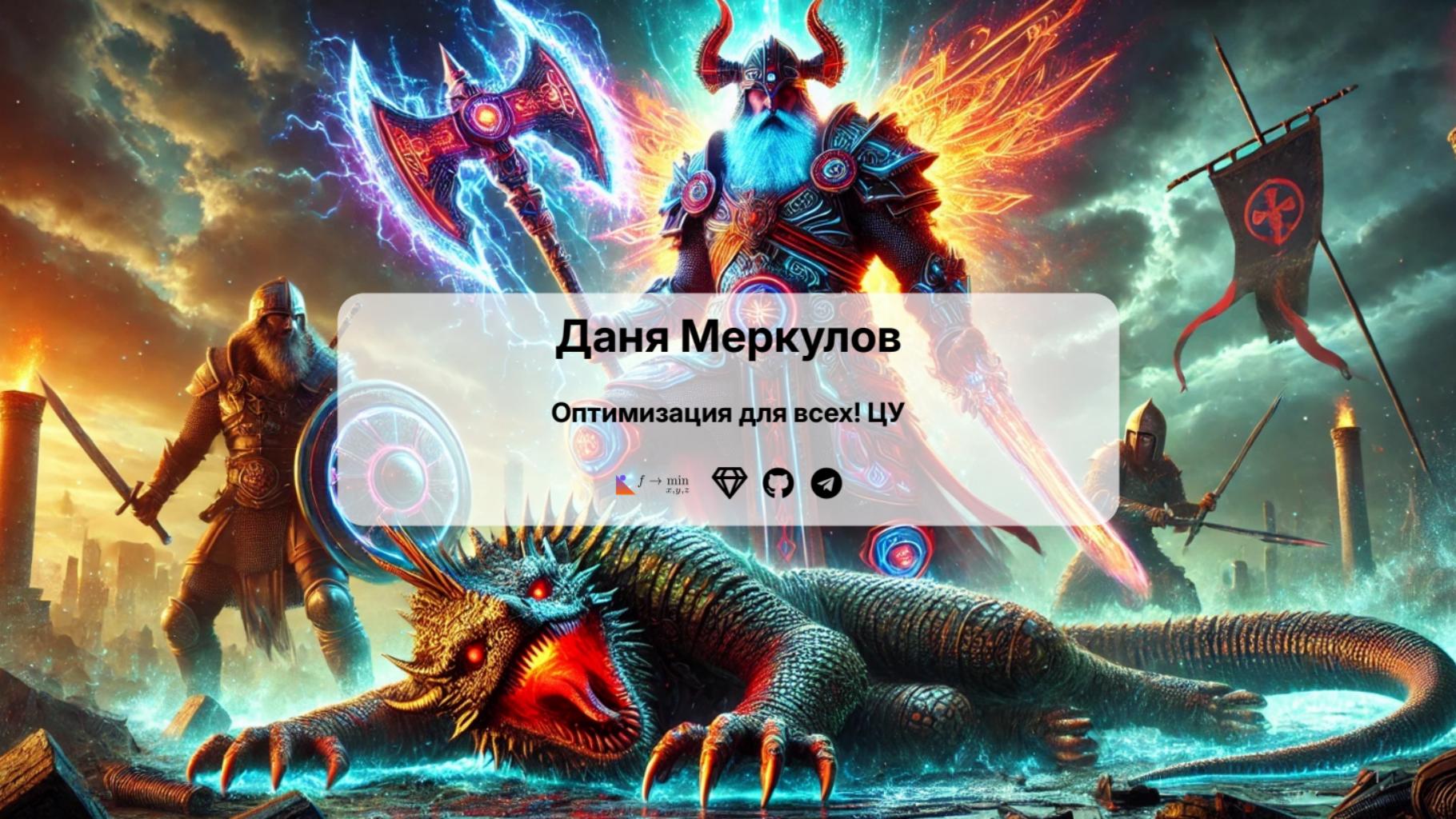


# Сюжеты из обучения нейронных сетей

МЕТОДЫ ВЫПУКЛОЙ ОПТИМИЗАЦИИ

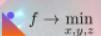
НЕДЕЛЯ 14

Даня Меркулов



# Даня Меркулов

Оптимизация для всех! ЦУ



# **Оптимизация для глубокого обучения с практической точки зрения**

# Задача оптимизации в нейросетях

Нейронная сеть — это функция, которая принимает на вход данные  $x$  и текущий набор весов (параметров)  $w$  и предсказывает некоторый вектор на выходе. Заметим, что многие нейронные сети прямого распространения (feed-forward) могут быть представлены в виде последовательности линейных преобразований, за которыми следует некоторая нелинейная функция (например,  $\text{ReLU}(x)$  или сигмоида):

# Задача оптимизации в нейросетях

Нейронная сеть — это функция, которая принимает на вход данные  $x$  и текущий набор весов (параметров)  $\mathbf{w}$  и предсказывает некоторый вектор на выходе. Заметим, что многие нейронные сети прямого распространения (feed-forward) могут быть представлены в виде последовательности линейных преобразований, за которыми следует некоторая нелинейная функция (например,  $\text{ReLU}(x)$  или сигмоида):

$$\mathcal{NN}(\mathbf{w}, x) = \sigma_L \circ w_L \circ \dots \circ \sigma_1 \circ w_1 \circ x, \quad \mathbf{w} = (W_1, b_1, \dots, W_L, b_L),$$

# Задача оптимизации в нейросетях

Нейронная сеть — это функция, которая принимает на вход данные  $x$  и текущий набор весов (параметров)  $\mathbf{w}$  и предсказывает некоторый вектор на выходе. Заметим, что многие нейронные сети прямого распространения (feed-forward) могут быть представлены в виде последовательности линейных преобразований, за которыми следует некоторая нелинейная функция (например,  $\text{ReLU}(x)$  или сигмоида):

$$\mathcal{NN}(\mathbf{w}, x) = \sigma_L \circ w_L \circ \dots \circ \sigma_1 \circ w_1 \circ x, \quad \mathbf{w} = (W_1, b_1, \dots, W_L, b_L),$$

где  $L$  — число слоёв,  $\sigma_i$  — нелинейная функция активации, а  $w_i = W_i x + b_i$  — линейный слой.

# Задача оптимизации в нейросетях

Нейронная сеть — это функция, которая принимает на вход данные  $x$  и текущий набор весов (параметров)  $\mathbf{w}$  и предсказывает некоторый вектор на выходе. Заметим, что многие нейронные сети прямого распространения (feed-forward) могут быть представлены в виде последовательности линейных преобразований, за которыми следует некоторая нелинейная функция (например,  $\text{ReLU}(x)$  или сигмоида):

$$\mathcal{NN}(\mathbf{w}, x) = \sigma_L \circ w_L \circ \dots \circ \sigma_1 \circ w_1 \circ x, \quad \mathbf{w} = (W_1, b_1, \dots, W_L, b_L),$$

где  $L$  — число слоёв,  $\sigma_i$  — нелинейная функция активации, а  $w_i = W_i x + b_i$  — линейный слой.

Обычно наша цель — найти такие параметры  $\mathbf{w}$ , которые позволяют решить некоторую задачу (например, добиться того, чтобы  $\mathcal{NN}(\mathbf{w}, x_i) \sim y_i$  для обучающих данных  $(x_i, y_i)$ ). Для этого мы решаем задачу оптимизации:

# Задача оптимизации в нейросетях

Нейронная сеть — это функция, которая принимает на вход данные  $x$  и текущий набор весов (параметров)  $\mathbf{w}$  и предсказывает некоторый вектор на выходе. Заметим, что многие нейронные сети прямого распространения (feed-forward) могут быть представлены в виде последовательности линейных преобразований, за которыми следует некоторая нелинейная функция (например,  $\text{ReLU}(x)$  или сигмоида):

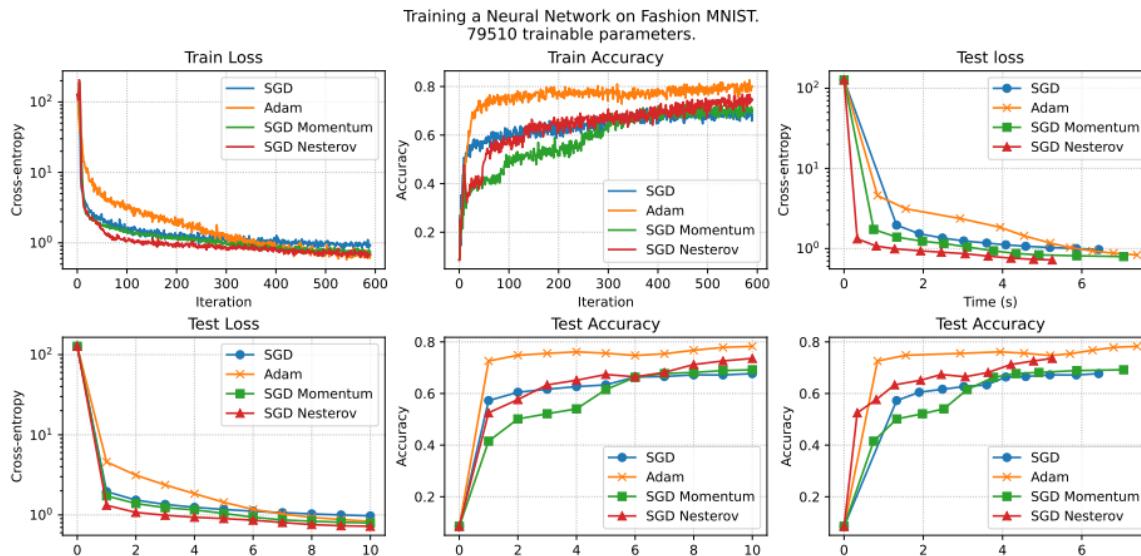
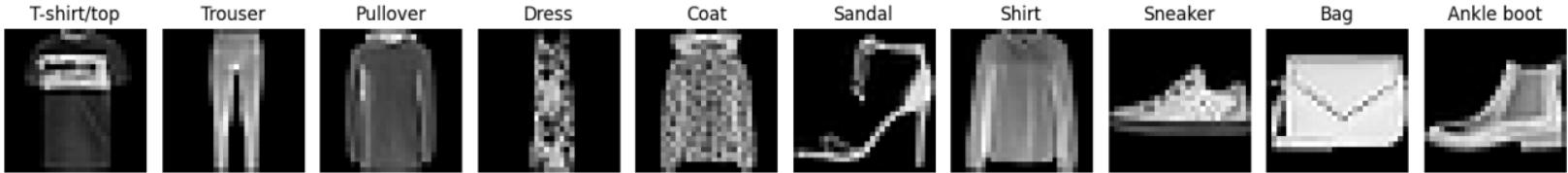
$$\mathcal{NN}(\mathbf{w}, x) = \sigma_L \circ w_L \circ \dots \circ \sigma_1 \circ w_1 \circ x, \quad \mathbf{w} = (W_1, b_1, \dots, W_L, b_L),$$

где  $L$  — число слоёв,  $\sigma_i$  — нелинейная функция активации, а  $w_i = W_i x + b_i$  — линейный слой.

Обычно наша цель — найти такие параметры  $\mathbf{w}$ , которые позволяют решить некоторую задачу (например, добиться того, чтобы  $\mathcal{NN}(\mathbf{w}, x_i) \sim y_i$  для обучающих данных  $(x_i, y_i)$ ). Для этого мы решаем задачу оптимизации:

$$L(\mathbf{w}, X, y) \rightarrow \min_{\mathbf{w}} \quad \frac{1}{N} \sum_{i=1}^N l(\mathbf{w}, x_i, y_i) \rightarrow \min_{\mathbf{w}}$$

# Простой пример: задача классификации Fashion MNIST



## Notable AI Models

### Training compute (FLOP)

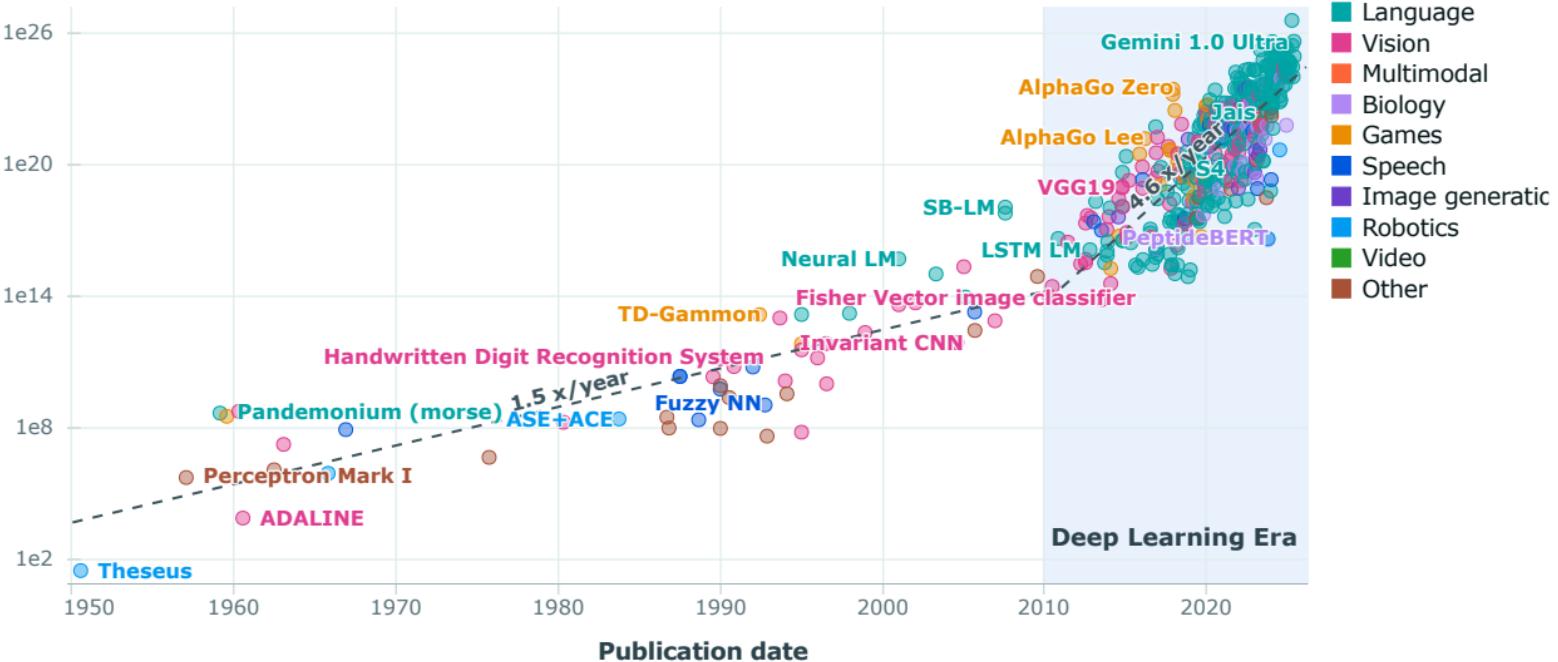


Рисунок 2. Динамика вычислений, необходимых для обучения моделей. Источник

## Notable AI Models

### Training compute (FLOP)

483 Results

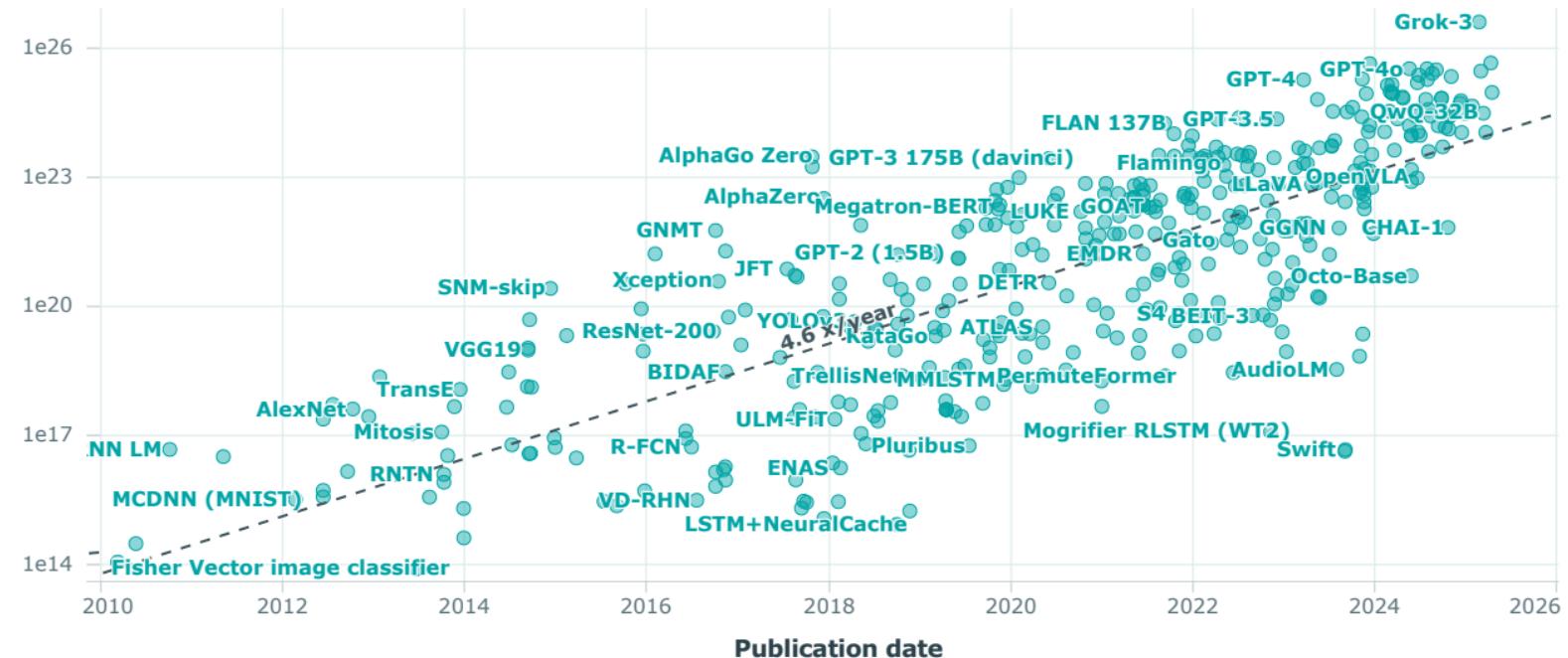


Рисунок 3. Динамика вычислений, необходимых для обучения нейросетевых моделей. Источник

## Notable AI Models

### Number of trainable parameters

636 Results

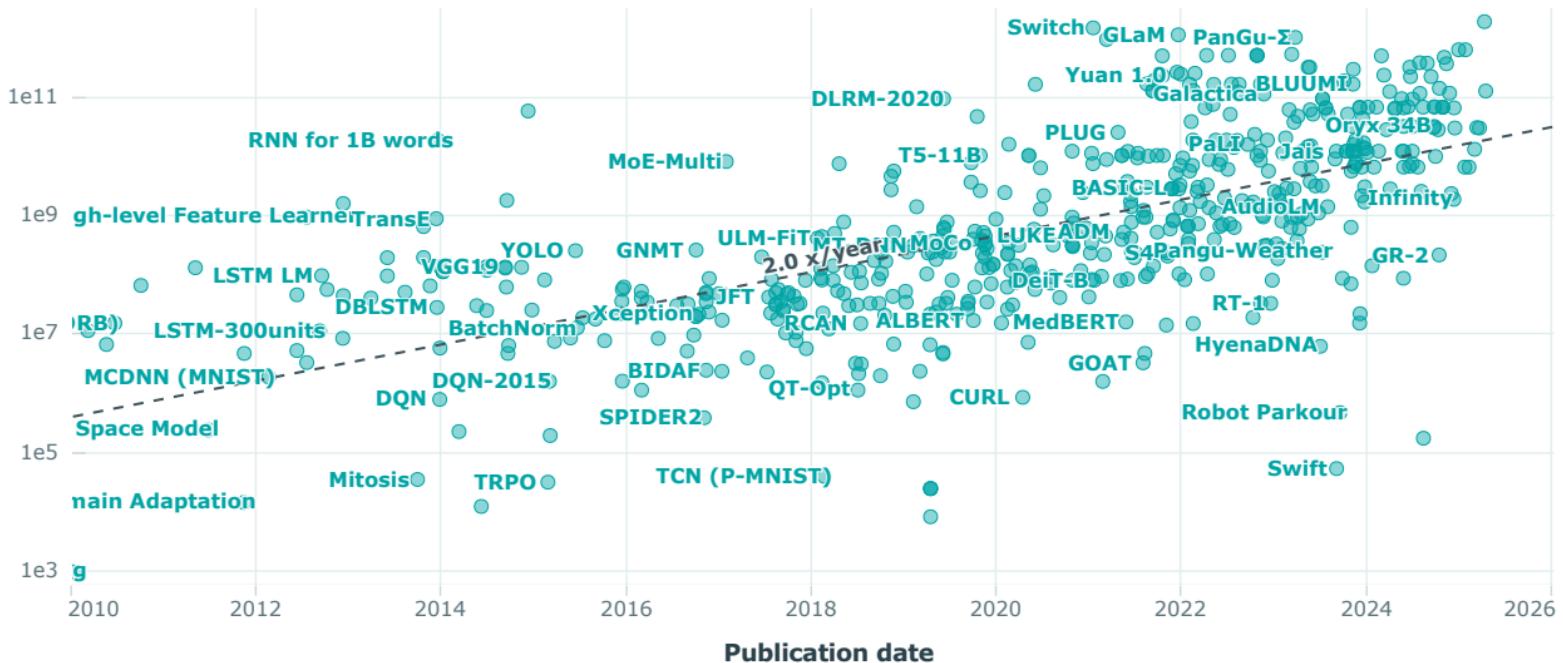


Рисунок 4. Динамика количества обучаемых параметров нейросетевых моделей. Источник

# Основные результаты сходимости SGD

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

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

# Основные результаты сходимости SGD

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

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

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

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

# Основные результаты сходимости SGD

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

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

# Основные результаты сходимости SGD

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

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

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

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

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



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

EMA:

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

Коррекция смещения:

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

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

Обновление:

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

**Заметки:**

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

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



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

EMA:

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

Коррекция смещения:

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

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

Обновление:

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

**Заметки:**

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

# Adam (Kingma and Ba, 2014) 12



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

EMA:

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

Коррекция смещения:

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

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

Обновление:

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

## Заметки:

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

# Adam (Kingma and Ba, 2014) 12



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

EMA:

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

Коррекция смещения:

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

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

Обновление:

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

## Заметки:

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

# Adam (Kingma and Ba, 2014)



12

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

EMA:

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

Коррекция смещения:

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

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

Обновление:

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

**Заметки:**

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

# Adam (Kingma and Ba, 2014)<sup>1</sup><sup>2</sup>



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

EMA:

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

Коррекция смещения:

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

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

Обновление:

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

## Заметки:

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

---

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

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

# AdamW (Loshchilov & Hutter, 2017)

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

Правило обновления:

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

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

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

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

**Заметки:**

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

# AdamW (Loshchilov & Hutter, 2017)

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

Правило обновления:

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

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

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

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

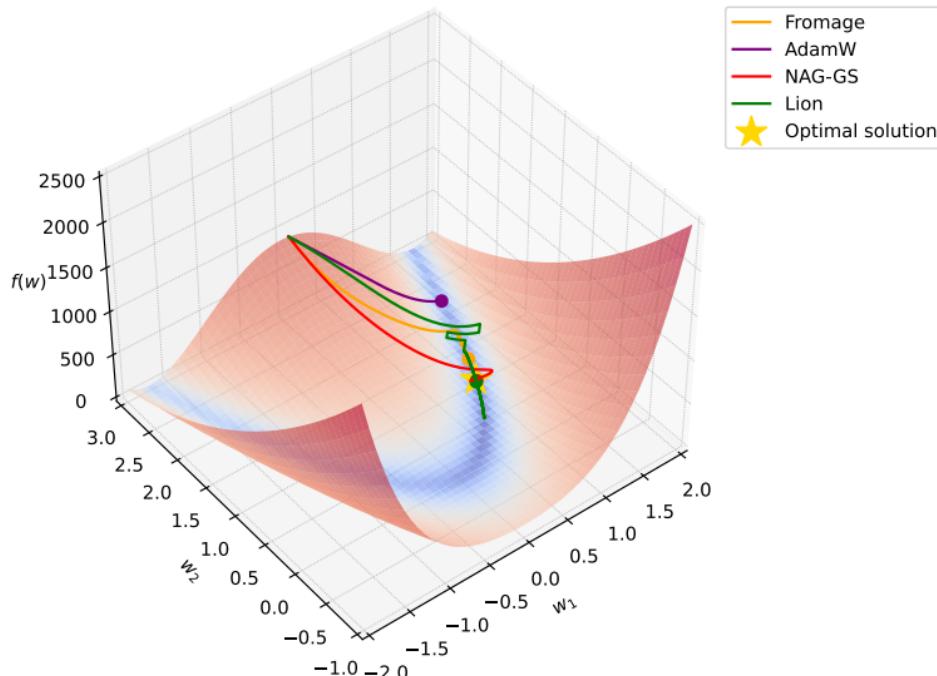
**Заметки:**

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

# Много методов



Rosenbrock Function.  
Adaptive stochastic gradient algorithms.  
Learning rate 0.003

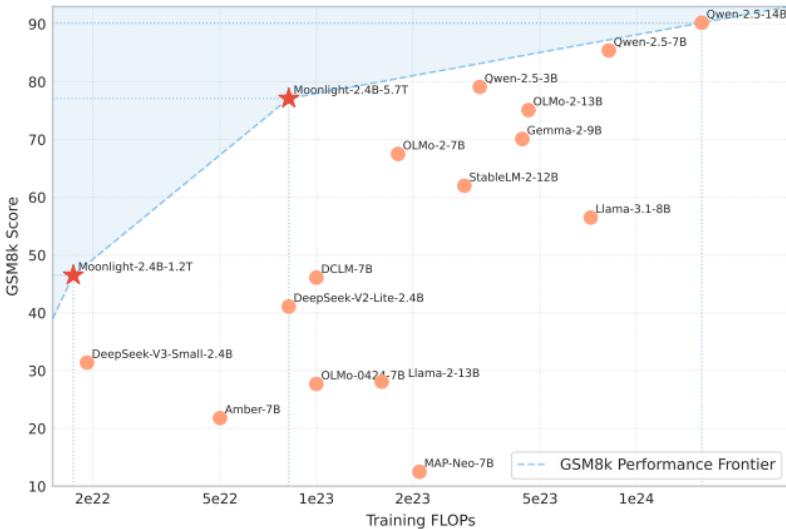
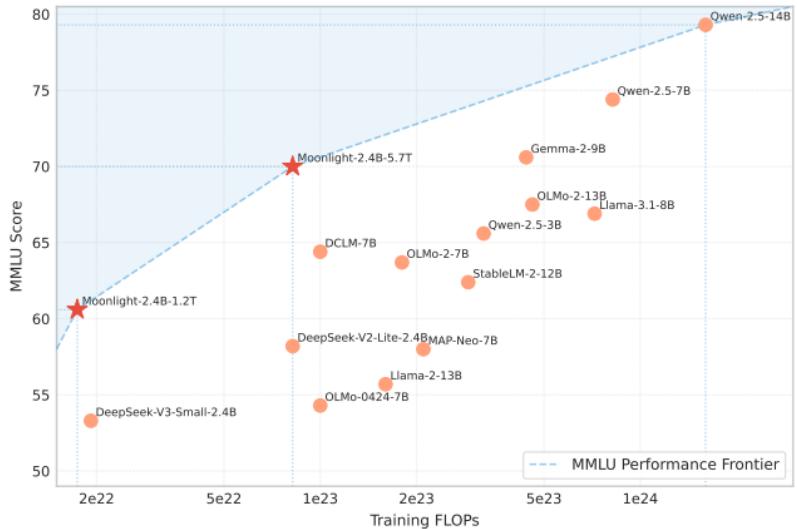




# Новый подход к оптимизации



3



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

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

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



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

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

<sup>4</sup>Презентация R. Gower

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

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

Функция потерь

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

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

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

Функция потерь

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

Линейная  
аппроксимация

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

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

Функция потерь

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

Хорошее приближение  
в окрестности  $x_k$

# Интуиция за методом Мион. Градиентный спуск



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

# Интуиция за методом Мион. Градиентный спуск



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

# Интуиция за методом Мион. Градиентный спуск



Штраф за  
дальность от  $x_k$

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

# Интуиция за методом Мион. Градиентный спуск



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

Шаг обучения /  
коэффициент регуляризации

Штраф за  
дальность от  $x_k$



# Интуиция за методом Мион. Нормированный градиентный спуск

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

# Интуиция за методом Мион. Нормированный градиентный спуск

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

# Интуиция за методом Мион. Нормированный градиентный спуск

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

Ограничение на  
длину шага



# Интуиция за методом Мион. Нормированный градиентный спуск

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

Параметр ограничения / шаг обучения

Ограничение на длину шага



# Что насчёт других норм?



Unit disk in the  $p$ -th norm

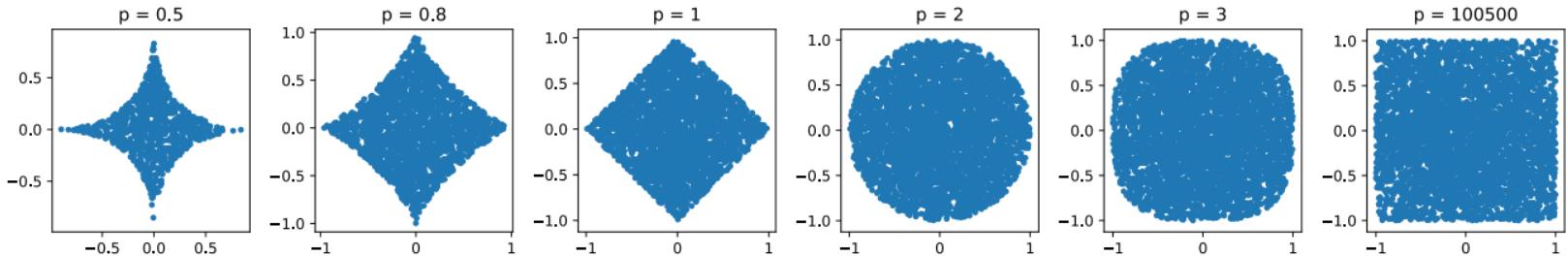


Рисунок 5. Примеры шаров в разных нормах

# Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

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

# Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

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

- Linear Minimization Oracle:

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

# Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

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

- Linear Minimization Oracle:

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

- Важное свойство, связывающее эти два понятия:

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

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



## ! Неевклидов градиентный спуск

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

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

---

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

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

## ! Неевклидов градиентный спуск

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

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

## ! Неевклидов нормированный градиентный спуск

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

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

---

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

# В нейросетях параметры — матрицы

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

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

# В нейросетях параметры — матрицы

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

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

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

# В нейросетях параметры — матрицы

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

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

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

# В нейросетях параметры — матрицы

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

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

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

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

# Неевклидов нормированный спуск для матриц

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

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

где

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

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

# Операторная норма и быстрый расчёт ( $UV^\top$ )

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

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

— редуцированное SVD градиента. Тогда

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

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

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

# Операторная норма и быстрый расчёт ( $UV^\top$ )

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

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

— редуцированное SVD градиента. Тогда

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

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

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

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

# Операторная норма и быстрый расчёт ( $UV^\top$ )

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

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

— редуцированное SVD градиента. Тогда

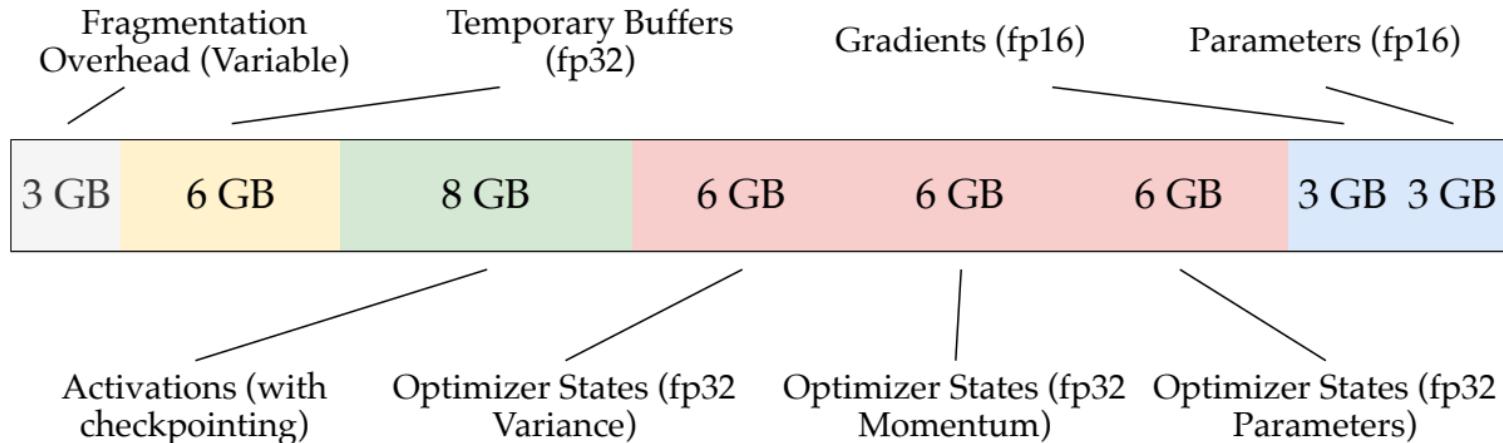
- LMO (с «max»-формулировкой) по операторной норме:

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

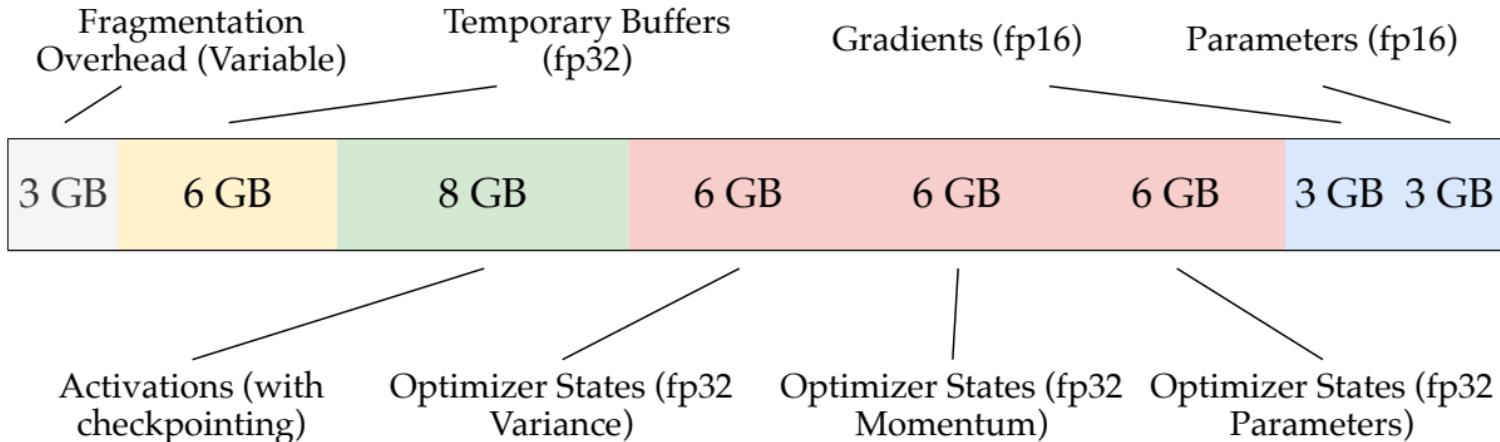
то есть оптимальное направление — **polar factor** (matrix sign) матрицы  $G_k$ .

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

# Потребление памяти при обучении GPT-2<sup>6</sup>

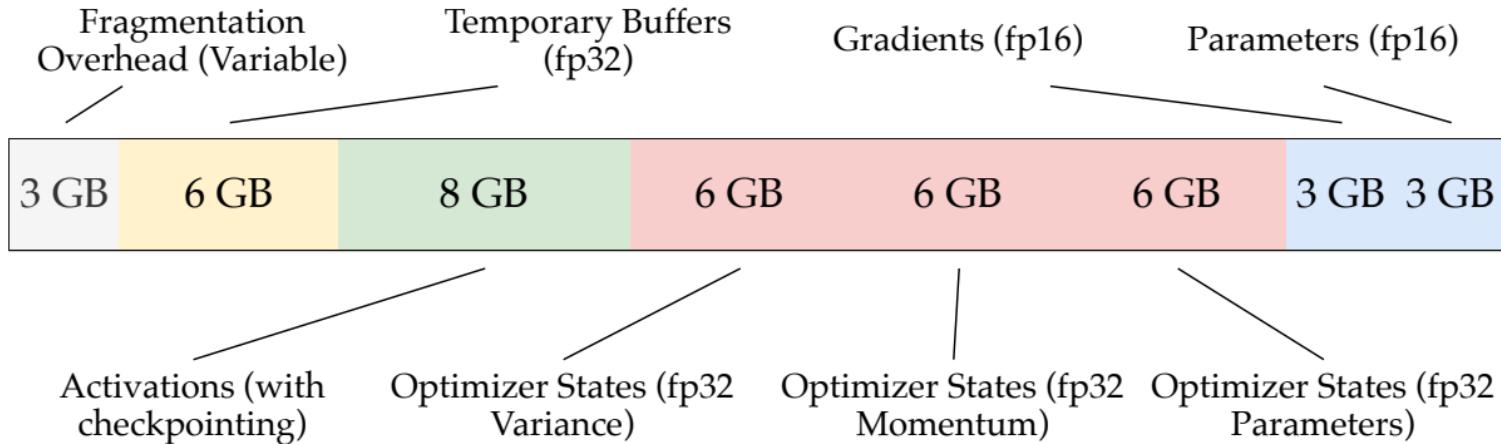


# Потребление памяти при обучении GPT-2<sup>6</sup>



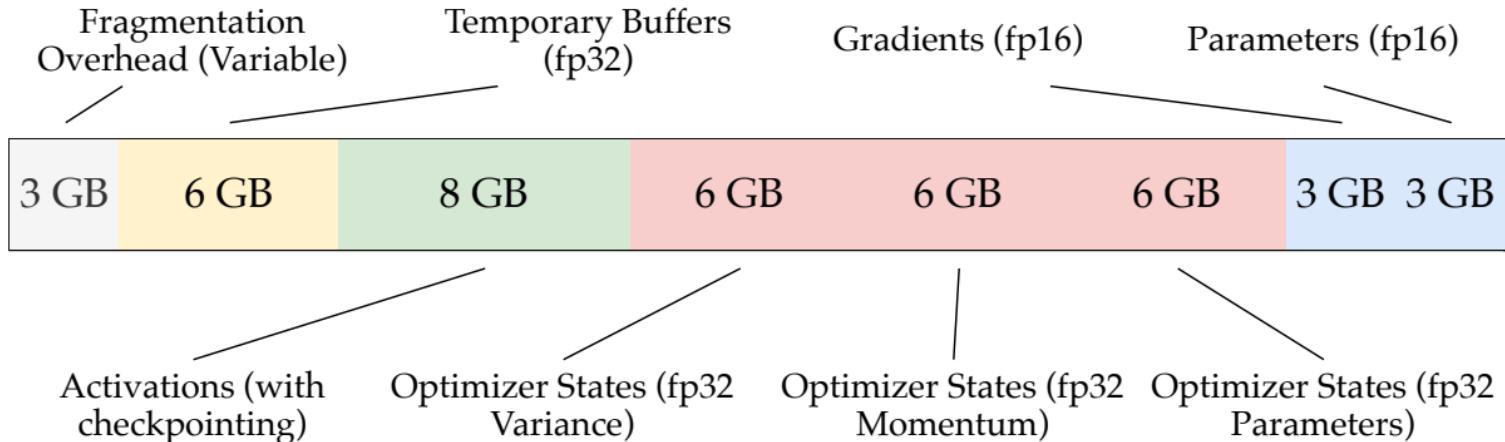
- Размер модели 1.5 В. Веса модели в fp16 занимают всего 3 GB, однако, для наивного обучения не хватит GPU даже на 32 GB

# Потребление памяти при обучении GPT-2<sup>6</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>6</sup>



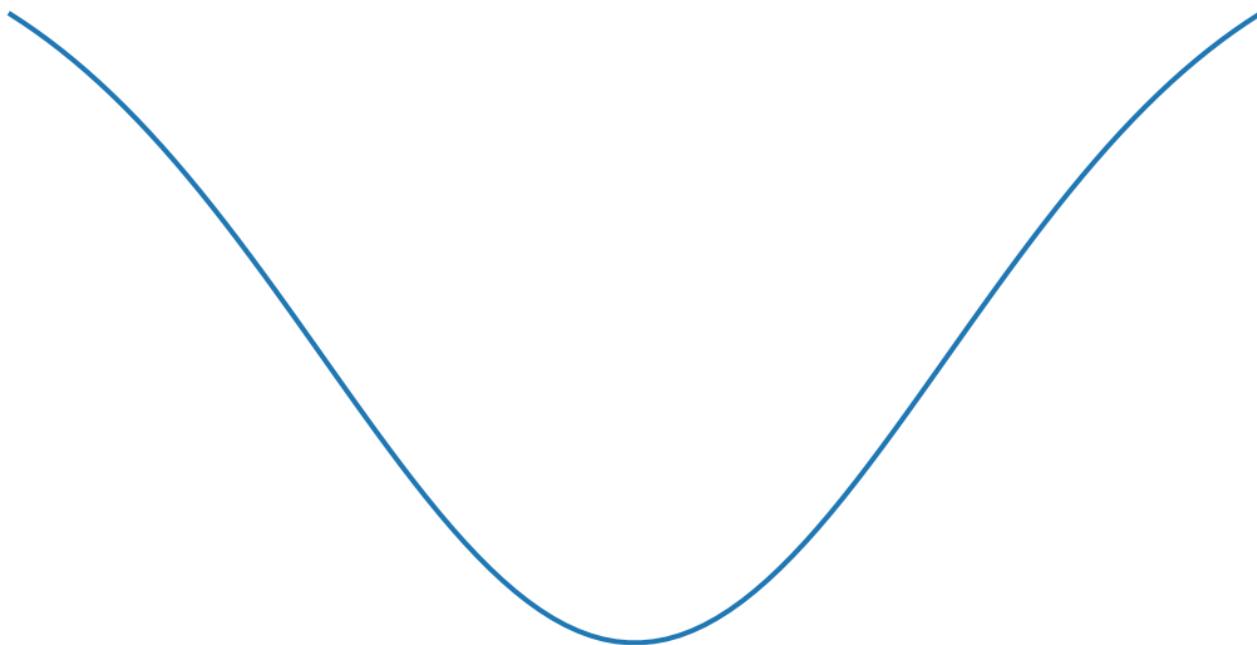
- Размер модели 1.5 В. Веса модели в fp16 занимают всего 3 GB, однако, для наивного обучения не хватит GPU даже на 32 GB
- Для использования Adam в режиме mixed precision необходимо хранить 3 (!) копии модели в fp32.
- Активации в наивном режиме могут занимать гораздо больше памяти: для длины последовательности 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>6</sup>ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

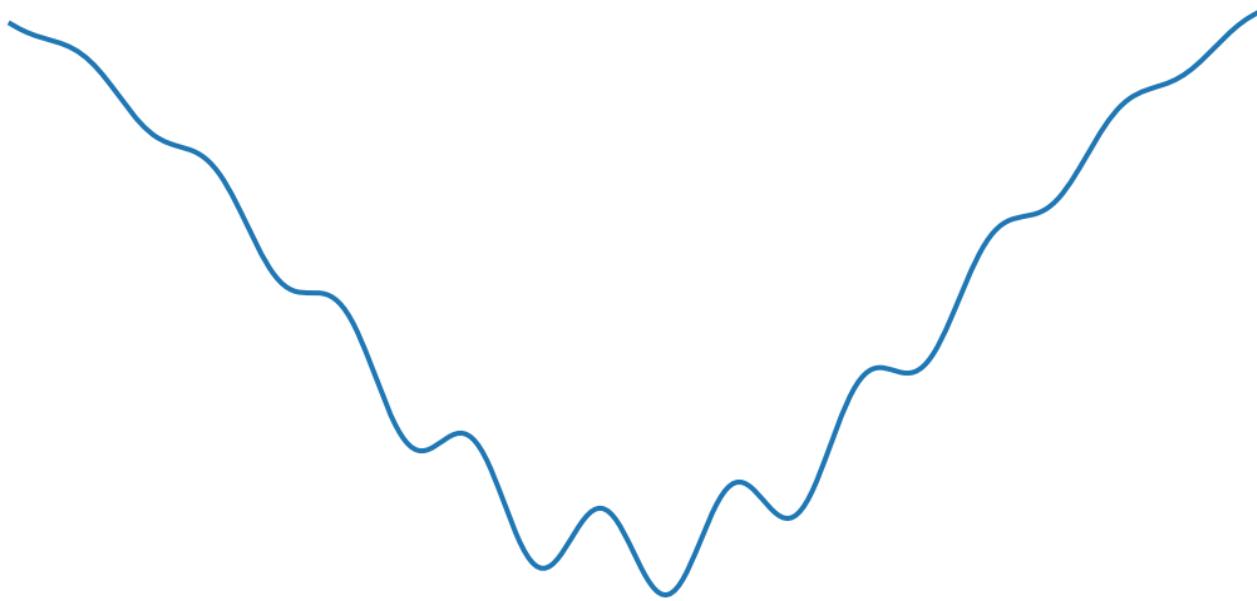


# Кто виноват?

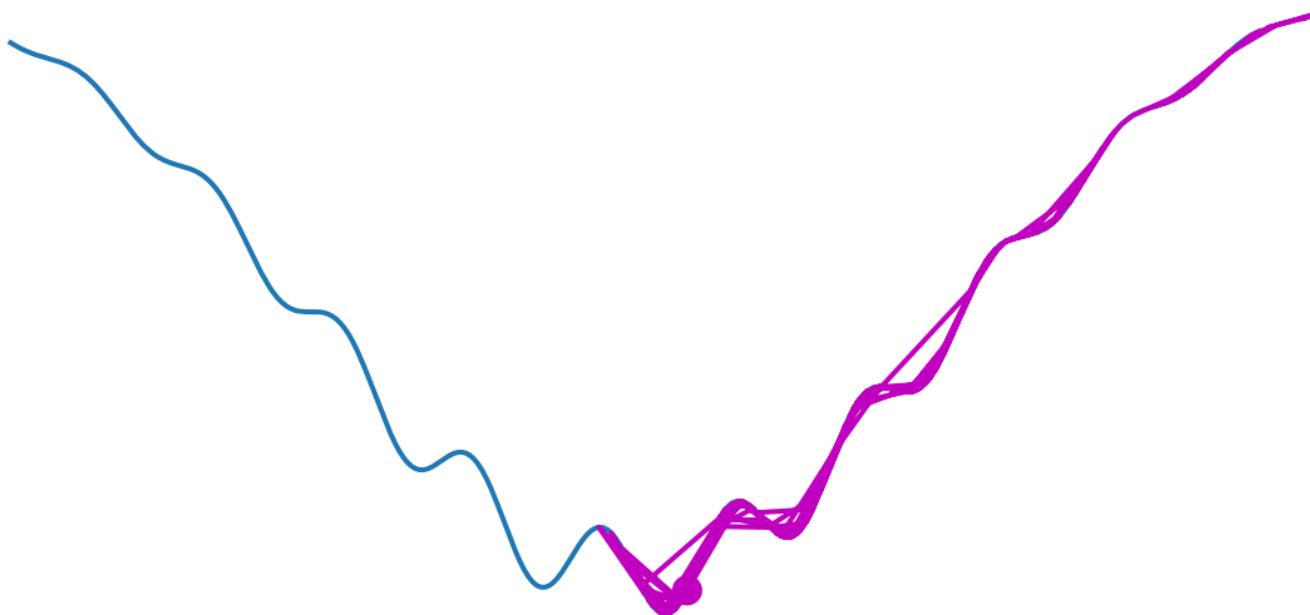
Градиентный спуск сходится к локальному минимуму



Градиентный спуск  
сходится к локальному минимуму



Стохастический градиентный спуск  
выпрыгивает из локальных минимумов



# Визуализация с помощью проекции на прямую

- Обозначим через  $w_0$  начальные веса нейронной сети. Веса, полученные после обучения, обозначим  $\hat{w}$ .

# Визуализация с помощью проекции на прямую

- Обозначим через  $w_0$  начальные веса нейронной сети. Веса, полученные после обучения, обозначим  $\hat{w}$ .

# Визуализация с помощью проекции на прямую

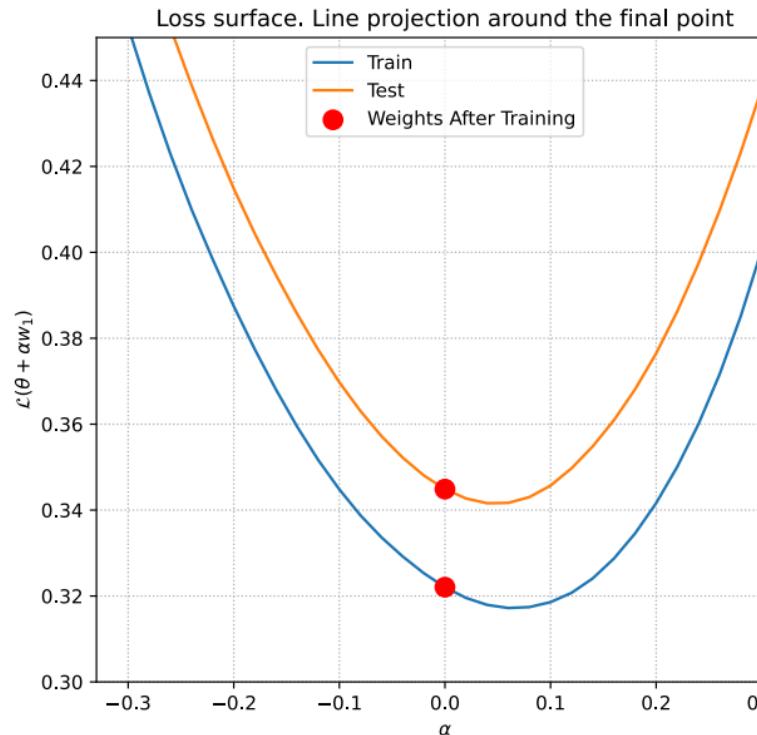
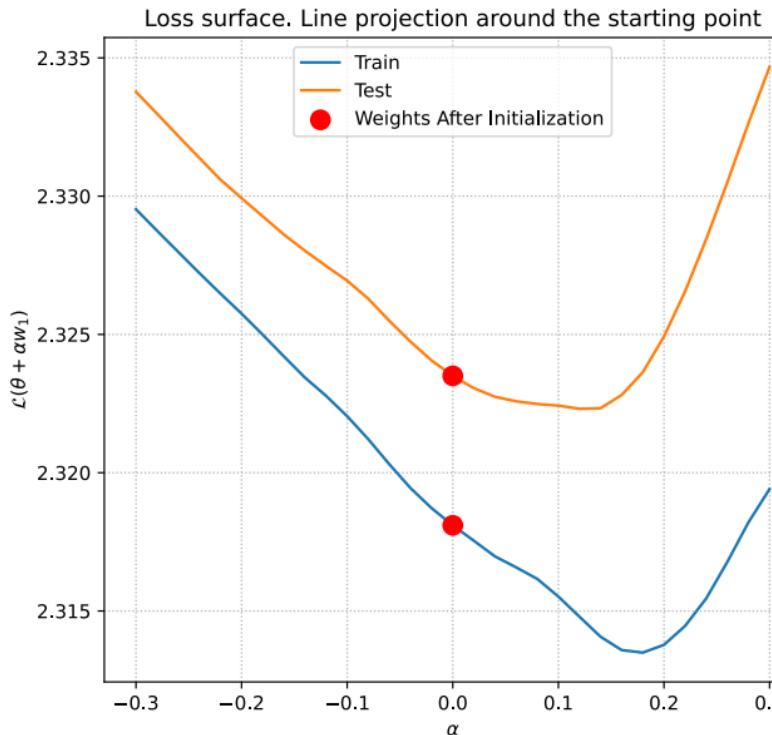
- Обозначим через  $w_0$  начальные веса нейронной сети. Веса, полученные после обучения, обозначим  $\hat{w}$ .
- Сгенерируем случайное направление  $w_1 \in \mathbb{R}^p$  той же размерности, затем вычислим значение функции потерь вдоль этого направления:

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

# Проекция функции потерь нейронной сети на прямую



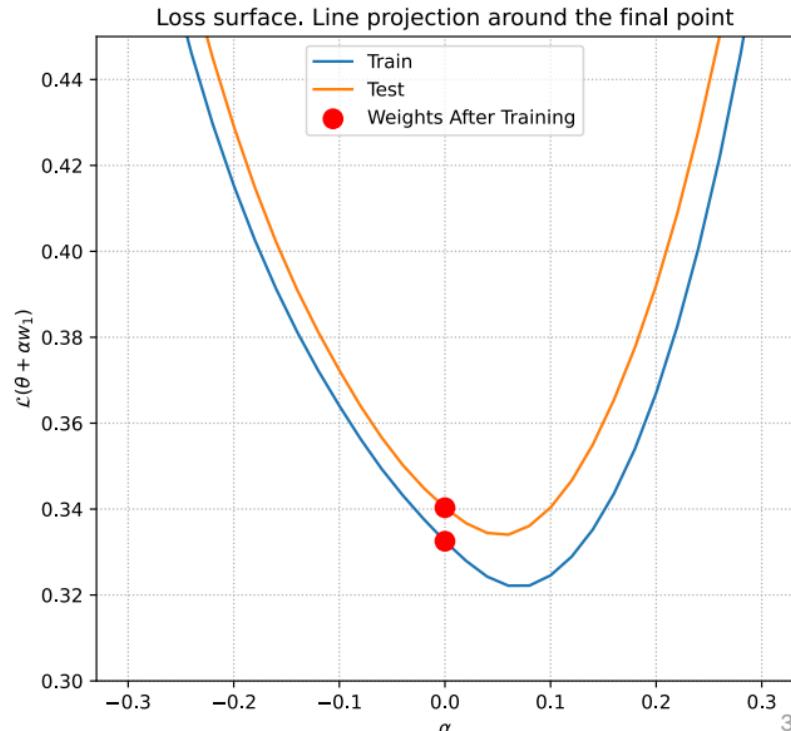
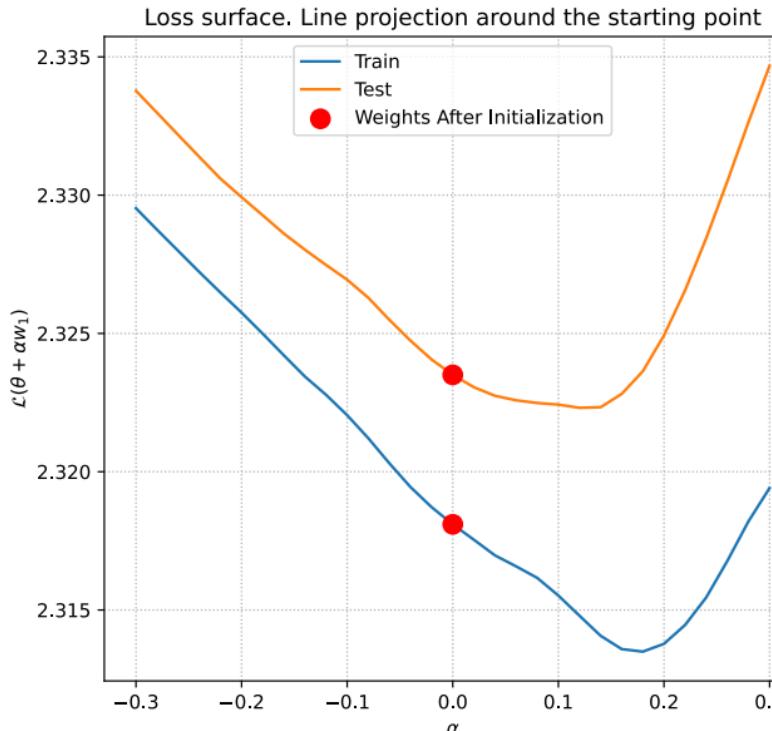
No Dropout



# Проекция функции потерь нейронной сети на прямую



Dropout 0.2



# Проекция функции потерь нейронной сети на плоскость

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

# Проекция функции потерь нейронной сети на плоскость

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

# 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.

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

# 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.

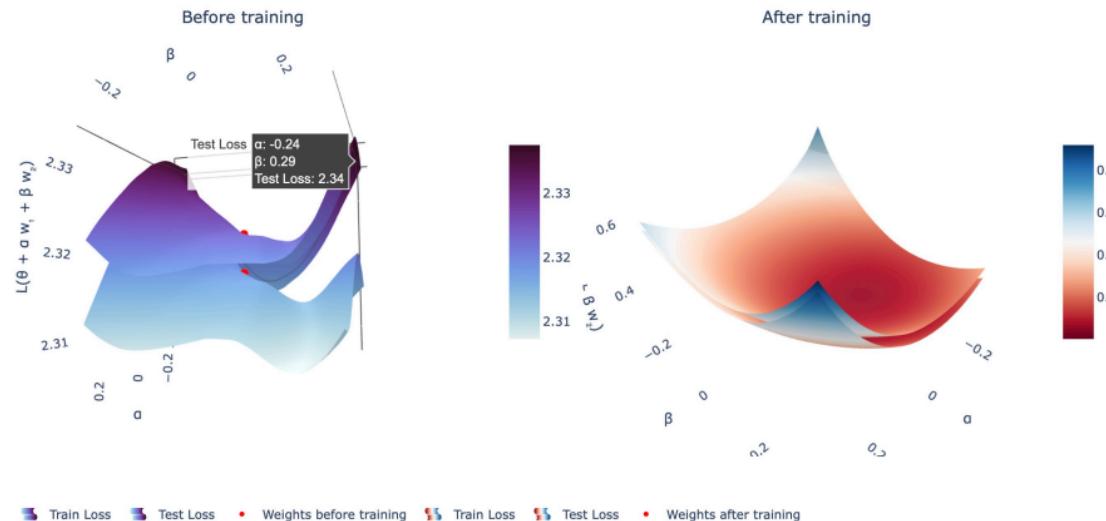
$$L(\alpha, \beta) = L(w_0 + \alpha w_1 + \beta w_2), \quad \text{где } \alpha, \beta \in [-b, b]^2.$$

# 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.

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

No Dropout. Plane projection of loss surface.



# Может ли быть полезно изучение таких проекций?



7

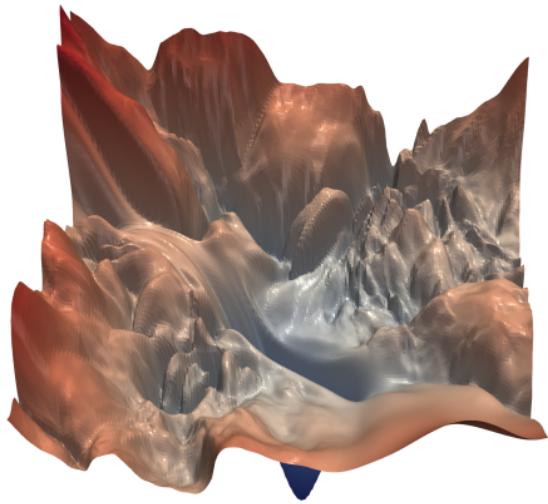


Рисунок 9. The loss surface of ResNet-56  
without skip connections

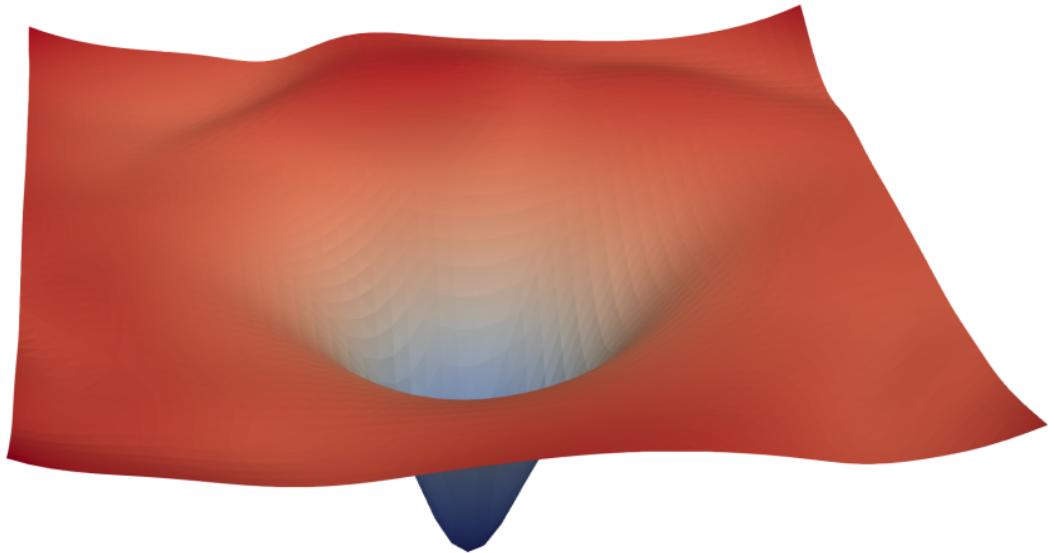


Рисунок 10. The loss surface of ResNet-56 with skip connections

<sup>7</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>8</sup>

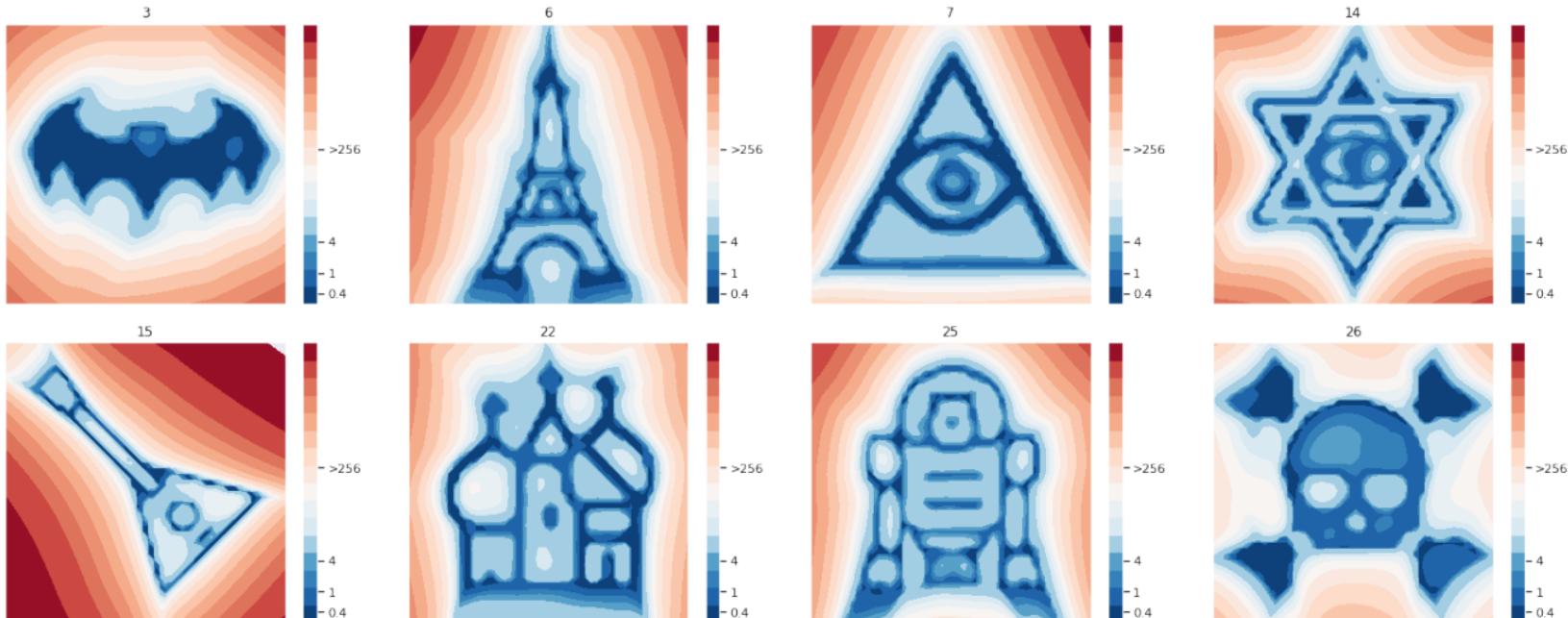
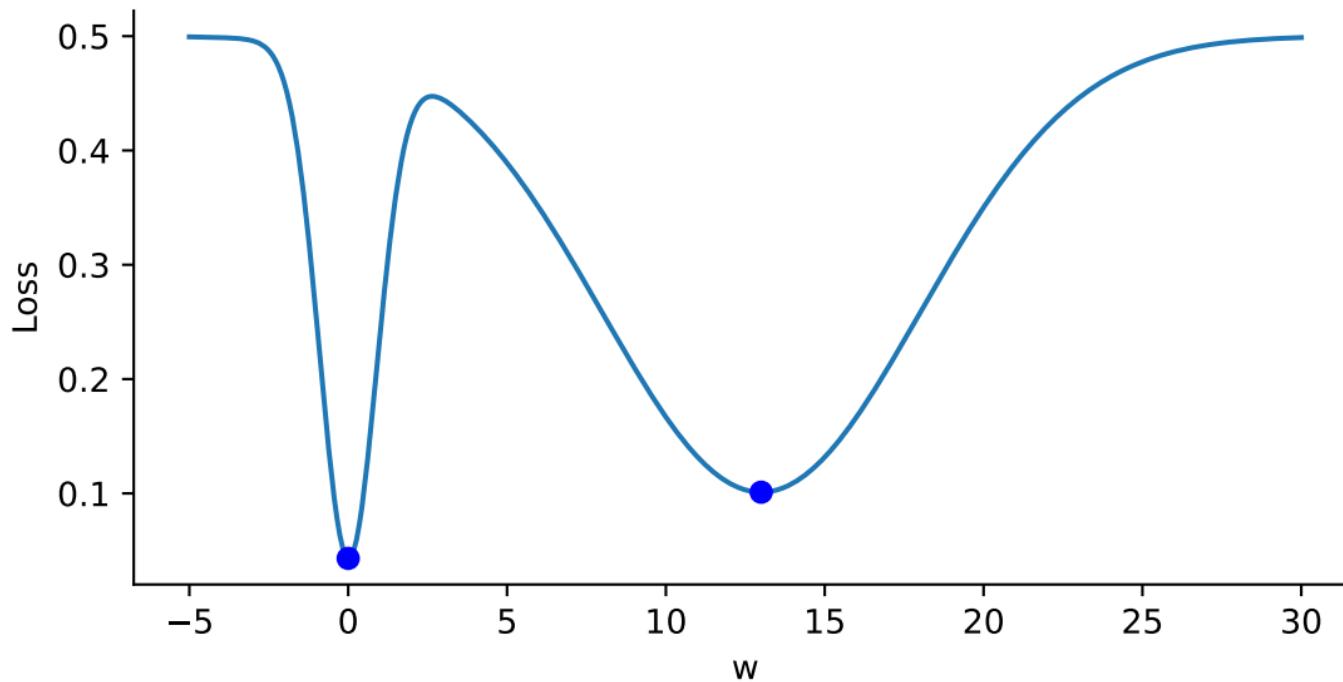


Рисунок 11. Examples of a loss landscape of a typical CNN model on FashionMNIST and CIFAR10 datasets found with MPO. Loss values are color-coded according to a logarithmic scale

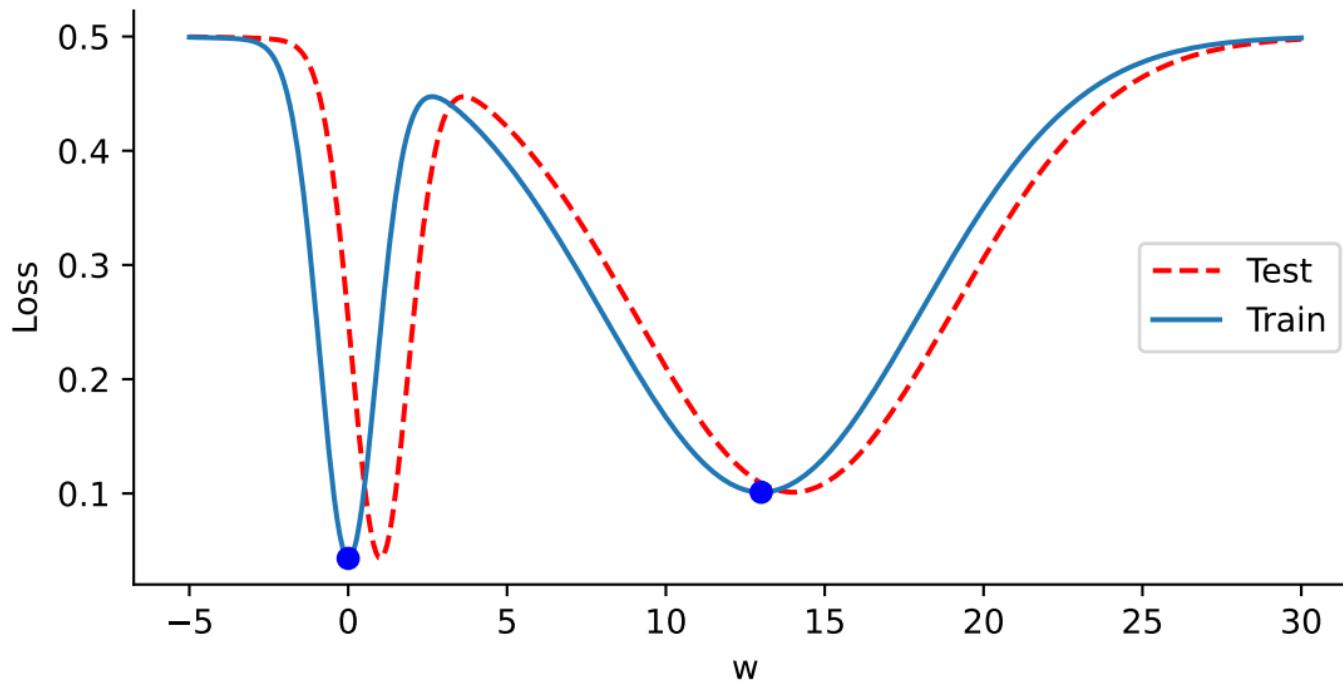
# Ширина локальных минимумов

Узкие и широкие локальные минимумы



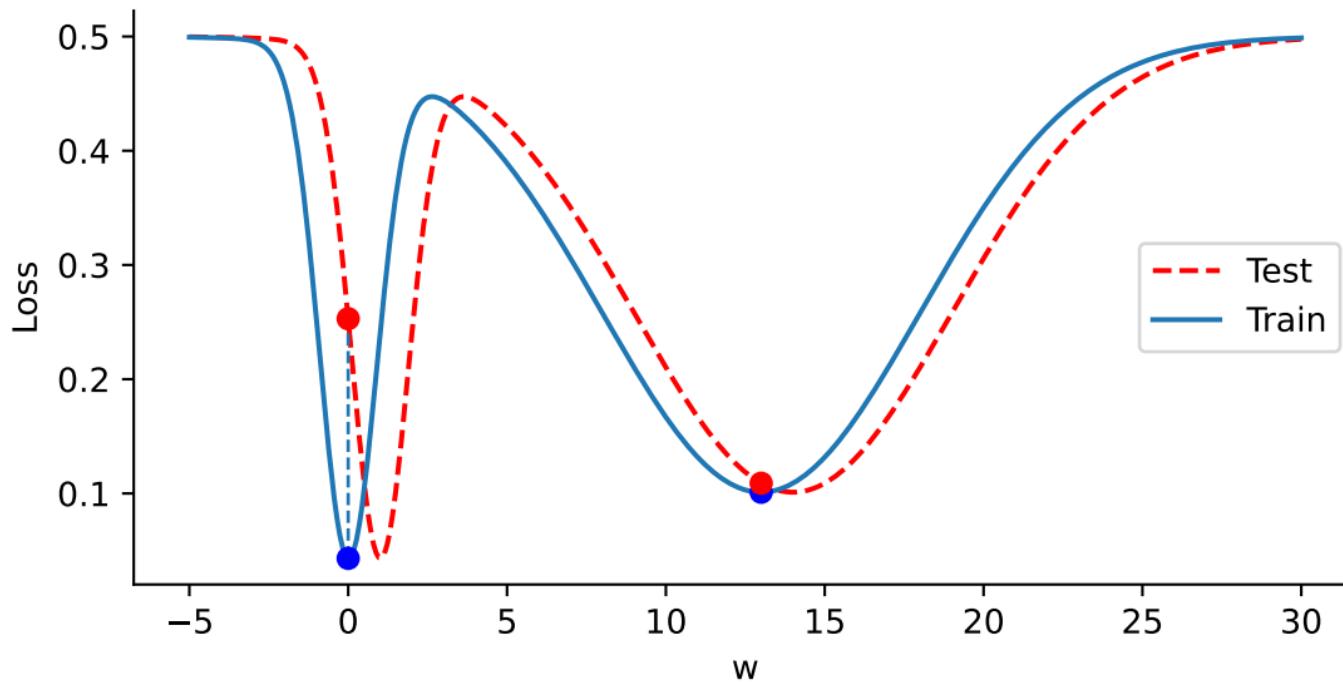
# Ширина локальных минимумов

Узкие и широкие локальные минимумы



# Ширина локальных минимумов

Узкие и широкие локальные минимумы



# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!)<sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?

# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!)<sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?
- Авторы предлагают **экспоненциально растущую** стратегию LR:

$$\eta_t = \eta_0(1 + \alpha)^t, \quad \alpha > 0$$

# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!)<sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?
- Авторы предлагают **экспоненциально растущую** стратегию LR:

$$\eta_t = \eta_0(1 + \alpha)^t, \quad \alpha > 0$$

- Несмотря на быстрое «взрывание» шага, обучение **всё ещё возможно**.

# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!)<sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?
- Авторы предлагают **экспоненциально растущую** стратегию LR:

$$\eta_t = \eta_0(1 + \alpha)^t, \quad \alpha > 0$$

- Несмотря на быстрое «взрывание» шага, обучение **всё ещё возможно**.
- Экспериментальный факт:

# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!)<sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?
- Авторы предлагают **экспоненциально растущую** стратегию LR:

$$\eta_t = \eta_0(1 + \alpha)^t, \quad \alpha > 0$$

- Несмотря на быстрое «взрывание» шага, обучение **всё ещё возможно**.
- Экспериментальный факт:
  - стандартные архитектуры для CIFAR-10 (например, PreResNet-32) успешно обучаются с ExpLR;

# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!)<sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?
- Авторы предлагают **экспоненциально растущую** стратегию LR:

$$\eta_t = \eta_0(1 + \alpha)^t, \quad \alpha > 0$$

- Несмотря на быстрое «взрывание» шага, обучение **всё ещё возможно**.
- Экспериментальный факт:
  - стандартные архитектуры для CIFAR-10 (например, PreResNet-32) успешно обучаются с ExpLR;
  - при корректном выборе  $\alpha$  траектория оптимизации оказывается близка к классической стратегии: фиксированный LR + weight decay.

# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!)<sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?
- Авторы предлагают **экспоненциально растущую** стратегию LR:

$$\eta_t = \eta_0(1 + \alpha)^t, \quad \alpha > 0$$

- Несмотря на быстрое «взрывание» шага, обучение **всё ещё возможно**.
- Экспериментальный факт:
  - стандартные архитектуры для CIFAR-10 (например, PreResNet-32) успешно обучаются с ExpLR;
  - при корректном выборе  $\alpha$  траектория оптимизации оказывается близка к классической стратегии: фиксированный LR + weight decay.
- Наблюдение: нормализация + weight decay создают эффект, напоминающий «эффективное увеличение» LR в процессе обучения.

# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!) <sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?
- Авторы предлагают **экспоненциально растущую** стратегию LR:

$$\eta_t = \eta_0(1 + \alpha)^t, \quad \alpha > 0$$

- Несмотря на быстрое «взрывание» шага, обучение **всё ещё возможно**.
- Экспериментальный факт:
  - стандартные архитектуры для CIFAR-10 (например, PreResNet-32) успешно обучаются с ExpLR;
  - при корректном выборе  $\alpha$  траектория оптимизации оказывается близка к классической стратегии: фиксированный LR + weight decay.
- Наблюдение: нормализация + weight decay создают эффект, напоминающий «эффективное увеличение» LR в процессе обучения.

<sup>9</sup>Exponential Learning Rate Schedules for Deep Learning (2020)

# Немного про LR schedulers: Экспоненциально растущий LR (ExpLR) (??!??!)<sup>9</sup>

- Вопрос авторов: действительно ли уменьшение LR является необходимым условием успешного обучения глубоких сетей?
- Авторы предлагают **экспоненциально растущую** стратегию LR:

$$\eta_t = \eta_0(1 + \alpha)^t, \quad \alpha > 0$$

- Несмотря на быстрое «взрывание» шага, обучение **всё ещё возможно**.
- Экспериментальный факт:
  - стандартные архитектуры для CIFAR-10 (например, PreResNet-32) успешно обучаются с ExpLR;
  - при корректном выборе  $\alpha$  траектория оптимизации оказывается близка к классической стратегии: фиксированный LR + weight decay.
- Наблюдение: нормализация + weight decay создают эффект, напоминающий «эффективное увеличение» LR в процессе обучения.

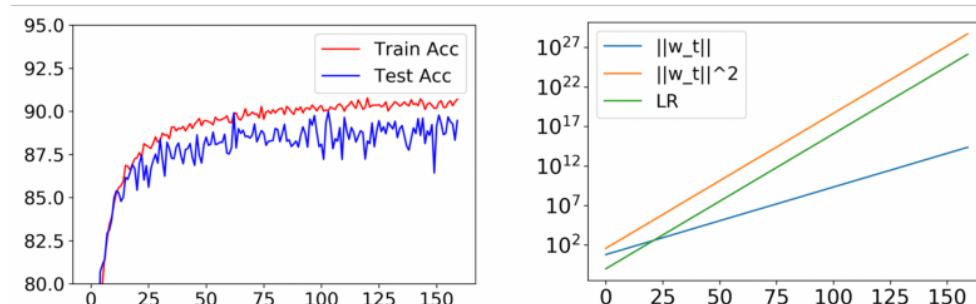
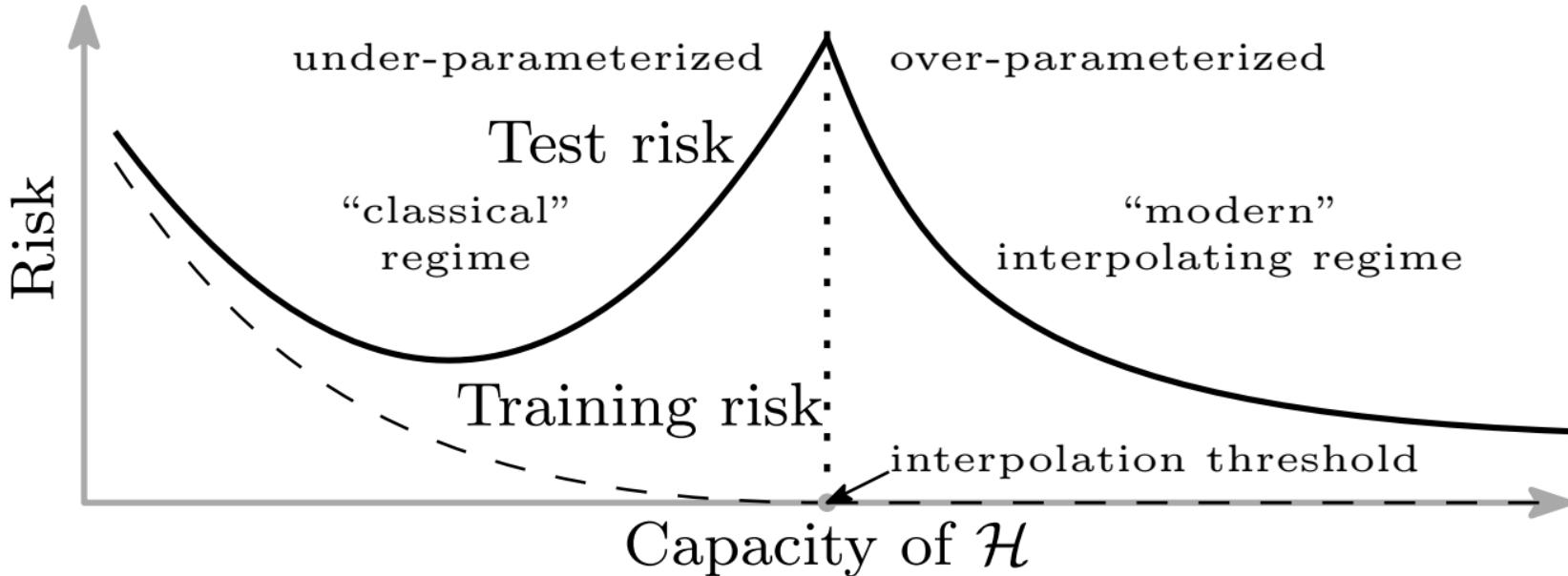


Рисунок 12. Обучение PreResNet32 на CIFAR10. Траектория с фиксированным LR и WD совпадает с ExpLR ( $\tilde{\eta}_t = 0.1 \times 1.481^t$ ) без WD. Справа: норма весов растет экспоненциально,  $|w_t|_2^2 / \tilde{\eta}_t = \text{const}$ .

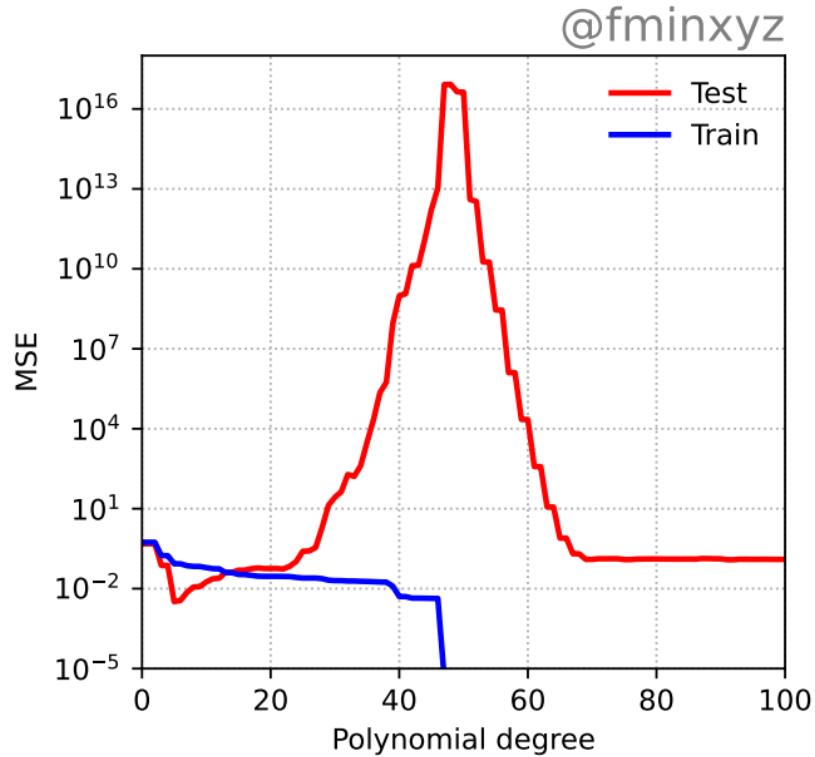
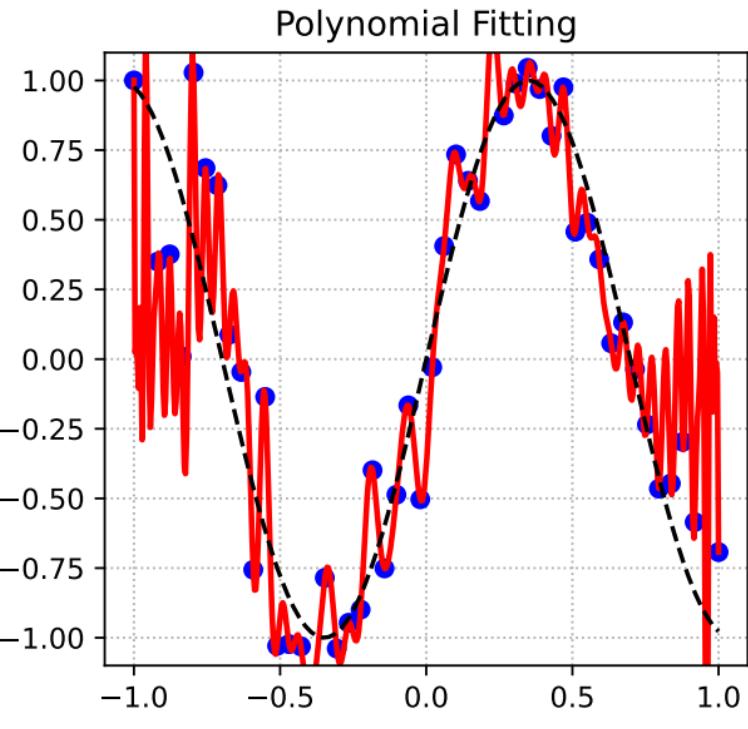
<sup>9</sup>Exponential Learning Rate Schedules for Deep Learning (2020)

# Double Descent<sup>10</sup>



<sup>10</sup> Reconciling modern machine learning practice and the bias-variance trade-off, Mikhail Belkin, Daniel Hsu, Siyuan Ma, Soumik Mandal

# Double Descent



# Grokking<sup>11</sup>

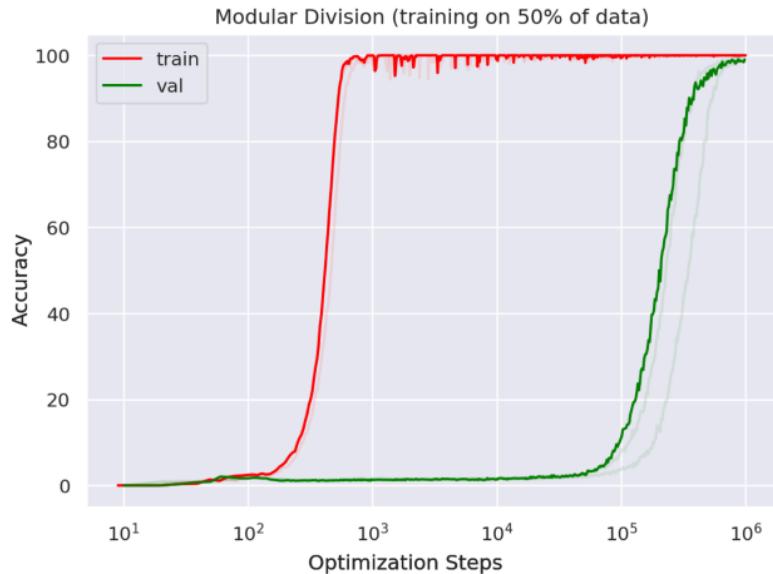


Рисунок 13. Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments (~ half an hour) is available here

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

# Grokking<sup>11</sup>

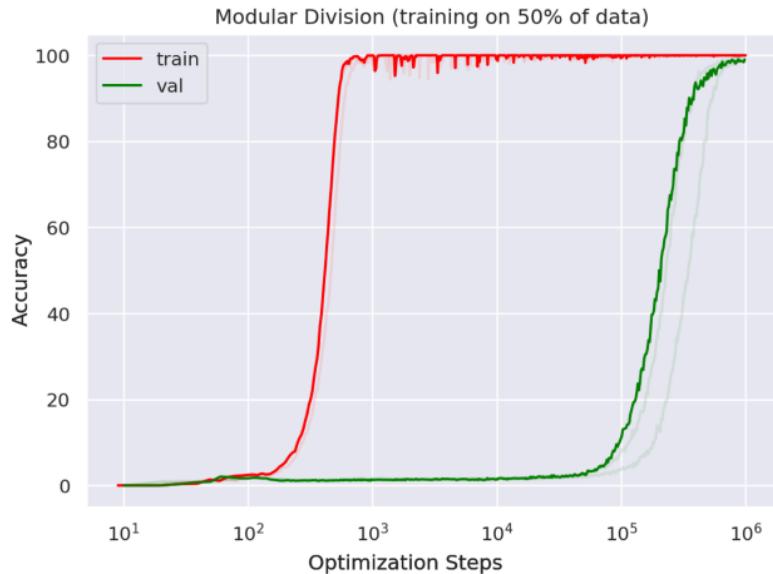


Рисунок 13. Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments (~ half an hour) is available here

- 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.

# Grokking<sup>11</sup>

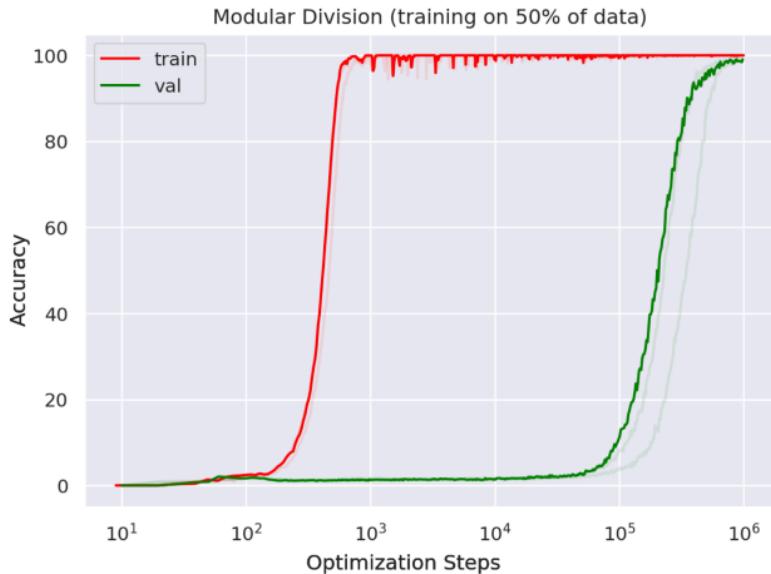


Рисунок 13. Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments (~ half an hour) is available here

- 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>11</sup>Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, Vedant Misra

# Что делать?

# Как сравнивать методы? Бенчмарк AlgoPerf<sup>12 13</sup>



- **Бенчмарк AlgoPerf:** Сравнивает алгоритмы обучения нейросетей в двух режимах:

# Как сравнивать методы? Бенчмарк AlgoPerf

12 13



- **Бенчмарк AlgoPerf:** Сравнивает алгоритмы обучения нейросетей в двух режимах:
  - Внешняя настройка (*External Tuning*): моделирует подбор гиперпараметров при ограниченных ресурсах (5 запусков, квазислучайный поиск).  
Оценка — медианное минимальное время достижения цели по 5 наборам задач.

# Как сравнивать методы? Бенчмарк AlgoPerf

12 13



- **Бенчмарк AlgoPerf:** Сравнивает алгоритмы обучения нейросетей в двух режимах:
  - **Внешняя настройка (External Tuning):** моделирует подбор гиперпараметров при ограниченных ресурсах (5 запусков, квазислучайный поиск). Оценка — медианное минимальное время достижения цели по 5 наборам задач.
  - **Самонастройка (Self-Tuning):** моделирует автоматический подбор на одной машине (фиксированный или внутренний подбор, бюджет x3). Оценка — медианное время выполнения по 5 наборам задач.

# Как сравнивать методы? Бенчмарк AlgoPerf

12 13



- **Бенчмарк AlgoPerf:** Сравнивает алгоритмы обучения нейросетей в двух режимах:
  - **Внешняя настройка (External Tuning):** моделирует подбор гиперпараметров при ограниченных ресурсах (5 запусков, квазислучайный поиск). Оценка — медианное минимальное время достижения цели по 5 наборам задач.
  - **Самонастройка (Self-Tuning):** моделирует автоматический подбор на одной машине (фиксированный или внутренний подбор, бюджет x3). Оценка — медианное время выполнения по 5 наборам задач.
- **Оценка:** результаты агрегируются с помощью профилей производительности. Профили показывают долю задач, решённых за время, не превышающее множитель  $\tau$  относительно самой быстрой посылки. Итоговая оценка — нормированная площадь под кривой профиля (1.0 = самая быстрая на всех задачах).

# Как сравнивать методы? Бенчмарк AlgoPerf<sup>12 13</sup>



- **Бенчмарк AlgoPerf:** Сравнивает алгоритмы обучения нейросетей в двух режимах:
  - **Внешняя настройка (External Tuning):** моделирует подбор гиперпараметров при ограниченных ресурсах (5 запусков, квазислучайный поиск). Оценка — медианное минимальное время достижения цели по 5 наборам задач.
  - **Самонастройка (Self-Tuning):** моделирует автоматический подбор на одной машине (фиксированный или внутренний подбор, бюджет x3). Оценка — медианное время выполнения по 5 наборам задач.
- **Оценка:** результаты агрегируются с помощью профилей производительности. Профили показывают долю задач, решённых за время, не превышающее множитель  $\tau$  относительно самой быстрой посылки. Итоговая оценка — нормированная площадь под кривой профиля (1.0 = самая быстрая на всех задачах).
- **Затраты ресурсов:** оценка требует  $\sim 49,240$  часов суммарно на 8x NVIDIA V100 GPUs (в среднем  $\sim 3469$  ч/внешняя настройка,  $\sim 1847$  ч/самонастройка).

---

<sup>12</sup>Benchmarking Neural Network Training Algorithms

<sup>13</sup>Accelerating neural network training: An analysis of the AlgoPerf competition

# Бенчмарк AlgoPerf



**Сводка фиксированных базовых задач в бенчмарке AlgoPerf.** Функции потерь включают кросс-энтропию (CE), среднюю абсолютную ошибку (L1) и функцию потерь CTC (Connectionist Temporal Classification). Дополнительные метрики оценки: индекс структурного сходства (SSIM), коэффициент ошибок (ER), доля ошибок по словам (WER), средняя усреднённая точность (mAP) и метрика BLEU (*bilingual evaluation understudy*). Бюджет времени выполнения соответствует правилам внешней настройки; правила самонастройки допускают обучение, в 3 раза более длительное.

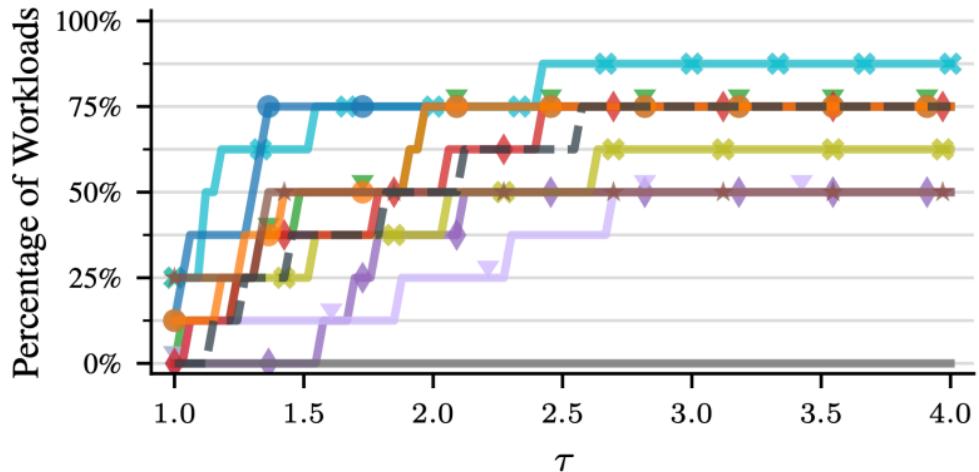
Задача	Датасет	Модель	Функция потерь	Метрика	Целевое значение (валидация)	Бюджет времени
Clickthrough rate prediction	CRITEO 1TB	DLRMSMALL	CE	CE	0.123735	7703
MRI reconstruction	FASTMRI	U-NET	L1	SSIM	0.7344	8859
Image classification	IMAGENET	ResNet-50	CE	ER	0.22569	63,008
		ViT	CE	ER	0.22691	77,520
Speech 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

# Бенчмарк AlgoPerf



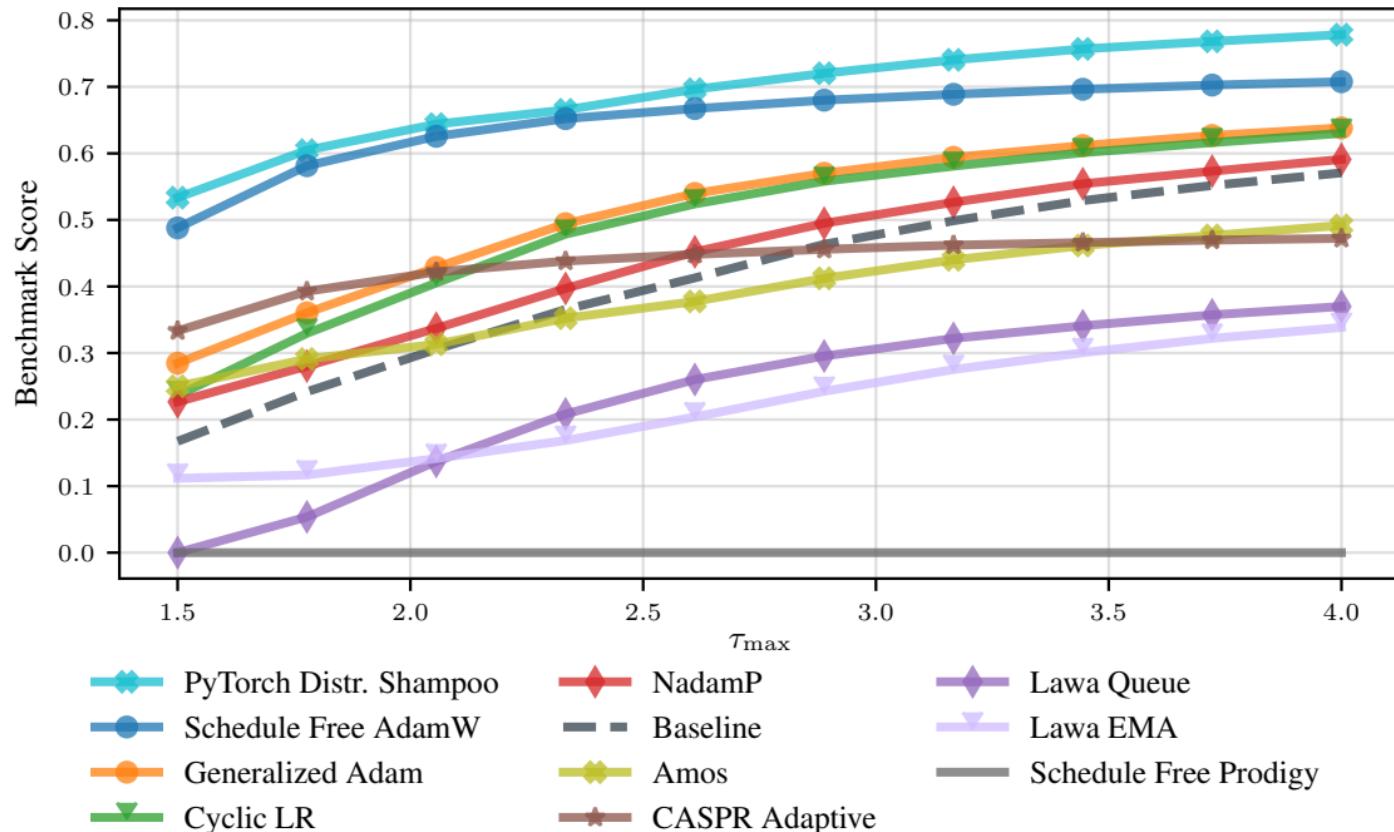
Submission	Line	Score
PYTORCH DISTRIBUTED SHAMPOO		0.7784
SCHEDULE FREE ADAMW		0.7077
GENERALIZED ADAM		0.6383
CYCLIC LR		0.6301
NADAMP		0.5909
BASELINE		0.5707
AMOS		0.4918
CASPR ADAPTIVE		0.4722
LAWA QUEUE		0.3699
LAWA EMA		0.3384
SCHEDULE FREE PRODIGY		0

(a) External tuning leaderboard

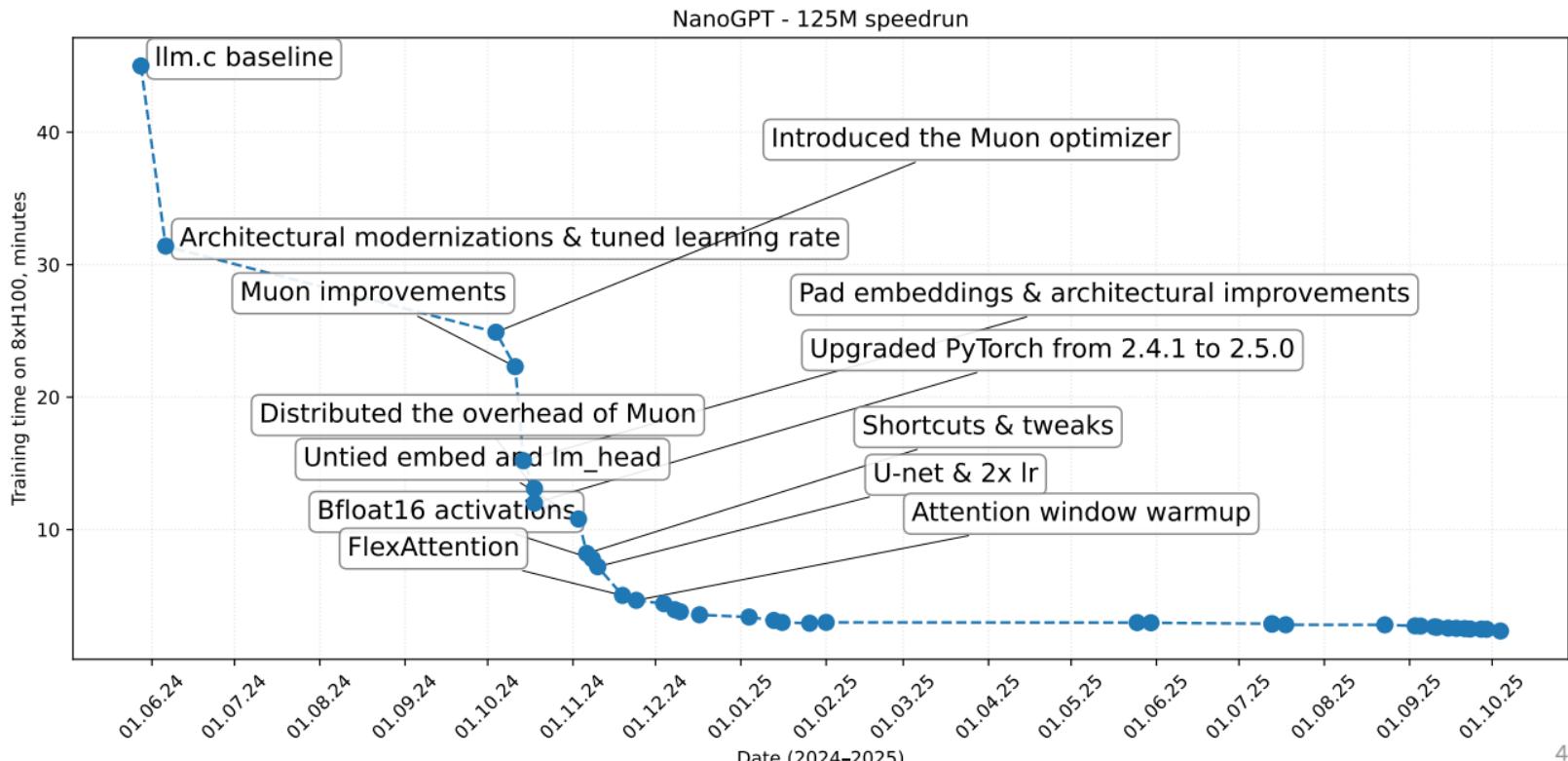


(b) External tuning performance profiles

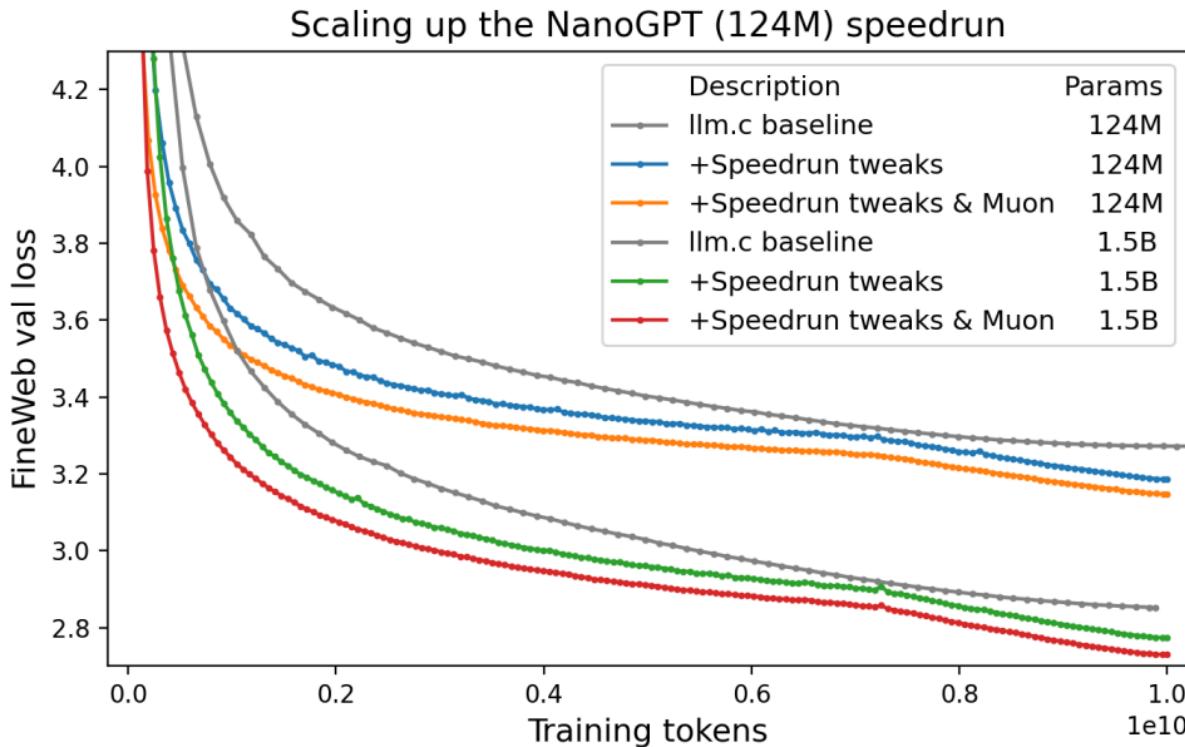
# Бенчмарк AlgoPerf



# NanoGPT speedrun



# Работают ли трюки, если увеличить размер модели?



# Работают ли трюки, если увеличить размер модели?

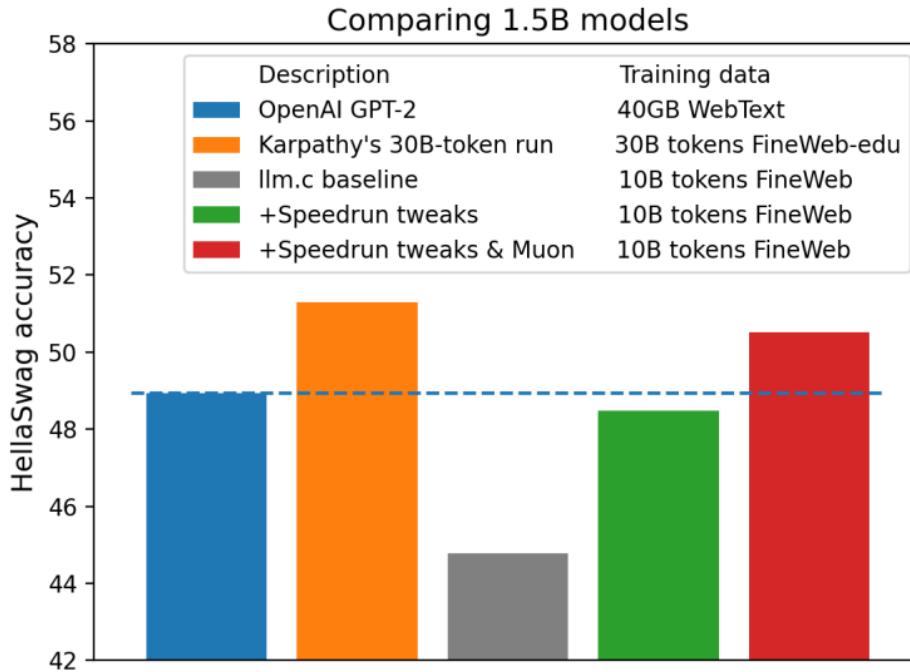


Рисунок 16. Источник

# Неожиданные истории

# Adam работает хуже для CV, чем для LLM? <sup>14</sup>

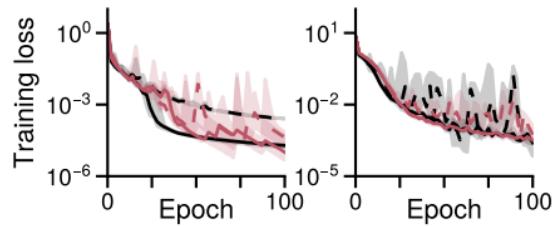


Рисунок 17. CNNs on MNIST and CIFAR10

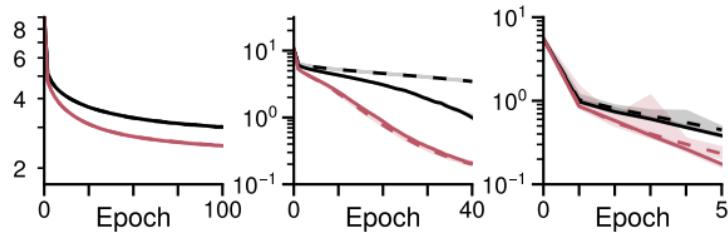


Рисунок 18. Transformers on PTB, WikiText2, and SQuAD

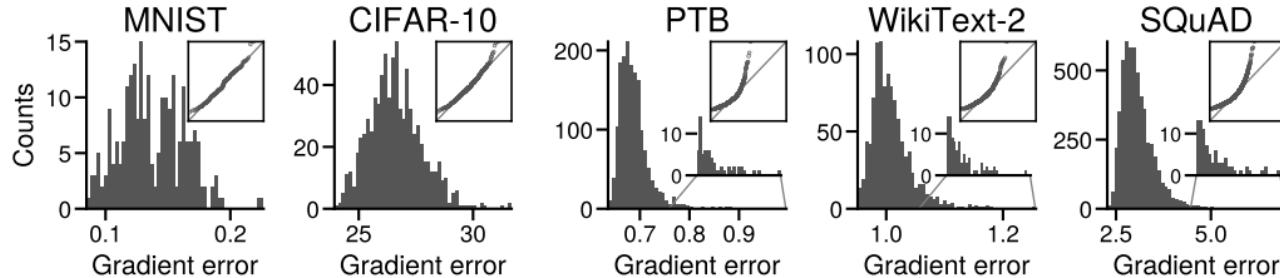
Чёрные линии — SGD, красные — Adam.

<sup>14</sup>Linear attention is (maybe) all you need (to understand transformer optimization)

# Почему Adam работает хуже для CV, чем для LLM?<sup>15</sup>



Потому что шум градиентов в языковых моделях имеет тяжелые хвосты?



<sup>15</sup>Linear attention is (maybe) all you need (to understand transformer optimization)

# Почему Adam работает хуже для CV, чем для LLM? <sup>16</sup>



Нет! Распределение меток имеет тяжёлые 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.

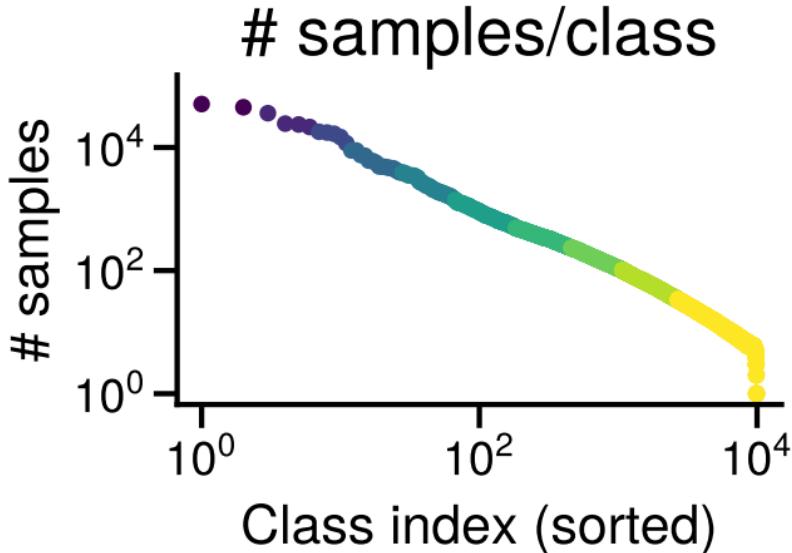


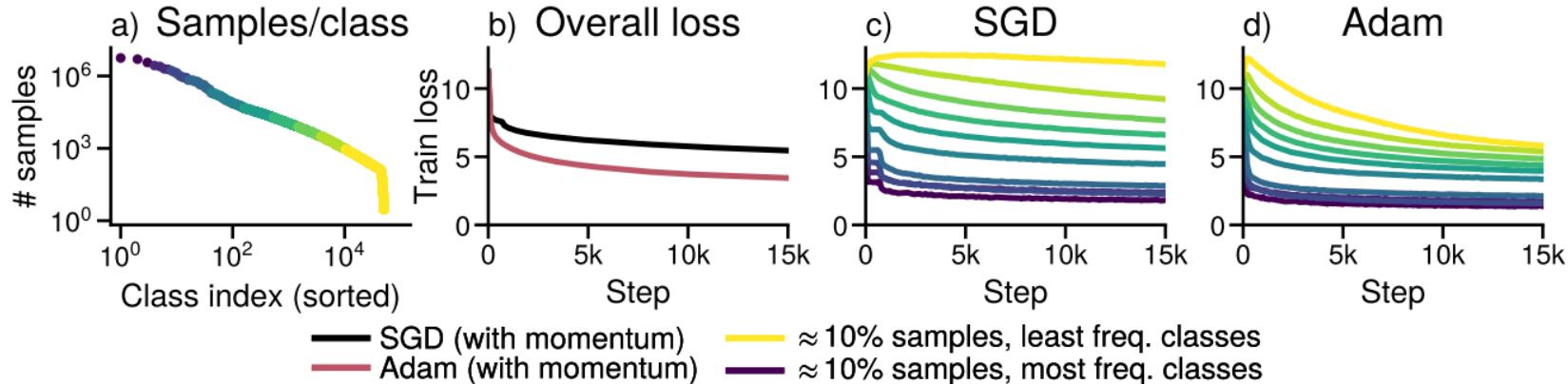
Рисунок 19. Распределение частоты токенов в PTB

<sup>16</sup> Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

# Почему Adam работает хуже для CV, чем для LLM? <sup>17</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>17</sup> Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

# Влияние инициализации



Правильная инициализация нейронной сети важна. Функция потерь нейронной сети сильно невыпукла; оптимизировать её для достижения «хорошего» решения трудно, это требует тщательной настройки.

# Влияние инициализации

💡 Правильная инициализация нейронной сети важна. Функция потерь нейронной сети сильно невыпукла; оптимизировать её для достижения «хорошего» решения трудно, это требует тщательной настройки.

- Не инициализируйте все веса одинаково — почему?

# Влияние инициализации

💡 Правильная инициализация нейронной сети важна. Функция потерь нейронной сети сильно невыпукла; оптимизировать её для достижения «хорошего» решения трудно, это требует тщательной настройки.

- Не инициализируйте все веса одинаково — почему?
- Случайная инициализация: инициализируйте 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*).

# Влияние инициализации<sup>18</sup>

💡 Правильная инициализация нейронной сети важна. Функция потерь нейронной сети сильно невыпукла; оптимизировать её для достижения «хорошего» решения трудно, это требует тщательной настройки.

- Не инициализируйте все веса одинаково — почему?
- Случайная инициализация: инициализируйте 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*).
- Можно найти более полезные советы здесь

<sup>18</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>19</sup>

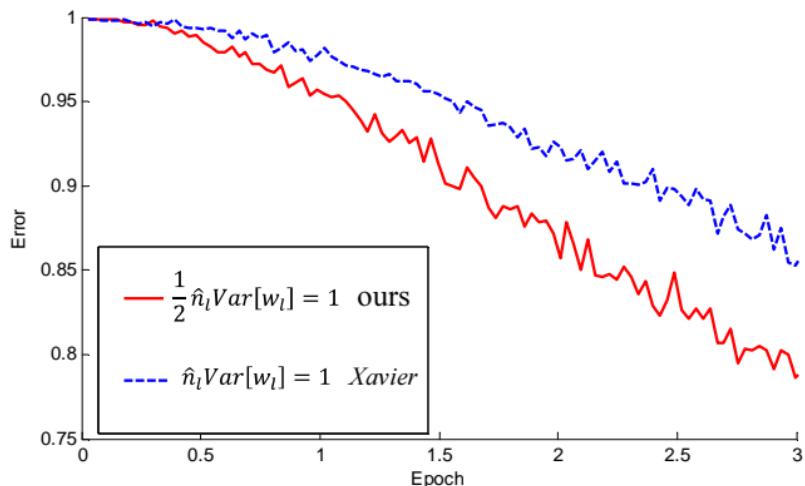


Рисунок 20. 22-layer ReLU net: good init converges faster

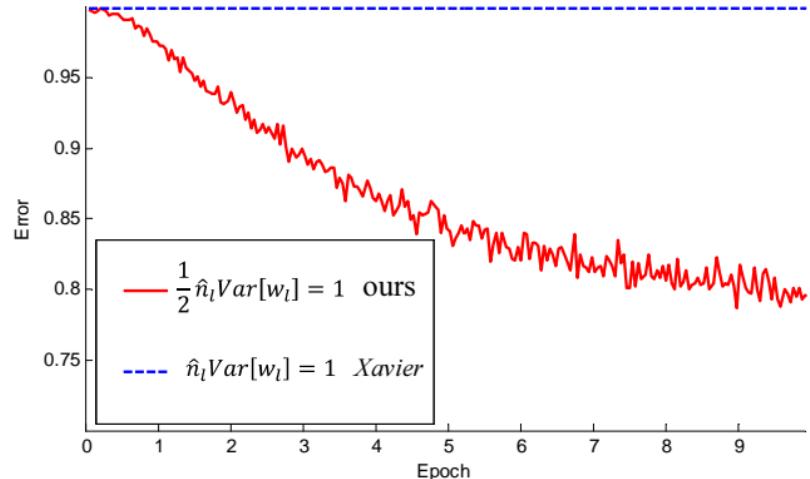


Рисунок 21. 30-layer ReLU net: good init is able to converge

<sup>19</sup>Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun