



# Перспективные направления исследований для совместной лаборатории ЦУ x AIRI

Даниил Меркулов

30 ноября 2024

# Optimizers

# Adam работает хуже для CV, чем для LLM? <sup>1</sup>

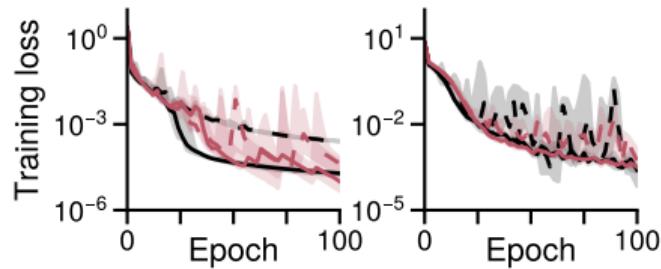


Рис. 1: CNNs on MNIST and CIFAR10

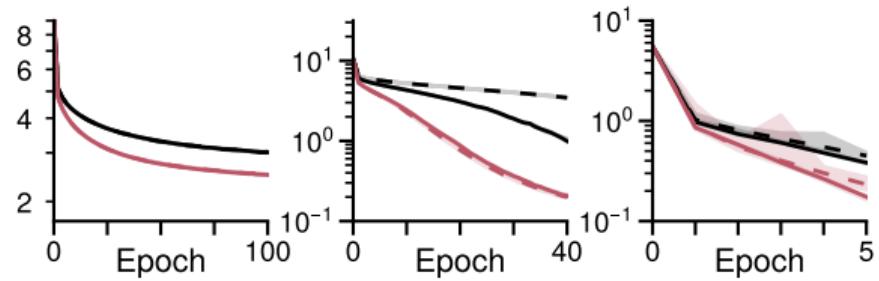


Рис. 2: Transformers on PTB, WikiText2, and SQuAD

<sup>1</sup>Linear attention is (maybe) all you need (to understand transformer optimization)

# Почему Adam работает хуже для CV, чем для LLM? <sup>2</sup>

Потому что шум градиентов в языковых моделях имеет тяжелые хвосты?



<sup>2</sup>Linear attention is (maybe) all you need (to understand transformer optimization)

## Почему Adam работает хуже для CV, чем для LLM? <sup>3</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

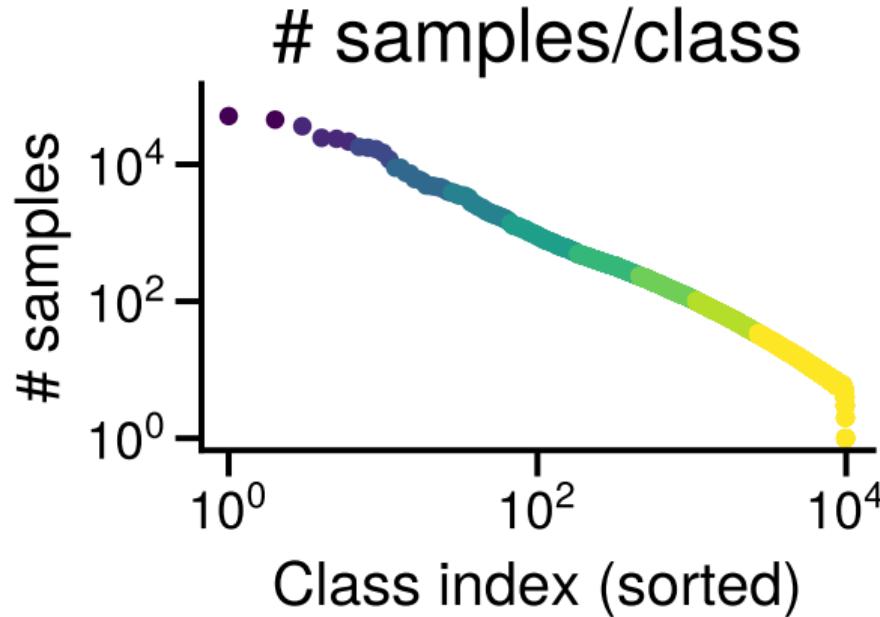
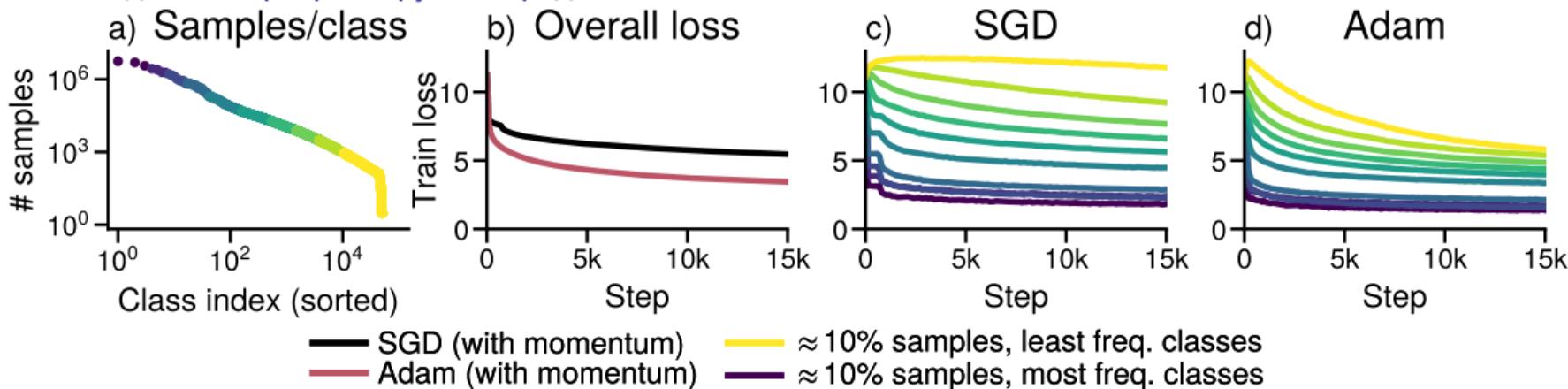


Рис. 3: Распределение частоты токенов в PTB

<sup>3</sup>Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

# Почему Adam работает хуже для CV, чем для LLM? <sup>4</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>4</sup>Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

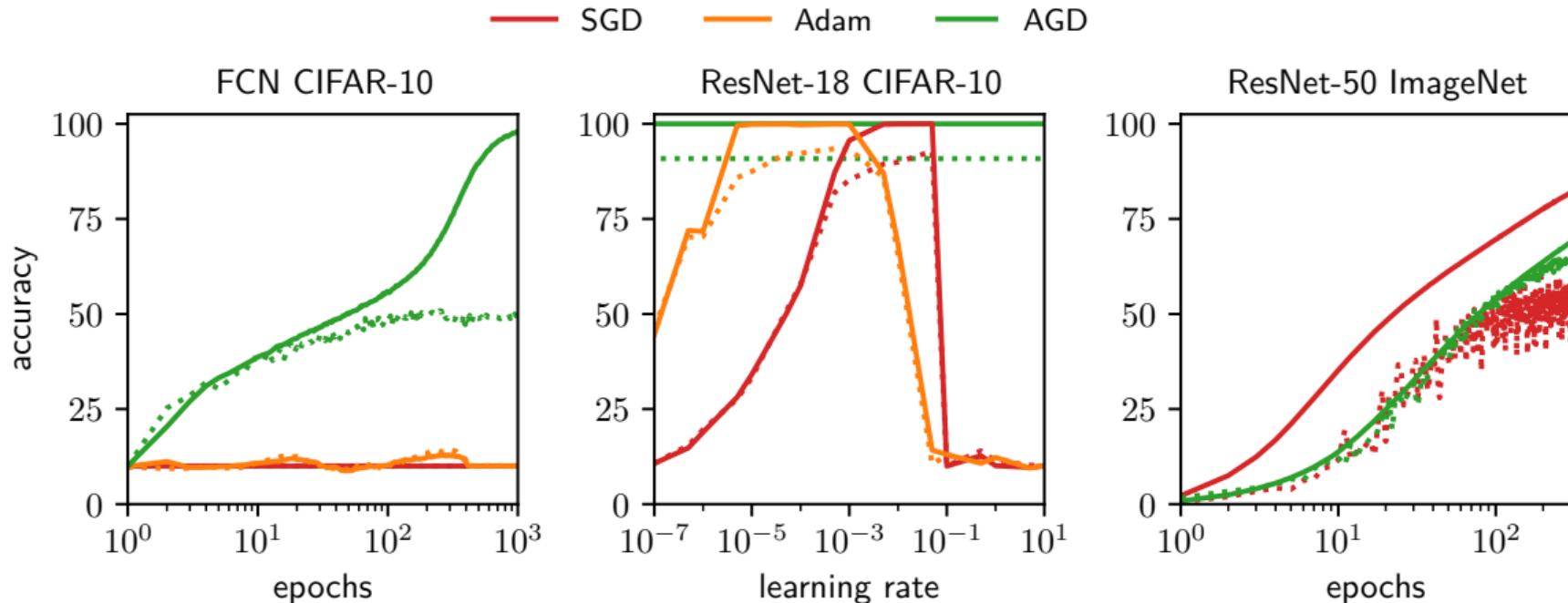
## Automatic Gradient Descent<sup>5</sup>

```
def initialise_weights():
    for layer k in {1,...,L}:
         $\mathbf{W}_k \sim \text{UNIFORM}(\text{orthogonal}(d_k, d_{k-1}))$  # sample a semi-orthogonal matrix
         $\mathbf{W}_k \leftarrow \mathbf{W}_k \cdot \sqrt{\frac{d_k}{d_{k-1}}}$  # rescale its singular values
    def update_weights():
         $G \leftarrow \frac{1}{L} \sum_{l=1}^L \|\nabla_{\mathbf{W}_k} \mathcal{L}\|_F \cdot \sqrt{\frac{d_k}{d_{k-1}}}$  # get gradient summary
         $\eta \leftarrow \log \frac{1+\sqrt{1+4G}}{2}$  # set automatic learning rate
        for layer k in {1,...,L}:
             $\mathbf{W}_k \leftarrow \mathbf{W}_k - \frac{\eta}{L} \cdot \frac{\nabla_{\mathbf{W}_k} \mathcal{L}}{\|\nabla_{\mathbf{W}_k} \mathcal{L}\|_F} \cdot \sqrt{\frac{d_k}{d_{k-1}}}$  # update weights
```

**Algorithm 1: Automatic gradient descent.** The matrix  $\mathbf{W}_k$  in  $\mathbb{R}^{d_k \times d_{k-1}}$  is the weight matrix at layer  $k$ . The gradient  $\nabla_{\mathbf{W}_k} \mathcal{L}$  is with respect to the objective  $\mathcal{L}$  evaluated on a mini-batch  $B$  of training samples.

<sup>5</sup>Automatic Gradient Descent: Deep Learning without Hyperparameters

## Automatic Gradient Descent<sup>6</sup>



<sup>6</sup>Automatic Gradient Descent: Deep Learning without Hyperparameters

# Prodigy<sup>7</sup>

## Algorithm 1 Prodigy (GD version)

```
1: Input:  $d_0 > 0, x_0, G \geq 0$ 
2: for  $k = 0$  to  $n$  do
3:    $g_k \in \partial f(x_k)$ 
4:   Choose weight  $\lambda_k$  (default:  $\lambda_k = 1$ )
5:    $\eta_k = \frac{d_k^2 \lambda_k}{\sqrt{d_k^2 G^2 + \sum_{i=0}^k d_i^2 \lambda_i^2 \|g_i\|^2}}$ 
6:    $x_{k+1} = x_k - \eta_k g_k$ 
7:    $\hat{d}_{k+1} = \frac{\sum_{i=0}^k \eta_i \langle g_i, x_0 - x_i \rangle}{\|x_{k+1} - x_0\|}$ 
8:    $d_{k+1} = \max(d_k, \hat{d}_{k+1})$ 
9: end for
10: Return  $\hat{x}_n = \frac{1}{n+1} \sum_{k=0}^n \eta_k x_k$ 
```

## Algorithm 2 Prodigy (Dual Averaging version)

```
1: Input:  $d_0 > 0, x_0, G \geq 0; s_0 = 0 \in \mathbb{R}^p$ 
2: for  $k = 0$  to  $n$  do
3:    $g_k \in \partial f(x_k)$ 
4:    $\lambda_k = d_k^2$ 
5:    $s_{k+1} = s_k + \lambda_k g_k$ 
6:    $\hat{d}_{k+1} = \frac{\sum_{i=0}^k \lambda_i \langle g_i, x_0 - x_i \rangle}{\|s_{k+1}\|}$ 
7:    $d_{k+1} = \max(d_k, \hat{d}_{k+1})$ 
8:    $\gamma_{k+1} = \frac{1}{\sqrt{\lambda_{k+1} G^2 + \sum_{i=0}^k \lambda_i \|g_i\|^2}}$ 
9:    $x_{k+1} = x_0 - \gamma_{k+1} s_{k+1}$ 
10: end for
11: Return  $\bar{x}_n = \frac{1}{n+1} \sum_{k=0}^n \lambda_k x_k$ 
```

<sup>7</sup>Prodigy: An Expeditiously Adaptive Parameter-Free Learner

---

## Algorithm 1 Proposed optimizer

---

**Require:** Learning rate  $\eta$ , momentum  $\mu$ , weight decay  $\lambda$

- 1: Initialize  $B_0 \leftarrow 0$
  - 2: **for**  $t = 1, \dots$  **do**
  - 3:     Compute gradient  $G_t \leftarrow \nabla_{\theta} \mathcal{L}_t(\theta_{t-1})$
  - 4:      $B_t \leftarrow \mu B_{t-1} + G_t$
  - 5:      $G_t \leftarrow \mu B_t + G_t$
  - 6:      $O_t \leftarrow \text{NewtonSchulz5}(G_t)$
  - 7:     Update parameters  $\theta_t \leftarrow \theta_{t-1} - \eta(O_t + \lambda \theta_{t-1})$
  - 8: **end for**
  - 9: **return**  $\theta_t$
-

## Scaling Laws

## Cooldown<sup>10 11</sup>



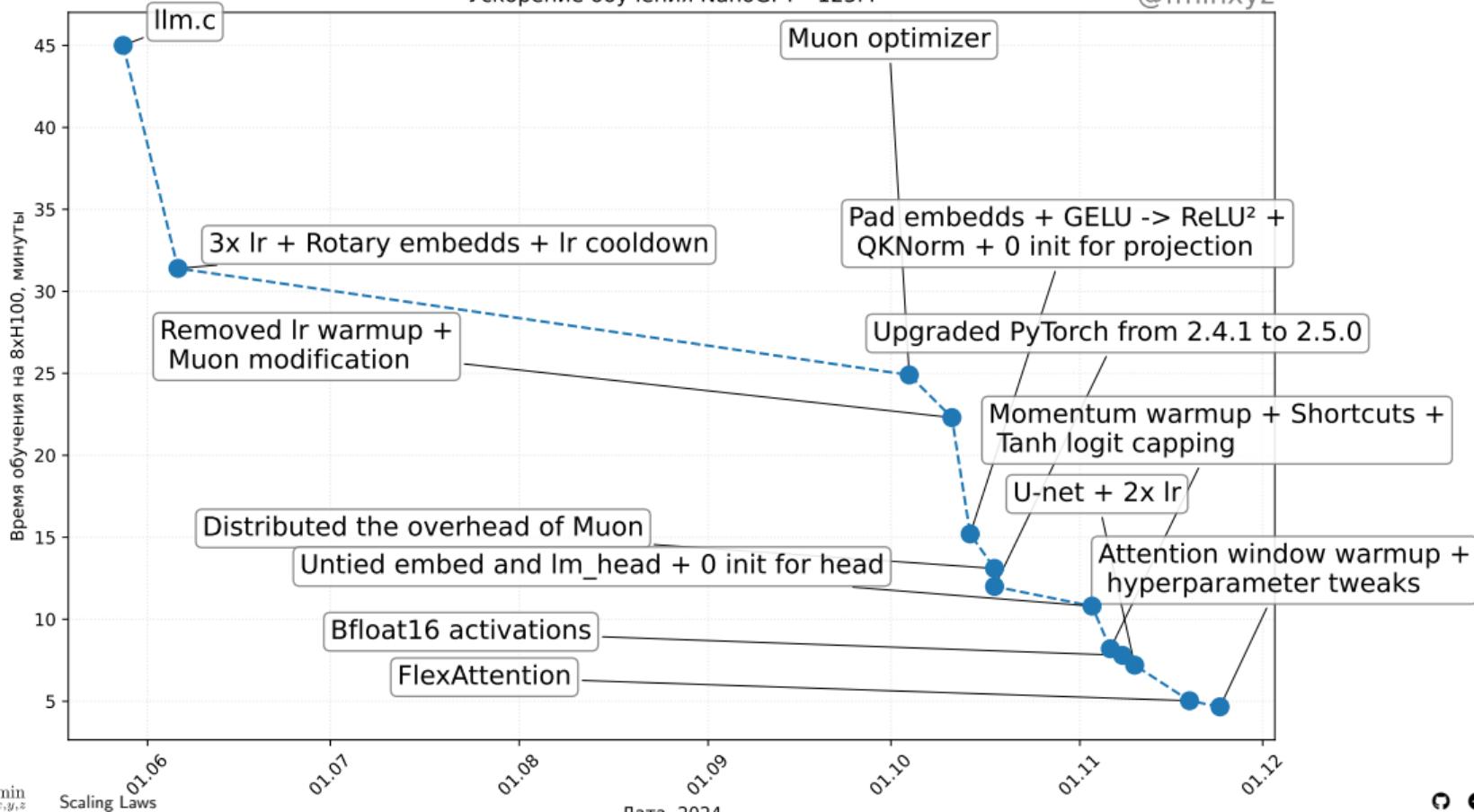
<sup>10</sup>Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations

<sup>11</sup>Scaling Vision Transformers

# NanoGPT speedrun

Ускорение обучения NanoGPT - 125M

@fminxyz



## Работают ли трюки, если увеличить размер модели?

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



Рис. 5: Источник

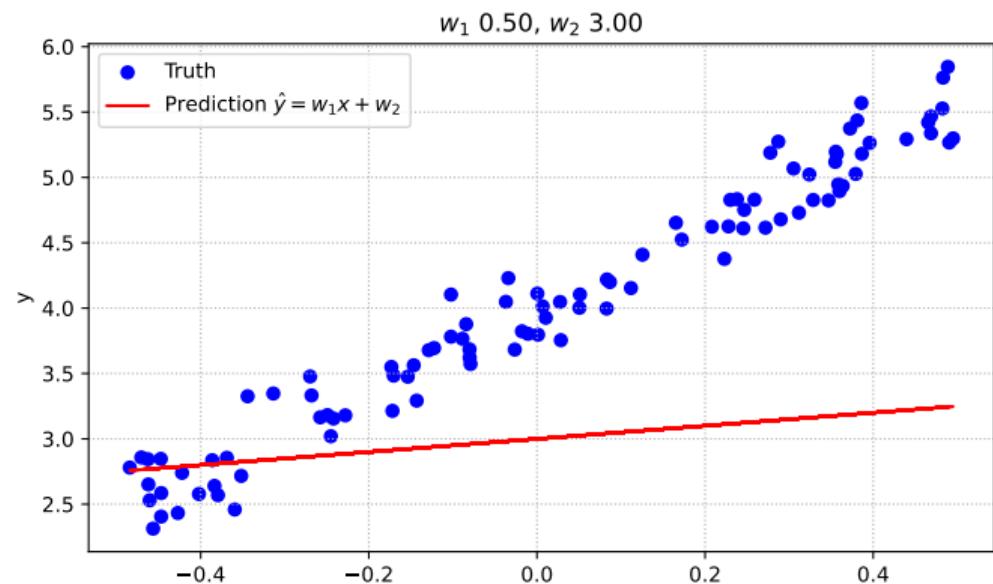
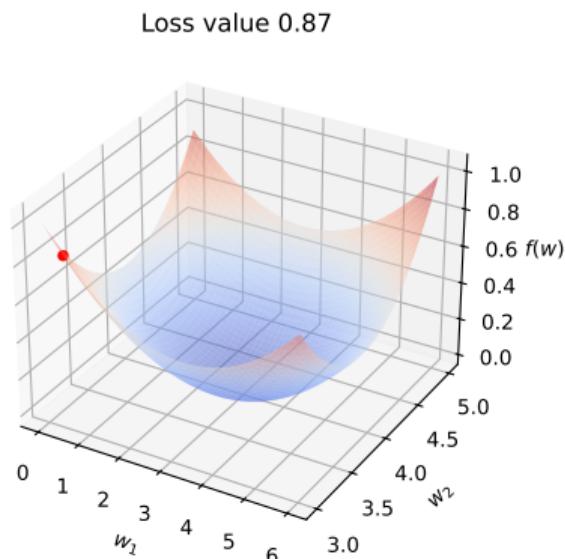
## Работают ли трюки, если увеличить размер модели?



Рис. 6: Источник



# Градиентный спуск для линейной регрессии



# Методы оптимизации

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

## Методы нулевого порядка

- Метод Нелдера - Мида
- Эволюционные методы
- Генетические алгоритмы
- Безградиентные методы
- и другие...

# Методы оптимизации

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

## Методы нулевого порядка

- Метод Нелдера - Мида
- Эволюционные методы
- Генетические алгоритмы
- Безградиентные методы
- и другие...

## Методы первого порядка

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

# Методы оптимизации

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

## Методы нулевого порядка

- Метод Нелдера - Мида
- Эволюционные методы
- Генетические алгоритмы
- Безградиентные методы
- и другие...

## Методы первого порядка

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

## Методы второго порядка

$$x_{k+1} = x_k - \alpha_k [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

## Проклятие размерности методов нулевого порядка

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

## Проклятие размерности методов нулевого порядка

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

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

Zero order GD:  $x_{k+1} = x_k - \alpha_k G,$

где  $G$  - двухточечная или многоточечная оценка градиента по значениям функции

## Проклятие размерности методов нулевого порядка

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

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

Zero order GD:  $x_{k+1} = x_k - \alpha_k G,$

где  $G$  - двухточечная или многоточечная оценка градиента по значениям функции

	$f(x)$ - гладкая	$f(x)$ - гладкая и выпуклая	$f(x)$ - гладкая и сильно выпуклая
GD	$\ \nabla f(x_k)\ ^2 \approx \mathcal{O}\left(\frac{1}{k}\right)$	$f(x_k) - f^* \approx \mathcal{O}\left(\frac{1}{k}\right)$	$\ x_k - x^*\ ^2 \approx \mathcal{O}\left(\left(1 - \frac{\mu}{L}\right)^k\right)$
Zero order GD	$\ \nabla f(x_k)\ ^2 \approx \mathcal{O}\left(\frac{n}{k}\right)$	$f(x_k) - f^* \approx \mathcal{O}\left(\frac{n}{k}\right)$	$\ x_k - x^*\ ^2 \approx \mathcal{O}\left(\left(1 - \frac{\mu}{nL}\right)^k\right)$

## Пример: задача многомерного шкалирования

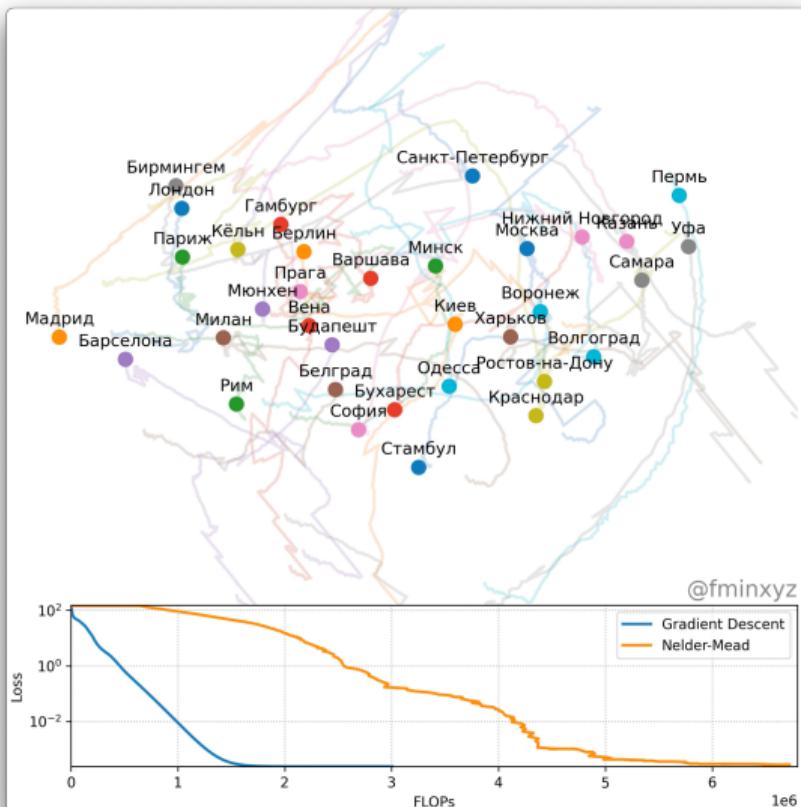
Пусть у нас есть матрица попарных расстояний для  $N$   $d$ -мерных объектов  $D \in \mathbb{R}^{N \times N}$ . По этой матрице мы должны восстановить начальные координаты  $W_i \in \mathbb{R}^d$ ,  $i = 1, \dots, N$ .

## Пример: задача многомерного шкалирования

Пусть у нас есть матрица попарных расстояний для  $N$   $d$ -мерных объектов  $D \in \mathbb{R}^{N \times N}$ . По этой матрице мы должны восстановить начальные координаты  $W_i \in \mathbb{R}^d$ ,  $i = 1, \dots, N$ .

$$L(W) = \sum_{i,j=1}^N (\|W_i - W_j\|_2^2 - D_{i,j})^2 \rightarrow \min_{W \in \mathbb{R}^{N \times d}}$$

## Пример: задача многомерного шкалирования



## В машинном обучении задача оптимизации часто имеет особенный вид

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

- $f_i(x)$  - значение функции потерь модели при весах  $x$  на  $i$ -ом объекте обучающей выборки

## В машинном обучении задача оптимизации часто имеет особенный вид

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

- $f_i(x)$  - значение функции потерь модели при весах  $x$  на  $i$ -ом объекте обучающей выборки
- $n$  - число обучаемых параметров модели ( $175 \cdot 10^9$  для GPT-3.5,  $405 \cdot 10^9$  для Llama 3.2)

## В машинном обучении задача оптимизации часто имеет особенный вид

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

- $f_i(x)$  - значение функции потерь модели при весах  $x$  на  $i$ -ом объекте обучающей выборки
- $n$  - число обучаемых параметров модели ( $175 \cdot 10^9$  для GPT-3.5,  $405 \cdot 10^9$  для Llama 3.2)
- $N$  - размер обучающей выборки (для ImageNet  $\approx 1.4 \cdot 10^7$ , для WikiText  $\approx 10^8$ , для FineWeb-Edu  $\approx 1.3 \cdot 10^{12}$ ).

## В машинном обучении задача оптимизации часто имеет особенный вид

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

- $f_i(x)$  - значение функции потерь модели при весах  $x$  на  $i$ -ом объекте обучающей выборки
- $n$  - число обучаемых параметров модели ( $175 \cdot 10^9$  для GPT-3.5,  $405 \cdot 10^9$  для Llama 3.2)
- $N$  - размер обучающей выборки (для ImageNet  $\approx 1.4 \cdot 10^7$ , для WikiText  $\approx 10^8$ , для FineWeb-Edu  $\approx 1.3 \cdot 10^{12}$ ).

## В машинном обучении задача оптимизации часто имеет особенный вид

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

- $f_i(x)$  - значение функции потерь модели при весах  $x$  на  $i$ -ом объекте обучающей выборки
- $n$  - число обучаемых параметров модели ( $175 \cdot 10^9$  для GPT-3.5,  $405 \cdot 10^9$  для Llama 3.2)
- $N$  - размер обучающей выборки (для ImageNet  $\approx 1.4 \cdot 10^7$ , для WikiText  $\approx 10^8$ , для FineWeb-Edu  $\approx 1.3 \cdot 10^{12}$ ).

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

## В машинном обучении задача оптимизации часто имеет особенный вид

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

- $f_i(x)$  - значение функции потерь модели при весах  $x$  на  $i$ -ом объекте обучающей выборки
- $n$  - число обучаемых параметров модели ( $175 \cdot 10^9$  для GPT-3.5,  $405 \cdot 10^9$  для Llama 3.2)
- $N$  - размер обучающей выборки (для ImageNet  $\approx 1.4 \cdot 10^7$ , для WikiText  $\approx 10^8$ , для FineWeb-Edu  $\approx 1.3 \cdot 10^{12}$ ).

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

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

Тяжело считать при больших  $N$ !

## В машинном обучении задача оптимизации часто имеет особенный вид

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

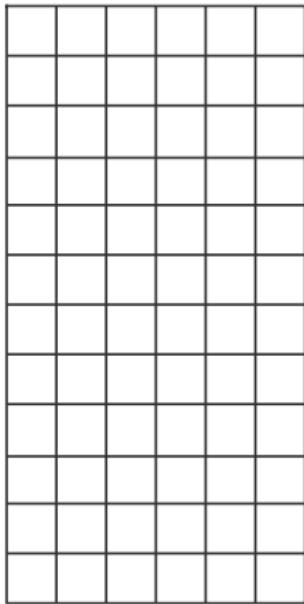
- $f_i(x)$  - значение функции потерь модели при весах  $x$  на  $i$ -ом объекте обучающей выборки
- $n$  - число обучаемых параметров модели ( $175 \cdot 10^9$  для GPT-3.5,  $405 \cdot 10^9$  для Llama 3.2)
- $N$  - размер обучающей выборки (для ImageNet  $\approx 1.4 \cdot 10^7$ , для WikiText  $\approx 10^8$ , для FineWeb-Edu  $\approx 1.3 \cdot 10^{12}$ ).

$$\nabla f(x_k) \approx \frac{1}{b} \sum_{i=1}^b \nabla f_{j_i}(x)$$

$$x_{k+1} = x_k - \frac{\alpha_k}{b} \sum_{i=1}^b \nabla f_{j_i}(x) \tag{SGD}$$

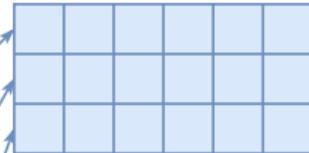
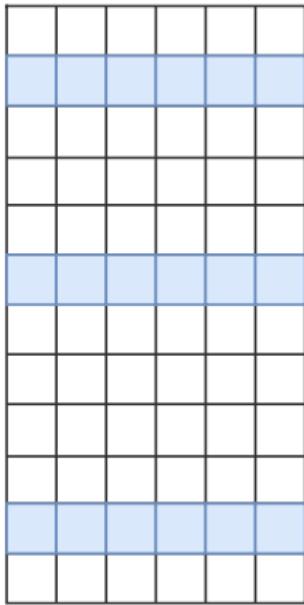
Можно считать при больших  $N$ !

### Данные



## Идея SGD и батчей

Данные



1 Батч

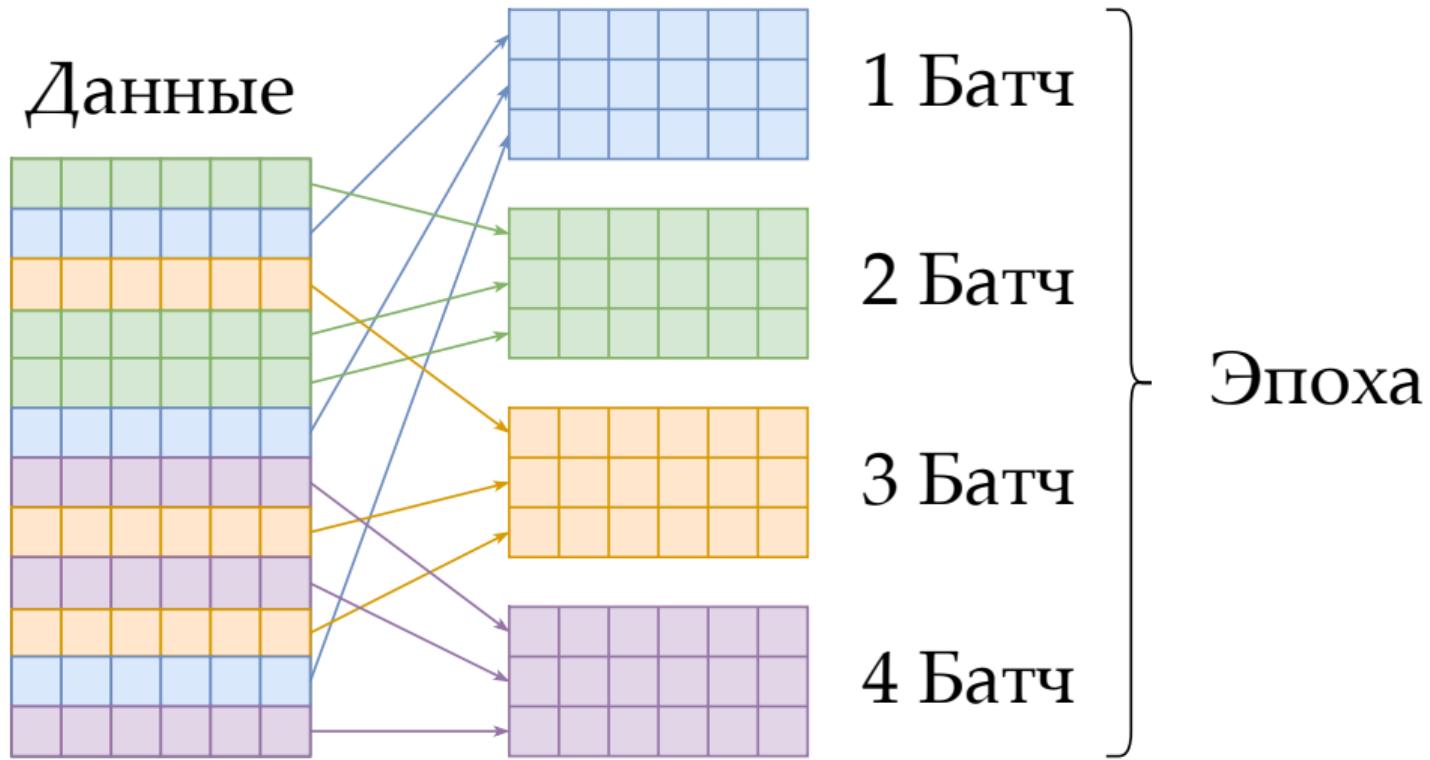
## Идея SGD и батчей



## Идея SGD и батчей



## Идея SGD и батчей



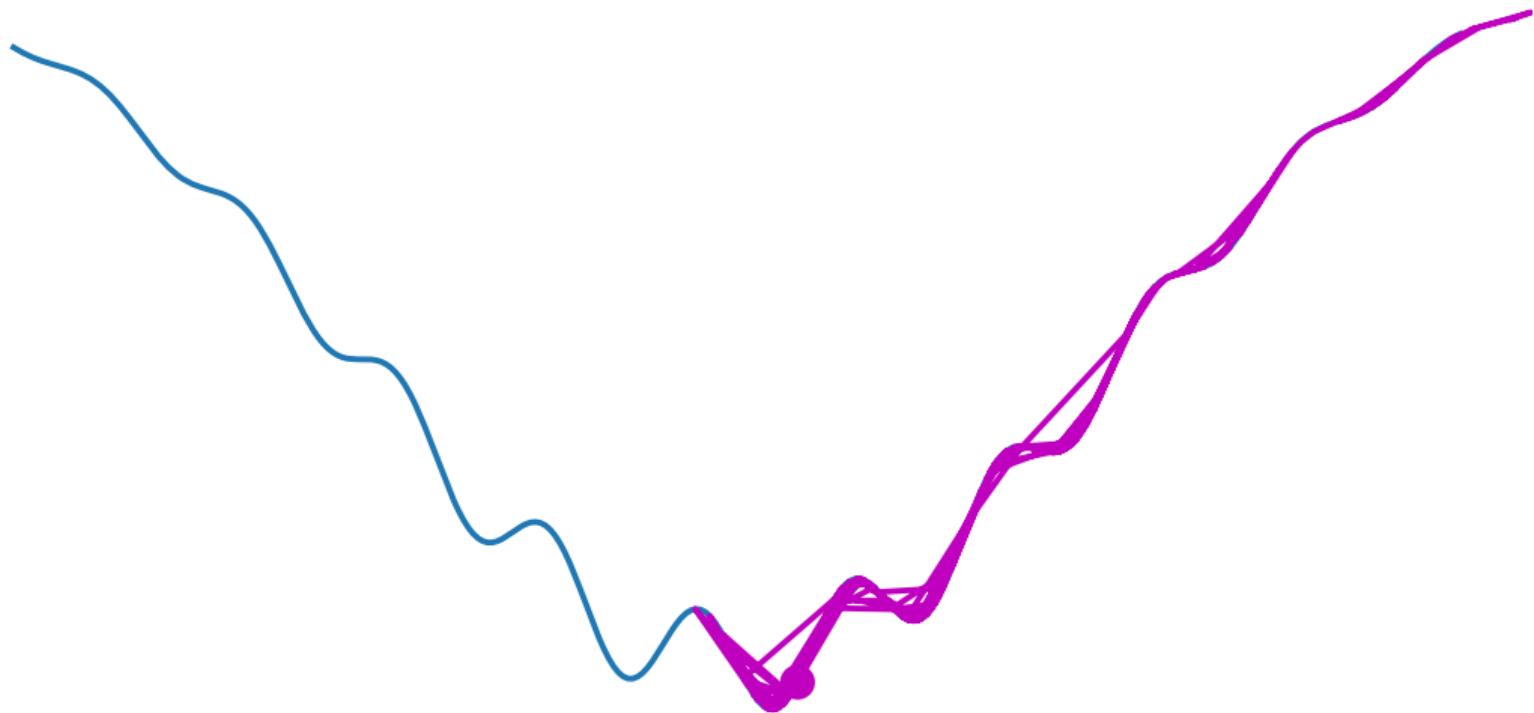
Градиентный спуск сходится к локальному минимуму



# Градиентный спуск сходится к локальному минимуму



Стохастический градиентный спуск  
выпрыгивает из локальных минимумов



## SGD не сходится с постоянным шагом для выпуклой функции

Stochastic Gradient Descent. Batch = 1



@fminxyz

## 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:

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

## 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:

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

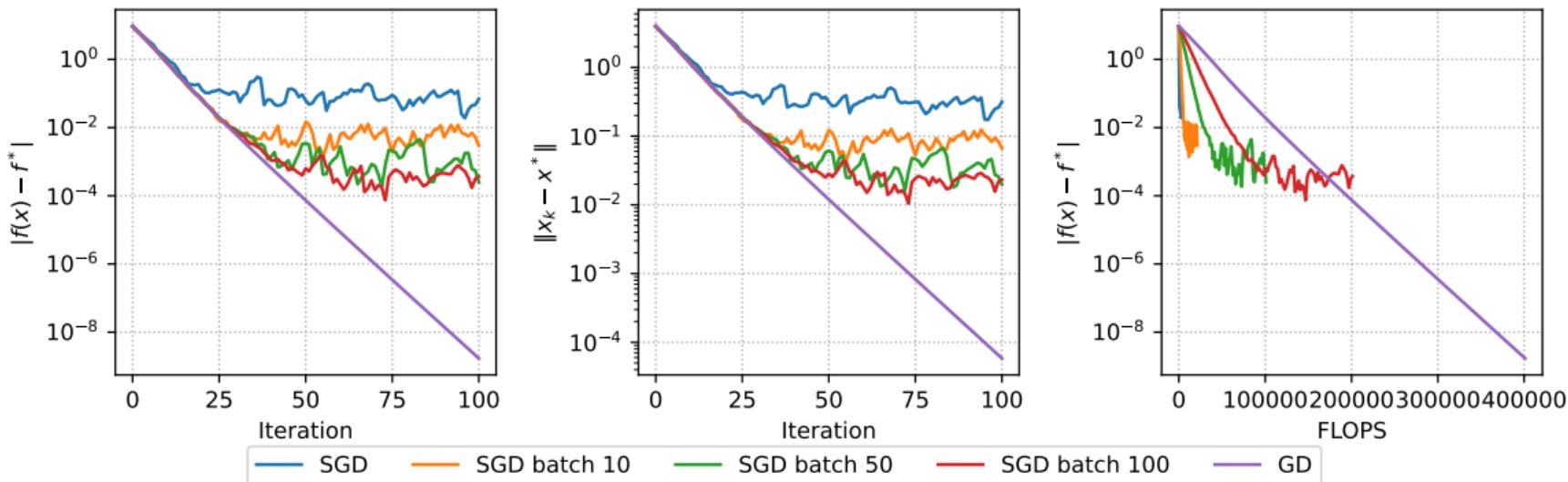
- Пусть  $f$  -  $L$ -гладкая  $\mu$ -сильно выпуклая функция, а дисперсия стохастического градиента конечна ( $\mathbb{E}[\|\nabla f_i(x_k)\|^2] \leq \sigma^2$ ). Тогда стохастический градиентный шум с уменьшающимся шагом  $\alpha_k = \frac{2k+1}{2\mu(k+1)^2}$  будет сходиться сублинейно:

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

## Сходимость в зависимости от размера батча

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

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



**Эта задача оптимизации даже сложнее, чем кажется**

## Улучшаем SGD - адаптивные методы (Adam) <sup>12 13</sup>

- Одна из самых цитируемых научных работ в мире

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

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

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

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

## Улучшаем SGD - адаптивные методы (Adam) <sup>12 13</sup>

- Одна из самых цитируемых научных работ в мире
- В 2018-2019 годах вышли статьи, указывающие на ошибку в оригинальной статье

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

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

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

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

## Улучшаем SGD - адаптивные методы (Adam) <sup>12 13</sup>

- 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)

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

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

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

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

## Улучшаем SGD - адаптивные методы (Adam) 12 13

- 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

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

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

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

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

## Улучшаем SGD - адаптивные методы (Adam) <sup>12 13</sup>

- 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?

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

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

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

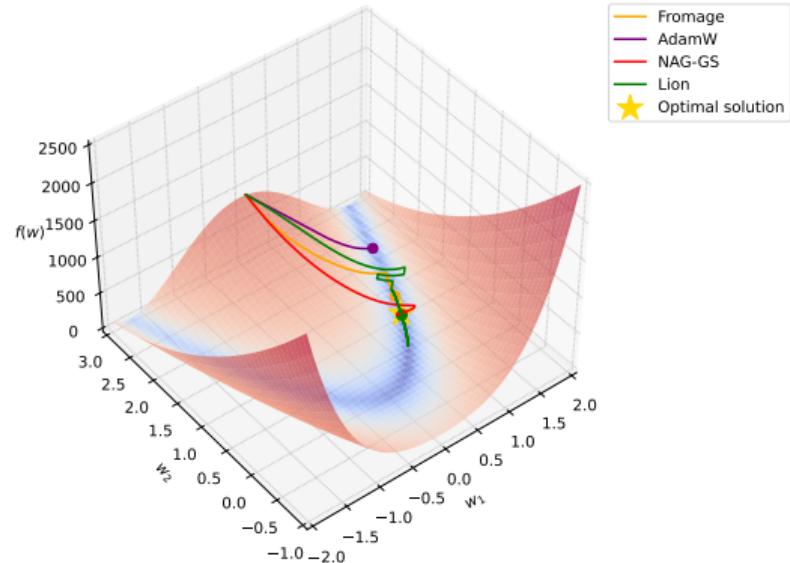
$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

<sup>12</sup>Adam: A Method for Stochastic Optimization

<sup>13</sup>On the Convergence of Adam and Beyond

- Требует хранения одного дополнительного вектора, вместо двух, как в Adam.

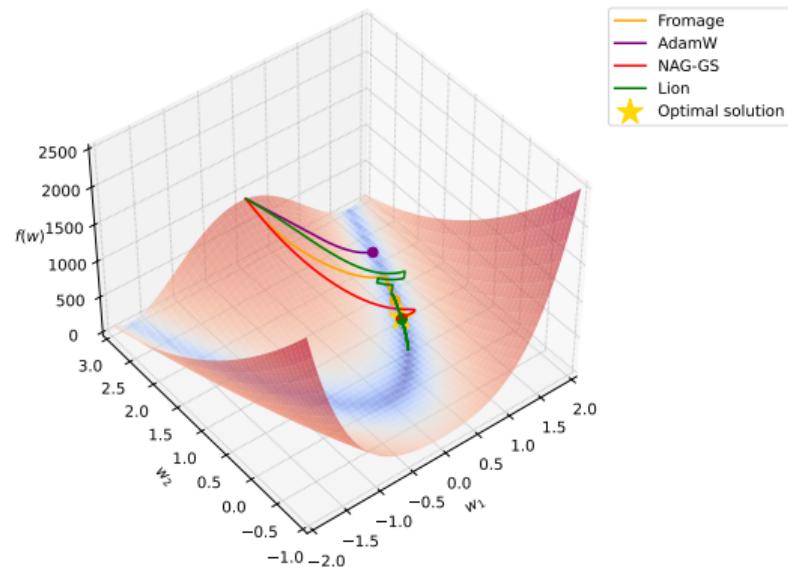
Rosenbrock Function.  
Adaptive stochastic gradient algorithms.  
Learning rate 0.003



## NAG-GS<sup>14</sup>

- Требует хранения одного дополнительного вектора, вместо двух, как в Adam.
- Качество в ряде задач сопоставимо с AdamW

Rosenbrock Function.  
Adaptive stochastic gradient algorithms.  
Learning rate 0.003



<sup>14</sup>NAG-GS: Semi-Implicit, Accelerated and Robust Stochastic Optimizer

## 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

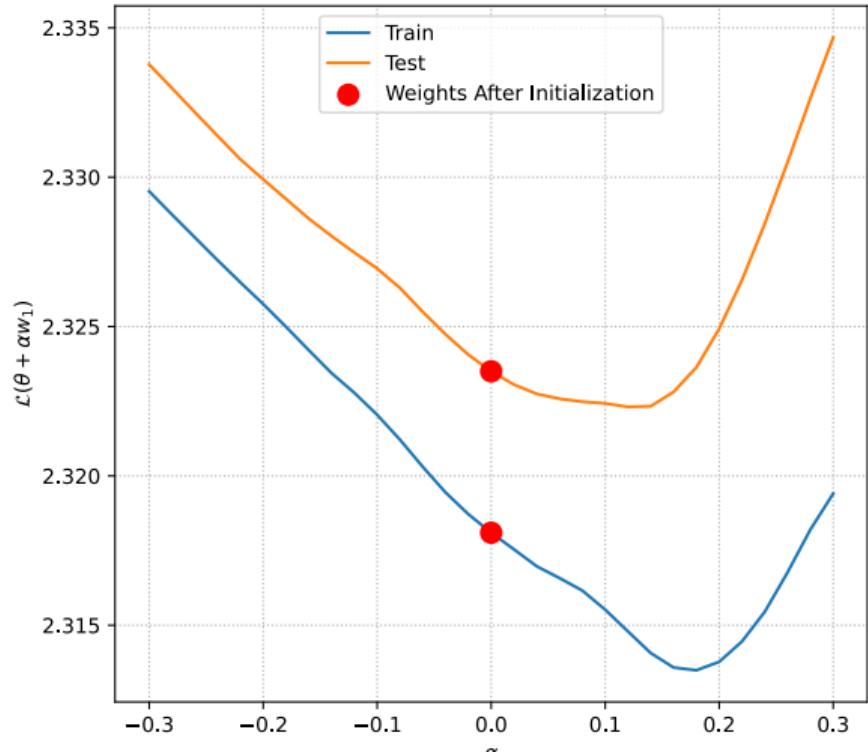
- Обозначим начальную точку как  $w_0$ , представляющую собой веса нейронной сети при инициализации. Веса, полученные после обучения, обозначим как  $\hat{w}$ .
- Генерируем случайный вектор такой же размерности и нормы  $w_1 \in \mathbb{R}^p$ , затем вычисляем значение функции потерь вдоль этого вектора:

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

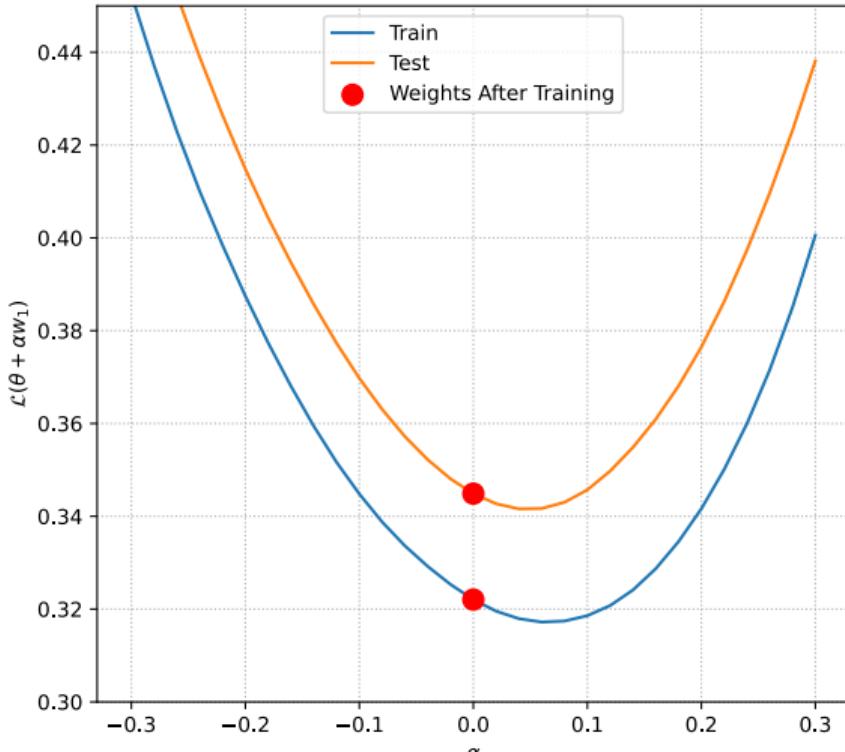
# Проекция функции потерь нейронной сети на прямую

No Dropout

Loss surface, Line projection around the starting point



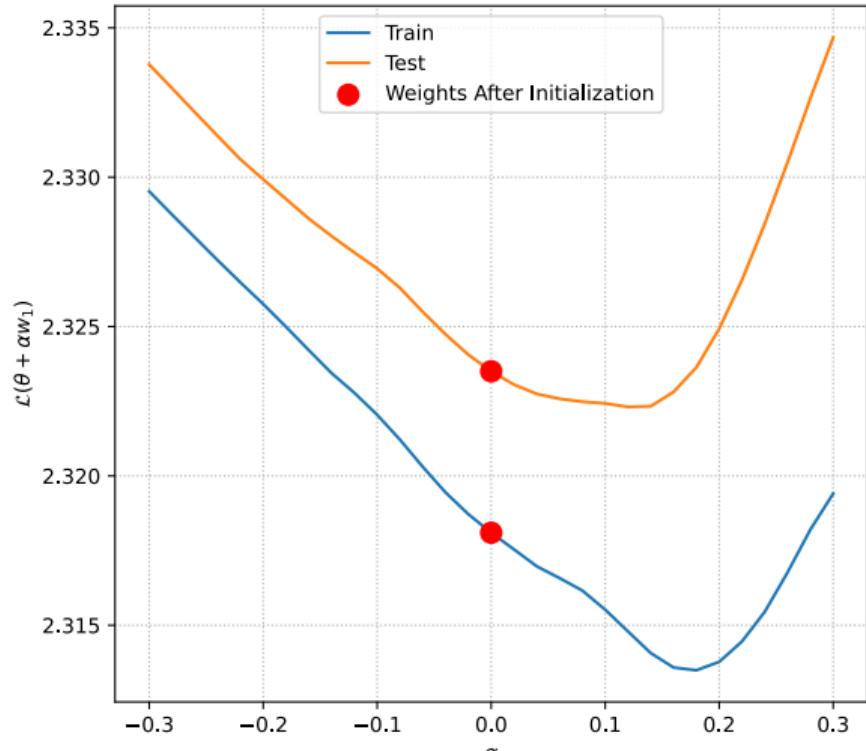
Loss surface, Line projection around the final point



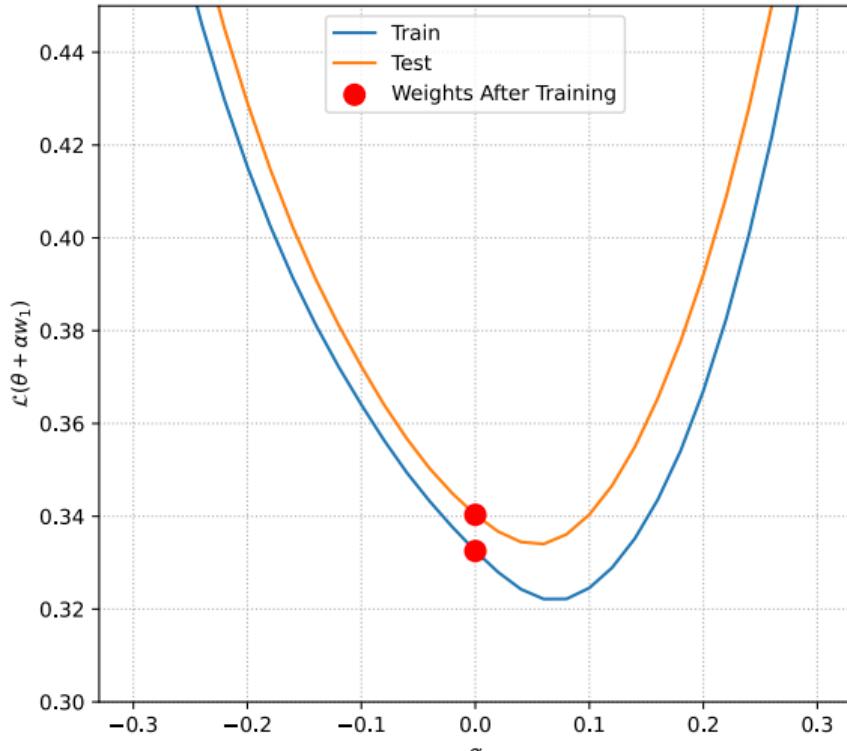
# Проекция функции потерь нейронной сети на прямую

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.

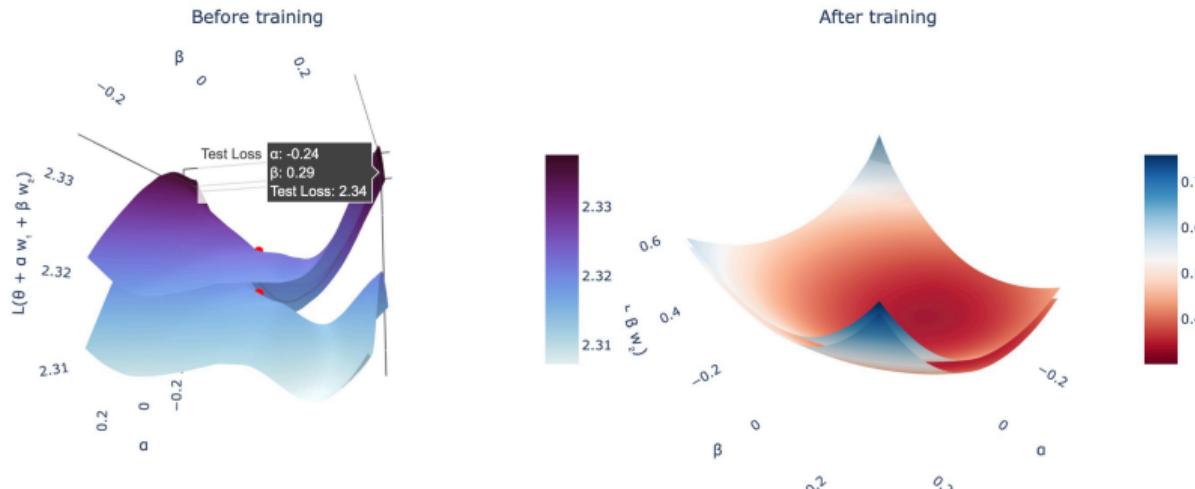


## Проекция функции потерь нейронной сети на плоскость

- 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>15</sup>

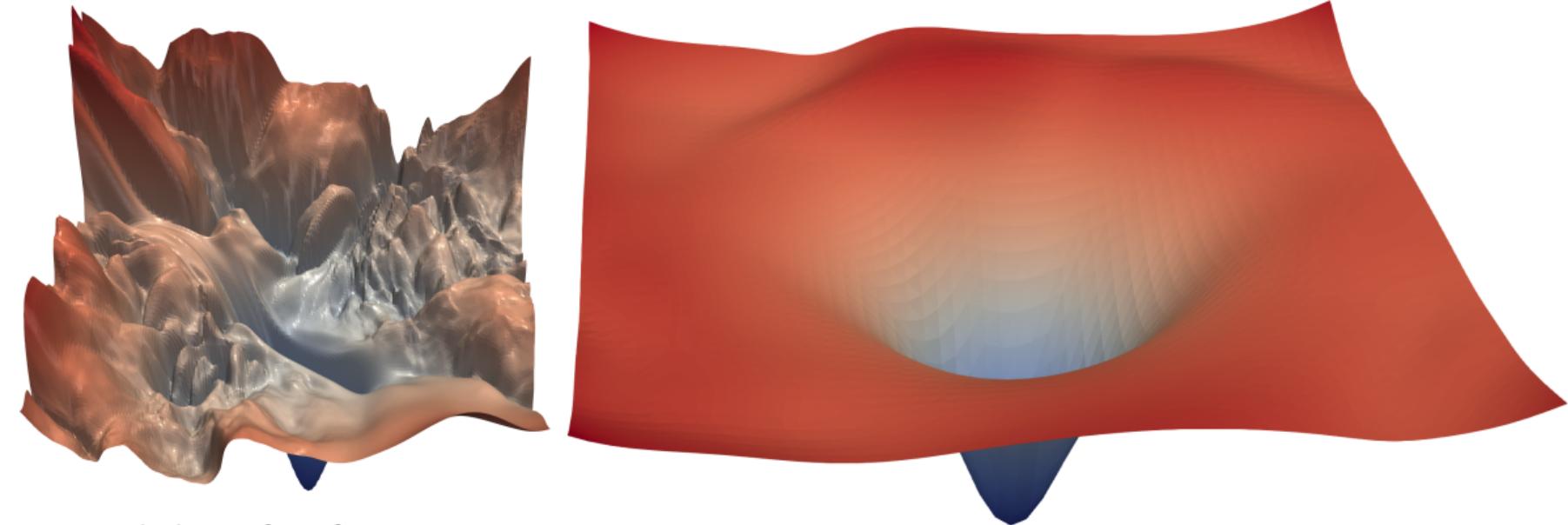


Рис. 10: The loss surface of ResNet-56  
without skip connections

Рис. 11: The loss surface of ResNet-56 with skip connections

<sup>15</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>16</sup>

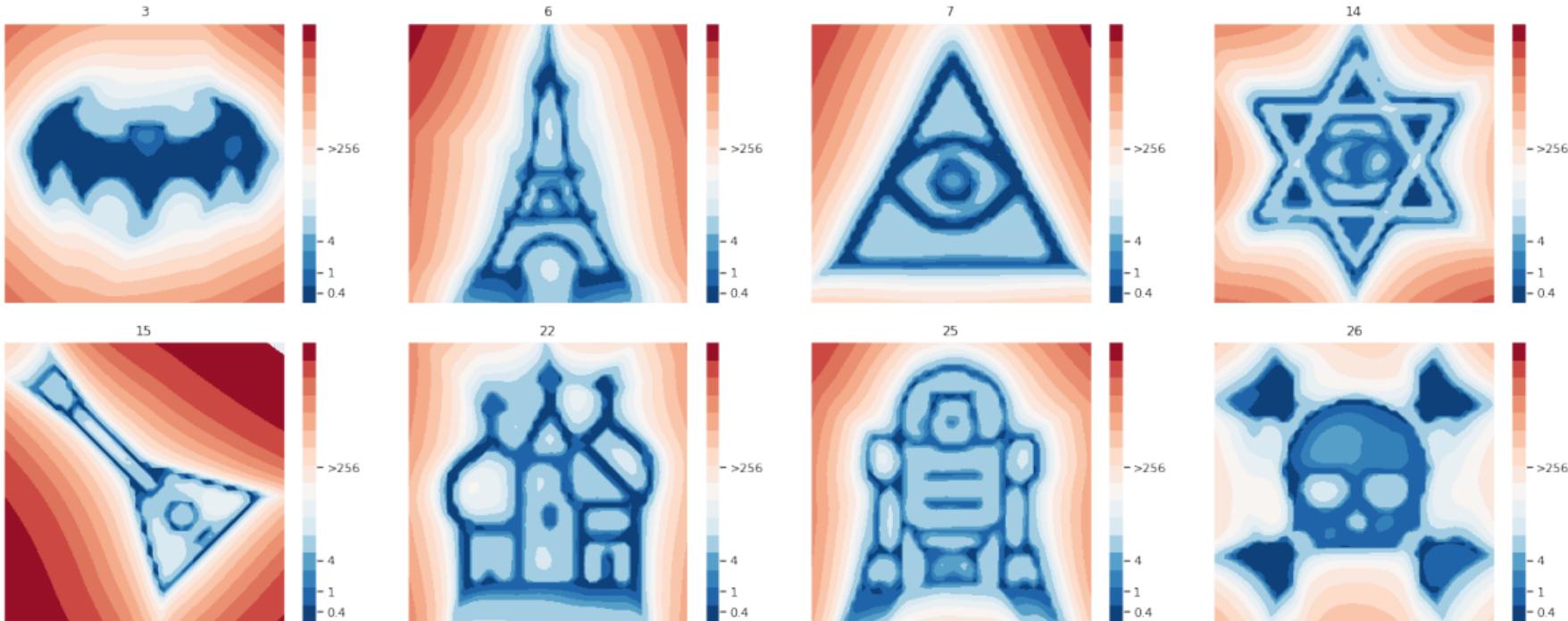


Рис. 12: Examples of a loss landscape of a typical CNN model on FashionMNIST and CIFAR10 datasets found with MPO. Loss values are color-coded according to a logarithmic scale

<sup>16</sup>Loss Landscape Sightseeing with Multi-Point Optimization, Ivan Skorokhodov, Mikhail Burtsev

## Ширина локальных минимумов

Узкие и широкие локальные минимумы



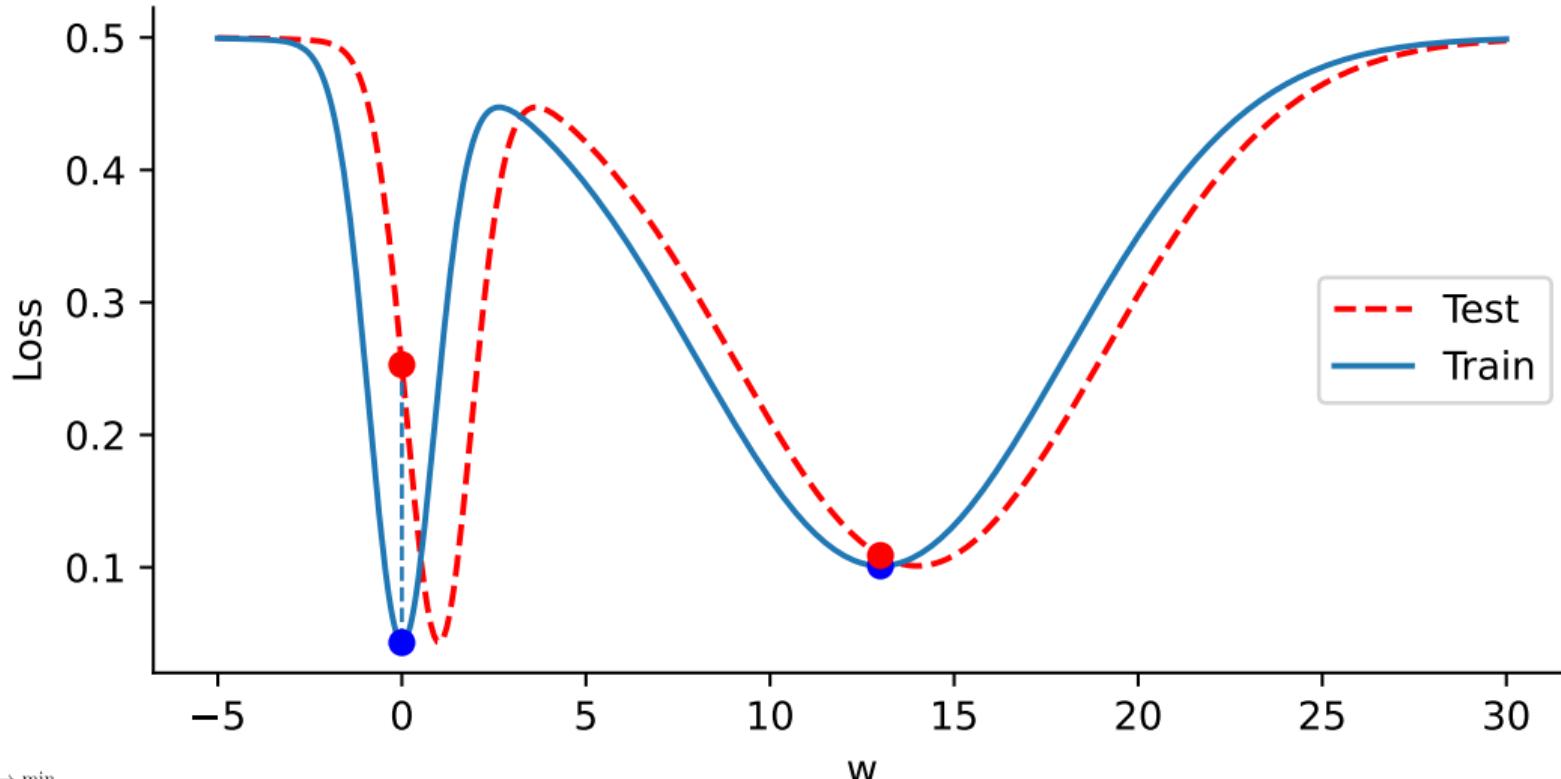
## Ширина локальных минимумов

### Узкие и широкие локальные минимумы



## Ширина локальных минимумов

### Узкие и широкие локальные минимумы



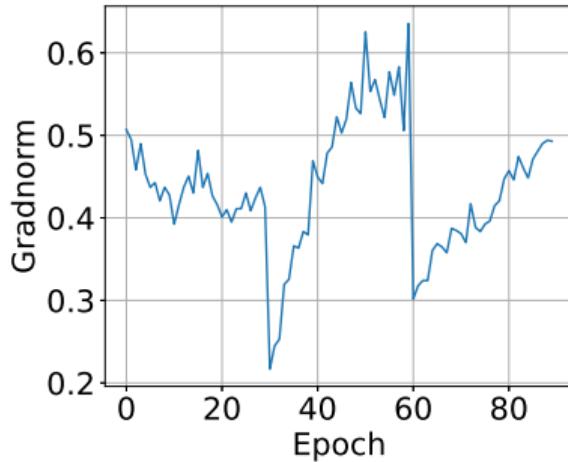
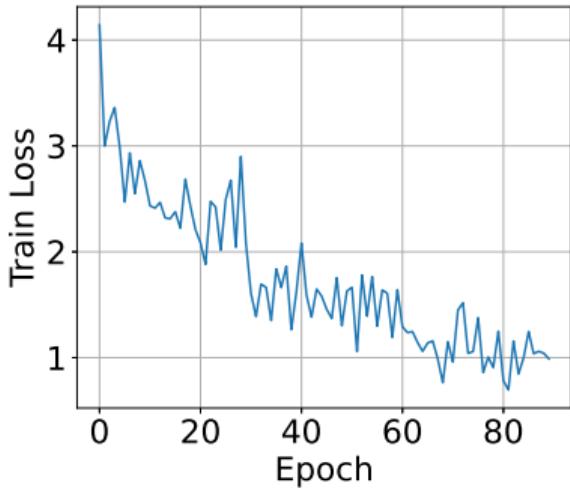
Градиентный спуск с маленьким шагом  
сходится в узкий локальный минимум



Градиентный спуск с большим шагом  
избегает узкого локального минимума



## Модели не сходятся к стационарным точкам, но это не страшно<sup>17</sup>



<sup>17</sup>NN Weights Do Not Converge to Stationary Points

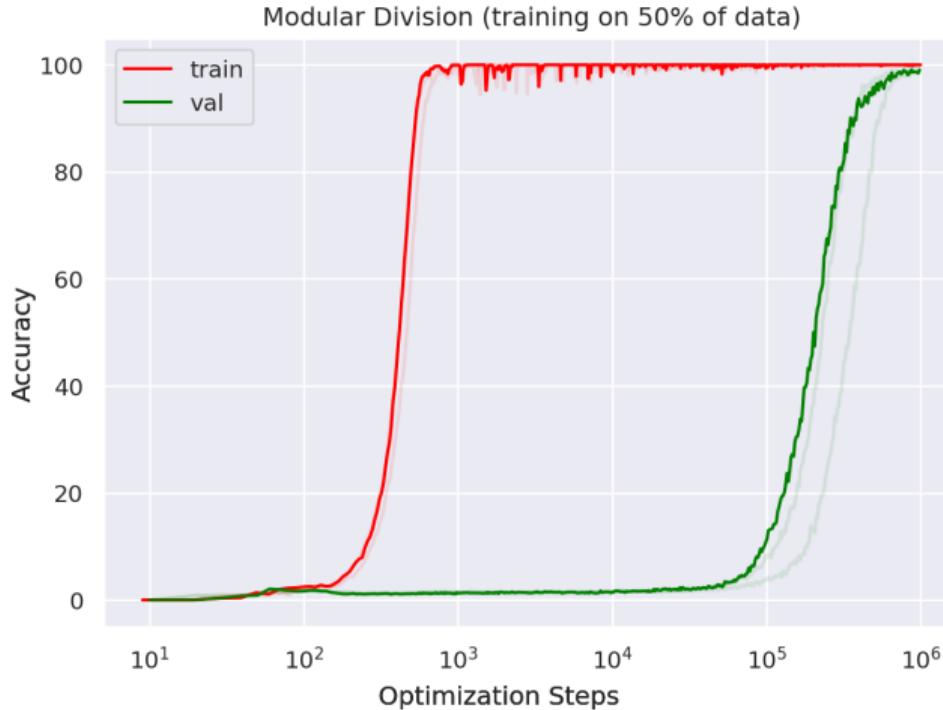
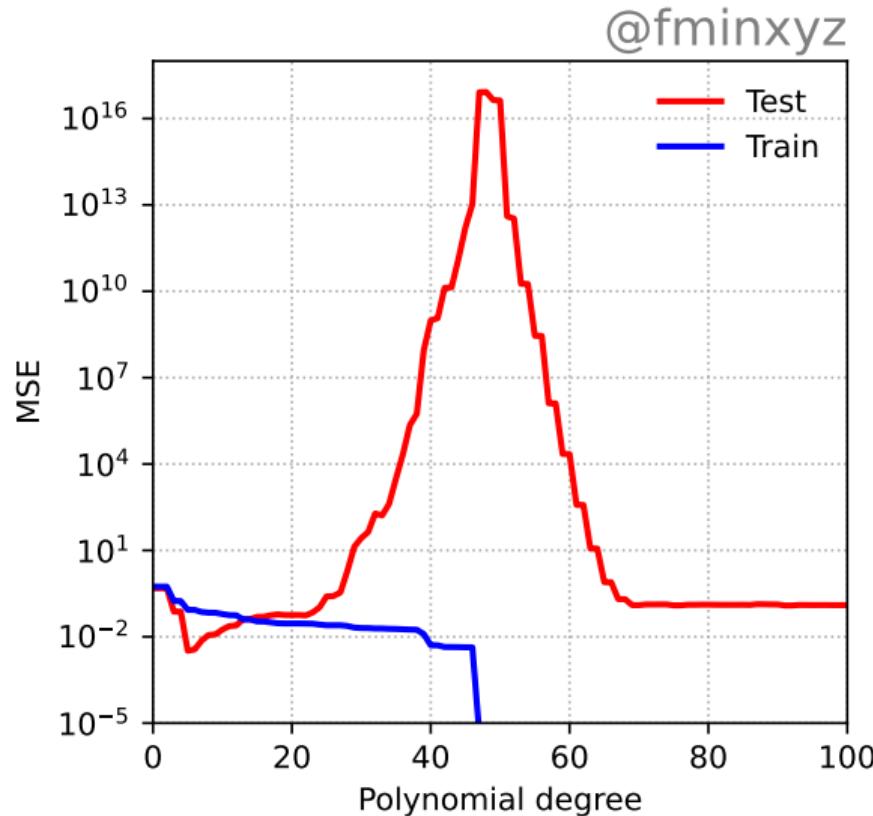


Рис. 13: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments (~ half an hour) is available here

<sup>18</sup>Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, Vedant Misra

## Double Descent<sup>19</sup>



<sup>19</sup>Reconciling modern machine learning practice and the bias-variance trade-off, Mikhail Belkin, Daniel Hsu, Siyuan Ma, Soumik Mandal  
 $f \rightarrow \min_{x,y,z}$  Эта задача оптимизации даже сложнее, чем кажется

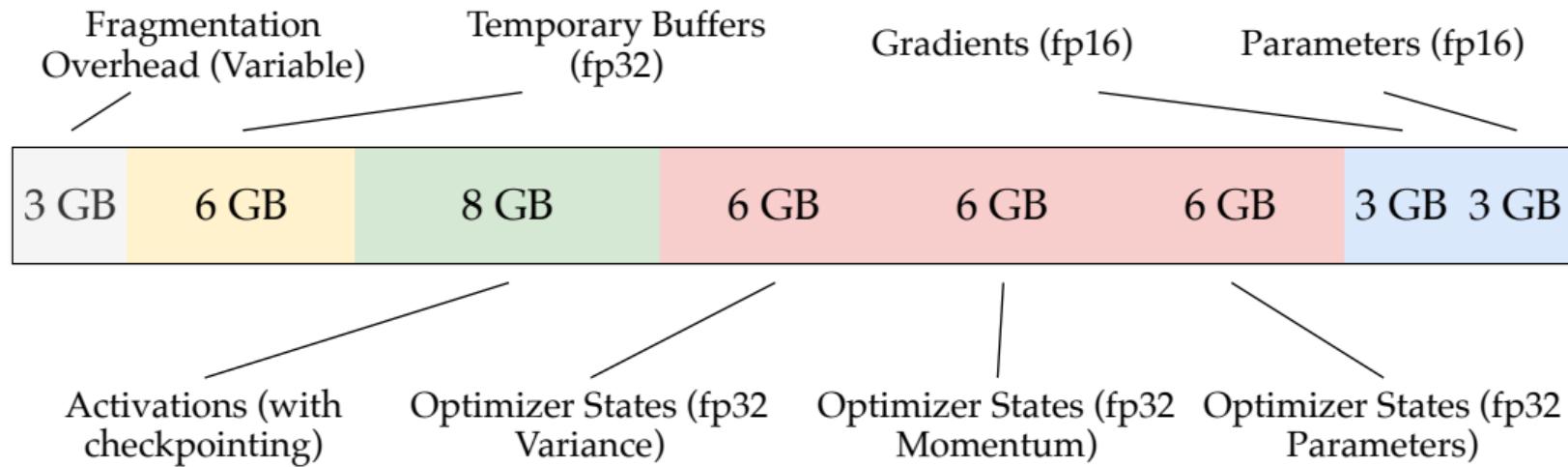
## Обучение больших моделей

## Потребление памяти при обучении GPT-2<sup>20</sup>



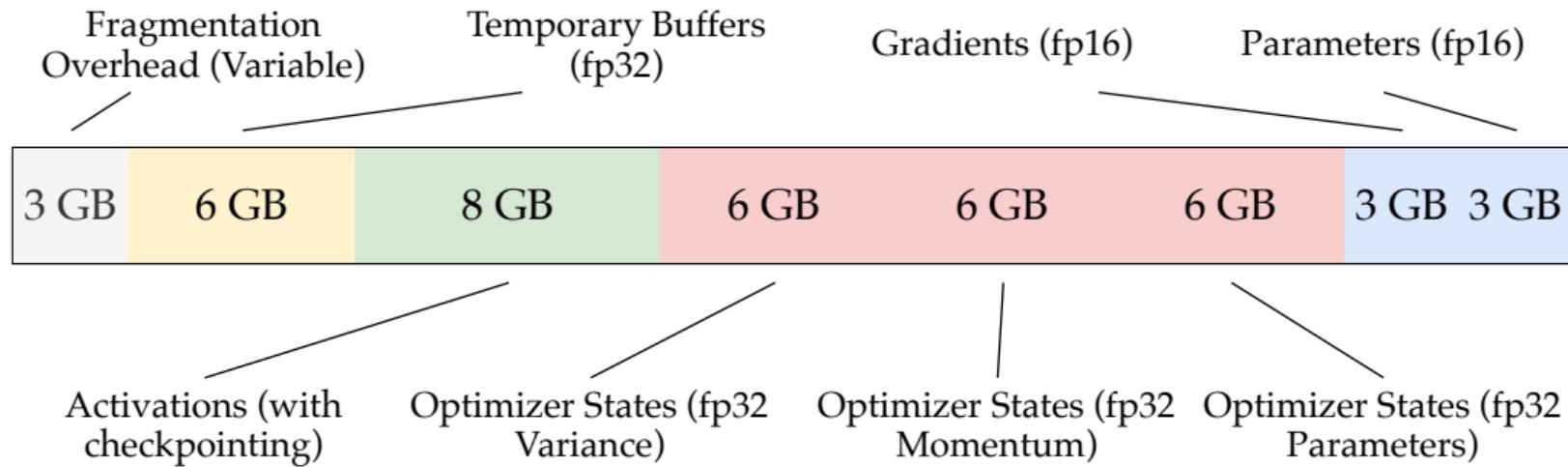
- Размер модели 1.5 В. Веса модели в fp16 занимают всего 3 GB, однако, для наивного обучения не хватит GPU даже на 32 GB

## Потребление памяти при обучении GPT-2<sup>20</sup>



- Размер модели 1.5 В. Веса модели в fp16 занимают всего 3 GB, однако, для наивного обучения не хватит GPU даже на 32 GB
- Для использования Adam в режиме mixed precision необходимо хранить 3 (!) копии модели в fp32.

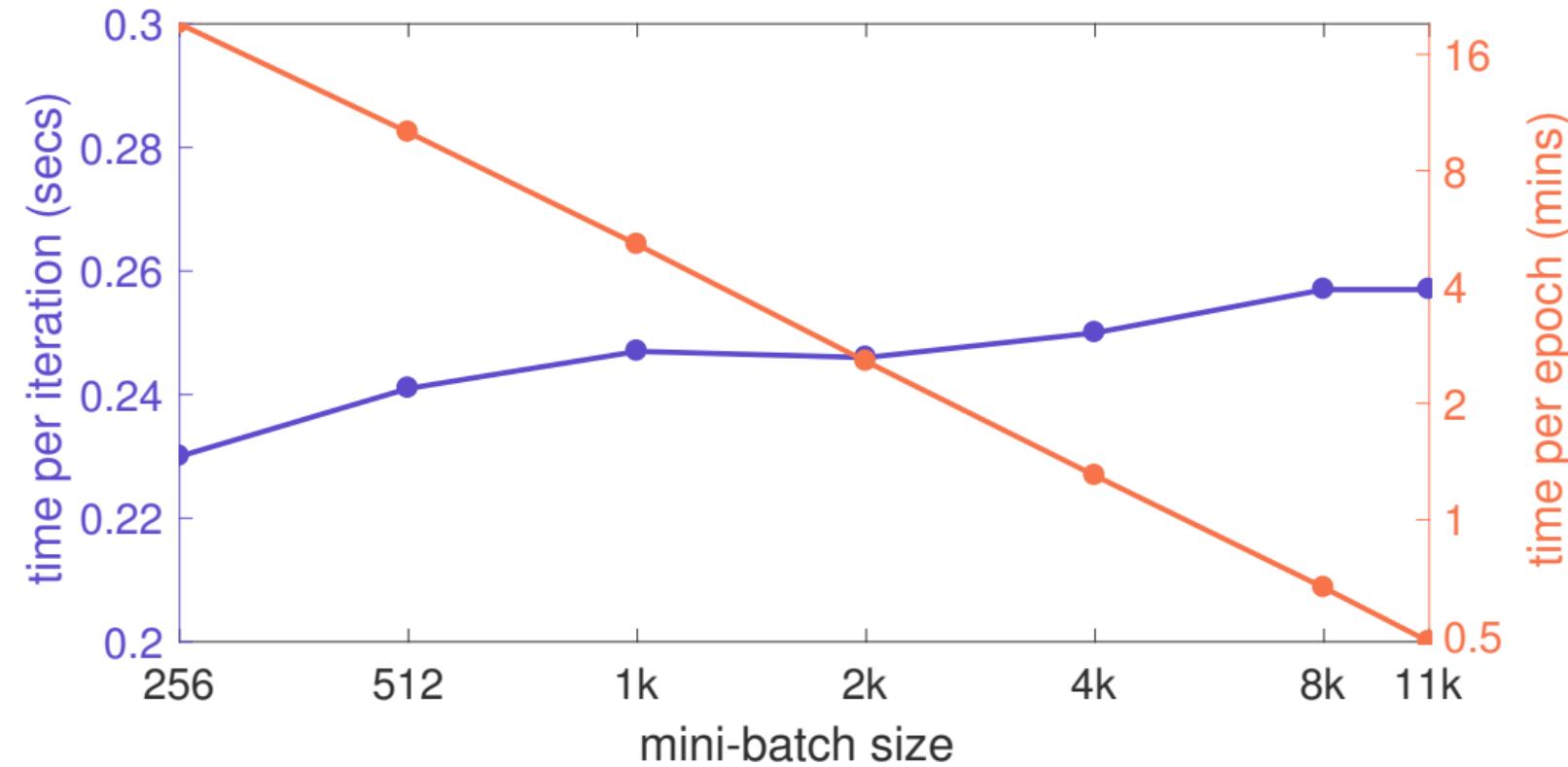
## Потребление памяти при обучении GPT-2<sup>20</sup>



- Размер модели 1.5 В. Веса модели в fp16 занимают всего 3 GB, однако, для наивного обучения не хватит GPU даже на 32 GB
- Для использования Adam в режиме mixed precision необходимо хранить 3 (!) копии модели в fp32.
- Активации в наивном режиме могут занимать гораздо больше памяти: для длины последовательности 1K и размера батча 32 нужно 60 GB для хранения всех промежуточных активаций. Чекпоинтинг активаций позволяет сократить потребление до 8 GB за счёт их перевычисления (33% computational overhead)

<sup>20</sup>ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

## Large batch training <sup>21</sup>

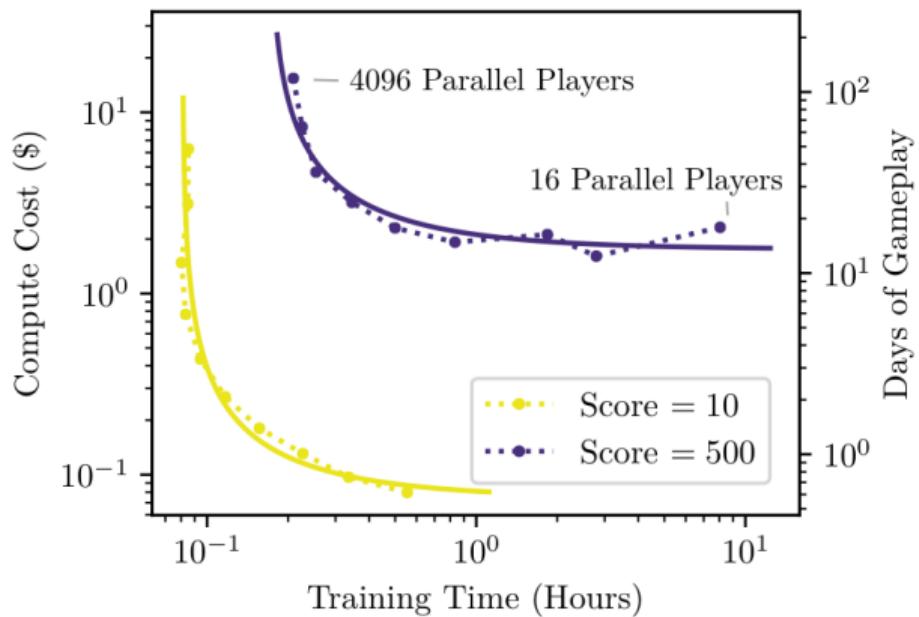


<sup>21</sup>Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

## Large batch training<sup>22</sup>

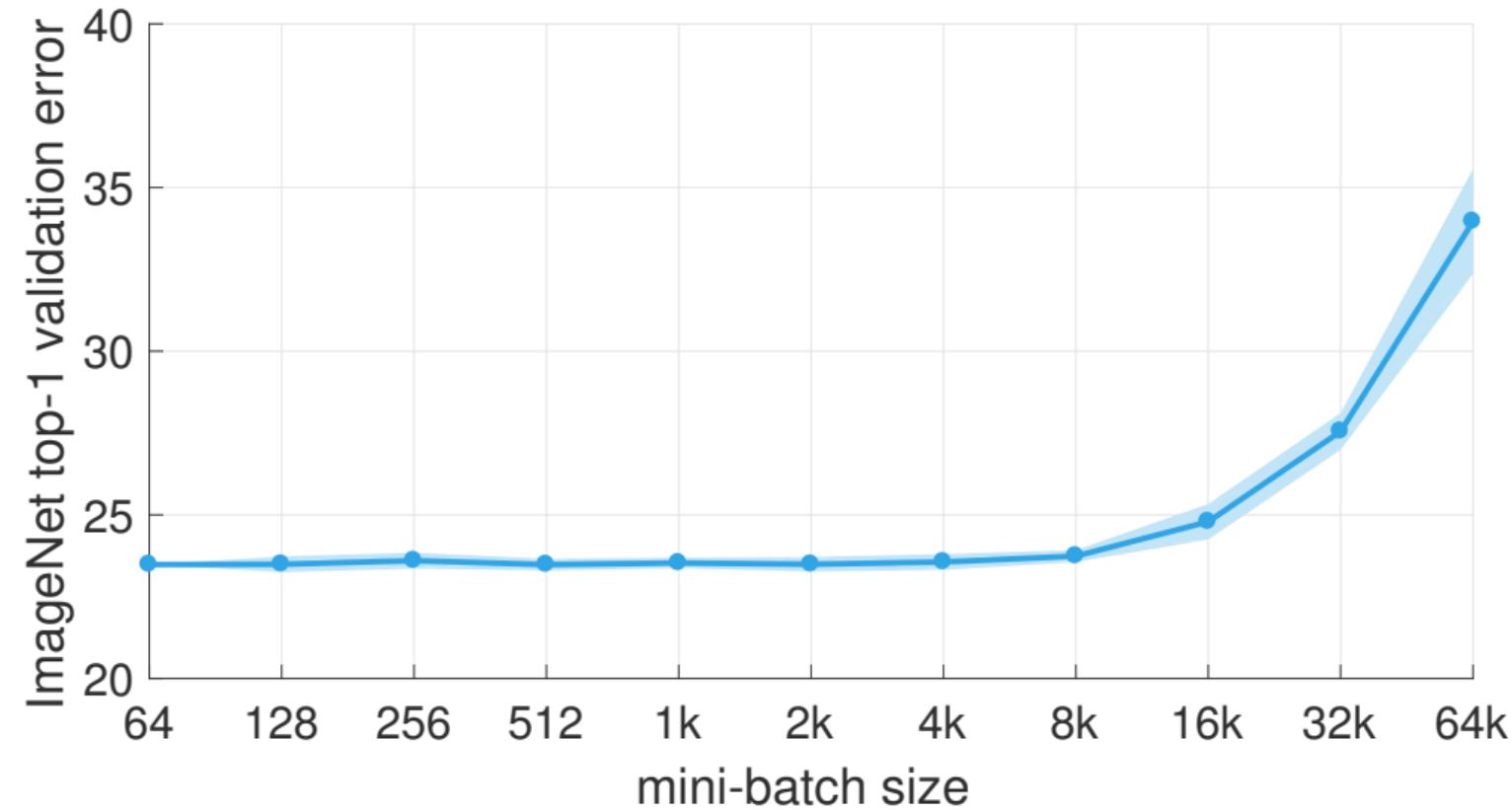


Atari Breakout - Pareto Fronts



<sup>22</sup>An Empirical Model of Large-Batch Training

## Large batch training<sup>23</sup>



<sup>23</sup>Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

## Linear and square root scaling rules

When training with large batches, the learning rate must be adjusted to maintain convergence speed and stability. The **linear scaling rule**<sup>24</sup> suggests multiplying the learning rate by the same factor as the increase in batch size:

$$\alpha_{\text{new}} = \alpha_{\text{base}} \cdot \frac{\text{Batch Size}_{\text{new}}}{\text{Batch Size}_{\text{base}}}$$

The **square root scaling rule**<sup>25</sup> proposes scaling the learning rate with the square root of the batch size increase:

$$\alpha_{\text{new}} = \alpha_{\text{base}} \cdot \sqrt{\frac{\text{Batch Size}_{\text{new}}}{\text{Batch Size}_{\text{base}}}}$$

Authors claimed, that it suits for adaptive optimizers like Adam, RMSProp and etc. while linear scaling rule serves well for SGD.

<sup>24</sup>Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

<sup>25</sup>Learning Rates as a Function of Batch Size: A Random Matrix Theory Approach to Neural Network Training

## Gradual warmup<sup>26</sup>

Gradual warmup helps to avoid instability when starting with large learning rates by slowly increasing the learning rate from a small value to the target value over a few epochs. This is defined as:

$$\alpha_t = \alpha_{\max} \cdot \frac{t}{T_w}$$

where  $t$  is the current iteration and  $T_w$  is the warmup duration in iterations. In the original paper, authors used first 5 epochs for gradual warmup.

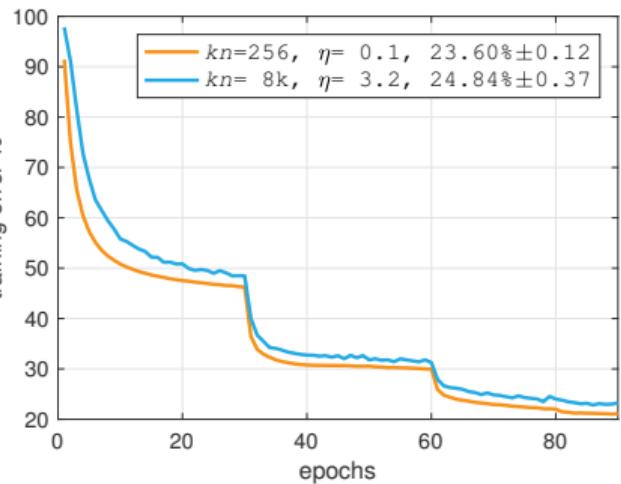


Рис. 14: no warmup

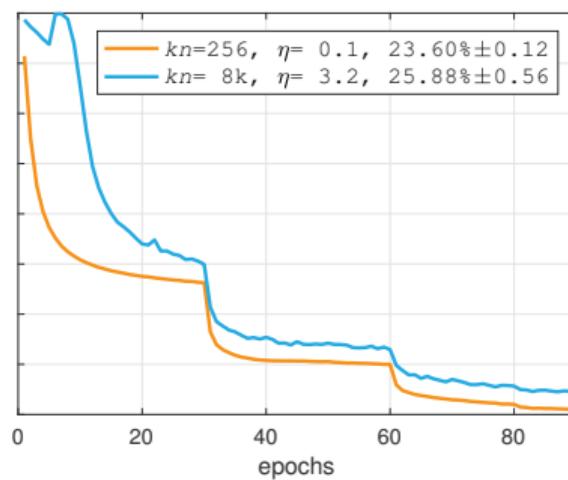


Рис. 15: constant warmup

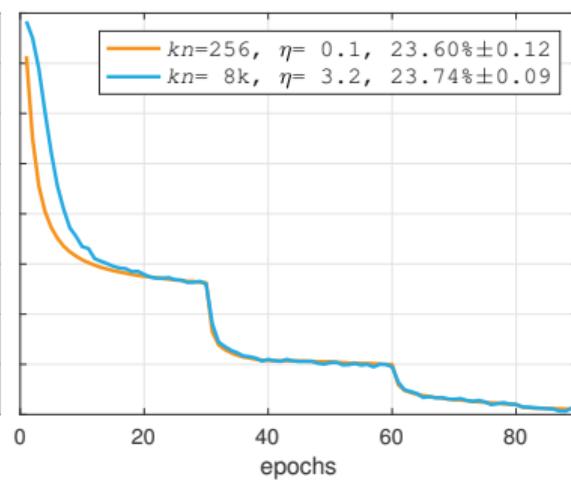


Рис. 16: gradual warmup

<sup>26</sup>Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Спасибо за внимание!

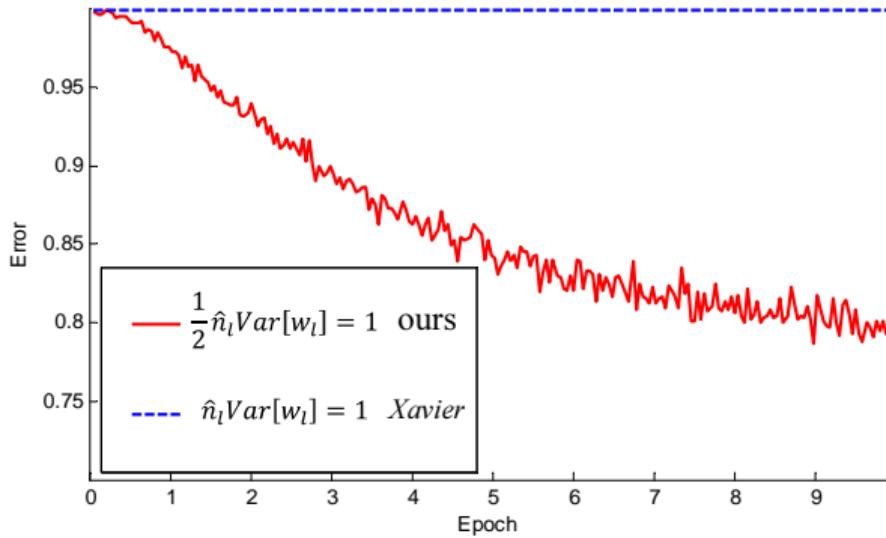


Рис. 17: Мои контакты

Запас

# Влияние инициализации весов нейронной сети на сходимость методов

27



<sup>27</sup>Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

## Методы уменьшения дисперсии? Не всё так просто на практике<sup>28</sup>

<sup>28</sup>On the Ineffectiveness of Variance Reduced Optimization for Deep Learning