизации и распараллеливания.

Например, имея 16 ядер, можно взять $|B_k| = 16$ и вычислить 16 градиентов параллельно.

Мини-батч как градиентный спуск с ошибкой

Метод SGD с батчем B_k («мини-батч») использует итерации:

$$x_{k+1} = x_k - \alpha_k \left(\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \right).$$

Мини-батч как градиентный спуск с ошибкой

Метод SGD с батчем B_k («мини-батч») использует итерации:

$$x_{k+1} = x_k - \alpha_k \left(\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \right).$$

Рассмотрим это как «градиентный метод с ошибкой»:

$$x_{k+1} = x_k - \alpha_k (\nabla f(x_k) + e_k),$$

где e_k — разница между аппроксимированным и истинным градиентом.

Мини-батч как градиентный спуск с ошибкой

Метод SGD с батчем B_k («мини-батч») использует итерации:

$$x_{k+1} = x_k - \alpha_k \left(\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \right).$$

Рассмотрим это как «градиентный метод с ошибкой»:

$$x_{k+1} = x_k - \alpha_k (\nabla f(x_k) + e_k),$$

где e_k — разница между аппроксимированным и истинным градиентом.

Если выбрать $\alpha_k = \frac{1}{L}$, то, согласно лемме о спуске:

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2 + \frac{1}{2L} \|e_k\|^2,$$

для любой ошибки e_k .

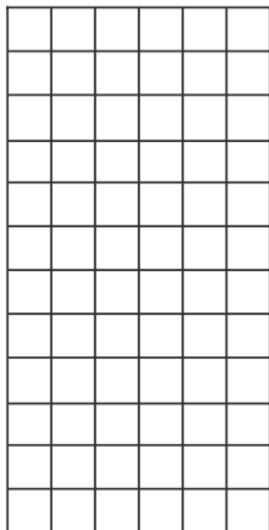
Влияние ошибки на скорость сходимости

Оценка прогресса при $\alpha_k = \frac{1}{L}$ и ошибке градиента e_k :

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2 + \frac{1}{2L} \|e_k\|^2.$$

Идея SGD и батчей

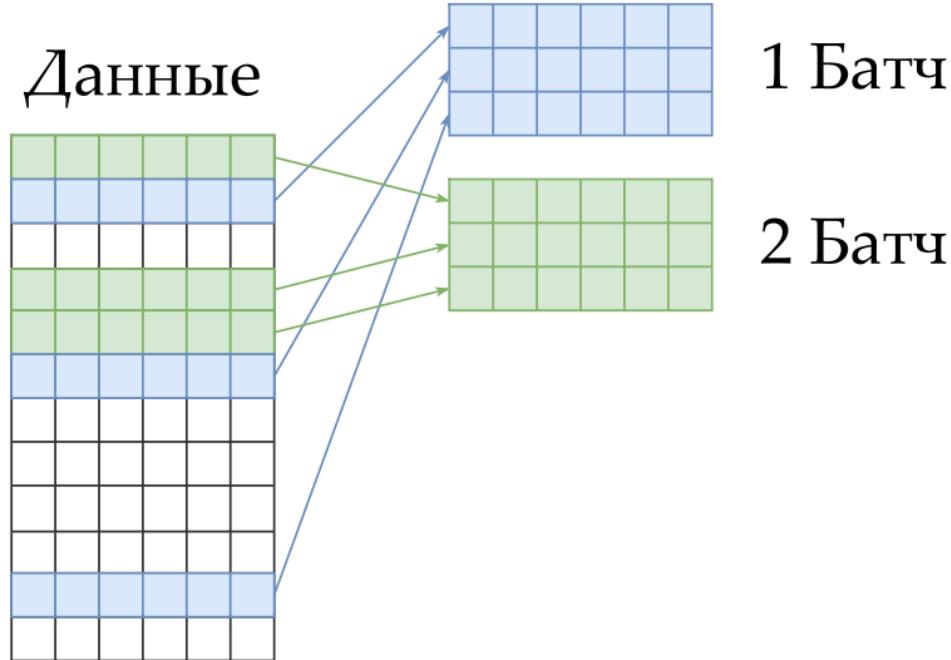
Данные



Идея SGD и батчей



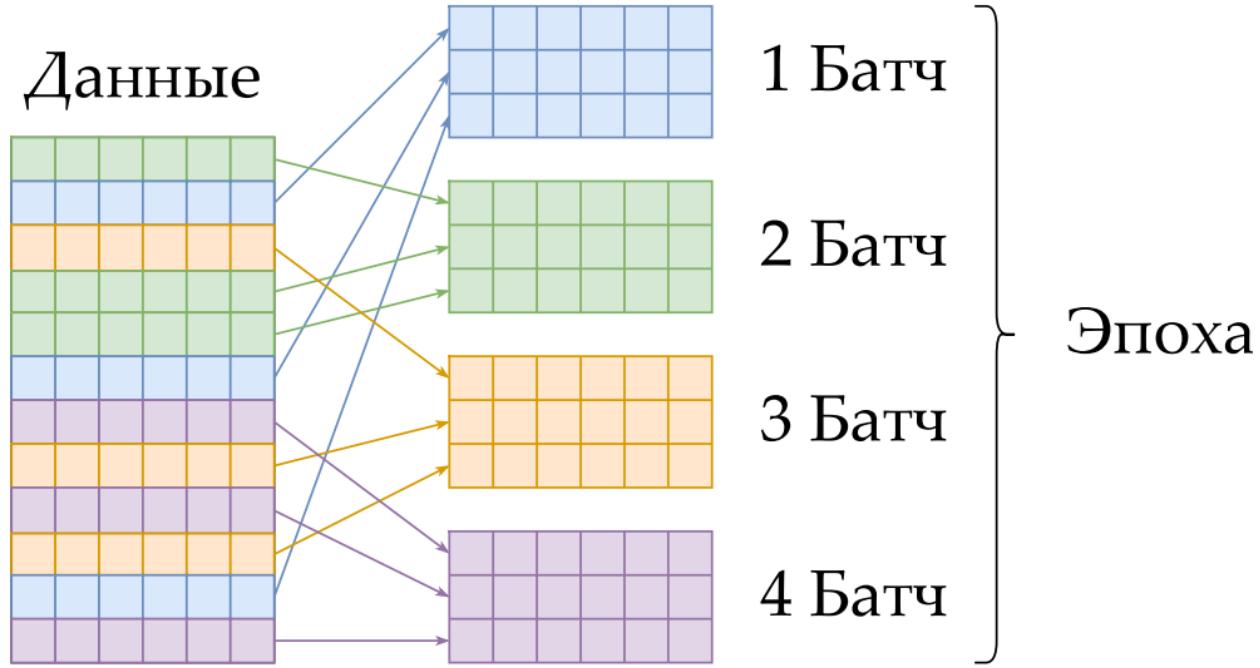
Идея SGD и батчей



Идея SGD и батчей



Идея SGD и батчей



Основная проблема SGD

$$f(x) = \frac{\mu}{2} \|x\|_2^2 + \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y_i \langle a_i, x \rangle)) \rightarrow \min_{x \in \mathbb{R}^n}$$

Strongly convex binary logistic regression. m=200, n=10, mu=1.



Основные результаты сходимости SGD

- Пусть f — L -гладкая μ -сильно выпуклая функция, а дисперсия стохастического градиента ограничена ($\mathbb{E}[\|\nabla f_i(x_k)\|^2] \leq \sigma^2$). Тогда траектория SGD с постоянным шагом $\alpha < \frac{1}{2\mu}$ будет гарантировать:

$$\mathbb{E}[f(x_{k+1}) - f^*] \leq (1 - 2\alpha\mu)^k [f(x_0) - f^*] + \frac{L\sigma^2\alpha}{4\mu}.$$

Основные результаты сходимости SGD

- i** Пусть f — L -гладкая μ -сильно выпуклая функция, а дисперсия стохастического градиента ограничена ($\mathbb{E}[\|\nabla f_i(x_k)\|^2] \leq \sigma^2$). Тогда траектория SGD с постоянным шагом $\alpha < \frac{1}{2\mu}$ будет гарантировать:

$$\mathbb{E}[f(x_{k+1}) - f^*] \leq (1 - 2\alpha\mu)^k [f(x_0) - f^*] + \frac{L\sigma^2\alpha}{4\mu}.$$

- i** Пусть f — L -гладкая μ -сильно выпуклая функция, а дисперсия стохастического градиента ограничена ($\mathbb{E}[\|\nabla f_i(x_k)\|^2] \leq \sigma^2$). Тогда SGD с убывающим шагом $\alpha_k = \frac{2k+1}{2\mu(k+1)^2}$ будет сходиться сублинейно:

$$\mathbb{E}[f(x_{k+1}) - f^*] \leq \frac{L\sigma^2}{2\mu^2(k+1)}$$

Summary

- SGD с постоянным шагом не сходится к оптимуму даже в PL (или сильно выпуклом) случае.

Summary

- SGD с постоянным шагом не сходится к оптимуму даже в PL (или сильно выпуклом) случае.
- SGD сходится сублинейно со скоростью $\mathcal{O}\left(\frac{1}{k}\right)$ для PL-случая.

Summary

- SGD с постоянным шагом не сходится к оптимуму даже в PL (или сильно выпуклом) случае.
- SGD сходится сублинейно со скоростью $\mathcal{O}\left(\frac{1}{k}\right)$ для PL-случая.
- Ускорение Нестерова/Поляка не улучшает скорость сходимости.

Summary

- SGD с постоянным шагом не сходится к оптимуму даже в PL (или сильно выпуклом) случае.
- SGD сходится сублинейно со скоростью $\mathcal{O}\left(\frac{1}{k}\right)$ для PL-случая.
- Ускорение Нестерова/Поляка не улучшает скорость сходимости.
- Двухфазный метод Ньютона достигает $\mathcal{O}\left(\frac{1}{k}\right)$ без сильной выпуклости.

Адаптивность или масштабирование



Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Популярный адаптивный метод. Обозначим $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Популярный адаптивный метод. Обозначим $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки шага обучения: $\alpha > 0$ — фиксированная константа, и шаг обучения автоматически уменьшается в ходе итераций.

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Популярный адаптивный метод. Обозначим $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки шага обучения: $\alpha > 0$ — фиксированная константа, и шаг обучения автоматически уменьшается в ходе итераций.
- Шаг обучения для редких информативных признаков убывает медленно.

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Популярный адаптивный метод. Обозначим $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки шага обучения: $\alpha > 0$ — фиксированная константа, и шаг обучения автоматически уменьшается в ходе итераций.
- Шаг обучения для редких информативных признаков убывает медленно.
- Может существенно превосходить SGD на разреженных задачах.

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Популярный адаптивный метод. Обозначим $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки шага обучения: $\alpha > 0$ — фиксированная константа, и шаг обучения автоматически уменьшается в ходе итераций.
- Шаг обучения для редких информативных признаков убывает медленно.
- Может существенно превосходить SGD на разреженных задачах.
- Основной недостаток — монотонное накопление квадратов градиентов в знаменателе. AdaDelta, Adam, AMSGrad и др. улучшают это, популярны в обучении глубоких нейронных сетей.

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Популярный адаптивный метод. Обозначим $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки шага обучения: $\alpha > 0$ — фиксированная константа, и шаг обучения автоматически уменьшается в ходе итераций.
- Шаг обучения для редких информативных признаков убывает медленно.
- Может существенно превосходить SGD на разреженных задачах.
- Основной недостаток — монотонное накопление квадратов градиентов в знаменателе. AdaDelta, Adam, AMSGrad и др. улучшают это, популярны в обучении глубоких нейронных сетей.
- Константа ϵ обычно устанавливается в 10^{-6} для предотвращения деления на ноль.

RMSProp (Tieleman and Hinton, 2012)

Модификация AdaGrad, устраняющая проблему агрессивного монотонного убывания шага. Использует экспоненциальное скользящее среднее квадратов градиентов для настройки шага по каждой координате переменной. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1 - \gamma)(g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

RMSProp (Tieleman and Hinton, 2012)

Модификация AdaGrad, устраняющая проблему агрессивного монотонного убывания шага. Использует экспоненциальное скользящее среднее квадратов градиентов для настройки шага по каждой координате переменной. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

$$\begin{aligned}v_j^{(k)} &= \gamma v_j^{(k-1)} + (1 - \gamma)(g_j^{(k)})^2 \\x_j^{(k)} &= x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}\end{aligned}$$

Заметки:

- RMSProp нормирует шаг обучения на корень из скользящего среднего квадратов градиентов.

RMSProp (Tieleman and Hinton, 2012)

Модификация AdaGrad, устраняющая проблему агрессивного монотонного убывания шага. Использует экспоненциальное скользящее среднее квадратов градиентов для настройки шага по каждой координате переменной. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

$$\begin{aligned}v_j^{(k)} &= \gamma v_j^{(k-1)} + (1 - \gamma)(g_j^{(k)})^2 \\x_j^{(k)} &= x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}\end{aligned}$$

Заметки:

- RMSProp нормирует шаг обучения на корень из скользящего среднего квадратов градиентов.
- Обеспечивает более тонкую настройку шагов обучения, чем AdaGrad, что делает его подходящим для нестационарных задач.

RMSProp (Tieleman and Hinton, 2012)

Модификация AdaGrad, устраняющая проблему агрессивного монотонного убывания шага. Использует экспоненциальное скользящее среднее квадратов градиентов для настройки шага по каждой координате переменной. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

$$\begin{aligned}v_j^{(k)} &= \gamma v_j^{(k-1)} + (1 - \gamma)(g_j^{(k)})^2 \\x_j^{(k)} &= x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}\end{aligned}$$

Заметки:

- RMSProp нормирует шаг обучения на корень из скользящего среднего квадратов градиентов.
- Обеспечивает более тонкую настройку шагов обучения, чем AdaGrad, что делает его подходящим для нестационарных задач.
- Широко используется при обучении нейронных сетей, особенно рекуррентных.

Adam (Kingma and Ba, 2014)¹²



Объединяет элементы из AdaGrad и RMSProp. Использует экспоненциальное скользящее среднее как градиентов, так и их квадратов.

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2$$

Коррекция смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Adam (Kingma and Ba, 2014) ¹²



Объединяет элементы из AdaGrad и RMSProp. Использует экспоненциальное скользящее среднее как градиентов, так и их квадратов.

EMA:

$$\begin{aligned}m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2\end{aligned}$$

Коррекция смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Компенсирует смещение к нулю на начальных итерациях, наблюдаемое в других методах (например, RMSProp), что делает оценки более точными.

Adam (Kingma and Ba, 2014) ¹²



Объединяет элементы из AdaGrad и RMSProp. Использует экспоненциальное скользящее среднее как градиентов, так и их квадратов.

EMA:

$$\begin{aligned}m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2\end{aligned}$$

Коррекция смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Компенсирует смещение к нулю на начальных итерациях, наблюдаемое в других методах (например, RMSProp), что делает оценки более точными.
- Одна из самых цитируемых научных работ в мире.

Adam (Kingma and Ba, 2014) ¹²



Объединяет элементы из AdaGrad и RMSProp. Использует экспоненциальное скользящее среднее как градиентов, так и их квадратов.

EMA:

$$\begin{aligned}m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2\end{aligned}$$

Коррекция смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Компенсирует смещение к нулю на начальных итерациях, наблюдаемое в других методах (например, RMSProp), что делает оценки более точными.
- Одна из самых цитируемых научных работ в мире.
- В 2018-2019 годах вышли статьи, указывающие на ошибку в оригинальной статье

Adam (Kingma and Ba, 2014) ¹²



Объединяет элементы из AdaGrad и RMSProp. Использует экспоненциальное скользящее среднее как градиентов, так и их квадратов.

EMA:

$$\begin{aligned}m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2\end{aligned}$$

Коррекция смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Компенсирует смещение к нулю на начальных итерациях, наблюдаемое в других методах (например, RMSProp), что делает оценки более точными.
- Одна из самых цитируемых научных работ в мире.
- В 2018-2019 годах вышли статьи, указывающие на ошибку в оригинальной статье
- Не сходится для некоторых простых задач (даже выпуклых)

Adam (Kingma and Ba, 2014) ¹²



Объединяет элементы из AdaGrad и RMSProp. Использует экспоненциальное скользящее среднее как градиентов, так и их квадратов.

EMA:

$$\begin{aligned}m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2\end{aligned}$$

Коррекция смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Компенсирует смещение к нулю на начальных итерациях, наблюдаемое в других методах (например, RMSProp), что делает оценки более точными.
- Одна из самых цитируемых научных работ в мире.
- В 2018-2019 годах вышли статьи, указывающие на ошибку в оригинальной статье
- Не сходится для некоторых простых задач (даже выпуклых)
- Почему-то очень хорошо работает для некоторых сложных задач

Adam (Kingma and Ba, 2014)¹²



Объединяет элементы из AdaGrad и RMSProp. Использует экспоненциальное скользящее среднее как градиентов, так и их квадратов.

EMA:

$$\begin{aligned}m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2\end{aligned}$$

Коррекция смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Компенсирует смещение к нулю на начальных итерациях, наблюдаемое в других методах (например, RMSProp), что делает оценки более точными.
- Одна из самых цитируемых научных работ в мире.
- В 2018-2019 годах вышли статьи, указывающие на ошибку в оригинальной статье
- Не сходится для некоторых простых задач (даже выпуклых)
- Почему-то очень хорошо работает для некоторых сложных задач
- Работает для языковых моделей значительно лучше, чем для задач компьютерного зрения. Почему?

¹Adam: A Method for Stochastic Optimization

²On the Convergence of Adam and Beyond

AdamW (Loshchilov & Hutter, 2017)

Решает проблему ℓ_2 -регуляризации в Adam. Стандартная ℓ_2 -регуляризация добавляет $\lambda \|x\|^2$ к функции потерь, что дает добавку λx к градиенту. В Adam эта добавка масштабируется адаптивным шагом обучения $(\sqrt{\hat{v}_j} + \epsilon)$, связывая затухание весов (weight decay) с величиной шага. AdamW отделяет затухание весов от адаптации шага.

Правило обновления:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2$$

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)$$

AdamW (Loshchilov & Hutter, 2017)

Решает проблему ℓ_2 -регуляризации в Adam. Стандартная ℓ_2 -регуляризация добавляет $\lambda \|x\|^2$ к функции потерь, что дает добавку λx к градиенту. В Adam эта добавка масштабируется адаптивным шагом обучения $(\sqrt{\hat{v}_j} + \epsilon)$, связывая затухание весов (weight decay) с величиной шага. AdamW отделяет затухание весов от адаптации шага.

Правило обновления:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2$$

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)$$

Заметки:

- Слагаемое затухания весов $\lambda x_j^{(k-1)}$ добавляется после адаптивного шага по градиенту.

AdamW (Loshchilov & Hutter, 2017)

Решает проблему ℓ_2 -регуляризации в Adam. Стандартная ℓ_2 -регуляризация добавляет $\lambda \|x\|^2$ к функции потерь, что дает добавку λx к градиенту. В Adam эта добавка масштабируется адаптивным шагом обучения $(\sqrt{\hat{v}_j} + \epsilon)$, связывая затухание весов (weight decay) с величиной шага. AdamW отделяет затухание весов от адаптации шага.

Правило обновления:

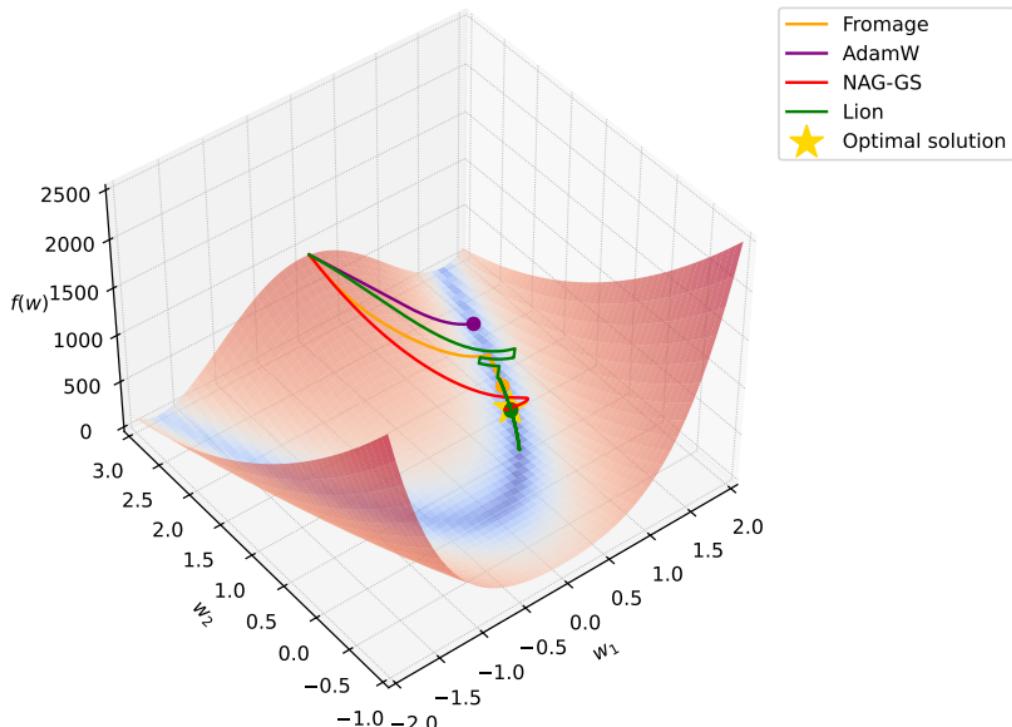
$$\begin{aligned} m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\ v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2 \\ \hat{m}_j &= \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k} \\ x_j^{(k)} &= x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right) \end{aligned}$$

Заметки:

- Слагаемое затухания весов $\lambda x_j^{(k-1)}$ добавляется после адаптивного шага по градиенту.
- Широко используется в обучении трансформаторов и других крупных моделей. Вариант по умолчанию для Hugging Face Trainer.

Много методов

Rosenbrock Function.
Adaptive stochastic gradient algorithms.
Learning rate 0.003

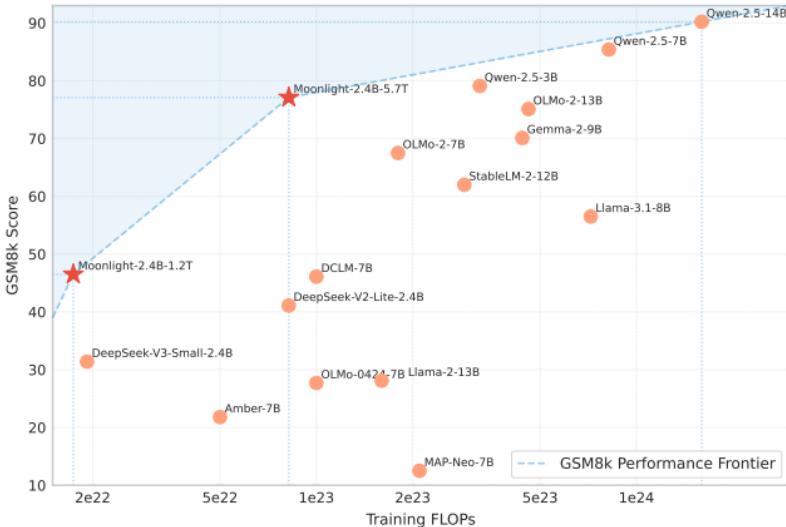
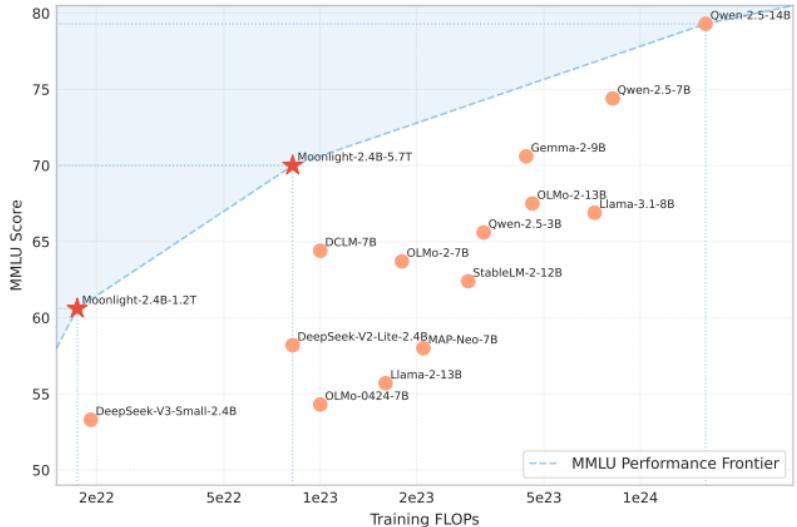




Новый подход к оптимизации



3



Модели, отмеченные звёздочкой, были обучены методом Мион, остальные модели были обучены другими алгоритмами оптимизации.

³KIMI K2: OPEN AGENTIC INTELLIGENCE

Интуиция за методом Мион⁴



$$\min_{x \in \mathbb{R}^p} f(x)$$

$$f(x) = f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \mathcal{O}(\|x - x_k\|_2^2).$$

⁴Презентация R. Gower

Интуиция за методом Мион⁴

$$\min_{x \in \mathbb{R}^p} f(x)$$

Функция потерь

$$f(x) = f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \mathcal{O}(\|x - x_k\|_2^2).$$

Интуиция за методом Мион⁴

$$\min_{x \in \mathbb{R}^p} f(x)$$

Функция потерь

$$f(x) = \underbrace{f(x_k) + \langle \nabla f(x_k), x - x_k \rangle}_{\text{Линейная аппроксимация}} + \mathcal{O}(\|x - x_k\|_2^2).$$

Линейная
аппроксимация

Интуиция за методом Мион⁴

$$\min_{x \in \mathbb{R}^p} f(x)$$

Функция потерь

$$f(x) = \underbrace{f(x_k) + \langle \nabla f(x_k), x - x_k \rangle}_{\text{Линейная аппроксимация}} + \mathcal{O}(\|x - x_k\|_2^2).$$

Хорошее приближение
в окрестности x_k

Интуиция за методом Мион. Градиентный спуск



$$x_{k+1} = \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right)$$

Интуиция за методом Мион. Градиентный спуск



$$\begin{aligned}x_{k+1} &= \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right) \\&= x_k - \alpha \nabla f(x_k)\end{aligned}$$

Интуиция за методом Мион. Градиентный спуск



Штраф за
дальность от x_k

$$\begin{aligned}x_{k+1} &= \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right) \\&= x_k - \alpha \nabla f(x_k)\end{aligned}$$

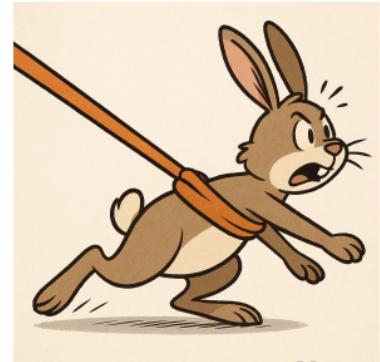
Интуиция за методом Мион. Градиентный спуск



$$\begin{aligned}x_{k+1} &= \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right) \\&= x_k - \alpha \nabla f(x_k)\end{aligned}$$

Шаг обучения /
коэффициент регуляризации

Штраф за
дальность от x_k



Интуиция за методом Мион. Нормированный градиентный спуск

$$x_{k+1} = \underset{\|x - x_k\|_2 = \alpha}{\operatorname{argmin}} (f(x_k) + \langle \nabla f(x_k), x - x_k \rangle)$$

Интуиция за методом Мион. Нормированный градиентный спуск

$$\begin{aligned}x_{k+1} &= \underset{\|x - x_k\|_2 = \alpha}{\operatorname{argmin}} (f(x_k) + \langle \nabla f(x_k), x - x_k \rangle) \\&= x_k - \alpha \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|_2}\end{aligned}$$

Интуиция за методом Мион. Нормированный градиентный спуск

$$\begin{aligned}x_{k+1} &= \underset{\|x - x_k\|_2 = \alpha}{\operatorname{argmin}} (f(x_k) + \langle \nabla f(x_k), x - x_k \rangle) \\&= x_k - \alpha \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|_2}\end{aligned}$$

Ограничение на
длину шага



Интуиция за методом Мион. Нормированный градиентный спуск

$$\begin{aligned}x_{k+1} &= \underset{\|x - x_k\|_2 = \alpha}{\operatorname{argmin}} (f(x_k) + \langle \nabla f(x_k), x - x_k \rangle) \\&= x_k - \alpha \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|_2}\end{aligned}$$

Параметр ограничения / шаг обучения

Ограничение на длину шага



Что насчёт других норм?

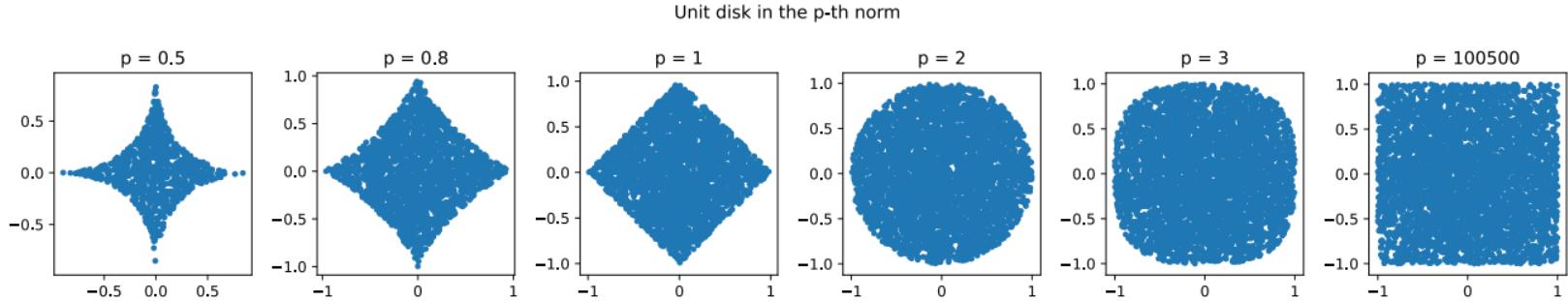


Рисунок 2. Примеры шаров в разных нормах

Неевклидовы записи методов⁵

Linear Minimization Oracle:

$$\text{LMO}_{\|\cdot\|}(g) = \underset{\|x\|=1}{\operatorname{argmin}} \langle g, x \rangle$$

! Неевклидов градиентный спуск

Для вектора градиента $g = \nabla f(x_k)$ и шага $\alpha > 0$:

$$x_{k+1} = \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left(f(x_k) + \langle g, x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|^2 \right)$$

⁵Old Optimizer, New Norm: An Anthology

Неевклидовы записи методов⁵

Linear Minimization Oracle:

$$\text{LMO}_{\|\cdot\|}(g) = \underset{\|x\|=1}{\operatorname{argmin}} \langle g, x \rangle$$

! Неевклидов градиентный спуск

Для вектора градиента $g = \nabla f(x_k)$ и шага $\alpha > 0$:

$$x_{k+1} = \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left(f(x_k) + \langle g, x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|^2 \right)$$

! Неевклидов нормированный градиентный спуск

Для вектора градиента $g = \nabla f(x_k)$ и шага $\alpha > 0$:

$$\begin{aligned} x_{k+1} &= \underset{\|x-x_k\|=\alpha}{\operatorname{argmin}} (f(x_k) + \langle g, x - x_k \rangle) \\ &= x_k + \alpha \text{LMO}_{\|\cdot\|}(g) \end{aligned}$$

⁵Old Optimizer, New Norm: An Anthology

В нейросетях параметры — матрицы

- В линейных слоях, attention, embedding-слоях параметр — матрица весов

$$W \in \mathbb{R}^{d \times n}, \quad G_k = \nabla_W f(W_k) \in \mathbb{R}^{d \times n}.$$

В нейросетях параметры — матрицы

- В линейных слоях, attention, embedding-слоях параметр — матрица весов

$$W \in \mathbb{R}^{d \times n}, \quad G_k = \nabla_W f(W_k) \in \mathbb{R}^{d \times n}.$$

- Естественно использовать **матричные нормы**: операторную $\|\cdot\|_{\text{op}}$, ядерную $\|\cdot\|_{\text{nuc}}$, Фробениуса $\|\cdot\|_F$ и т.п.

В нейросетях параметры — матрицы

- В линейных слоях, attention, embedding-слоях параметр — матрица весов

$$W \in \mathbb{R}^{d \times n}, \quad G_k = \nabla_W f(W_k) \in \mathbb{R}^{d \times n}.$$

- Естественно использовать **матричные нормы**: операторную $\|\cdot\|_{\text{оп}}$, ядерную $\|\cdot\|_{\text{nuc}}$, Фробениуса $\|\cdot\|_F$ и т.п.
- Вся логика переносится: вместо вектора ищем «лучшее направление спуска» среди матриц заданной длины.

В нейросетях параметры — матрицы

- В линейных слоях, attention, embedding-слоях параметр — матрица весов

$$W \in \mathbb{R}^{d \times n}, \quad G_k = \nabla_W f(W_k) \in \mathbb{R}^{d \times n}.$$

- Естественно использовать **матричные нормы**: операторную $\|\cdot\|_{\text{оп}}$, ядерную $\|\cdot\|_{\text{nuc}}$, Фробениуса $\|\cdot\|_F$ и т.п.
- Вся логика переносится: вместо вектора ищем «лучшее направление спуска» среди матриц заданной длины.
- Скалярное произведение:

$$\langle A, B \rangle := \text{tr}(A^\top B) = \sum_{ij} A_{ij} B_{ij}.$$

Неевклидов нормированный спуск для матриц

Пусть заданы матричная норма $\|\cdot\|$ и шаг $\lambda > 0$. Тогда нормированный шаг по матрице W :

$$W_{k+1} = \underset{\|W-W_k\|=\lambda}{\operatorname{argmin}} \left(f(W_k) + \langle G_k, W - W_k \rangle \right) = W_k + \lambda \text{LMO}_{\|\cdot\|}(G_k),$$

где

$$\text{LMO}_{\|\cdot\|}(G) = \underset{\|W\|=1}{\operatorname{argmin}} \langle G, W \rangle$$

— тот же самый LMO, только теперь он ищет **матрицу** единичной нормы, дающую наибольшее убывание линейного приближения.

Операторная норма и быстрый расчёт (UV^\top)

Рассмотрим операторную (спектральную) норму $\|\cdot\|_{\text{op}}$. Пусть

$$G_k = U\Sigma V^\top$$

— редуцированное SVD градиента. Тогда

Операторная норма и быстрый расчёт (UV^\top)

Рассмотрим операторную (спектральную) норму $\|\cdot\|_{\text{op}}$. Пусть

$$G_k = U\Sigma V^\top$$

— редуцированное SVD градиента. Тогда

- LMO (с «max»-формулировкой) по операторной норме:

$$\text{LMO}_{\|\cdot\|}(G) = -UV^\top,$$

то есть оптимальное направление — **polar factor** (matrix sign) матрицы G_k .

Операторная норма и быстрый расчёт (UV^\top)

Рассмотрим операторную (спектральную) норму $\|\cdot\|_{\text{оп}}$. Пусть

$$G_k = U\Sigma V^\top$$

— редуцированное SVD градиента. Тогда

- LMO (с «max»-формулировкой) по операторной норме:

$$\text{LMO}_{\|\cdot\|}(G) = -UV^\top,$$

то есть оптимальное направление — **polar factor** (matrix sign) матрицы G_k .

- Проблема: полное SVD на каждом шаге дорого. Хорошая новость: нам нужен только (UV^\top) , его можно считать гораздо быстрее:

Операторная норма и быстрый расчёт (UV^\top)

Рассмотрим операторную (спектральную) норму $\|\cdot\|_{\text{оп}}$. Пусть

$$G_k = U\Sigma V^\top$$

— редуцированное SVD градиента. Тогда

- LMO (с «max»-формулировкой) по операторной норме:

$$\text{LMO}_{\|\cdot\|}(G) = -UV^\top,$$

то есть оптимальное направление — **polar factor** (matrix sign) матрицы G_k .

- Проблема: полное SVD на каждом шаге дорого. Хорошая новость: нам нужен только (UV^\top) , его можно считать гораздо быстрее:
- итерациями **Newton–Schulz/ Polar Express**, которые используют только матричные умножения, дают приближение UV^\top за несколько шагов и снимают узкое место полного SVD внутри Muon.

Обучение GPT-2 (124М) на FineWeb



Рисунок 3. NanoGPT speedrun

Обучение GPT-2 (124М) на FineWeb

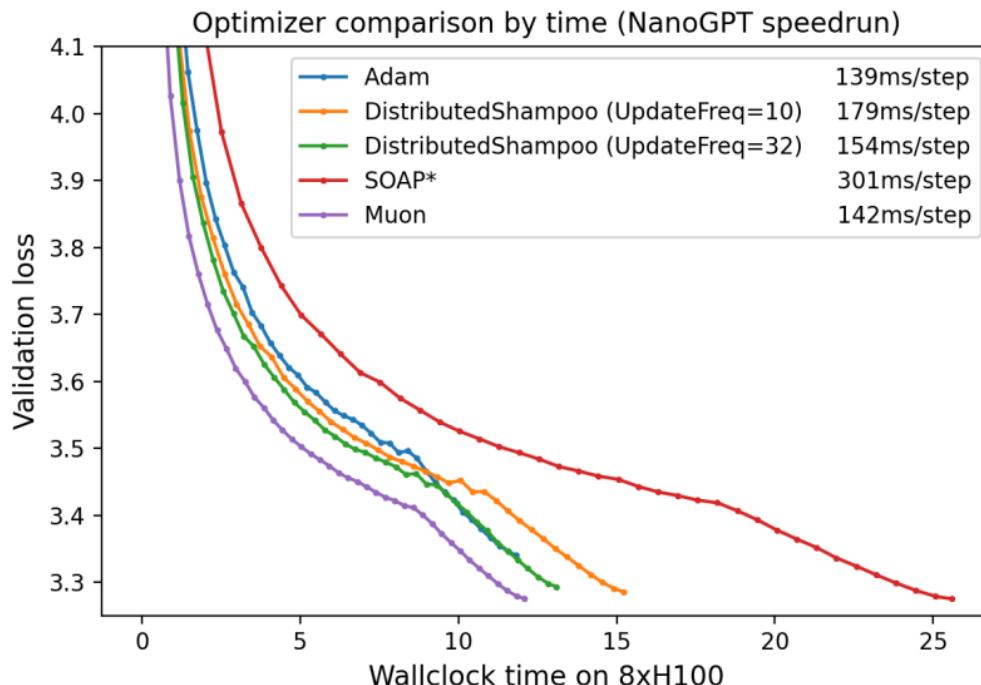


Рисунок 4. NanoGPT speedrun

Бонус: больше методов

Shampoo (Gupta, Anil, et al., 2018; Anil et al., 2020)



Расшифровывается как **Stochastic Hessian-Approximation Matrix Preconditioning for Optimization Of** deep networks: стохастическое предобуславливание матрицей, основанной на аппроксимации гессиана, для оптимизации глубоких сетей. Это метод, вдохновлённый оптимизацией второго порядка и рассчитанный на крупномасштабное глубокое обучение.

Основная идея: аппроксимировать полноматричный предобуславливатель AdaGrad с помощью эффективных матричных структур, в частности произведений Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$, обновление включает предобуславливание с использованием приближений статистических матриц $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощённая концепция:

1. Вычислить градиент G_k .

Shampoo (Gupta, Anil, et al., 2018; Anil et al., 2020)



Расшифровывается как **Stochastic Hessian-Approximation Matrix Preconditioning for Optimization Of** deep networks: стохастическое предобуславливание матрицей, основанной на аппроксимации гессиана, для оптимизации глубоких сетей. Это метод, вдохновлённый оптимизацией второго порядка и рассчитанный на крупномасштабное глубокое обучение.

Основная идея: аппроксимировать полноматричный предобуславливатель AdaGrad с помощью эффективных матричных структур, в частности произведений Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$, обновление включает предобуславливание с использованием приближений статистических матриц $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощённая концепция:

1. Вычислить градиент G_k .
2. Обновить статистику $L_k = \beta L_{k-1} + (1 - \beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1 - \beta) G_k^T G_k$.

Shampoo (Gupta, Anil, et al., 2018; Anil et al., 2020)



Расшифровывается как **Stochastic Hessian-Approximation Matrix Preconditioning for Optimization Of** deep networks: стохастическое предобуславливание матрицей, основанной на аппроксимации гессиана, для оптимизации глубоких сетей. Это метод, вдохновлённый оптимизацией второго порядка и рассчитанный на крупномасштабное глубокое обучение.

Основная идея: аппроксимировать полноматричный предобуславливатель AdaGrad с помощью эффективных матричных структур, в частности произведений Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$, обновление включает предобуславливание с использованием приближений статистических матриц $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощённая концепция:

1. Вычислить градиент G_k .
2. Обновить статистику $L_k = \beta L_{k-1} + (1 - \beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1 - \beta) G_k^T G_k$.
3. Вычислить предобуславливатели $P_L = L_k^{-1/4}$ и $P_R = R_k^{-1/4}$. (Обратный корень матрицы)

Shampoo (Gupta, Anil, et al., 2018; Anil et al., 2020)



Расшифровывается как **Stochastic Hessian-Approximation Matrix Preconditioning for Optimization Of** deep networks: стохастическое предобуславливание матрицей, основанной на аппроксимации гессиана, для оптимизации глубоких сетей. Это метод, вдохновлённый оптимизацией второго порядка и рассчитанный на крупномасштабное глубокое обучение.

Основная идея: аппроксимировать полноматричный предобуславливатель AdaGrad с помощью эффективных матричных структур, в частности произведений Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$, обновление включает предобуславливание с использованием приближений статистических матриц $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощённая концепция:

1. Вычислить градиент G_k .
2. Обновить статистику $L_k = \beta L_{k-1} + (1 - \beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1 - \beta) G_k^T G_k$.
3. Вычислить предобуславливатели $P_L = L_k^{-1/4}$ и $P_R = R_k^{-1/4}$. (Обратный корень матрицы)
4. Update: $W_{k+1} = W_k - \alpha P_L G_k P_R$.



Shampoo (Gupta, Anil, et al., 2018; Anil et al., 2020)

Заметки:

- Цель — эффективнее учитывать информацию о кривизне, чем методы первого порядка.

Shampoo (Gupta, Anil, et al., 2018; Anil et al., 2020)

Заметки:

- Цель — эффективнее учитывать информацию о кривизне, чем методы первого порядка.
- Вычислительно дороже, чем Adam, но может сходиться быстрее или приводить к лучшим решениям по числу шагов.

Shampoo (Gupta, Anil, et al., 2018; Anil et al., 2020)



Заметки:

- Цель — эффективнее учитывать информацию о кривизне, чем методы первого порядка.
- Вычислительно дороже, чем Adam, но может сходиться быстрее или приводить к лучшим решениям по числу шагов.
- Требует аккуратной реализации для эффективности (например, эффективного вычисления корней из обратной матрицы, обработки больших матриц).

Shampoo (Gupta, Anil, et al., 2018; Anil et al., 2020)



Заметки:

- Цель — эффективнее учитывать информацию о кривизне, чем методы первого порядка.
- Вычислительно дороже, чем Adam, но может сходиться быстрее или приводить к лучшим решениям по числу шагов.
- Требует аккуратной реализации для эффективности (например, эффективного вычисления корней из обратной матрицы, обработки больших матриц).
- Существуют варианты для разных форм тензоров (например, для свёрточных слоёв).

Muon⁶



$$\begin{aligned}W_{t+1} &= W_t - \eta(G_t G_t^\top)^{-1/4} G_t (G_t^\top G_t)^{-1/4} \\&= W_t - \eta(US^2U^\top)^{-1/4}(USV^\top)(VS^2V^\top)^{-1/4} \\&= W_t - \eta(US^{-1/2}U^\top)(USV^\top)(VS^{-1/2}V^\top) \\&= W_t - \eta US^{-1/2} SS^{-1/2} V^\top \\&= W_t - \eta UV^\top\end{aligned}$$

⁶Deriving Muon