

# Стохастический градиентный спуск. Адаптивные методы. Оптимизация нейронных сетей

МЕТОДЫ ВЫПУКЛОЙ ОПТИМИЗАЦИИ

НЕДЕЛЯ 14

Даня Меркулов  
Пётр Остроухов



# Даня Меркулов, Петр Остроухов

Оптимизация для всех! ЦУ



# Задача с конечной суммой

# Задача с конечной суммой

Рассмотрим классическую задачу минимизации среднего по 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 классическую задачу минимизации среднего по конечной выборке:

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

Градиентный спуск для этой задачи записывается следующим образом:

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

# Задача с конечной суммой

Рассмотрим классическую задачу минимизации среднего по конечной выборке:

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

Градиентный спуск для этой задачи записывается следующим образом:

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

- Сходимость с постоянным  $\alpha$  или линейным поиском.

# Задача с конечной суммой

Рассмотрим классическую задачу минимизации среднего по конечной выборке:

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

Градиентный спуск для этой задачи записывается следующим образом:

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

- Сходимость с постоянным  $\alpha$  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 классическую задачу минимизации среднего по конечной выборке:

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

Градиентный спуск для этой задачи записывается следующим образом:

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

- Сходимость с постоянным  $\alpha$  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 классическую задачу минимизации среднего по конечной выборке:

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

Градиентный спуск для этой задачи записывается следующим образом:

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

- Сходимость с постоянным  $\alpha$  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.

Давайте перейдем от полного вычисления градиента к его несмешенной оценке, когда мы случайным образом выбираем индекс  $i_k$  точки на каждой итерации равномерно:

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

С  $p(i_k = i) = \frac{1}{n}$ , стохастический градиент является несмешенной оценкой градиента, которая задается следующим образом:

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

Это означает, что математическое ожидание стохастического градиента равно фактическому градиенту  $f(x)$ .

# Результаты для градиентного спуска

Стохастические итерации в  $n$  раз быстрее, но сколько итераций потребуется для достижения заданной точности?

Если  $\nabla f$  является липшицевым, то мы получаем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$\mathcal{O}(\log(1/\varepsilon))$	
Выпуклый	$\mathcal{O}(1/\varepsilon)$	
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	

# Результаты для градиентного спуска

Стохастические итерации в  $n$  раз быстрее, но сколько итераций потребуется для достижения заданной точности?

Если  $\nabla f$  является липшицевым, то мы получаем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- Стохастический градиентный спуск имеет низкую стоимость итерации, но медленную скорость сходимости.

# Результаты для градиентного спуска

Стохастические итерации в  $n$  раз быстрее, но сколько итераций потребуется для достижения заданной точности?

Если  $\nabla f$  является липшицевым, то мы получаем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- Стохастический градиентный спуск имеет низкую стоимость итерации, но медленную скорость сходимости.
  - Сублинейная скорость даже в сильно выпуклом случае.

# Результаты для градиентного спуска

Стохастические итерации в  $n$  раз быстрее, но сколько итераций потребуется для достижения заданной точности?

Если  $\nabla f$  является липшицевым, то мы получаем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

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

# Результаты для градиентного спуска

Стохастические итерации в  $n$  раз быстрее, но сколько итераций потребуется для достижения заданной точности?

Если  $\nabla f$  является липшицевым, то мы получаем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- Стохастический градиентный спуск имеет низкую стоимость итерации, но медленную 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

Стохастические итерации в  $n$  раз быстрее, но сколько итераций потребуется для достижения заданной точности?

Если  $\nabla f$  является липшицевым, то мы получаем:

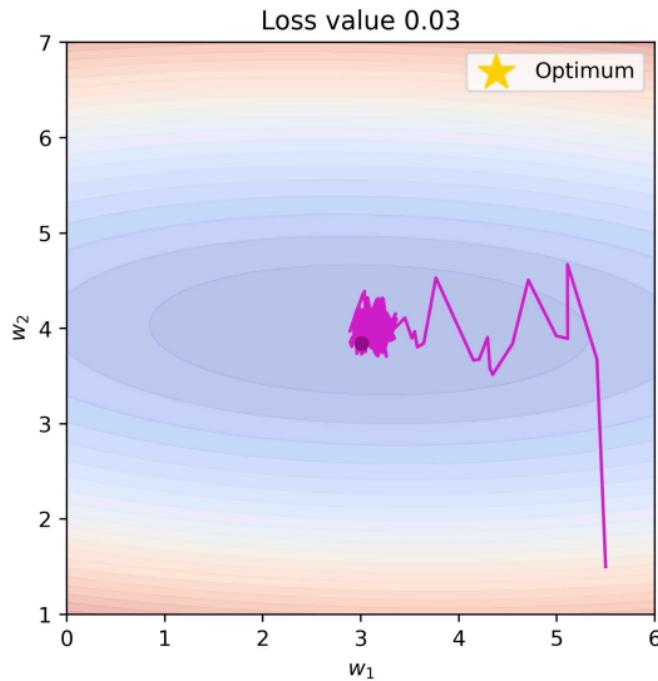
Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$\mathcal{O}(\log(1/\varepsilon))$	$\mathcal{O}(1/\varepsilon)$
Выпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$
Невыпуклый	$\mathcal{O}(1/\varepsilon)$	$\mathcal{O}(1/\varepsilon^2)$

- Стохастический градиентный спуск имеет низкую стоимость итерации, но медленную 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 моментом и квази-Ньютоновские методы не улучшают скорость в стохастическом случае, а только могут улучшить константные множители (быть локальное горлышко — дисперсия, а не число обусловленности).

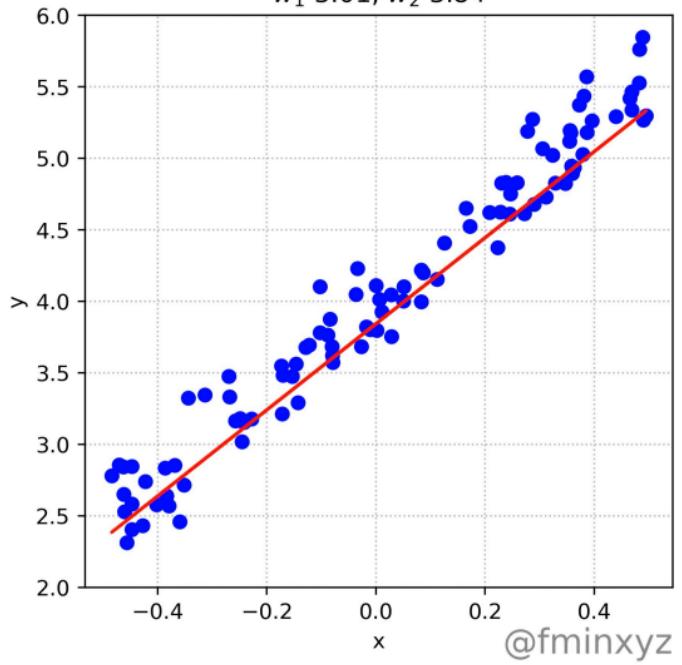
# **Стохастический градиентный спуск (SGD)**

# Типичное поведение

Stochastic Gradient Descent. Batch = 2



$w_1$  3.01,  $w_2$  3.84



@fminxyz

# 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

Подход 1: контролировать размер выборки

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 1 элемент:

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

Распространнёный 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 параллелизации.

Например, с 16 ядрами установите  $|B_k| = 16$  и вычислите 16 градиентов одновременно.

# Мини-батч как градиентный спуск с ошибкой

Метод SG с выборкой  $B_k$  ("мини-батч") использует итерации:

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

Посмотрим на это как на "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 вы используете  $\alpha_k = \frac{1}{L}$ , то используя лемму о спуске, этот алгоритм имеет:

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

для любой ошибки  $e_k$ .

# Влияние ошибки на скорость сходимости

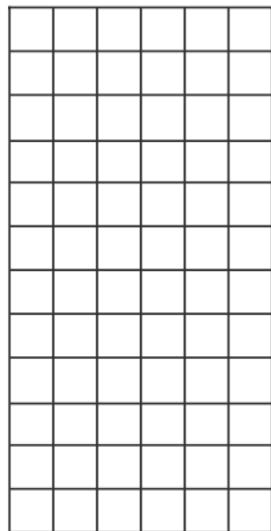
Оценка прогресса с  $\alpha_k = \frac{1}{L}$  и ошибкой в градиенте  $e_k$  выглядит следующим образом:

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

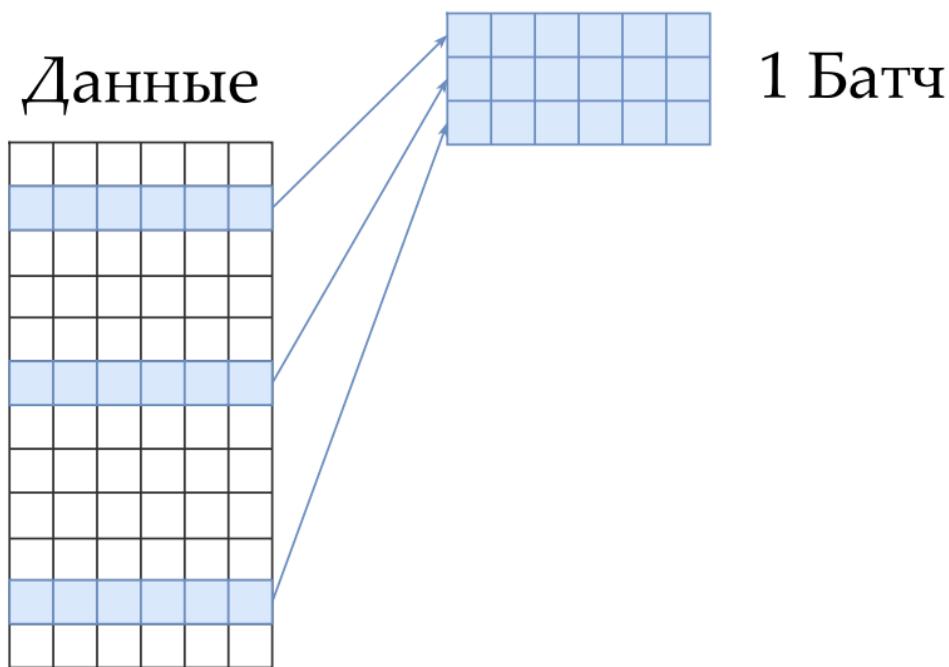
## Идея SGD и батчей



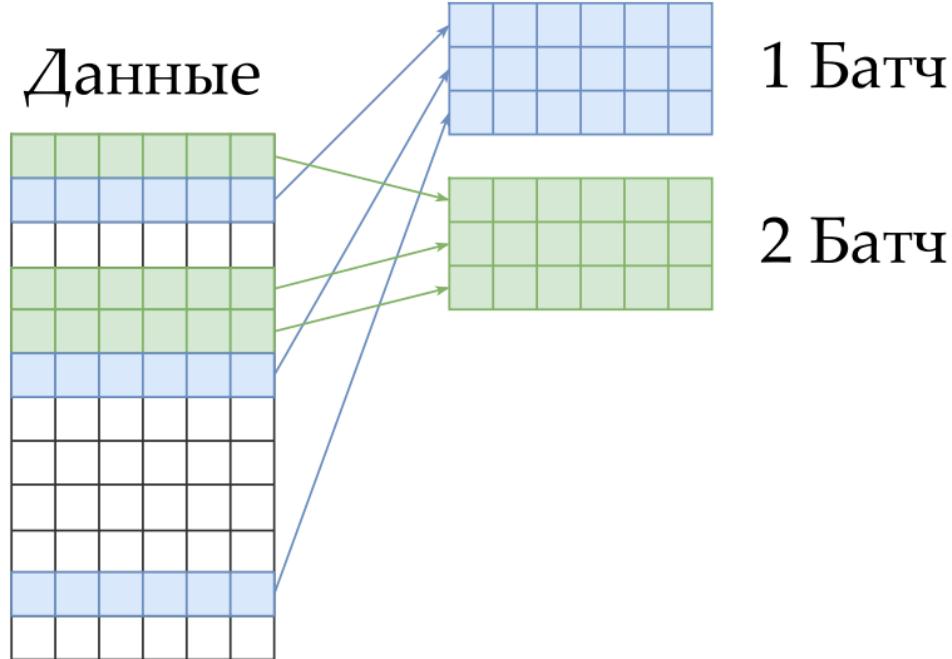
## Данные



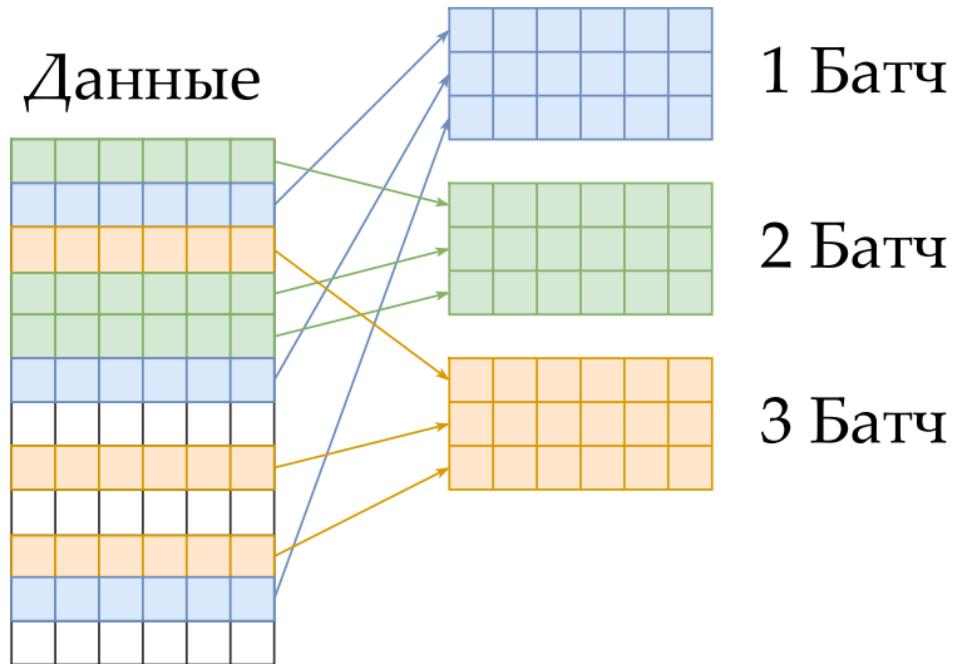
# Идея SGD и батчей



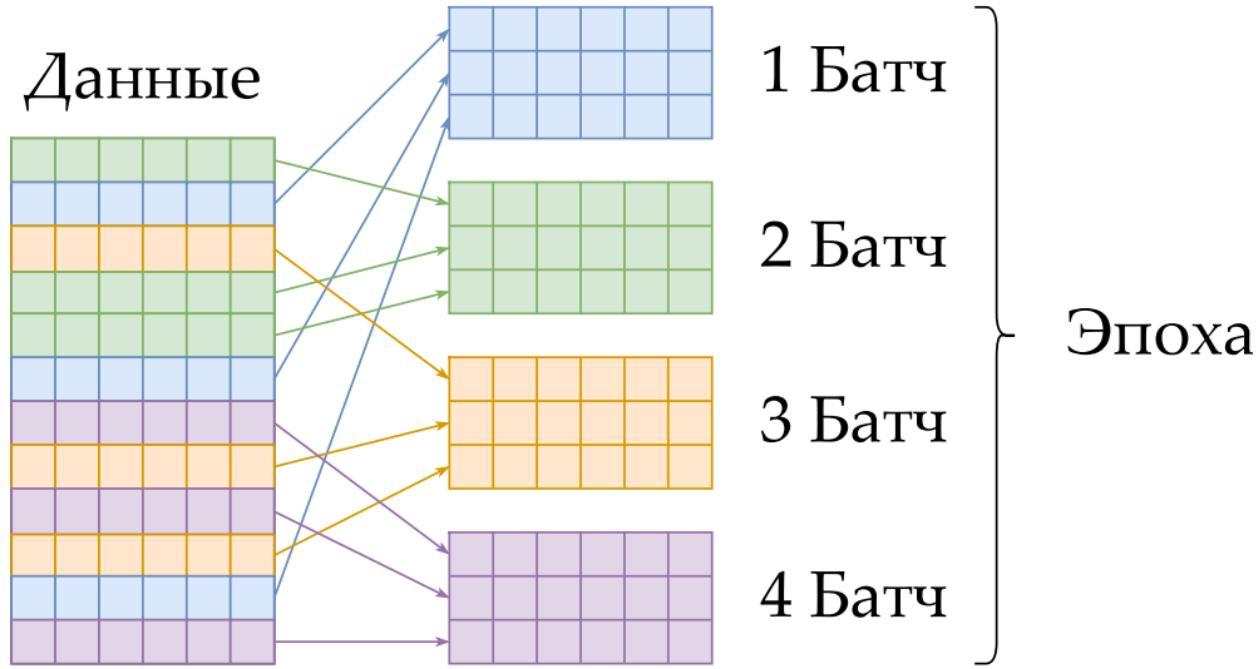
# Идея SGD и батчей



# Идея SGD и батчей



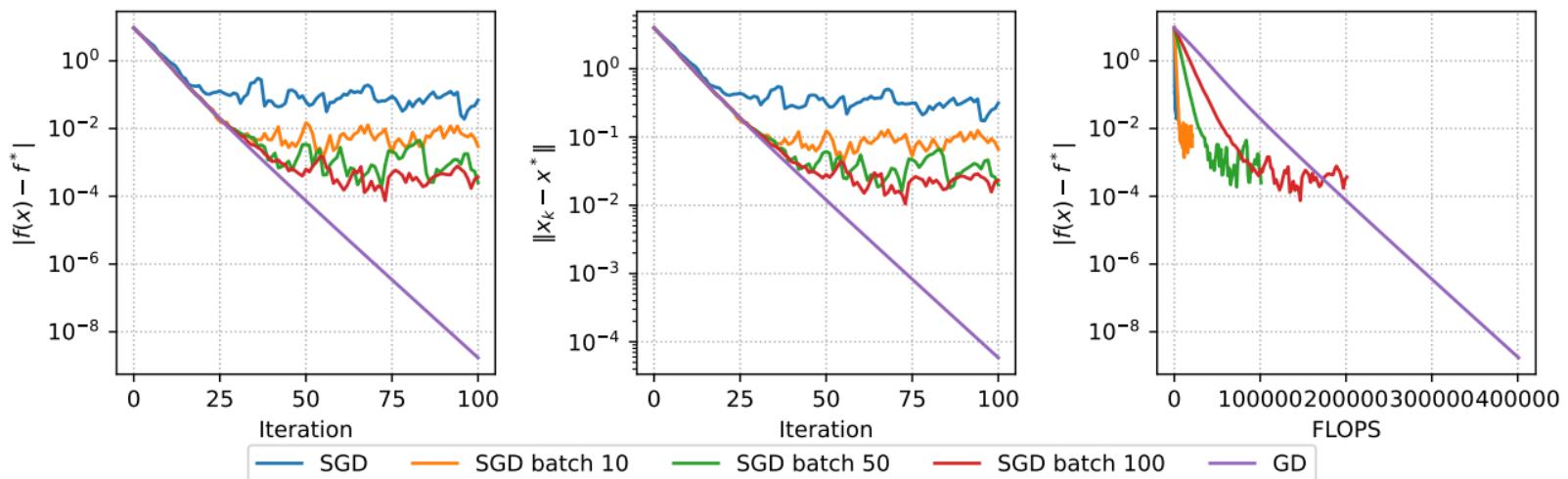
# Идея SGD и батчей



# 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  $\mu$ -сильно выпуклая функция, а дисперсия стохастического градиента конечна ( $\mathbb{E}[\|\nabla f_i(x_k)\|^2] \leq \sigma^2$ ). Тогда траектория стохастического градиентного спуска 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  $\mu$ -сильно выпуклая функция, а дисперсия стохастического градиента конечна ( $\mathbb{E}[\|\nabla f_i(x_k)\|^2] \leq \sigma^2$ ). Тогда траектория стохастического градиентного спуска 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  $\mu$ -сильно выпуклая функция, а дисперсия стохастического градиента конечна ( $\mathbb{E}[\|\nabla f_i(x_k)\|^2] \leq \sigma^2$ ). Тогда стохастический градиентный шум с уменьшающимся 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 даже для PL (сильно выпуклого) случая

# Summary

- SGD с постоянным шагом не сходится даже для PL (сильно выпуклого) случая
- SGD достигает сублинейной сходимости с скоростью  $\mathcal{O}\left(\frac{1}{k}\right)$  для PL-случая.

# Summary

- SGD с постоянным шагом не сходится даже для PL (сильно выпуклого) случая
- SGD достигает сублинейной сходимости с скоростью  $\mathcal{O}\left(\frac{1}{k}\right)$  для PL-случая.
- Ускорения Нестерова/Поляка не улучшают скорость сходимости

# Summary

- SGD с постоянным шагом не сходится даже для PL (сильно выпуклого) случая
- SGD достигает сублинейной сходимости с скоростью  $\mathcal{O}\left(\frac{1}{k}\right)$  для PL-случая.
- Ускорения Нестерова/Поляка не улучшают скорость сходимости
- Двухфазный Ньютоновский метод достигает  $\mathcal{O}\left(\frac{1}{k}\right)$  без сильной выпуклости.

# Адаптивность или масштабирование

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

Очень популярный адаптивный метод. Пусть  $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ , правило обновления для  $j = 1, \dots, p$ :

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

## Заметки:

- AdaGrad не требует настройки шага обучения:  $\alpha > 0$  — фиксированная константа, и скорость обучения естественно уменьшается по итерациям.

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

Очень популярный адаптивный метод. Пусть  $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ , правило обновления для  $j = 1, \dots, p$ :

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

## Заметки:

- AdaGrad не требует настройки шага обучения:  $\alpha > 0$  — фиксированная константа, и скорость обучения естественно уменьшается по итерациям.
- Шаг обучения для редких информативных признаков убывает медленно.

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

Очень популярный адаптивный метод. Пусть  $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ , 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 скорость обучения естественно уменьшается по итерациям.
- 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)

Очень популярный адаптивный метод. Пусть  $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ , правило обновления для  $j = 1, \dots, p$ :

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

## Заметки:

- AdaGrad не требует настройки шага обучения:  $\alpha > 0$  — фиксированная константа, и скорость обучения естественно уменьшается по итерациям.
- Шаг обучения для редких информативных признаков убывает медленно.
- Может существенно превосходить SGD на разреженных задачах.
- Основной недостаток — монотонное накопление 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)

Очень популярный адаптивный метод. Пусть  $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ , правило обновления для  $j = 1, \dots, p$ :

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

## Заметки:

- AdaGrad не требует настройки шага обучения:  $\alpha > 0$  — фиксированная константа, и скорость обучения естественно уменьшается по итерациям.
- Шаг обучения для редких информативных признаков убывает медленно.
- Может существенно превосходить SGD на разреженных задачах.
- Основной недостаток — монотонное накопление градиентов в знаменателе. AdaDelta, Adam, AMSGrad и др. улучшают это, популярны в обучении глубоких нейронных сетей.
- Константа  $\epsilon$  обычно устанавливается в  $10^{-6}$  для обеспечения отсутствия деления на ноль или слишком больших шагов.

# RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, которое устраняет его агрессивный, монотонно убывающий шаг обучения. Использует скользящее среднее квадратов градиентов для настройки шага обучения для каждого веса. 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 делит шаг обучения для веса на скользящее среднее величин недавних градиентов для этого веса.

# RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, которое устраняет его агрессивный, монотонно убывающий шаг обучения. Использует скользящее среднее квадратов градиентов для настройки шага обучения для каждого веса. 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 делит шаг обучения для веса на скользящее среднее величин недавних градиентов для этого веса.
- Обеспечивает более тонкую настройку шагов обучения, чем AdaGrad, что делает его подходящим для нестационарных задач.

# RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, которое устраняет его агрессивный, монотонно убывающий шаг обучения. Использует скользящее среднее квадратов градиентов для настройки шага обучения для каждого веса. 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 делит шаг обучения для веса на скользящее среднее величин недавних градиентов для этого веса.
- 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)



Объединяет элементы из AdaGrad и RMSProp. Учитывает экспоненциально убывающее среднее прошлых градиентов и квадратов градиентов.

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 в начальных моментах, наблюдаемое в других методах (например, RMSProp), что делает оценки более точными.

# Adam (Kingma and Ba, 2014) <sup>12</sup>



Объединяет элементы из AdaGrad и RMSProp. Учитывает экспоненциально убывающее среднее прошлых градиентов и квадратов градиентов.

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 в начальных моментах, 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. Учитывает экспоненциально убывающее среднее прошлых градиентов и квадратов градиентов.

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 в начальных моментах, 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. Учитывает экспоненциально убывающее среднее прошлых градиентов и квадратов градиентов.

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 в начальных моментах, 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. Учитывает экспоненциально убывающее среднее прошлых градиентов и квадратов градиентов.

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 в начальных моментах, 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)<sup>1</sup><sup>2</sup>



Объединяет элементы из AdaGrad и RMSProp. Учитывает экспоненциально убывающее среднее прошлых градиентов и квадратов градиентов.

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 в начальных моментах, 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
- Гораздо лучше работает для языковых моделей, чем для задач компьютерного зрения - почему?

---

<sup>1</sup>Adam: A Method for Stochastic Optimization

<sup>2</sup>On the Convergence of Adam and Beyond

# AdamW (Loshchilov & Hutter, 2017)

Устраняет распространенную проблему с  $\ell_2$ -регуляризацией в адаптивных оптимизаторах, таких как Adam. Стандартная  $\ell_2$ -регуляризация добавляет  $\lambda \|x\|^2$  к функции потерь, что приводит к градиентному слагаемому  $\lambda x$ . В Adam это слагаемое масштабируется адаптивным шагом обучения  $(\sqrt{\hat{v}_j} + \epsilon)$ , связывая затухание весов (weight decay) с величинами градиента. AdamW разделяет затухание весов от шага адаптации градиентов.

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)

Устраняет распространенную проблему с  $\ell_2$ -регуляризацией в адаптивных оптимизаторах, таких как Adam. Стандартная  $\ell_2$ -регуляризация добавляет  $\lambda \|x\|^2$  к функции потерь, что приводит к градиентному слагаемому  $\lambda x$ . В Adam это слагаемое масштабируется адаптивным шагом обучения  $(\sqrt{\hat{v}_j} + \epsilon)$ , связывая затухание весов (weight decay) с величинами градиента. AdamW разделяет затухание весов от шага адаптации градиентов.

Правило обновления:

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

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

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

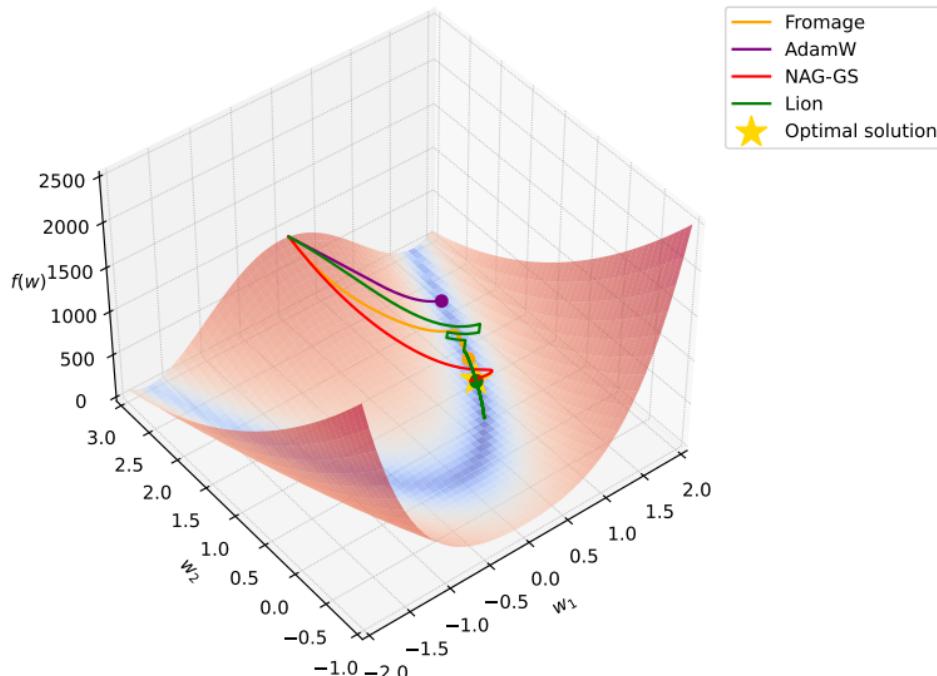
$$x_j^{(k)} = x_j^{(k-1)} - \alpha \left( \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)$$

**Заметки:**

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

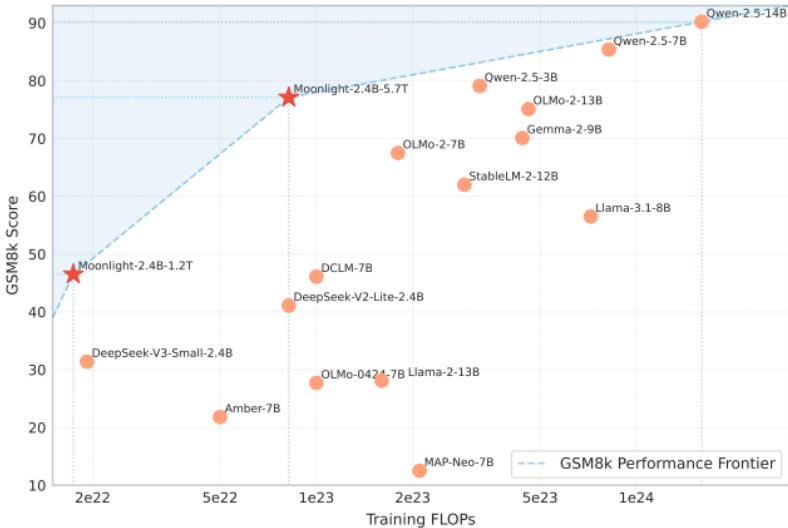
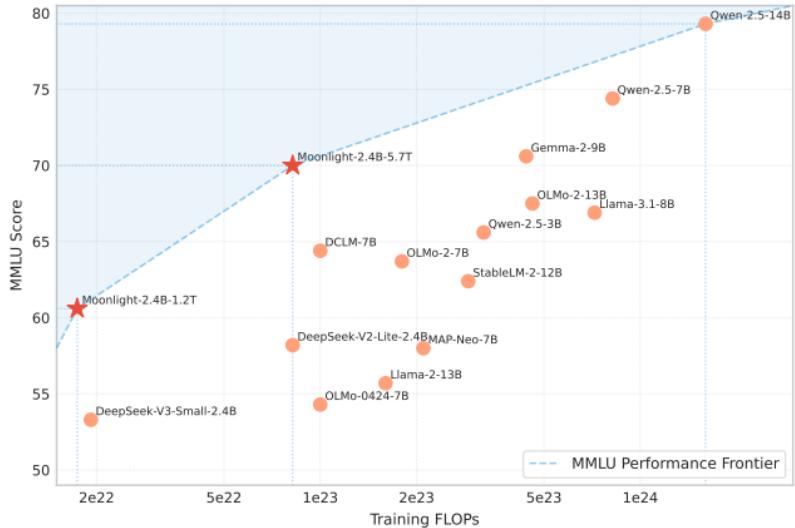




# Новый подход к оптимизации



3



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

<sup>3</sup>KIMI K2: OPEN AGENTIC INTELLIGENCE

# Интуиция за методом Мион<sup>4</sup>



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

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

<sup>4</sup>Презентация R. Gower

# Интуиция за методом Мион<sup>4</sup>

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

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 Мион<sup>4</sup>

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

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 Мион<sup>4</sup>

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

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  $x_k$

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

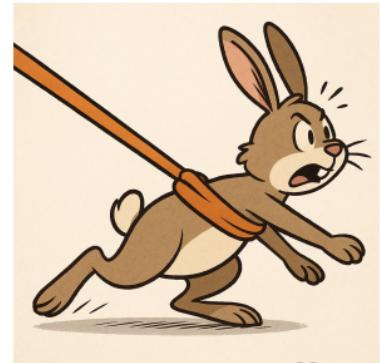
# Интуиция за методом Мион. Градиентный спуск



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

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?

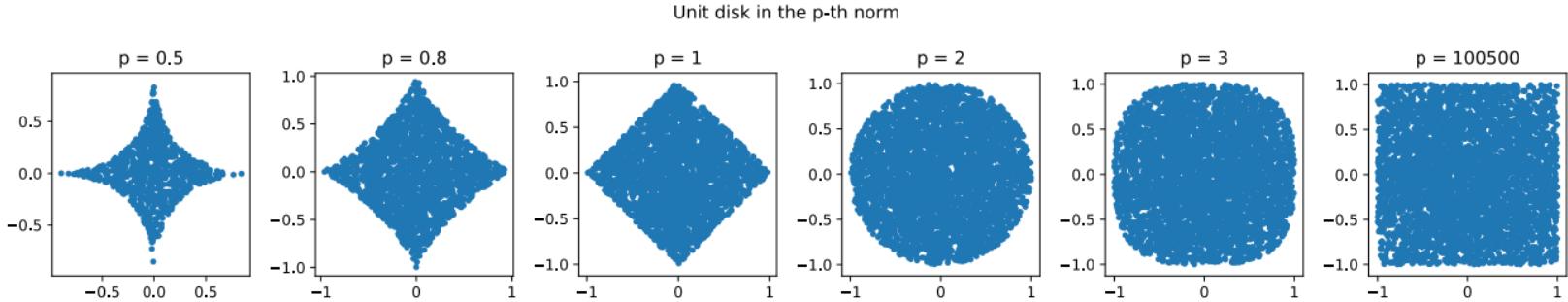


Рисунок 2. Примеры шаров в разных нормах

# Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

$$\|g\|^* = \sup_{\|x\|=1} \langle g, x \rangle$$

# Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

$$\|g\|^* = \sup_{\|x\|=1} \langle g, x \rangle$$

- Linear Minimization Oracle:

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

# Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

$$\|g\|^* = \sup_{\|x\|=1} \langle g, x \rangle$$

- Linear Minimization Oracle:

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

- Важное свойство, связывающее эти два понятия:

$$\langle g, \text{LMO}_{\|\cdot\|}(g) \rangle = -\|g\|^*$$

# Неевклидовы записи методов<sup>5</sup>



## ! Неевклидов градиентный спуск

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

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

---

<sup>5</sup>Old Optimizer, New Norm: An Anthology

# Неевклидовы записи методов<sup>5</sup>

## ! Неевклидов градиентный спуск

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

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

## ! Неевклидов нормированный градиентный спуск

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

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

---

<sup>5</sup>Old 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

- 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

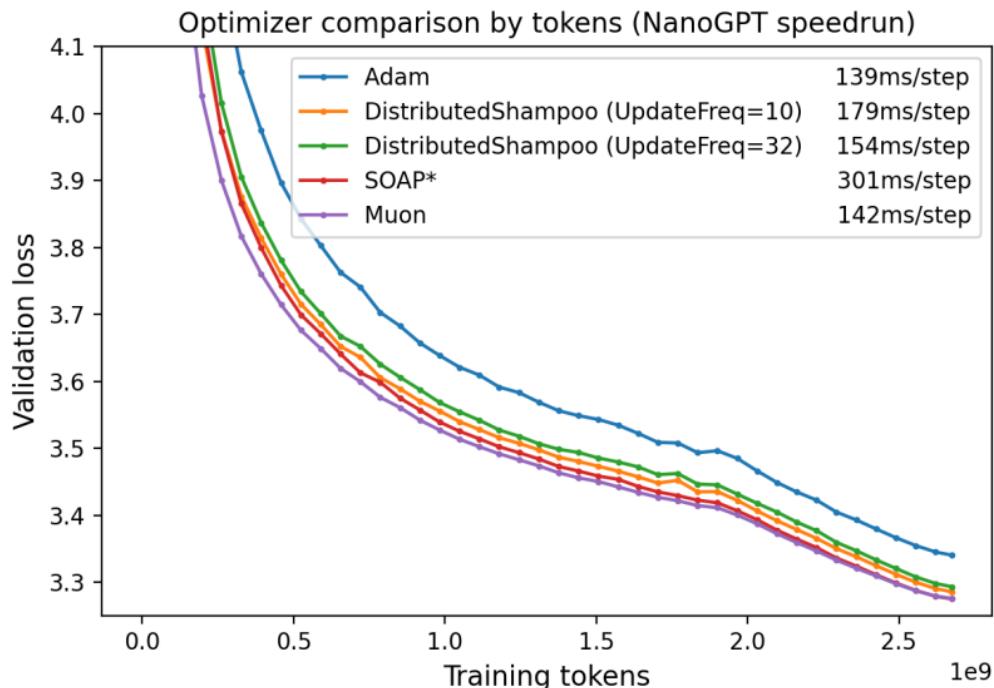


Рисунок 3. NanoGPT speedrun

# Обучение GPT-2 (124М) на FineWeb

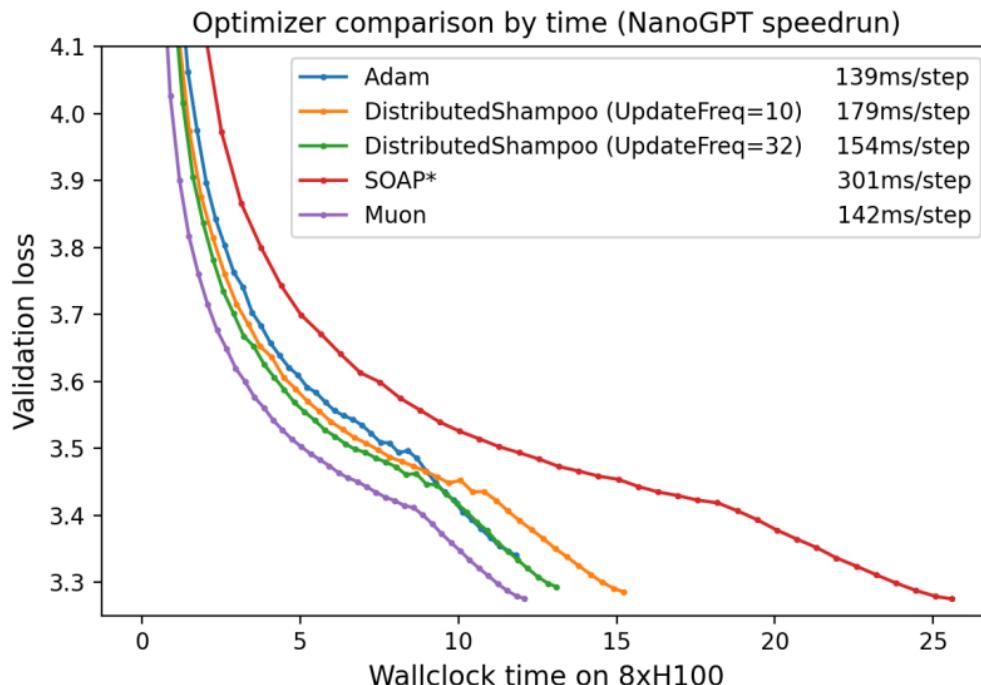


Рисунок 4. NanoGPT speedrun

# 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).

# Muon<sup>6</sup>



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

---

<sup>6</sup>Deriving Muon