



Методы редукции дисперсии: SAG, SVRG,  
SAGA. Адаптивные стохастические  
градиентные методы. Современные методы  
оптимизации для обучения нейронных  
сетей

Даня Меркулов

Оптимизация для всех! ЦУ

## Задача с конечной суммой

## Задача с конечной суммой

Рассмотрим классическую задачу минимизации среднего по конечной выборке:

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

Градиентный спуск действует следующим образом:

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

- Стоимость итерации линейна по  $n$ .

## 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 действует следующим образом:

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

- Стоимость итерации линейна по  $n$ .
- Сходимость с постоянным шагом  $\alpha$  или с линейным поиском шага.

## 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 действует следующим образом:

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

- Стоимость итерации линейна по  $n$ .
- Сходимость с постоянным шагом  $\alpha$  или с линейным поиском шага.

## 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 действует следующим образом:

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

- Стоимость итерации линейна по  $n$ .
- Сходимость с постоянным шагом  $\alpha$  или с линейным поиском шага.

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

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

При  $p(i_k = i) = \frac{1}{n}$  стохастический градиент является несмешённой оценкой градиента и задаётся так:

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

Это показывает, что ожидаемое значение 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	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- 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	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- 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	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- 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	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- 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 несмешённую 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	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- 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 несмешённую аппроксимацию градиента с ограниченной дисперсией.
- Методы с моментом и квази-Ньютоновские методы не улучшают скорость в стохастическом случае, а только могут улучшить константные множители (бутилочное горлышко — дисперсия, а не число обусловленности).

## SGD с постоянным шагом не сходится

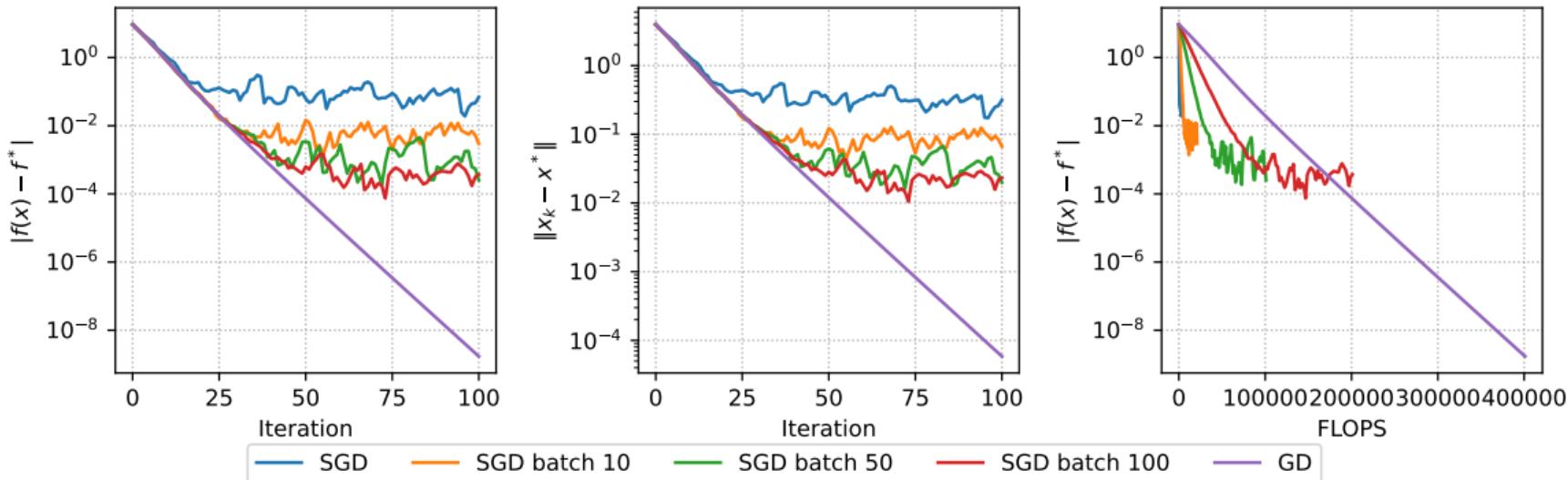
Stochastic Gradient Descent. Batch = 2



# 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.



## Методы редукции дисперсии

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$
- $\text{var}(Z_\alpha) = \alpha^2 (\text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y))$

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$
- $\text{var}(Z_\alpha) = \alpha^2 (\text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y))$ 
  - Если  $\alpha = 1$ : нет смещения

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$
- $\text{var}(Z_\alpha) = \alpha^2 (\text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y))$ 
  - Если  $\alpha = 1$ : нет смещения
  - Если  $\alpha < 1$ : потенциальное смещение (но уменьшенная дисперсия).

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$
- $\text{var}(Z_\alpha) = \alpha^2 (\text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y))$ 
  - Если  $\alpha = 1$ : нет смещения
  - Если  $\alpha < 1$ : потенциальное смещение (но уменьшенная дисперсия).
- Полезно, если  $Y$  положительно коррелирован с  $X$ .

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$
- $\text{var}(Z_\alpha) = \alpha^2 (\text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y))$ 
  - Если  $\alpha = 1$ : нет смещения
  - Если  $\alpha < 1$ : потенциальное смещение (но уменьшенная дисперсия).
- Полезно, если  $Y$  положительно коррелирован с  $X$ .

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$
- $\text{var}(Z_\alpha) = \alpha^2 (\text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y))$ 
  - Если  $\alpha = 1$ : нет смещения
  - Если  $\alpha < 1$ : потенциальное смещение (но уменьшенная дисперсия).
- Полезно, если  $Y$  положительно коррелирован с  $X$ .

### Применение к оценке градиента?

- SVRG: Пусть  $X = \nabla f_{i_k}(x^{(k-1)})$  и  $Y = \nabla f_{i_k}(\tilde{x})$ , при  $\alpha = 1$  и с сохранённой точкой  $\tilde{x}$ .

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$
- $\text{var}(Z_\alpha) = \alpha^2 (\text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y))$ 
  - Если  $\alpha = 1$ : нет смещения
  - Если  $\alpha < 1$ : потенциальное смещение (но уменьшенная дисперсия).
- Полезно, если  $Y$  положительно коррелирован с  $X$ .

### Применение к оценке градиента?

- SVRG: Пусть  $X = \nabla f_{i_k}(x^{(k-1)})$  и  $Y = \nabla f_{i_k}(\tilde{x})$ , при  $\alpha = 1$  и с сохранённой точкой  $\tilde{x}$ .
- $\mathbb{E}[Y] = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$  полный градиент в точке  $\tilde{x}$ ;

## Основная идея редукции дисперсии

**Принцип:** уменьшение дисперсии выборки  $X$  путём использования выборки из другой случайной переменной  $Y$  с известным математическим ожиданием:

$$Z_\alpha = \alpha(X - Y) + \mathbb{E}[Y]$$

- $\mathbb{E}[Z_\alpha] = \alpha\mathbb{E}[X] + (1 - \alpha)\mathbb{E}[Y]$
- $\text{var}(Z_\alpha) = \alpha^2 (\text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y))$ 
  - Если  $\alpha = 1$ : нет смещения
  - Если  $\alpha < 1$ : потенциальное смещение (но уменьшенная дисперсия).
- Полезно, если  $Y$  положительно коррелирован с  $X$ .

### Применение к оценке градиента?

- SVRG: Пусть  $X = \nabla f_{i_k}(x^{(k-1)})$  и  $Y = \nabla f_{i_k}(\tilde{x})$ , при  $\alpha = 1$  и с сохранённой точкой  $\tilde{x}$ .
- $\mathbb{E}[Y] = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$  полный градиент в точке  $\tilde{x}$ ;
- $X - Y = \nabla f_{i_k}(x^{(k-1)}) - \nabla f_{i_k}(\tilde{x})$ .

## SAG (Стохастический средний градиент, Schmidt, Le Roux, и Bach 2013)

- Хранить таблицу, содержащую градиент  $g_i$  функции  $f_i$ ,  $i = 1, \dots, n$

## SAG (Стохастический средний градиент, Schmidt, Le Roux, и Bach 2013)

- Хранить таблицу, содержащую градиент  $g_i$  функции  $f_i$ ,  $i = 1, \dots, n$
- Инициализировать  $x^{(0)}$ , и  $g_i^{(0)} = \nabla f_i(x^{(0)})$ ,  $i = 1, \dots, n$

## SAG (Стохастический средний градиент, Schmidt, Le Roux, и Bach 2013)

- Хранить таблицу, содержащую градиент  $g_i$  функции  $f_i$ ,  $i = 1, \dots, n$
- Инициализировать  $x^{(0)}$ , и  $g_i^{(0)} = \nabla f_i(x^{(0)})$ ,  $i = 1, \dots, n$
- На шагах  $k = 1, 2, 3, \dots$ , случайно выбирать  $i_k \in \{1, \dots, n\}$ , тогда пусть

$$g_{i_k}^{(k)} = \nabla f_{i_k}(x^{(k-1)}) \quad (\text{наиболее свежий градиент } f_{i_k})$$

Все остальные  $g_i^{(k)} = g_i^{(k-1)}$ ,  $i \neq i_k$ , т.е., они остаются такими же

## SAG (Стохастический средний градиент, Schmidt, Le Roux, и Bach 2013)

- Хранить таблицу, содержащую градиент  $g_i$  функции  $f_i$ ,  $i = 1, \dots, n$
- Инициализировать  $x^{(0)}$ , и  $g_i^{(0)} = \nabla f_i(x^{(0)})$ ,  $i = 1, \dots, n$
- На шагах  $k = 1, 2, 3, \dots$ , случайно выбирать  $i_k \in \{1, \dots, n\}$ , тогда пусть

$$g_{i_k}^{(k)} = \nabla f_{i_k}(x^{(k-1)}) \quad (\text{наиболее свежий градиент } f_{i_k})$$

Все остальные  $g_i^{(k)} = g_i^{(k-1)}$ ,  $i \neq i_k$ , т.е., они остаются такими же

- Обновление

$$x^{(k)} = x^{(k-1)} - \alpha_k \frac{1}{n} \sum_{i=1}^n g_i^{(k)}$$

## SAG (Стохастический средний градиент, Schmidt, Le Roux, и Bach 2013)

- Хранить таблицу, содержащую градиент  $g_i$  функции  $f_i$ ,  $i = 1, \dots, n$
- Инициализировать  $x^{(0)}$ , и  $g_i^{(0)} = \nabla f_i(x^{(0)})$ ,  $i = 1, \dots, n$
- На шагах  $k = 1, 2, 3, \dots$ , случайно выбирать  $i_k \in \{1, \dots, n\}$ , тогда пусть

$$g_{i_k}^{(k)} = \nabla f_{i_k}(x^{(k-1)}) \quad (\text{наиболее свежий градиент } f_{i_k})$$

Все остальные  $g_i^{(k)} = g_i^{(k-1)}$ ,  $i \neq i_k$ , т.е., они остаются такими же

- Обновление

$$x^{(k)} = x^{(k-1)} - \alpha_k \frac{1}{n} \sum_{i=1}^n g_i^{(k)}$$

- Оценки градиента SAG больше не являются несмешёнными, но у них значительно уменьшена дисперсия

## SAG (Стохастический средний градиент, Schmidt, Le Roux, и Bach 2013)

- Хранить таблицу, содержащую градиент  $g_i$  функции  $f_i$ ,  $i = 1, \dots, n$
- Инициализировать  $x^{(0)}$ , и  $g_i^{(0)} = \nabla f_i(x^{(0)})$ ,  $i = 1, \dots, n$
- На шагах  $k = 1, 2, 3, \dots$ , случайно выбирать  $i_k \in \{1, \dots, n\}$ , тогда пусть

$$g_{i_k}^{(k)} = \nabla f_{i_k}(x^{(k-1)}) \quad (\text{наиболее свежий градиент } f_{i_k})$$

Все остальные  $g_i^{(k)} = g_i^{(k-1)}$ ,  $i \neq i_k$ , т.е., они остаются такими же

- Обновление

$$x^{(k)} = x^{(k-1)} - \alpha_k \frac{1}{n} \sum_{i=1}^n g_i^{(k)}$$

- Оценки градиента SAG больше не являются несмешёнными, но у них значительно уменьшена дисперсия
- Разве усреднение всех этих градиентов не слишком затратно? В целом это столь же эффективно, как SGD, если подойти к делу с умом:

$$x^{(k)} = x^{(k-1)} - \alpha_k \underbrace{\left( \frac{1}{n} g_i^{(k)} - \frac{1}{n} g_i^{(k-1)} + \underbrace{\frac{1}{n} \sum_{i=1}^n g_i^{(k-1)}}_{\text{среднее старой таблицы}} \right)}_{\text{среднее новой таблицы}}$$

# Сходимость SAG

Пусть  $f(x) = \frac{1}{n} \sum_{i=1}^n f_i(x)$ , где каждая  $f_i$  дифференцируема, и  $\nabla f_i$  липшицева с константой  $L$ .

Обозначим  $\bar{x}^{(k)} = \frac{1}{k} \sum_{l=0}^{k-1} x^{(l)}$ , среднюю итерацию после  $k - 1$  шагов.

## Theorem

SAG, с постоянным шагом  $\alpha = \frac{1}{16L}$ , и инициализацией

$$g_i^{(0)} = \nabla f_i(x^{(0)}) - \nabla f(x^{(0)}), \quad i = 1, \dots, n$$

удовлетворяет

$$\mathbb{E}[f(\bar{x}^{(k)})] - f^\star \leq \frac{48n}{k} [f(x^{(0)}) - f^\star] + \frac{128L}{k} \|x^{(0)} - x^\star\|^2$$

где ожидание берётся по случайным индексам.

## Сходимость SAG

- Результат сформулирован в терминах средней итерации  $\bar{x}^{(k)}$ , но также может выполняться для наилучшей итерации  $x_{best}^{(k)}$ , найденной на данный момент.

## Сходимость SAG

- Результат сформулирован в терминах средней итерации  $\bar{x}^{(k)}$ , но также может выполняться для наилучшей итерации  $x_{best}^{(k)}$ , найденной на данный момент.
- Это скорость сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для SAG. Сравните со скоростью сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для GD и  $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$  для SGD.

## Сходимость SAG

- Результат сформулирован в терминах средней итерации  $\bar{x}^{(k)}$ , но также может выполняться для наилучшей итерации  $x_{best}^{(k)}$ , найденной на данный момент.
- Это скорость сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для SAG. Сравните со скоростью сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для GD и  $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$  для SGD.
- Но константы разные! Оценки после  $k$  шагов:

## Сходимость SAG

- Результат сформулирован в терминах средней итерации  $\bar{x}^{(k)}$ , но также может выполняться для наилучшей итерации  $x_{best}^{(k)}$ , найденной на данный момент.
- Это скорость сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для SAG. Сравните со скоростью сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для GD и  $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$  для SGD.
- Но константы разные! Оценки после  $k$  шагов:
  - GD:  $\frac{L\|x^{(0)} - x^*\|^2}{2k}$

## Сходимость SAG

- Результат сформулирован в терминах средней итерации  $\bar{x}^{(k)}$ , но также может выполняться для наилучшей итерации  $x_{best}^{(k)}$ , найденной на данный момент.
- Это скорость сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для SAG. Сравните со скоростью сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для GD и  $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$  для SGD.
- Но константы разные! Оценки после  $k$  шагов:
  - GD:  $\frac{L\|x^{(0)} - x^*\|^2}{2k}$
  - SAG:  $\frac{48n[f(x^{(0)}) - f^*] + 128L\|x^{(0)} - x^*\|^2}{k}$

## Сходимость SAG

- Результат сформулирован в терминах средней итерации  $\bar{x}^{(k)}$ , но также может выполняться для наилучшей итерации  $x_{best}^{(k)}$ , найденной на данный момент.
- Это скорость сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для SAG. Сравните со скоростью сходимости  $\mathcal{O}\left(\frac{1}{k}\right)$  для GD и  $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$  для SGD.
- Но константы разные! Оценки после  $k$  шагов:
  - GD:  $\frac{L\|x^{(0)} - x^*\|^2}{2k}$
  - SAG:  $\frac{48n[f(x^{(0)}) - f^*] + 128L\|x^{(0)} - x^*\|^2}{k}$
- Таким образом, первый член в оценке для SAG страдает от фактора  $n$ ; авторы предлагают более удачную инициализацию, чтобы сделать  $f(x^{(0)}) - f^*$  малым (например, они предлагают использовать результат  $n$  шагов SGD).

# Сходимость SAG

Дополнительно предположим, что каждая  $f_i$   $\mu$ -сильно выпукла.

## Theorem

SAG, с шагом  $\alpha = \frac{1}{16L}$  и той же инициализацией, что и раньше, удовлетворяет

$$\mathbb{E}[f(x^{(k)})] - f^* \leq \left(1 - \min\left(\frac{\mu}{16L}, \frac{1}{8n}\right)\right)^k \left(\frac{3}{2} (f(x^{(0)}) - f^*) + \frac{4L}{n} \|x^{(0)} - x^*\|^2\right)$$

## Замечания:

- Это линейная скорость сходимости  $\mathcal{O}(\gamma^k)$  для SAG. Для сравнения:  $\mathcal{O}(\gamma^k)$  для GD и лишь  $\mathcal{O}\left(\frac{1}{k}\right)$  для SGD.

# Сходимость SAG

Дополнительно предположим, что каждая  $f_i$   $\mu$ -сильно выпукла.

## Theorem

SAG, с шагом  $\alpha = \frac{1}{16L}$  и той же инициализацией, что и раньше, удовлетворяет

$$\mathbb{E}[f(x^{(k)})] - f^* \leq \left(1 - \min\left(\frac{\mu}{16L}, \frac{1}{8n}\right)\right)^k \left(\frac{3}{2} (f(x^{(0)}) - f^*) + \frac{4L}{n} \|x^{(0)} - x^*\|^2\right)$$

## Замечания:

- Это линейная скорость сходимости  $\mathcal{O}(\gamma^k)$  для SAG. Для сравнения:  $\mathcal{O}(\gamma^k)$  для GD и лишь  $\mathcal{O}(\frac{1}{k})$  для SGD.
- Подобно GD, метод SAG адаптивен к сильной выпуклости.

# Сходимость SAG

Дополнительно предположим, что каждая  $f_i$   $\mu$ -сильно выпукла.

## Theorem

SAG, с шагом  $\alpha = \frac{1}{16L}$  и той же инициализацией, что и раньше, удовлетворяет

$$\mathbb{E}[f(x^{(k)})] - f^* \leq \left(1 - \min\left(\frac{\mu}{16L}, \frac{1}{8n}\right)\right)^k \left(\frac{3}{2} (f(x^{(0)}) - f^*) + \frac{4L}{n} \|x^{(0)} - x^*\|^2\right)$$

## Замечания:

- Это линейная скорость сходимости  $\mathcal{O}(\gamma^k)$  для SAG. Для сравнения:  $\mathcal{O}(\gamma^k)$  для GD и лишь  $\mathcal{O}(\frac{1}{k})$  для SGD.
- Подобно GD, метод SAG адаптивен к сильной выпуклости.
- Доказательства этих результатов нетривиальны: 15 страниц с компьютерными вычислениями.

## Замечания по сходимости SAG

- Заметьте, что метод в базовой формулировке неприменим к обучению больших нейронных сетей из-за требований к памяти.

## Замечания по сходимости SAG

- Заметьте, что метод в базовой формулировке неприменим к обучению больших нейронных сетей из-за требований к памяти.
- На практике можно использовать стратегию backtracking для оценки константы Липшица.

## Замечания по сходимости SAG

- Заметьте, что метод в базовой формулировке неприменим к обучению больших нейронных сетей из-за требований к памяти.
- На практике можно использовать стратегию backtracking для оценки константы Липшица.
  - Выбрать начальное  $L_0$

## Замечания по сходимости SAG

- Заметьте, что метод в базовой формулировке неприменим к обучению больших нейронных сетей из-за требований к памяти.
- На практике можно использовать стратегию backtracking для оценки константы Липшица.
  - Выбрать начальное  $L_0$
  - Увеличить  $L$ , пока не выполнится следующее:

$$f_{i_k}(x^{k+1}) \leq f_{i_k}(x^k) + \nabla f_{i_k}(x^k)(x^{k+1} - x^k) + \frac{L}{2} \|x^{k+1} - x^k\|_2^2$$

## Замечания по сходимости SAG

- Заметьте, что метод в базовой формулировке неприменим к обучению больших нейронных сетей из-за требований к памяти.
- На практике можно использовать стратегию backtracking для оценки константы Липшица.
  - Выбрать начальное  $L_0$
  - Увеличить  $L$ , пока не выполнится следующее:

$$f_{i_k}(x^{k+1}) \leq f_{i_k}(x^k) + \nabla f_{i_k}(x^k)(x^{k+1} - x^k) + \frac{L}{2} \|x^{k+1} - x^k\|_2^2$$

- Уменьшить  $L$  между итерациями.

## Замечания по сходимости SAG

- Заметьте, что метод в базовой формулировке неприменим к обучению больших нейронных сетей из-за требований к памяти.
- На практике можно использовать стратегию backtracking для оценки константы Липшица.
  - Выбрать начальное  $L_0$
  - Увеличить  $L$ , пока не выполнится следующее:

$$f_{i_k}(x^{k+1}) \leq f_{i_k}(x^k) + \nabla f_{i_k}(x^k)(x^{k+1} - x^k) + \frac{L}{2} \|x^{k+1} - x^k\|_2^2$$

- Уменьшить  $L$  между итерациями.
- Поскольку стохастический градиент  $g(x^k) \rightarrow \nabla f(x^k)$ , можно использовать его норму для отслеживания сходимости (что неверно для SGD!).

## Замечания по сходимости SAG

- Заметьте, что метод в базовой формулировке неприменим к обучению больших нейронных сетей из-за требований к памяти.
- На практике можно использовать стратегию backtracking для оценки константы Липшица.
  - Выбрать начальное  $L_0$
  - Увеличить  $L$ , пока не выполнится следующее:

$$f_{i_k}(x^{k+1}) \leq f_{i_k}(x^k) + \nabla f_{i_k}(x^k)(x^{k+1} - x^k) + \frac{L}{2} \|x^{k+1} - x^k\|_2^2$$

- Уменьшить  $L$  между итерациями.
- Поскольку стохастический градиент  $g(x^k) \rightarrow \nabla f(x^k)$ , можно использовать его норму для отслеживания сходимости (что неверно для SGD!).
- Для обобщённых линейных моделей (включая LogReg, LLS) требуется гораздо меньше памяти  $\mathcal{O}(n)$  вместо  $\mathcal{O}(pn)$ .

$$f_i(w) = \varphi(w^T x_i) \leftrightarrow \nabla f_i(w) = \varphi'(w^T x_i) x_i$$

## SAG с неравномерной выборкой

- Шаг  $\alpha_k$  и скорость сходимости метода определяются константой  $L$  для  $f(x)$ , где  $L = \max_{1 \leq i \leq n} L_i$ ,  $L_i$  — константа Липшица для функции  $f_i$

## SAG с неравномерной выборкой

- Шаг  $\alpha_k$  и скорость сходимости метода определяются константой  $L$  для  $f(x)$ , где  $L = \max_{1 \leq i \leq n} L_i$ ,  $L_i$  — константа Липшица для функции  $f_i$
- При выборе компонент с вероятностью, пропорциональной  $L_i$ , константа  $L$  может быть уменьшена от  $\max_i L_i$  до  $\bar{L} = \sum_i L_i/N$ :

$$\begin{aligned} g(x) &= \frac{1}{n} \sum_{i=1}^n f_i(x) \\ &= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^{L_i} \frac{f_i(x)}{L_i} \\ &= \frac{1}{\sum_k L_k} \sum_{i=1}^n \sum_{j=1}^{L_i} \left( \sum_k \frac{L_k}{n} \frac{f_i(x)}{L_i} \right) \end{aligned}$$

При таком подходе компонента с большим значением  $L_i$  выбирается чаще.

## SAG с неравномерной выборкой

- Шаг  $\alpha_k$  и скорость сходимости метода определяются константой  $L$  для  $f(x)$ , где  $L = \max_{1 \leq i \leq n} L_i$ ,  $L_i$  — константа Липшица для функции  $f_i$
- При выборе компонент с вероятностью, пропорциональной  $L_i$ , константа  $L$  может быть уменьшена от  $\max_i L_i$  до  $\bar{L} = \sum_i L_i / N$ :

$$\begin{aligned} g(x) &= \frac{1}{n} \sum_{i=1}^n f_i(x) \\ &= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^{L_i} \frac{f_i(x)}{L_i} \\ &= \frac{1}{\sum_k L_k} \sum_{i=1}^n \sum_{j=1}^{L_i} \left( \sum_k \frac{L_k}{n} \frac{f_i(x)}{L_i} \right) \end{aligned}$$

При таком подходе компонента с большим значением  $L_i$  выбирается чаще.

- Чтобы обеспечить сходимость, выбор компоненты выполняют по правилу: с вероятностью 0.5 выбирают равномерно, с вероятностью 0.5 — с вероятностями  $L_i / \sum_j L_j$ .

## SAG с неравномерной выборкой

- Шаг  $\alpha_k$  и скорость сходимости метода определяются константой  $L$  для  $f(x)$ , где  $L = \max_{1 \leq i \leq n} L_i$ ,  $L_i$  — константа Липшица для функции  $f_i$
- При выборе компонент с вероятностью, пропорциональной  $L_i$ , константа  $L$  может быть уменьшена от  $\max_i L_i$  до  $\bar{L} = \sum_i L_i / N$ :

$$\begin{aligned} g(x) &= \frac{1}{n} \sum_{i=1}^n f_i(x) \\ &= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^{L_i} \frac{f_i(x)}{L_i} \\ &= \frac{1}{\sum_k L_k} \sum_{i=1}^n \sum_{j=1}^{L_i} \left( \sum_k \frac{L_k}{n} \frac{f_i(x)}{L_i} \right) \end{aligned}$$

При таком подходе компонента с большим значением  $L_i$  выбирается чаще.

- Чтобы обеспечить сходимость, выбор компоненты выполняют по правилу: с вероятностью 0.5 выбирают равномерно, с вероятностью 0.5 — с вероятностями  $L_i / \sum_j L_j$ .
- Для генерации с вероятностями  $L_i / \sum_j L_j$  существует алгоритм со сложностью  $O(\log N)$ .

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$
  - Для  $t = 1$  до числа эпох (m)

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$
  - Для  $t = 1$  до числа эпох (m)
    - Выбрать  $i_t \in \{1, \dots, n\}$  равномерно случайно

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$
  - Для  $t = 1$  до числа эпох (m)
    - Выбрать  $i_t \in \{1, \dots, n\}$  равномерно случайно
    - $x_t = x_{t-1} - \alpha [\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \nabla f(\tilde{x})]$

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$
  - Для  $t = 1$  до числа эпох ( $m$ )
    - Выбрать  $i_t \in \{1, \dots, n\}$  равномерно случайно
    - $x_t = x_{t-1} - \alpha [\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \nabla f(\tilde{x})]$
  - Обновить  $\tilde{x} = x_m$

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$
  - Для  $t = 1$  до числа эпох ( $m$ )
    - Выбрать  $i_t \in \{1, \dots, n\}$  равномерно случайно
    - $x_t = x_{t-1} - \alpha [\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \nabla f(\tilde{x})]$
  - Обновить  $\tilde{x} = x_m$

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$
  - Для  $t = 1$  до числа эпох ( $m$ )
    - Выбрать  $i_t \in \{1, \dots, n\}$  равномерно случайно
    - $x_t = x_{t-1} - \alpha [\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \nabla f(\tilde{x})]$
  - Обновить  $\tilde{x} = x_m$

Заметки:

- Две оценки градиента на каждый внутренний шаг.

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$
  - Для  $t = 1$  до числа эпох ( $m$ )
    - Выбрать  $i_t \in \{1, \dots, n\}$  равномерно случайно
    - $x_t = x_{t-1} - \alpha [\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \nabla f(\tilde{x})]$
  - Обновить  $\tilde{x} = x_m$

Заметки:

- Две оценки градиента на каждый внутренний шаг.
- Два параметра: длина эпохи + шаг  $\alpha$ .

# Стохастический градиентный метод редукции дисперсии (SVRG)

- Инициализация:  $\tilde{x} \in \mathbb{R}^d$
- Для  $i_{epoch} = 1$  до # числа эпох
  - Вычислить все градиенты  $\nabla f_i(\tilde{x})$ ; сохранить  $\nabla f(\tilde{x}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\tilde{x})$
  - Инициализировать  $x_0 = \tilde{x}$
  - Для  $t = 1$  до числа эпох ( $m$ )
    - Выбрать  $i_t \in \{1, \dots, n\}$  равномерно случайно
    - $x_t = x_{t-1} - \alpha [\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \nabla f(\tilde{x})]$
  - Обновить  $\tilde{x} = x_m$

Заметки:

- Две оценки градиента на каждый внутренний шаг.
- Два параметра: длина эпохи + шаг  $\alpha$ .
- Линейная скорость сходимости, простое доказательство.



## Adagrad (Duchi, Hazan, and Singer 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.

## Adagrad (Duchi, Hazan, and Singer 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.

## Adagrad (Duchi, Hazan, and Singer 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)

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.
- 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)

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.
- 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  $j = 1, \dots, p$ :

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

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

### 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  $j = 1, \dots, p$ :

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

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

### 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  $j = 1, \dots, p$ :

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

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

### 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.

## Adadelta (Zeiler, 2012)

Расширение RMSProp, нацеленное на снижение зависимости от вручную заданного глобального шага обучения. Вместо накопления всех прошлых квадратов градиентов Adadelta ограничивает окно накопленных прошлых градиентов фиксированным размером  $w$ . Механизм обновления не требует шага обучения  $\alpha$ :

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

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \epsilon}}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}$$

$$\Delta x_j^{(k)} = \rho \Delta x_j^{(k-1)} + (1 - \rho)(\tilde{g}_j^{(k)})^2$$

### Заметки:

- Adadelta адаптирует шаги обучения на основе скользящего окна обновлений градиентов, а не накопления всех прошлых градиентов. Таким образом, настроенные шаги обучения более устойчивы к изменениям динамики модели.

## Adadelta (Zeiler, 2012)

Расширение RMSProp, нацеленное на снижение зависимости от вручную заданного глобального шага обучения. Вместо накопления всех прошлых квадратов градиентов Adadelta ограничивает окно накопленных прошлых градиентов фиксированным размером  $w$ . Механизм обновления не требует шага обучения  $\alpha$ :

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

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \epsilon}}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}$$

$$\Delta x_j^{(k)} = \rho \Delta x_j^{(k-1)} + (1 - \rho)(\tilde{g}_j^{(k)})^2$$

### Заметки:

- Adadelta адаптирует шаги обучения на основе скользящего окна обновлений градиентов, а не накопления всех прошлых градиентов. Таким образом, настроенные шаги обучения более устойчивы к изменениям динамики модели.
- Метод не требует начального установления шага обучения, что упрощает настройку.

## Adadelta (Zeiler, 2012)

Расширение RMSProp, нацеленное на снижение зависимости от вручную заданного глобального шага обучения. Вместо накопления всех прошлых квадратов градиентов Adadelta ограничивает окно накопленных прошлых градиентов фиксированным размером  $w$ . Механизм обновления не требует шага обучения  $\alpha$ :

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

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \epsilon}}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}$$

$$\Delta x_j^{(k)} = \rho \Delta x_j^{(k-1)} + (1 - \rho)(\tilde{g}_j^{(k)})^2$$

### Заметки:

- Adadelta адаптирует шаги обучения на основе скользящего окна обновлений градиентов, а не накопления всех прошлых градиентов. Таким образом, настроенные шаги обучения более устойчивы к изменениям динамики модели.
- Метод не требует начального установления шага обучения, что упрощает настройку.
- Часто используется в глубоком обучении, где масштабы параметров существенно различаются между слоями.

## 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:

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

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:

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

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>1</sup> <sup>2</sup>

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

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:

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

**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) <sup>1</sup> <sup>2</sup>

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

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:

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

**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) <sup>1</sup> <sup>2</sup>

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

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:

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

**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) <sup>1</sup> <sup>2</sup>

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

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:

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

**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:

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

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:

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

## 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:

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

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

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

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:

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

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

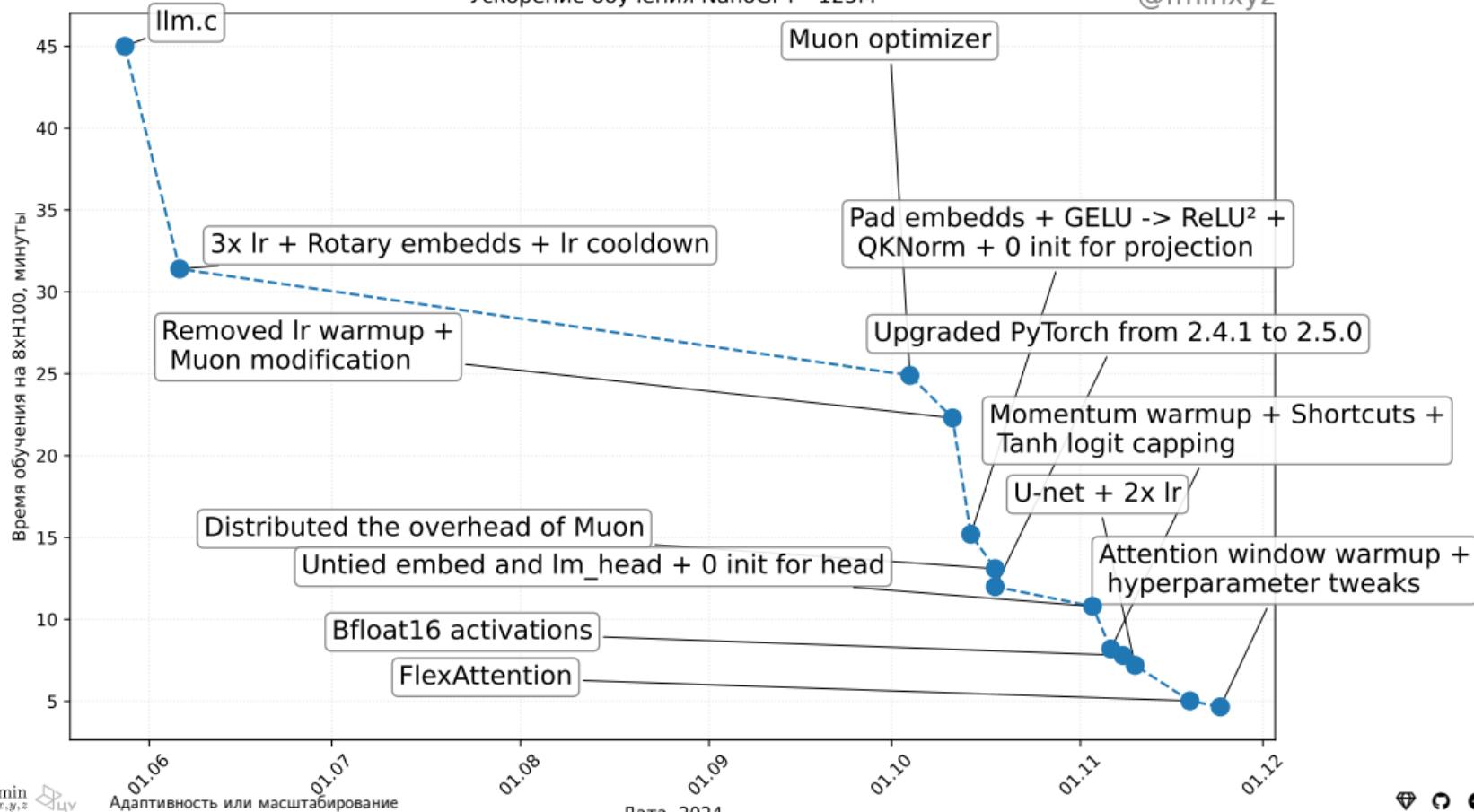


## Как их сравнить? AlgoPerf benchmark

# NanoGPT speedrun

Ускорение обучения NanoGPT - 125M

@fminxyz



## 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).

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

---

### <sup>3</sup>Deriving Muon

## Стохастическая оптимизация

# Много методов

Rosenbrock Function.  
Adaptive stochastic gradient algorithms.  
Learning rate 0.003



SGD расходится с любым шагом обучения для LLS

# Оптимизация для глубокого обучения с практической точки зрения

## Как их сравнить? AlgoPerf benchmark<sup>4 5</sup>

- **AlgoPerf Benchmark:** Сравнивает алгоритмы обучения нейросетей по двум режимам:

# Как их сравнить? AlgoPerf benchmark<sup>4 5</sup>

- **AlgoPerf Benchmark:** Сравнивает алгоритмы обучения нейросетей по двум режимам:
  - **Внешняя настройка (External Tuning):** моделирует тюнинг гиперпараметров при ограниченных ресурсах (5 запусков, квазислучайный поиск). Оценка — медианное минимальное время достижения цели по 5 наборам задач.

# Как их сравнить? AlgoPerf benchmark<sup>4 5</sup>

- **AlgoPerf Benchmark:** Сравнивает алгоритмы обучения нейросетей по двум режимам:
  - **Внешняя настройка (External Tuning):** моделирует тюнинг гиперпараметров при ограниченных ресурсах (5 запусков, квазислучайный поиск). Оценка — медианное минимальное время достижения цели по 5 наборам задач.
  - **Самонастройка (Self-Tuning):** моделирует автоматический тюнинг на одной машине (фиксированный/внутрипетлевой тюнинг, бюджет  $\times 3$ ). Оценка — медианное время выполнения по 5 наборам задач.

# Как их сравнить? AlgoPerf benchmark

4 5

- **AlgoPerf Benchmark:** Сравнивает алгоритмы обучения нейросетей по двум режимам:
  - **Внешняя настройка (External Tuning):** моделирует тюнинг гиперпараметров при ограниченных ресурсах (5 запусков, квазислучайный поиск). Оценка — медианное минимальное время достижения цели по 5 наборам задач.
  - **Самонастройка (Self-Tuning):** моделирует автоматический тюнинг на одной машине (фиксированный/внутрипетлевой тюнинг, бюджет  $\times 3$ ). Оценка — медианное время выполнения по 5 наборам задач.
- **Оценка:** результаты агрегируются с помощью профилей производительности. Профили показывают долю задач, решённых за время, не превышающее фактор  $\tau$  относительно самой быстрой посылки. Итоговый балл — нормированная площадь под кривой профиля ( $1.0 =$  самая быстрая на всех задачах).

# Как их сравнить? AlgoPerf benchmark<sup>4 5</sup>

- **AlgoPerf Benchmark:** Сравнивает алгоритмы обучения нейросетей по двум режимам:
  - **Внешняя настройка (External Tuning):** моделирует тюнинг гиперпараметров при ограниченных ресурсах (5 запусков, квазислучайный поиск). Оценка — медианное минимальное время достижения цели по 5 наборам задач.
  - **Самонастройка (Self-Tuning):** моделирует автоматический тюнинг на одной машине (фиксированный/внутрипетлевой тюнинг, бюджет  $\times 3$ ). Оценка — медианное время выполнения по 5 наборам задач.
- **Оценка:** результаты агрегируются с помощью профилей производительности. Профили показывают долю задач, решённых за время, не превышающее фактор  $\tau$  относительно самой быстрой посылки. Итоговый балл — нормированная площадь под кривой профиля (1.0 = самая быстрая на всех задачах).
- **Вычислительная стоимость:** оценка требует  $\sim 49,240$  суммарных часов на 8x NVIDIA V100 GPUs (в среднем  $\sim 3469$  ч/внешняя настройка,  $\sim 1847$  ч/самонастройка).

<sup>4</sup>Benchmarking Neural Network Training Algorithms

<sup>5</sup>Accelerating neural network training: An analysis of the AlgoPerf competition

## AlgoPerf benchmark

**Summary фиксированных базовых задач в AlgoPerf benchmark.** Функции потерь включают кросс-энтропию (CE), среднюю абсолютную ошибку (L1) и CTC-потерю (Connectionist Temporal Classification, CTC).

Дополнительные метрики оценки: индекс структурного сходства (SSIM), коэффициент ошибок (ER) и доля ошибок по словам (WER), средняя усреднённая точность (mAP) и метрика BLEU (bilingual evaluation understudy). Бюджет времени выполнения соответствует набору правил внешней настройки; набор правил самонастройки допускает обучение, в 3 раза более длительное.

Задача	Датасет	Модель	Потери	Метрика	Целевое значение на валидации	Бюджет времени
Clickthrough rate prediction	CRITEO 1TB	DLRMSMALL	CE	CE	0.123735	7703
MRI reconstruction	FASTMRI	U-NET	L1	SSIM	0.7344	8859
Image classification	IMAGENET	ResNet-50	CE	ER	0.22569	63,008
		ViT	CE	ER	0.22691	77,520
Speech recognition	LIBRISPEECH	Conformer	CTC	WER	0.085884	61,068
		DeepSpeech	CTC	WER	0.119936	55,506
Molecular property prediction	OGBG	GNN	CE	mAP	0.28098	18,477
Translation	WMT	Transformer	CE	BLEU	30.8491	48,151

# AlgoPerf benchmark

Submission	Line	Score
PYTORCH DISTRIBUTED SHAMPOO	■	0.7784
SCHEDULE FREE ADAMW	●	0.7077
GENERALIZED ADAM	○	0.6383
CYCLIC LR	▼	0.6301
NADAMP	◆	0.5909
BASELINE	---	0.5707
AMOS	◆	0.4918
CASPR ADAPTIVE	★	0.4722
LAWA QUEUE	▲	0.3699
LAWA EMA	▼	0.3384
SCHEDULE FREE PRODIGY	—	0

(a) External tuning leaderboard



(b) External tuning performance profiles

# AlgoPerf benchmark



- PyTorch Distr. Shampoo
- Schedule Free AdamW
- Generalized Adam
- Cyclic LR
- NadamP
- Baseline
- Amos
- CASPR Adaptive
- Lawa Queue
- Lawa EMA
- Schedule Free Prodigy

# NanoGPT speedrun

Ускорение обучения NanoGPT - 125M

@fminxyz



## Работают ли трюки, если увеличить размер модели?

### Scaling up the NanoGPT (124M) speedrun

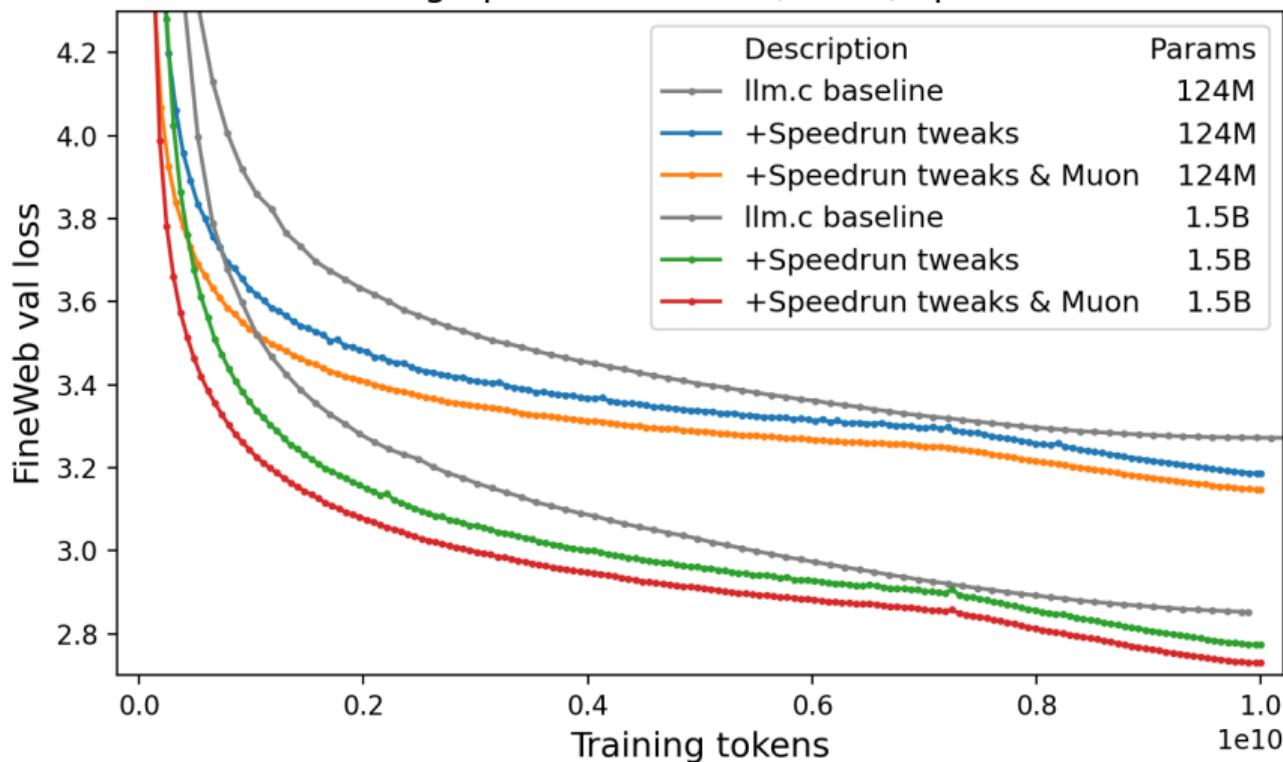


Рис. 4: Источник

## Работают ли трюки, если увеличить размер модели?



Рис. 5: Источник

## Неожиданные истории

## Adam работает хуже для CV, чем для LLM? <sup>6</sup>



Рис. 6: CNNs on MNIST and CIFAR10

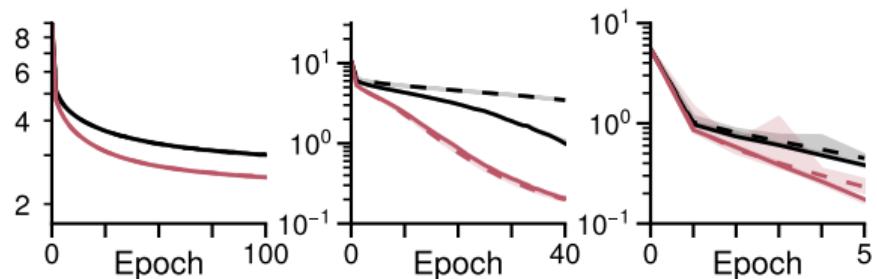


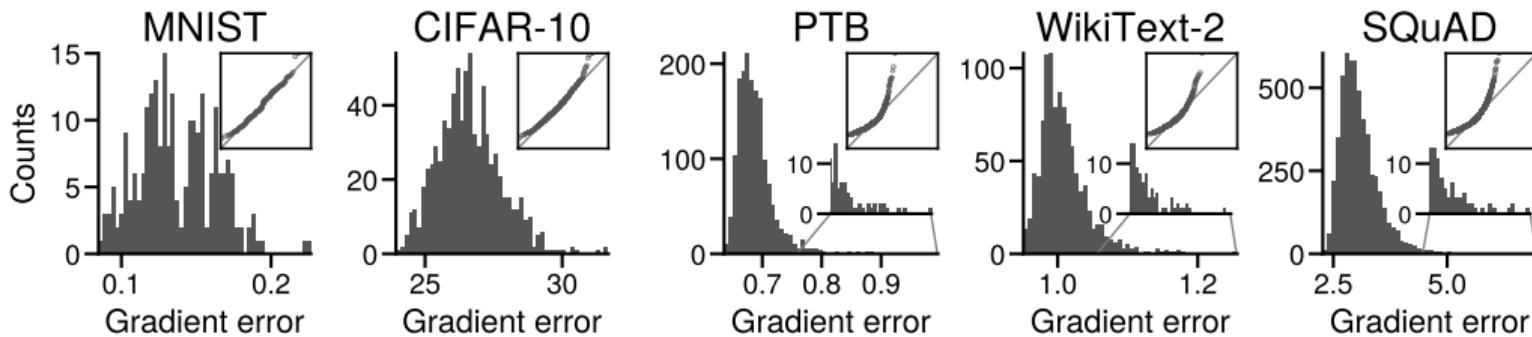
Рис. 7: Transformers on PTB, WikiText2, and SQuAD

Черные линии - SGD; красные линии - Adam.

<sup>6</sup>Linear attention is (maybe) all you need (to understand transformer optimization)

# Почему Adam работает хуже для CV, чем для LLM? <sup>7</sup>

Потому что шум градиентов в языковых моделях имеет тяжелые хвосты?



<sup>7</sup>Linear attention is (maybe) all you need (to understand transformer optimization)

## Почему Adam работает хуже для CV, чем для LLM? <sup>8</sup>

Нет! Метки имеют тяжелые хвосты!

В компьютерном зрении датасеты часто сбалансированы: 1000 котиков, 1000 песелей и т.д.  
В языковых датасетах почти всегда не так: слово *the* встречается часто, слово *tie* на порядки реже



Рис. 8: Распределение частоты токенов в PTB

<sup>8</sup>Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

# Почему Adam работает хуже для CV, чем для LLM? <sup>9</sup>

SGD медленно прогрессирует на редких классах



SGD не добивается прогресса на низкочастотных классах, в то время как Adam добивается. Обучение GPT-2 S на WikiText-103. (a) Распределение классов, отсортированных по частоте встречаемости, разбитых на группы, соответствующие  $\approx 10\%$  данных. (b) Значение функции потерь при обучении. (c, d) Значение функции потерь при обучении для каждой группы при использовании SGD и Adam.

<sup>9</sup>Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

- 💡 Правильная инициализация нейронной сети важна. Функция потерь нейронной сети является высоко невыпуклой; оптимизировать её для достижения «хорошего» решения трудно, требует тщательной настройки.
- Не инициализируйте все веса одинаково — почему?



Правильная инициализация нейронной сети важна. Функция потерь нейронной сети является высоко невыпуклой; оптимизировать её для достижения «хорошего» решения трудно, требует тщательной настройки.

- Не инициализируйте все веса одинаково — почему?
- Случайная инициализация: задавайте случайно, например, из гауссовского распределения  $N(0, \sigma^2)$ , где стандартное отклонение  $\sigma$  зависит от числа нейронов в слое. Это обеспечивает нарушение симметрии. *Symmetry breaking*.



Правильная инициализация нейронной сети важна. Функция потерь нейронной сети является высоко невыпуклой; оптимизировать её для достижения «хорошего» решения трудно, требует тщательной настройки.

- Не инициализируйте все веса одинаково — почему?
- Случайная инициализация: задавайте случайно, например, из гауссова распределения  $N(0, \sigma^2)$ , где стандартное отклонение  $\sigma$  зависит от числа нейронов в слое. Это обеспечивает нарушение симметрии. *Symmetry breaking*.
- Можно найти более полезные советы здесь

<sup>10</sup>On the importance of initialization and momentum in deep learning Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton

# Влияние инициализации весов нейронной сети на сходимость методов<sup>11</sup>



Рис. 9: 22-layer ReLU net: good init converges faster

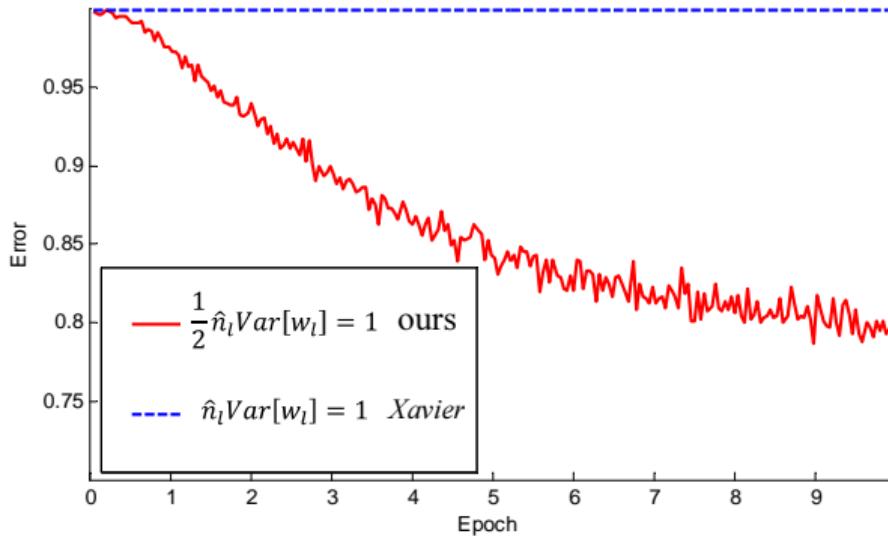


Рис. 10: 30-layer ReLU net: good init is able to converge

<sup>11</sup>Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

## Весёлые истории

Градиентный спуск сходится к локальному минимуму



# Градиентный спуск сходится к локальному минимуму



Стохастический градиентный спуск  
выпрыгивает из локальных минимумов



## Визуализация с помощью проекции на прямую

- Обозначим начальную точку как  $w_0$ , представляющую собой веса нейронной сети при инициализации. Веса, полученные после обучения, обозначим как  $\hat{w}$ .

$$L(\alpha) = L(w_0 + \alpha w_1), \text{ where } \alpha \in [-b, b].$$

## Визуализация с помощью проекции на прямую

- Обозначим начальную точку как  $w_0$ , представляющую собой веса нейронной сети при инициализации. Веса, полученные после обучения, обозначим как  $\hat{w}$ .
- Генерируем случайный вектор такой же размерности и нормы  $w_1 \in \mathbb{R}^p$ , затем вычисляем значение функции потерь вдоль этого вектора:

$$L(\alpha) = L(w_0 + \alpha w_1), \text{ where } \alpha \in [-b, b].$$

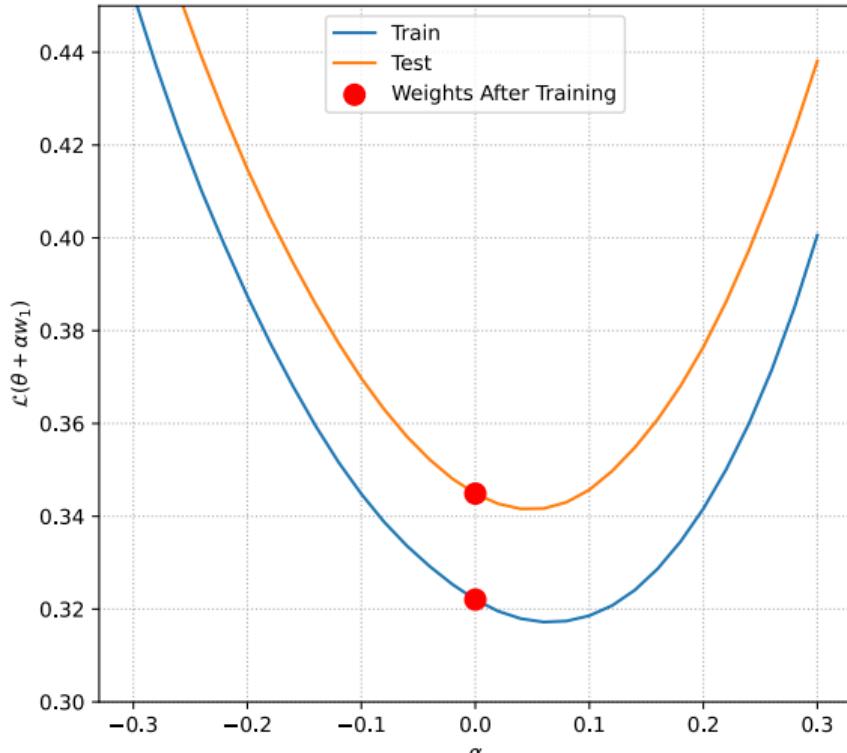
# Проекция функции потерь нейронной сети на прямую

No Dropout

Loss surface, Line projection around the starting point



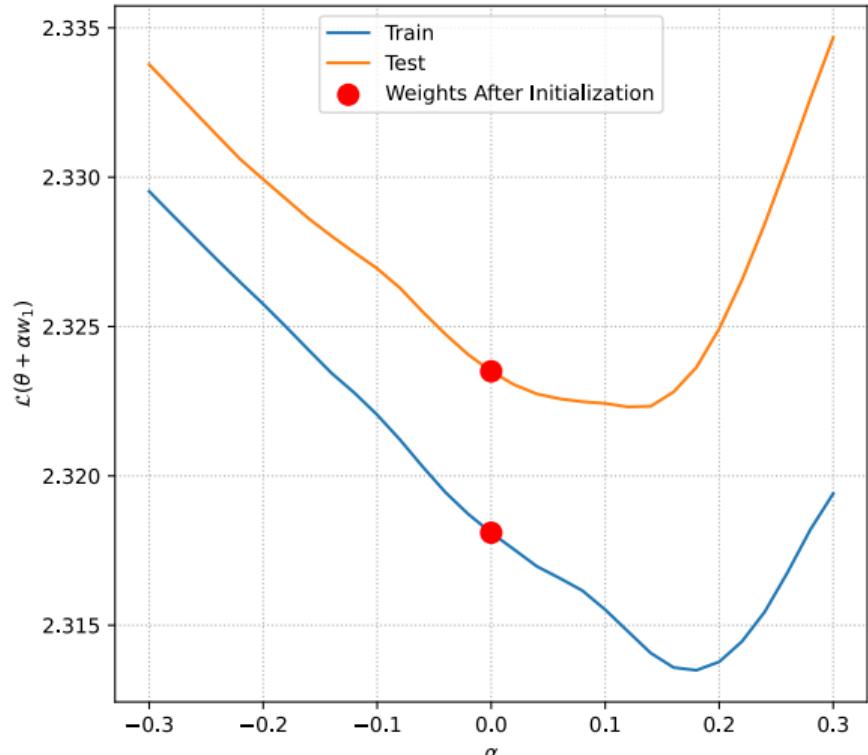
Loss surface, Line projection around the final point



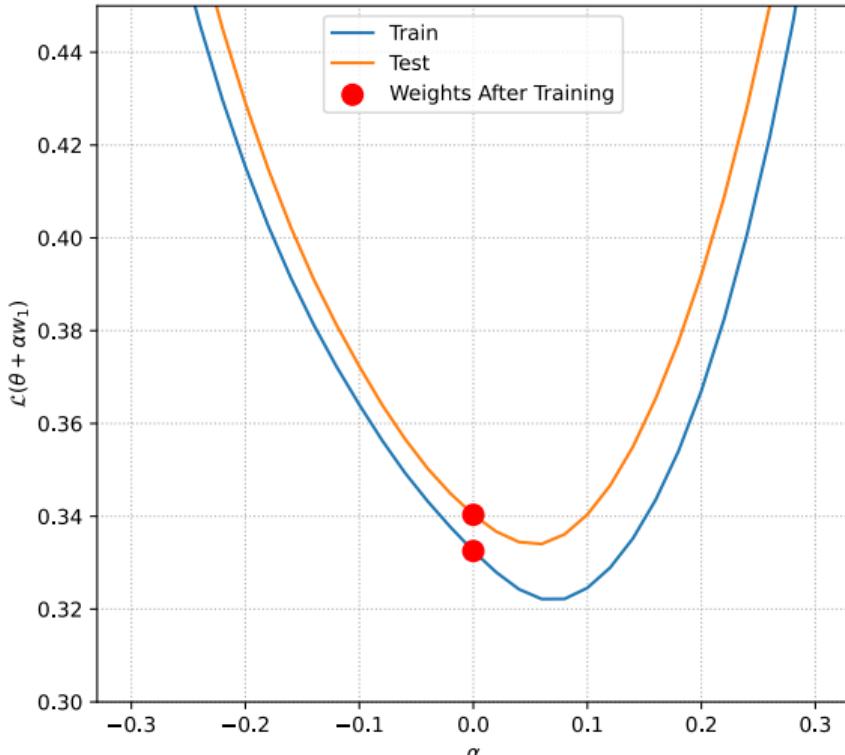
# Проекция функции потерь нейронной сети на прямую

Dropout 0.2

Loss surface, Line projection around the starting point



Loss surface, Line projection around the final point



# Проекция функции потерь нейронной сети на плоскость

- Мы можем расширить эту идею и построить проекцию поверхности потерь на плоскость, которая задается 2 случайными векторами.

$$L(\alpha, \beta) = L(w_0 + \alpha w_1 + \beta w_2), \text{ where } \alpha, \beta \in [-b, b]^2.$$

No Dropout. Plane projection of loss surface.

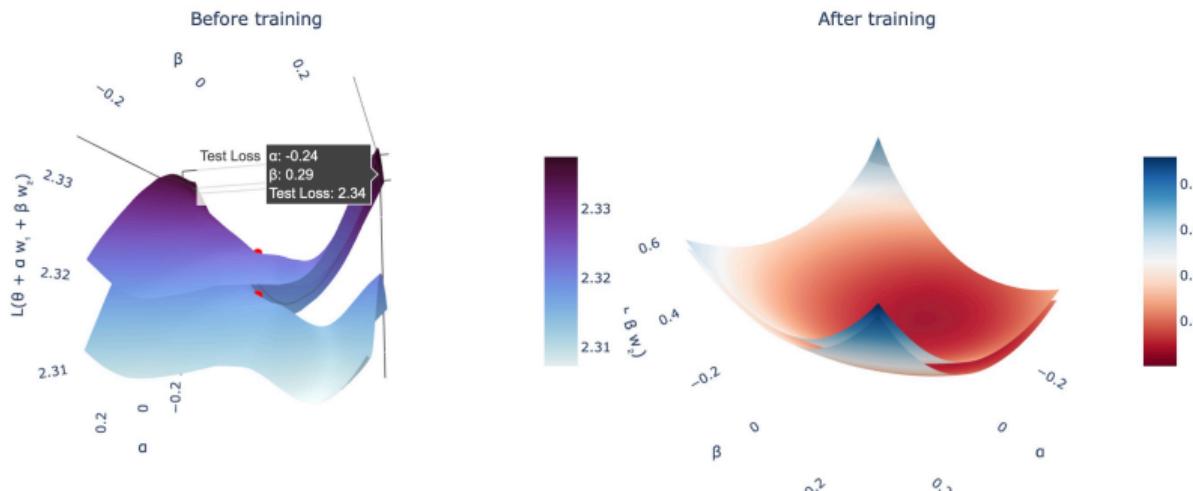


## Проекция функции потерь нейронной сети на плоскость

- Мы можем расширить эту идею и построить проекцию поверхности потерь на плоскость, которая задается 2 случайными векторами.
- Два случайных гауссовых вектора в пространстве большой размерности с высокой вероятностью ортогональны.

$$L(\alpha, \beta) = L(w_0 + \alpha w_1 + \beta w_2), \text{ where } \alpha, \beta \in [-b, b]^2.$$

No Dropout. Plane projection of loss surface.



Может ли быть полезно изучение таких проекций? <sup>12</sup>



Рис. 14: The loss surface of ResNet-56  
without skip connections



Рис. 15: The loss surface of ResNet-56 with skip connections

<sup>12</sup>Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein

# Может ли быть полезно изучение таких проекций, если серьезно? <sup>13</sup>



Рис. 16: Examples of a loss landscape of a typical CNN model on FashionMNIST and CIFAR10 datasets found with MPO. Loss values are color-coded according to a logarithmic scale

<sup>13</sup>Loss Landscape Sightseeing with Multi-Point Optimization, Ivan Skorokhodov, Mikhail Burtsev  
 $f \rightarrow \min_{x,y,z}$  Весёлые истории

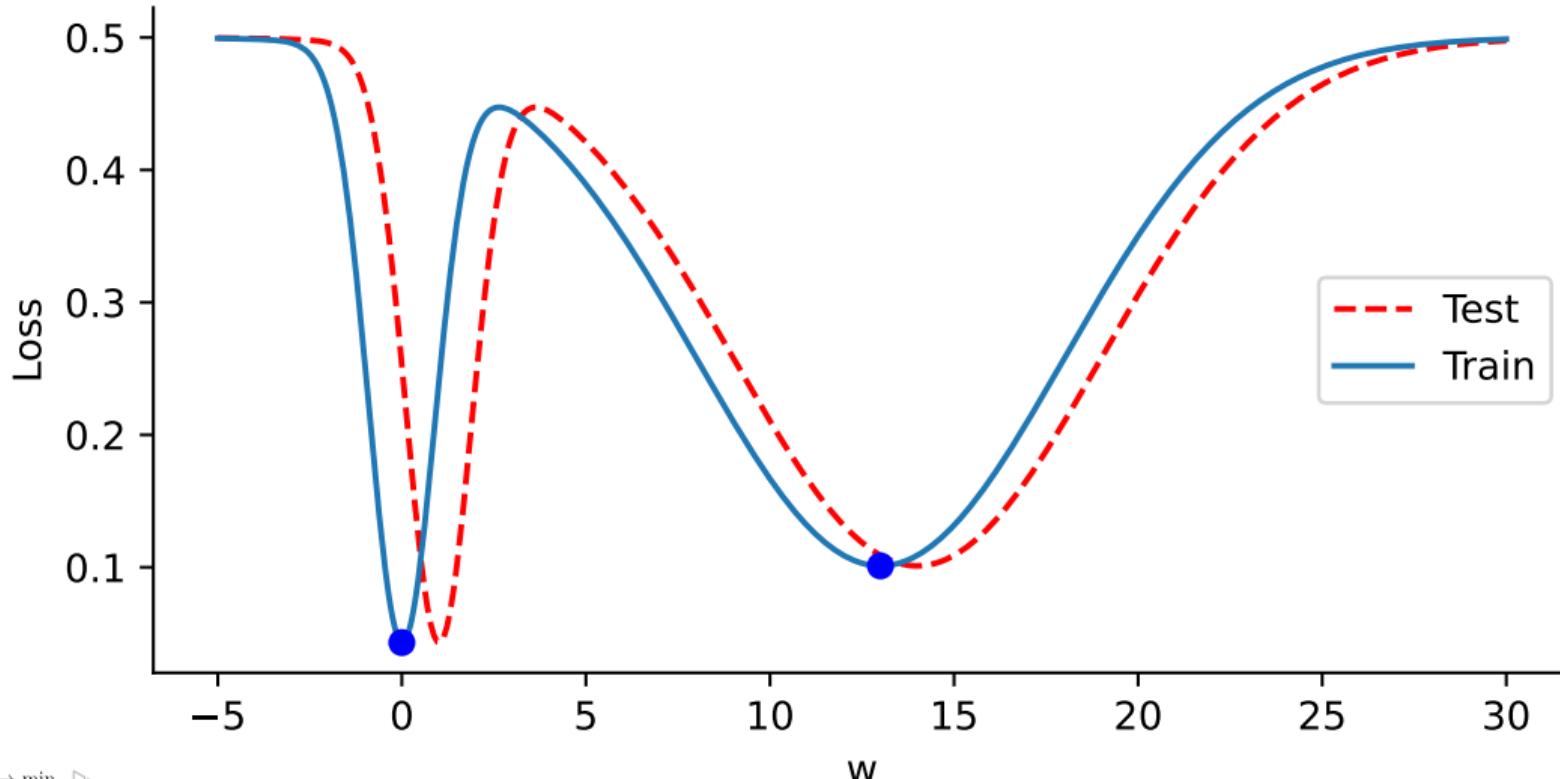
## Ширина локальных минимумов

Узкие и широкие локальные минимумы



## Ширина локальных минимумов

Узкие и широкие локальные минимумы



## Ширина локальных минимумов

Узкие и широкие локальные минимумы



# Экспоненциальный шаг обучения

- Exponential Learning Rate Schedules for Deep Learning

## Double Descent<sup>14</sup>



<sup>14</sup>Reconciling modern machine learning practice and the bias-variance trade-off, Mikhail Belkin, Daniel Hsu, Siyuan Ma, Soumik Mandal

## Double Descent

Polynomial Fitting



@fminxyz



Modular Division (training on 50% of data)

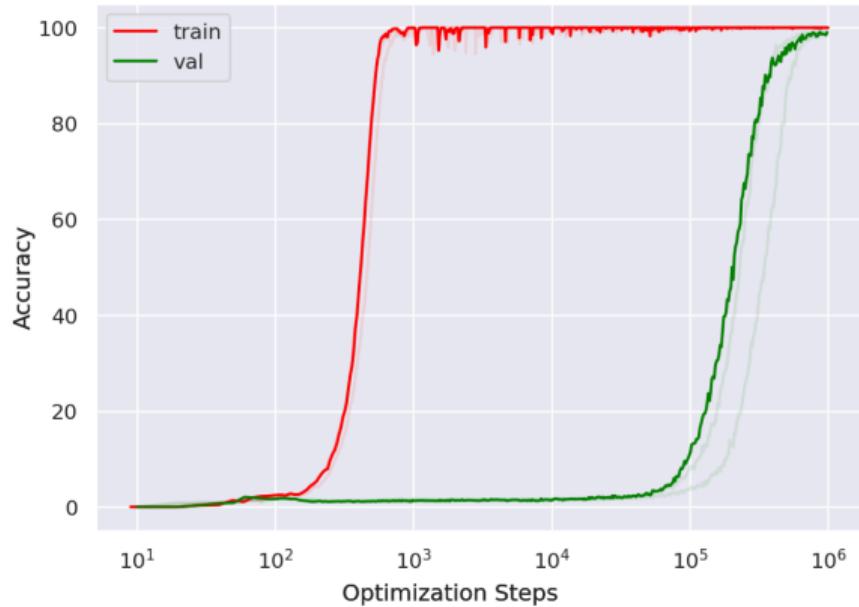


Рис. 17: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments (~ half an hour) is available [here](#)

- Рекомендую посмотреть лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети** (Surprising properties of loss landscape in overparameterized models). видео, Презентация

Modular Division (training on 50% of data)



Рис. 17: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments ( $\sim$  half an hour) is available here

- Рекомендую посмотреть лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети** (Surprising properties of loss landscape in overparameterized models). видео, Презентация
- Автор канала Свидетели Градиента собирает интересные наблюдения и эксперименты про гроккинг.

Modular Division (training on 50% of data)



Рис. 17: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments ( $\sim$  half an hour) is available here

- Рекомендую посмотреть лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети** (Surprising properties of loss landscape in overparameterized models). видео, Презентация
- Автор канала Свидетели Градиента собирает интересные наблюдения и эксперименты про гроккинг.
- Также есть видео с его докладом **Чем не является гроккинг**.

<sup>15</sup>Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, Vedant Misra