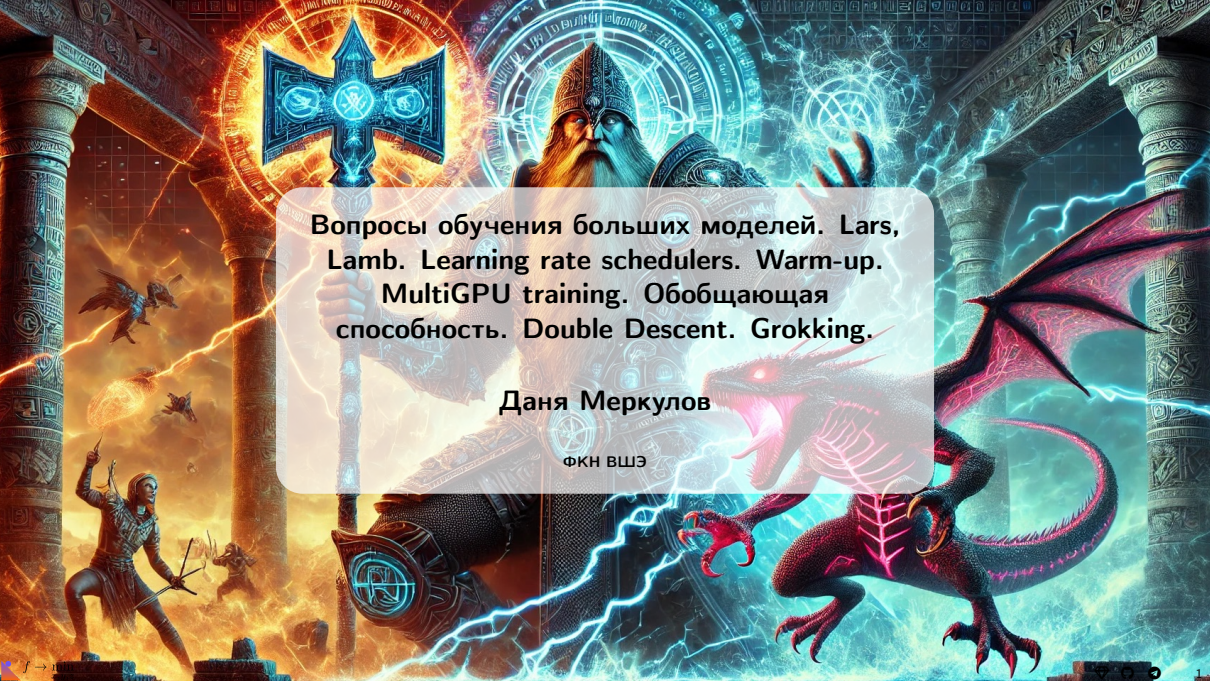


Вопросы обучения больших моделей. Lars, Lamb. Learning rate schedulers. Warm-up. MultiGPU training. Обобщающая способность. Double Descent. Grokking.

Даня Меркулов

ФКН ВШЭ

GPT-2

1.5 B

6 Гбайт

fp32

32 бита

4 байта

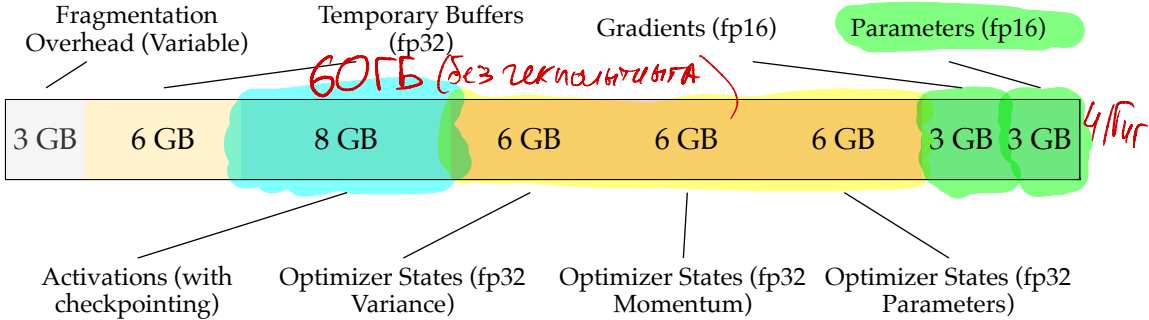
32 Гига

Потребление памяти при обучении

А для обучения влезет?

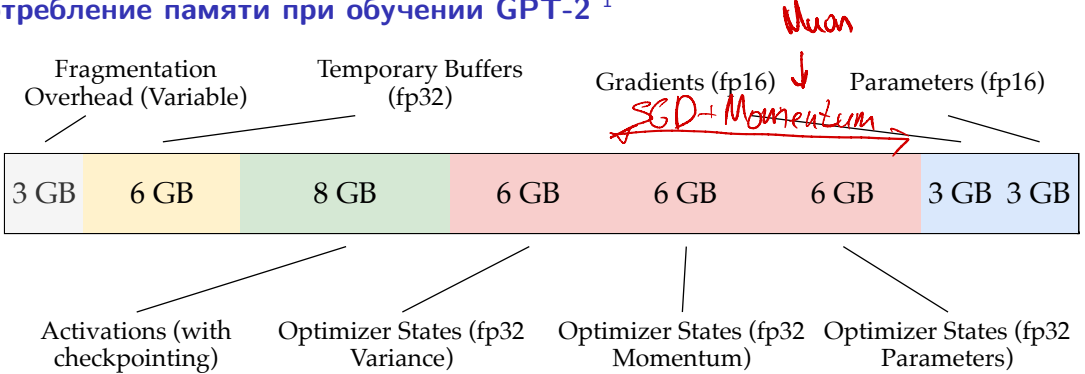
12 ГБ у Стаса при некоторых усилиях ВЛЕЗЕТ

Потребление памяти при обучении GPT-2 [1]



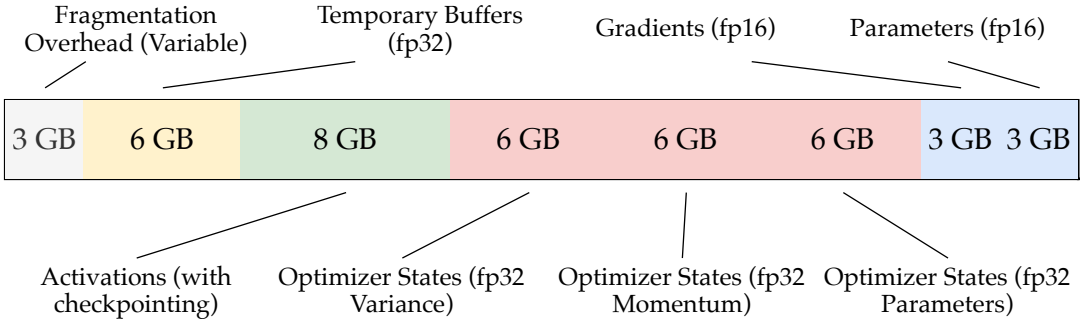
- Размер модели 1.5 B. Веса модели в fp16 занимают всего 3 GB, однако, для наивного обучения не хватит GPU даже на 32 GB

Потребление памяти при обучении GPT-2 [1]



- Размер модели 1.5 B. 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 [1]



- Размер модели 1.5 B. 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)

[1] ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

Из чего состоит память при обучении?

Для модели с N параметрами и оптимизатора Adam:

Компонент	FP32	Mixed Precision
Веса (forward/backward)	$4N$ байт	$2N$ байт (fp16)
Градиенты	$4N$ байт	$2N$ байт (fp16)
Adam: первый момент m	$4N$ байт	$4N$ байт (fp32)
Adam: второй момент v	$4N$ байт	$4N$ байт (fp32)
Master-копия весов	—	$4N$ байт (fp32)
Итого	$16N$	$16N$

Из чего состоит память при обучении?

Для модели с N параметрами и оптимизатора Adam:

Компонент	FP32	Mixed Precision
Веса (forward/backward)	$4N$ байт	$2N$ байт (fp16)
Градиенты	$4N$ байт	$2N$ байт (fp16)
Adam: первый момент m	$4N$ байт	$4N$ байт (fp32)
Adam: второй момент v	$4N$ байт	$4N$ байт (fp32)
Master-копия весов	—	$4N$ байт (fp32)
Итого	$16N$	$16N$

- Для GPT-2 (1.5B): минимум **24 GB** только на параметры + оптимизатор

Из чего состоит память при обучении?

Для модели с N параметрами и оптимизатора Adam:

Компонент	FP32	Mixed Precision
Веса (forward/backward)	$4N$ байт	$2N$ байт (fp16)
Градиенты	$4N$ байт	$2N$ байт (fp16)
Adam: первый момент m	$4N$ байт	$4N$ байт (fp32)
Adam: второй момент v	$4N$ байт	$4N$ байт (fp32)
Master-копия весов	—	$4N$ байт (fp32)
Итого	$16N$	$16N$

- Для GPT-2 (1.5B): минимум **24 GB** только на параметры + оптимизатор
- **Активации**: зависят от длины последовательности, размера батча и числа слоёв — часто **доминируют** в потреблении памяти

Числовые форматы для обучения нейросетей

Формат	Бит	Диапазон	Машинный ε	Применение
FP32	32	$\pm 3.4 \times 10^{38}$	1.2×10^{-7}	Мастер-веса, состояние оптимизатора
TF32	19	$\pm 3.4 \times 10^{38}$	9.8×10^{-4}	Matmul на Ampere+ (A100)
FP16	16	± 65504	9.8×10^{-4}	Forward/backward, требует loss scaling
BF16	16	$\pm 3.4 \times 10^{38}$	7.8×10^{-3}	Forward/backward, не требует loss scaling
FP8 (E4M3)	8	± 448	1.25×10^{-1}	Transformer Engine (H100+)

Числовые форматы для обучения нейросетей

Формат	Бит	Диапазон	Машинный ε	Применение
FP32	32	$\pm 3.4 \times 10^{38}$	1.2×10^{-7}	Мастер-веса, состояние оптимизатора
TF32	19	$\pm 3.4 \times 10^{38}$	9.8×10^{-4}	Matmul на Ampere+ (A100)
FP16	16	± 65504	9.8×10^{-4}	Forward/backward, требует loss scaling
BF16	16	$\pm 3.4 \times 10^{38}$	7.8×10^{-3}	Forward/backward, не требует loss scaling
FP8 (E4M3)	8	± 448	1.25×10^{-1}	Transformer Engine (H100+)

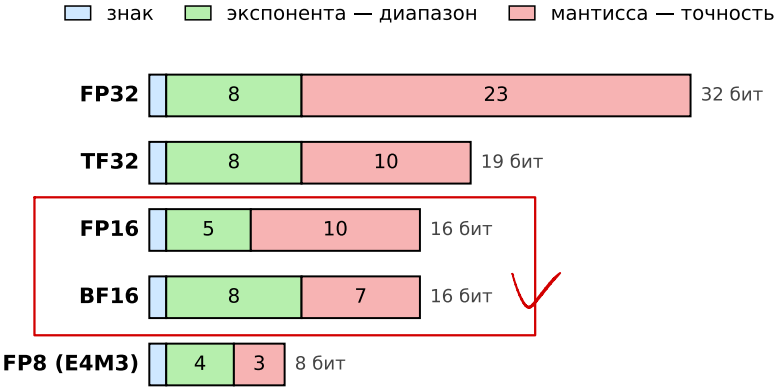
- **BF16 vs FP16**: BF16 имеет тот же диапазон, что FP32 (8 бит экспоненты), но меньшую точность. FP16 имеет больший диапазон мантиссы, но ограниченный диапазон — возникают overflow/underflow

Числовые форматы для обучения нейросетей

Формат	Бит	Диапазон	Машинный ε	Применение
FP32	32	$\pm 3.4 \times 10^{38}$	1.2×10^{-7}	Мастер-веса, состояние оптимизатора
TF32	19	$\pm 3.4 \times 10^{38}$	9.8×10^{-4}	Matmul на Ampere+ (A100)
FP16	16	± 65504	9.8×10^{-4}	Forward/backward, требует loss scaling
BF16	16	$\pm 3.4 \times 10^{38}$	7.8×10^{-3}	Forward/backward, не требует loss scaling
FP8 (E4M3)	8	± 448	1.25×10^{-1}	Transformer Engine (H100+)

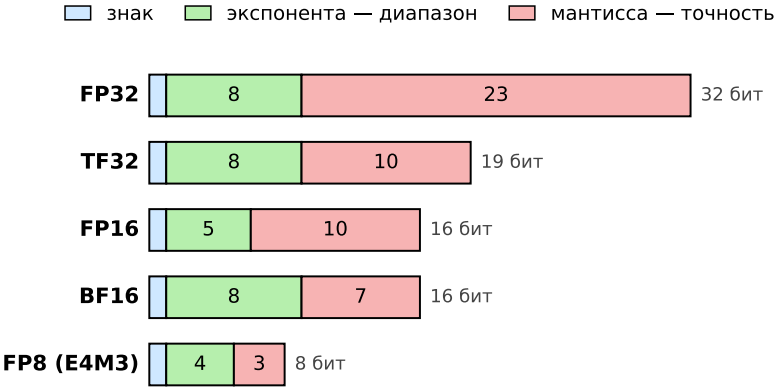
- **BF16 vs FP16**: BF16 имеет тот же диапазон, что FP32 (8 бит экспоненты), но меньшую точность. FP16 имеет больший диапазон мантиссы, но ограниченный диапазон — возникают overflow/underflow
- **TF32**: автоматически используется NVIDIA для `torch.matmul` на Ampere+ GPU — не требует изменения кода

Mixed Precision Training [2]



Ключевая идея: forward и backward pass в fp16/bf16, оптимизатор обновляет мастер-копию в fp32.

Mixed Precision Training [2]

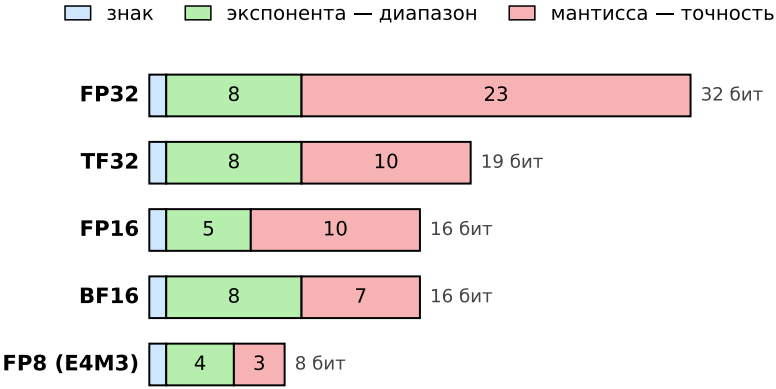


@fminxyz

Ключевая идея: forward и backward pass в fp16/bf16, оптимизатор обновляет мастер-копию в fp32.

Loss scaling (для FP16): градиенты в fp16 могут уходить в underflow. Решение — масштабировать loss перед backward:

Mixed Precision Training [2]



@fminxyz

Ключевая идея: forward и backward pass в fp16/bf16, оптимизатор обновляет мастер-копию в fp32.

Loss scaling (для FP16): градиенты в fp16 могут уходить в underflow. Решение — масштабировать loss перед backward:

Чекпоинтинг активаций [3]

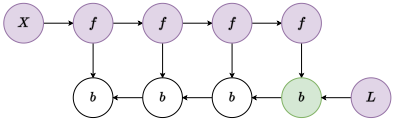


Figure 1: Стандартный backprop: все активации хранятся

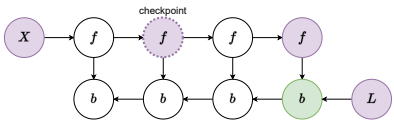


Figure 2: Чекпоинтинг: храним только отмеченные

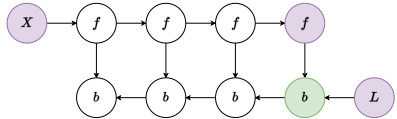


Figure 3: Минимум памяти: перевычисляем всё

Чекпоинтинг активаций [3]

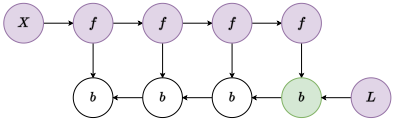


Figure 1: Стандартный backprop: все активации хранятся

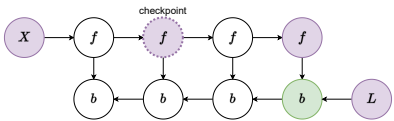


Figure 2: Чекпоинтинг: храним только отмеченные

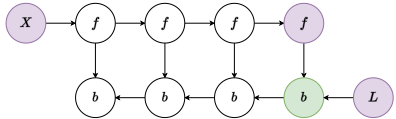


Figure 3: Минимум памяти: перевычисляем всё

- **Стандартный backprop**: хранит все промежуточные активации — $O(L)$ памяти для L слоёв

Чекпоинтинг активаций [3]

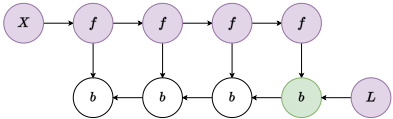


Figure 1: Стандартный backprop: все активации хранятся

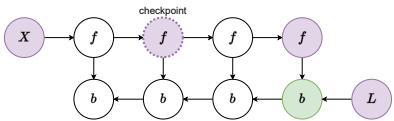


Figure 2: Чекпоинтинг: храним только отмеченные

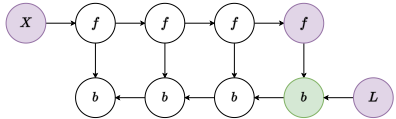


Figure 3: Минимум памяти: перевычисляем всё

- **Стандартный backprop**: хранит все промежуточные активации — $O(L)$ памяти для L слоёв
- **Чекпоинтинг**: сохраняет активации только в $\sqrt{L}$ «контрольных точках», остальные перевычисляет при backward — $O(\sqrt{L})$ памяти, $\sim 33\%$ вычислительный overhead

Чекпоинтинг активаций [3]

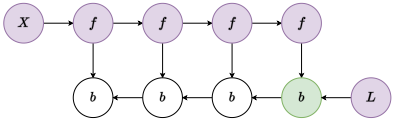


Figure 1: Стандартный backprop: все активации хранятся

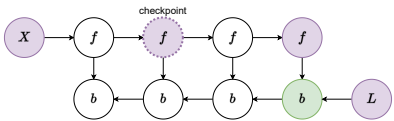


Figure 2: Чекпоинтинг: храним только отмеченные

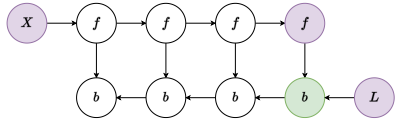


Figure 3: Минимум памяти: перевычисляем всё

- **Стандартный backprop**: хранит все промежуточные активации — $O(L)$ памяти для L слоёв
- **Чекпоинтинг**: сохраняет активации только в $\sqrt{L}$ «контрольных точках», остальные перевычисляет при backward — $O(\sqrt{L})$ памяти, $\sim 33\%$ вычислительный overhead
- **Минимальная память**: хранит только вход — $O(1)$ памяти, но $O(L)$ перевычислений

Чекпоинтинг активаций [3]

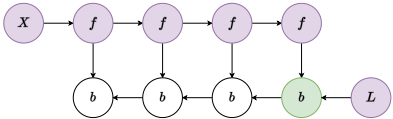


Figure 1: Стандартный backprop: все активации хранятся

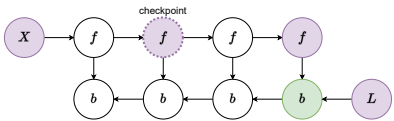


Figure 2: Чекпоинтинг: храним только отмеченные

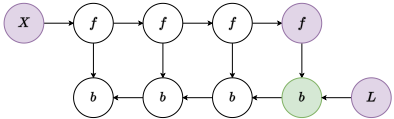


Figure 3: Минимум памяти: перевычисляем всё

- **Стандартный backprop**: хранит все промежуточные активации — $O(L)$ памяти для L слоёв
- **Чекпоинтинг**: сохраняет активации только в $\sqrt{L}$ «контрольных точках», остальные перевычисляет при backward — $O(\sqrt{L})$ памяти, $\sim 33\%$ вычислительный overhead
- **Минимальная память**: хранит только вход — $O(1)$ памяти, но $O(L)$ перевычислений

[3] Training Deep Nets with Sublinear Memory Cost

Чекпоинтинг активаций [3]

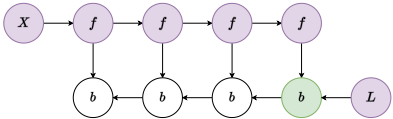


Figure 1: Стандартный backprop: все активации хранятся

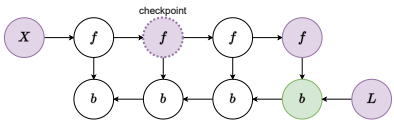


Figure 2: Чекпоинтинг: храним только отмеченные

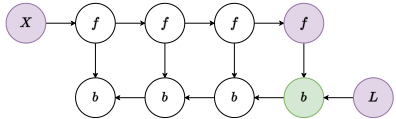


Figure 3: Минимум памяти: перевычисляем всё

- **Стандартный backprop**: хранит все промежуточные активации — $O(L)$ памяти для L слоёв
- **Чекпоинтинг**: сохраняет активации только в $\sqrt{L}$ «контрольных точках», остальные перевычисляет при backward — $O(\sqrt{L})$ памяти, $\sim 33\%$ вычислительный overhead
- **Минимальная память**: хранит только вход — $O(1)$ памяти, но $O(L)$ перевычислений

В PyTorch:

```
from torch.utils.checkpoint import checkpoint
output = checkpoint(layer, input, use_reentrant=False)
```

[3] Training Deep Nets with Sublinear Memory Cost

Scaling Laws

Scaling Laws [4]

- **Эмпирическое правило**: кросс-энтропия уменьшается по степенному закону

$$L(N, D, C) \propto N^{-\alpha} D^{-\beta} C^{-\gamma}$$

где N — параметры, D — токены, C — FLOPs.

Scaling Laws [4]

- **Эмпирическое правило**: кросс-энтропия уменьшается по степенному закону

$$L(N, D, C) \propto N^{-\alpha} D^{-\beta} C^{-\gamma}$$

где N — параметры, D — токены, C — FLOPs.

Scaling Laws [4]

- **Эмпирическое правило**: кросс-энтропия уменьшается по степенному закону

$$L(N, D, C) \propto N^{-\alpha} D^{-\beta} C^{-\gamma}$$

где N — параметры, D — токены, C — FLOPs.

- **Compute allocation**: при фиксированном C оптимально $N \propto D^{0.74}$ — крупнее модель, меньше данных.

Scaling Laws [4]

- **Эмпирическое правило**: кросс-энтропия уменьшается по степенному закону

$$L(N, D, C) \propto N^{-\alpha} D^{-\beta} C^{-\gamma}$$

где N — параметры, D — токены, C — FLOPs.

- **Compute allocation**: при фиксированном C оптимально $N \propto D^{0.74}$ — крупнее модель, меньше данных.
- **Предсказание качества**: линейность на $\log$—$\log$-графике сохраняется вплоть до GPT-3-scale.

Scaling Laws [4]

- **Эмпирическое правило**: кросс-энтропия уменьшается по степенному закону

$$L(N, D, C) \propto N^{-\alpha} D^{-\beta} C^{-\gamma}$$

где N — параметры, D — токены, C — FLOPs.

- **Compute allocation**: при фиксированном C оптимально $N \propto D^{0.74}$ — крупнее модель, меньше данных.
- **Предсказание качества**: линейность на $\log$—$\log$-графике сохраняется вплоть до GPT-3-scale.
- Практически scaling-законы помогают подбирать размеры корпуса и останавливать обучение до переобучения.

[4] Scaling Laws for Neural Language Models, Kaplan et al., 2020

Chinchilla [6]

- DeepMind обучили **Chinchilla 70 B** на **1.4 T** токенов при том же compute, что и Gopher 280 B.

Chinchilla [6]

- DeepMind обучили **Chinchilla 70 B** на **1.4 T** токенов при том же compute, что и Gopher 280 B.
- **Результат**: +7 pp на MMLU и существенный прирост на BIG-bench vs GPT-3.

Chinchilla [6]

- DeepMind обучили **Chinchilla 70 B** на **1.4 T** токенов при том же compute, что и Gopher 280 B.
- **Результат**: +7 pp на MMLU и существенный прирост на BIG-bench vs GPT-3.
- **Compute-optimal scaling**: при ограниченных FLOPs соотношение «токенов-на-параметр»

$$\frac{D}{N} \approx 20$$

обеспечивает максимум качества.

Chinchilla [6]

- DeepMind обучили **Chinchilla 70 B** на **1.4 T** токенов при том же compute, что и Gopher 280 B.
- **Результат**: +7 pp на MMLU и существенный прирост на BIG-bench vs GPT-3.
- **Compute-optimal scaling**: при ограниченных FLOPs соотношение «токенов-на-параметр»

$$\frac{D}{N} \approx 20$$

обеспечивает максимум качества.

Chinchilla [6]

- DeepMind обучили **Chinchilla 70 B** на **1.4 T** токенов при том же compute, что и Gopher 280 B.
- **Результат**: +7 pp на MMLU и существенный прирост на BIG-bench vs GPT-3.
- **Compute-optimal scaling**: при ограниченных FLOPs соотношение «токенов-на-параметр»

$$\frac{D}{N} \approx 20$$

обеспечивает максимум качества.

- **Вывод Chinchilla**: при фиксированном compute параметры и данные надо растить **поровну** ($N \propto \sqrt{C}$, $D \propto \sqrt{C}$) $\rightarrow$ многие большие модели до 2022 (GPT-3 175B, Gopher 280B, MT-NLG 530B) оказались **сильно недообучены**: слишком велики для своего объёма данных.

Chinchilla [6]

- DeepMind обучили **Chinchilla 70 B** на **1.4 T** токенов при том же compute, что и Gopher 280 B.
- **Результат**: +7 pp на MMLU и существенный прирост на BIG-bench vs GPT-3.
- **Compute-optimal scaling**: при ограниченных FLOPs соотношение «токенов-на-параметр»

$$\frac{D}{N} \approx 20$$

обеспечивает максимум качества.

- **Вывод Chinchilla**: при фиксированном compute параметры и данные надо растить **поровну** ($N \propto \sqrt{C}$, $D \propto \sqrt{C}$) $\rightarrow$ многие большие модели до 2022 (GPT-3 175B, Gopher 280B, MT-NLG 530B) оказались **сильно недообучены**: слишком велики для своего объёма данных.
- Современные LLM (LLaMA, Qwen) идут ещё дальше — $D/N \gg 20$ (LLaMA-3 8B: ~ 1900 токенов/параметр), но уже из **inference-оптимальности** (меньше модель $\rightarrow$ дешевле инференс), а не compute-оптимальности [5].

[5] Beyond Chinchilla-Optimal: Accounting for Inference, Sardana et al., 2023
[6] Training Compute-Optimal Large Language Models, Hoffmann et al., 2022

Chinchilla scaling laws

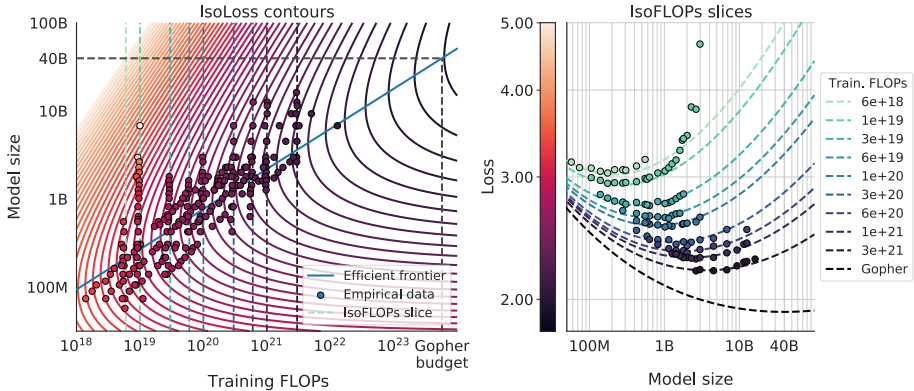


Figure 4: Параметрическое моделирование лосса $L(N, D)$: контурный график (слева) и isoFLOP-срезы (справа). Каждый isoFLOP-срез соответствует пунктирной линии слева. Эффективная граница показана синим — линия в log-log пространстве. Оптимальный размер модели для FLOP-бюджета Gopher проецируется в 40B параметров.

Kaplan vs Chinchilla: точные законы

Kaplan (2020): лосс — степень по числу параметров

$$L(N) = (N_c/N)^{\alpha_N}, \quad \alpha_N \approx 0.076$$

Распределение компьюта: $N \propto C^{0.73}$, $D \propto C^{0.27}$ — *«модель важнее данных»*.

Chinchilla (2022): параметрический фит на > 400 моделях

$$L(N, D) = 1.69 + \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.28}}$$

Распределение: $N \propto C^{0.50}$, $D \propto C^{0.50}$ — поровну, ≈ 20 токенов/параметр.

Kaplan vs Chinchilla: точные законы

Kaplan (2020): лосс — степень по числу параметров

$$L(N) = (N_c/N)^{\alpha_N}, \quad \alpha_N \approx 0.076$$

Распределение компьюта: $N \propto C^{0.73}$, $D \propto C^{0.27}$ — *«модель важнее данных»*.

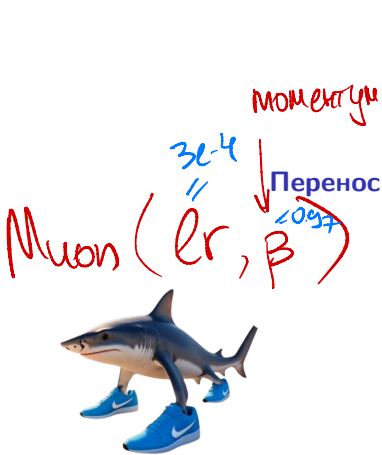
Chinchilla (2022): параметрический фит на > 400 моделях

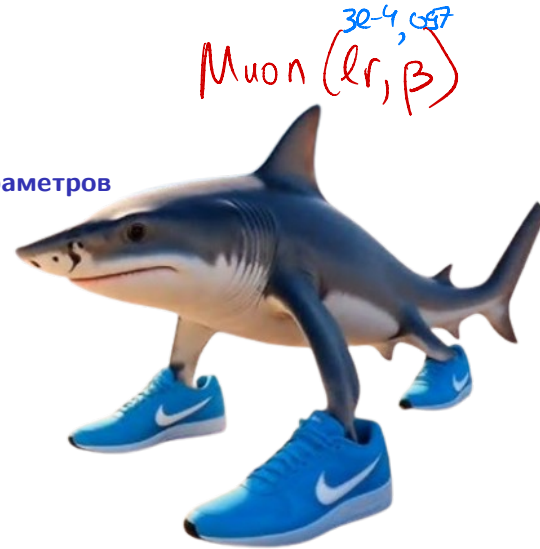
$$L(N, D) = 1.69 + \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.28}}$$

Распределение: $N \propto C^{0.50}$, $D \propto C^{0.50}$ — поровну, ≈ 20 токенов/параметр.

- **Почему законы разошлись?** Kaplan использовал фиксированное (не подстроенное под длину рана) lr-расписание и считал N *с эмбеддингами*. Chinchilla это исправил → другая оптимальная аллокация.

Перенос гиперпараметров





Проблема: оптимальный LR дрейфует с размером

- В **стандартной параметризации (SP)** оптимальный learning rate **смещается** при росте ширины сети: хорош для 100M $\rightarrow$ расходится/недоучивает 10B.

Проблема: оптимальный LR дрейфует с размером

- В **стандартной параметризации (SP)** оптимальный learning rate **смещается** при росте ширины сети: хорош для 100M → расходится/недоучивает 10B.
- Тюнить LR полным перебором на модели в миллиарды параметров — **запретительно дорого**.

Проблема: оптимальный LR дрейфует с размером

- В **стандартной параметризации (SP)** оптимальный learning rate **смещается** при росте ширины сети: хорош для 100M → расходится/недоучивает 10B.
- Тюнить LR полным перебором на модели в миллиарды параметров — **запретительно дорого**.

Проблема: оптимальный LR дрейфует с размером

- В **стандартной параметризации (SP)** оптимальный learning rate **смещается** при росте ширины сети: хорош для 100M → расходится/недоучивает 10B.
- Тюнить LR полным перебором на модели в миллиарды параметров — **запретительно дорого**.

Идея: настроить гиперпараметры на *маленькой* прокси-модели и **предсказать** их для большой. Два подхода:

1. **Перенос параметризацией** (μP / muTransfer) — сделать оптимальный LR инвариантным к размеру.

Проблема: оптимальный LR дрейфует с размером

- В **стандартной параметризации (SP)** оптимальный learning rate **смещается** при росте ширины сети: хорош для 100M → расходится/недоучивает 10B.
- Тюнить LR полным перебором на модели в миллиарды параметров — **запретительно дорого**.

Идея: настроить гиперпараметры на *маленькой* прокси-модели и **предсказать** их для большой. Два подхода:

1. **Перенос параметризацией** (μP / muTransfer) — сделать оптимальный LR инвариантным к размеру.
2. **Гиперпараметрические scaling laws** — фитнуть HP как явную степенную функцию от компьюта/размера.

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$

Рецепт muTransfer:

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$
- множитель readout-слоя $\propto 1/n$

Рецепт muTransfer:

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$
- множитель readout-слоя $\propto 1/n$
- LR для Adam на матричных слоях $\propto 1/n$

Рецепт muTransfer:

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$
- множитель readout-слоя $\propto 1/n$
- LR для Adam на матричных слоях $\propto 1/n$

Рецепт muTransfer:

1. параметризуем большую модель в µP;

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$
- множитель readout-слоя $\propto 1/n$
- LR для Adam на матричных слоях $\propto 1/n$

Рецепт muTransfer:

1. параметризуем большую модель в µP;
2. тюним HP на **маленькой** модели;

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$
- множитель readout-слоя $\propto 1/n$
- LR для Adam на матричных слоях $\propto 1/n$

Рецепт muTransfer:

1. параметризуем большую модель в µP;
2. тюним HP на **маленькой** модели;
3. **zero-shot** переносим на большую — её не тюним вовсе.

В идеале

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$
- множитель readout-слоя $\propto 1/n$
- LR для Adam на матричных слоях $\propto 1/n$

Рецепт muTransfer:

1. параметризуем большую модель в µP;
2. тюним HP на **маленькой** модели;
3. **zero-shot** переносим на большую — её не тюним вовсе.

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$
- множитель readout-слоя $\propto 1/n$
- LR для Adam на матричных слоях $\propto 1/n$

Рецепт muTransfer:

1. параметризуем большую модель в µP;
2. тюним HP на **маленькой** модели;
3. **zero-shot** переносим на большую — её не тюним вовсе.

- Маркеры: перенос с **40M → GPT-3 6.7B**, стоимость тюнинга **всего 7%** претрейна; **13M → BERT-large**.

```
pip install mup
```

µP / muTransfer [8]

Maximal Update Parametrization (µP): масштабируем инициализацию, LR и множители слоёв так, чтобы при ширине $n \to \infty$ каждый слой получал $O(1)$ feature-update. Следствие (доказано через Tensor Programs): **оптимальный LR почти инвариантен к ширине**.

Правила масштабирования по ширине n:

- init скрытых весов $\propto 1/\text{fan\_in}$
- множитель readout-слоя $\propto 1/n$
- LR для Adam на матричных слоях $\propto 1/n$

Рецепт muTransfer:

1. параметризуем большую модель в µP;
2. тюним HP на **маленькой** модели;
3. **zero-shot** переносим на большую — её не тюним вовсе.

- Маркеры: перенос с **40M → GPT-3 6.7B**, стоимость тюнинга **всего 7%** претрейна; **13M → BERT-large**. `pip install mup`
- Развитие **u-µP** [7]: transfer error $0.03 \to 0.005$, LR константен по ширине **и по глубине**, FP8-обучение «из коробки».

[7] u-µP: Unit-Scaled Maximal Update Parametrization, Blake et al., 2024

[8] Tensor Programs V: Tuning Large NNs via Zero-Shot HP Transfer, Yang & Hu et al., 2022

Гиперпараметрические scaling laws: формула для LR и batch

Гиперпараметрические scaling laws (DeepSeek LLM, 2401.02954): LR падает, batch растёт с компьютом

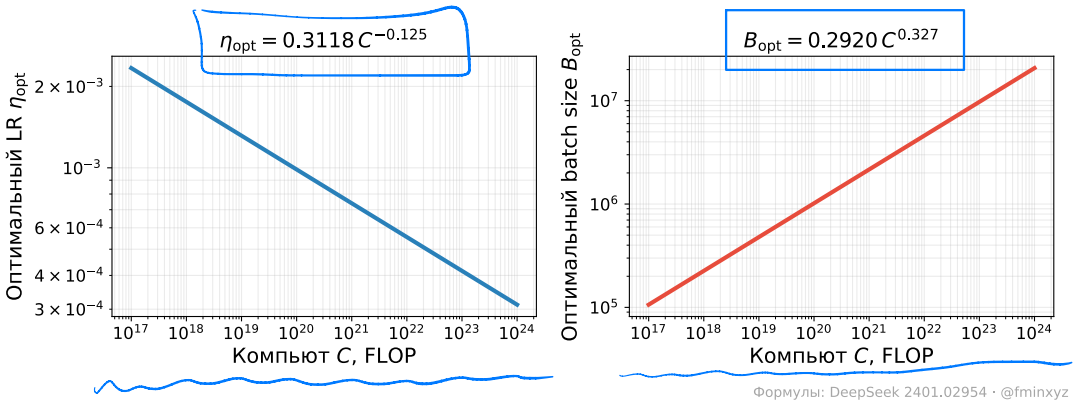


Figure 5: Опубликованные степенные законы DeepSeek LLM (2401.02954), фитнутые на реальных прогонах: оптимальный LR убывает, оптимальный batch растёт с компьютом.

Гиперпараметрические scaling laws: формула для LR и batch

- **DeepSeek LLM** [9]:

$$\eta_{\text{opt}} = 0.3118 \cdot C^{-0.125}, \qquad B_{\text{opt}} = 0.2920 \cdot C^{0.327}$$

[9] DeepSeek LLM: Scaling Open-Source Language Models, 2024

Гиперпараметрические scaling laws: формула для LR и batch

- **DeepSeek LLM** [9]:

$$\eta_{\text{opt}} = 0.3118 \cdot C^{-0.125}, \qquad B_{\text{opt}} = 0.2920 \cdot C^{0.327}$$

- **Step Law** [10]:

$$\eta(N, D) = 1.79 \cdot N^{-0.713} D^{0.307}, \qquad B(D) = 0.58 \cdot D^{0.571}$$

batch зависит в основном от D, LR — и от N, и от D.

[9] DeepSeek LLM: Scaling Open-Source Language Models, 2024
[10] Optimal Hyperparameter Scaling Law in LLM Pre-training, 2025

Гиперпараметрические scaling laws: формула для LR и batch

- **DeepSeek LLM** [9]:

$$\eta_{\text{opt}} = 0.3118 \cdot C^{-0.125}, \qquad B_{\text{opt}} = 0.2920 \cdot C^{0.327}$$

- **Step Law** [10]:

$$\eta(N, D) = 1.79 \cdot N^{-0.713} D^{0.307}, \qquad B(D) = 0.58 \cdot D^{0.571}$$

batch зависит в основном от D, LR — и от N, и от D.

[9] DeepSeek LLM: Scaling Open-Source Language Models, 2024
[10] Optimal Hyperparameter Scaling Law in LLM Pre-training, 2025

Гиперпараметрические scaling laws: формула для LR и batch

- **DeepSeek LLM** [9]:
$$\eta_{\text{opt}} = 0.3118 \cdot C^{-0.125}, \qquad B_{\text{opt}} = 0.2920 \cdot C^{0.327}$$
- **Step Law** [10]:
$$\eta(N, D) = 1.79 \cdot N^{-0.713} D^{0.307}, \qquad B(D) = 0.58 \cdot D^{0.571}$$
batch зависит в основном от D, LR — и от N, и от D.
- Фундамент для batch — **gradient noise scale** [11]: $B_{\text{simple}} = \operatorname{tr}(\Sigma)/\|G\|^2$ (растёт по ходу обучения $\rightarrow$ batch можно наращивать).

[9] DeepSeek LLM: Scaling Open-Source Language Models, 2024
[10] Optimal Hyperparameter Scaling Law in LLM Pre-training, 2025
[11] An Empirical Model of Large-Batch Training, McCandlish et al., 2018

Когда классические законы ломаются

- **Ограниченные данные** [12]: повтор данных до ~ 4 **эпох** $\approx$ как свежие; после ~ 16 — резкий спад.

[12] Scaling Data-Constrained Language Models, Muennighoff et al., 2023

Когда классические законы ломаются

- **Ограниченные данные** [12]: повтор данных до ~ 4 **эпох** $\approx$ как свежие; после ~ 16 — резкий спад.
- **Учёт инференса** [13]: при большом спросе на инференс выгодно брать **меньшую модель и учить дольше** (далеко за 20 токенов/параметр) — объясняет стратегию LLaMA/Mistral.

[12] Scaling Data-Constrained Language Models, Muennighoff et al., 2023
[13] Beyond Chinchilla-Optimal: Accounting for Inference, Sardana et al., 2023

Когда классические законы ломаются

- **Ограниченные данные** [12]: повтор данных до ~ 4 **эпох** $\approx$ как свежие; после ~ 16 — резкий спад.
- **Учёт инференса** [13]: при большом спросе на инференс выгодно брать **меньшую модель и учить дольше** (далеко за 20 токенов/параметр) — объясняет стратегию LLaMA/Mistral.
- **Точность** [14]: $N_{\text{eff}} = N(1 - e^{-P/\gamma})$; compute-optimal точность $\approx$ **7–8 бит**, дефолтные 16 могут быть субоптимальны.

[12] Scaling Data-Constrained Language Models, Muennighoff et al., 2023
[13] Beyond Chinchilla-Optimal: Accounting for Inference, Sardana et al., 2023
[14] Scaling Laws for Precision, Kumar et al., 2024

Практический рецепт обучения большой модели

1. **Параметризуй в μP/u-μP** → перенеси LR с маленькой прокси-модели.

Практический рецепт обучения большой модели

1. **Параметризуй в μP/u-μP** → перенеси LR с маленькой прокси-модели.
2. **Фитни HP-scaling law** (DeepSeek / Step Law) на дешёвых малых ранах → экстраполируй η, B на целевой масштаб.

Практический рецепт обучения большой модели

1. **Параметризуй в μP/u-μP** $\rightarrow$ перенеси LR с маленькой прокси-модели.
2. **Фитни HP-scaling law** (DeepSeek / Step Law) на дешёвых малых ранах $\rightarrow$ экстраполируй η, B на целевой масштаб.
3. **Выбери** N, D **по Chinchilla** (≈ 20 токенов/параметр) или сдвинь под inference-бюджет.

Практический рецепт обучения большой модели

1. **Параметризуй в µP/u-µP** $\rightarrow$ перенеси LR с маленькой прокси-модели.
2. **Фитни HP-scaling law** (DeepSeek / Step Law) на дешёвых малых ранах $\rightarrow$ экстраполируй η, B на целевой масштаб.
3. **Выбери** N, D **по Chinchilla** (≈ 20 токенов/параметр) или сдвинь под inference-бюджет.
4. **WSD/cooldown расписание** $\rightarrow$ снимай точки scaling law вдоль *одного* рана, не переобучая с нуля.

Практический рецепт обучения большой модели

1. **Параметризуй в µP/u-µP** $\rightarrow$ перенеси LR с маленькой прокси-модели.
2. **Фитни HP-scaling law** (DeepSeek / Step Law) на дешёвых малых ранах $\rightarrow$ экстраполируй η, B на целевой масштаб.
3. **Выбери** N, D **по Chinchilla** (≈ 20 токенов/параметр) или сдвинь под inference-бюджет.
4. **WSD/cooldown расписание** $\rightarrow$ снимай точки scaling law вдоль *одного* рана, не переобучая с нуля.
5. **Учти precision** (7–8 бит) и лимит уникальных данных.

Практический рецепт обучения большой модели

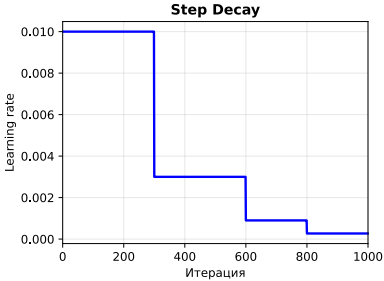
1. **Параметризуй в µP/u-µP** $\rightarrow$ перенеси LR с маленькой прокси-модели.
2. **Фитни HP-scaling law** (DeepSeek / Step Law) на дешёвых малых ранах $\rightarrow$ экстраполируй η, B на целевой масштаб.
3. **Выбери** N, D **по Chinchilla** (≈ 20 токенов/параметр) или сдвинь под inference-бюджет.
4. **WSD/cooldown расписание** $\rightarrow$ снимай точки scaling law вдоль *одного* рана, не переобучая с нуля.
5. **Учти precision** (7–8 бит) и лимит уникальных данных.

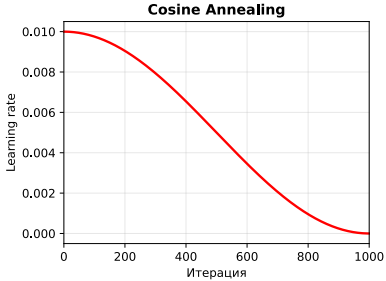
Практический рецепт обучения большой модели

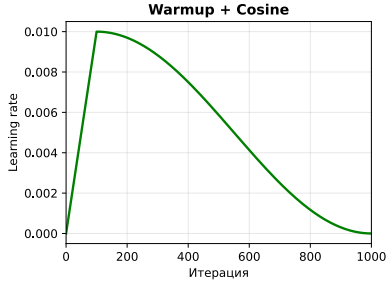
1. **Параметризуй в µP/u-µP** $\rightarrow$ перенеси LR с маленькой прокси-модели.
2. **Фитни HP-scaling law** (DeepSeek / Step Law) на дешёвых малых ранах $\rightarrow$ экстраполируй η, B на целевой масштаб.
3. **Выбери** N, D **по Chinchilla** (≈ 20 токенов/параметр) или сдвинь под inference-бюджет.
4. **WSD/cooldown расписание** $\rightarrow$ снимай точки scaling law вдоль *одного* рана, не переобучая с нуля.
5. **Учти precision** (7–8 бит) и лимит уникальных данных.
   От «обучай вслепую и молись» $\rightarrow$ к «обучи прокси, измерь степенной закон, экстраполируй гиперпараметры и бюджет».

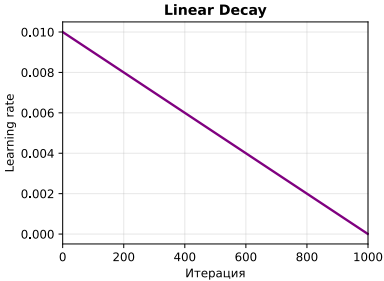
Расписания скорости обучения

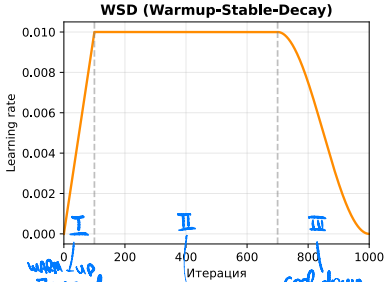
Расписания скорости обучения (Learning rate schedulers)

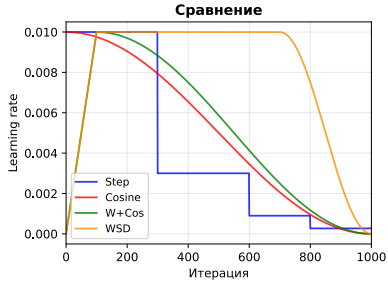












Gradual warmup [15]

Постепенный разогрев помогает избежать нестабильности при старте с большого learning rate — скорость обучения линейно растёт от малого значения до целевого:

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

где t — текущая итерация, T_w — длительность разогрева.

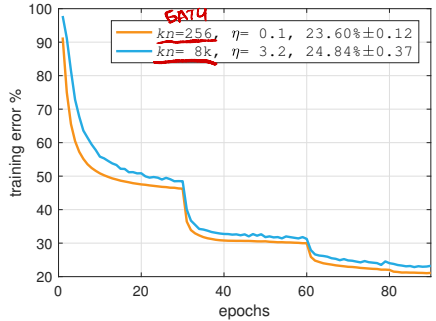


Figure 6: Без warmup

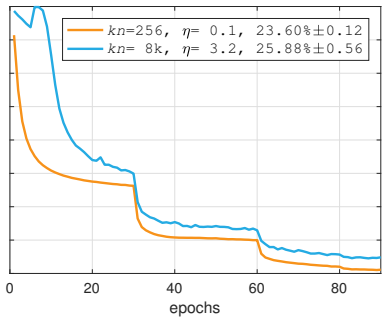


Figure 7: Постоянный warmup

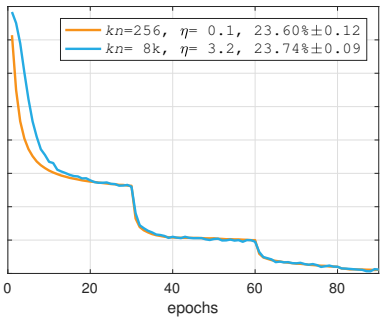


Figure 8: Постепенный warmup

[15] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Cosine Annealing [16]

$$\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min}) \left(1 + \cos\left(\frac{\pi t}{T}\right)\right)$$

- Плавное убывание от $\alpha_{\max}$ до $\alpha_{\min}$

Cosine Annealing [16]

$$\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min}) \left(1 + \cos\left(\frac{\pi t}{T}\right)\right)$$

- Плавное убывание от $\alpha_{\max}$ до $\alpha_{\min}$
- Без резких скачков — стабильнее step decay на практике

Cosine Annealing [16]

$$\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min}) \left(1 + \cos\left(\frac{\pi t}{T}\right)\right)$$

- Плавное убывание от $\alpha_{\max}$ до $\alpha_{\min}$
- Без резких скачков — стабильнее step decay на практике
- **Стандарт** для обучения трансформеров и vision моделей

Cosine Annealing [16]

$$\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min}) \left(1 + \cos\left(\frac{\pi t}{T}\right)\right)$$

- Плавное убывание от $\alpha_{\max}$ до $\alpha_{\min}$
- Без резких скачков — стабильнее step decay на практике
- **Стандарт** для обучения трансформеров и vision моделей

[16] SGDR: Stochastic Gradient Descent with Warm Restarts

Cosine Annealing [16]

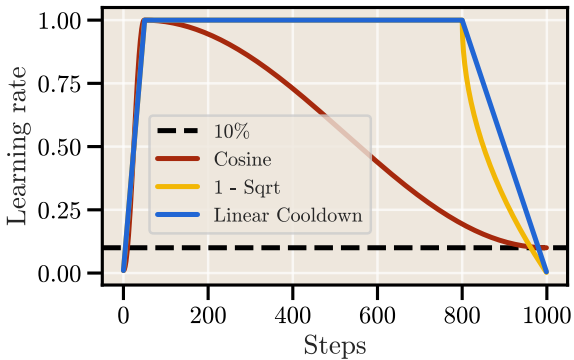
$$\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min})\left(1 + \cos\left(\frac{\pi t}{T}\right)\right)$$

- Плавное убывание от $\alpha_{\max}$ до $\alpha_{\min}$
- Без резких скачков — стабильнее step decay на практике
- **Стандарт** для обучения трансформеров и vision моделей

Вариант с warm restarts (SGDR): периодический сброс lr обратно к $\alpha_{\max}$ с удлиняющимися циклами.

[16] SGDR: Stochastic Gradient Descent with Warm Restarts

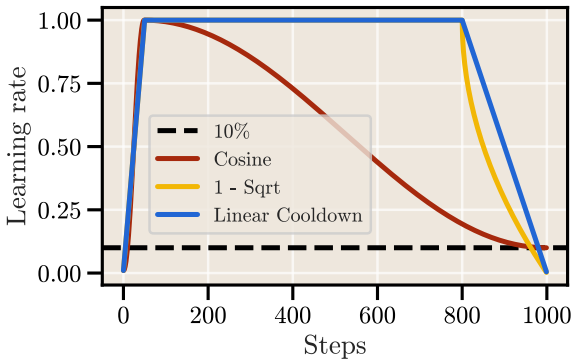
WSD (Warmup-Stable-Decay) [17]



Три фазы:

1. **Warmup**: линейный рост до $\alpha_{\max}$ за T_w шагов

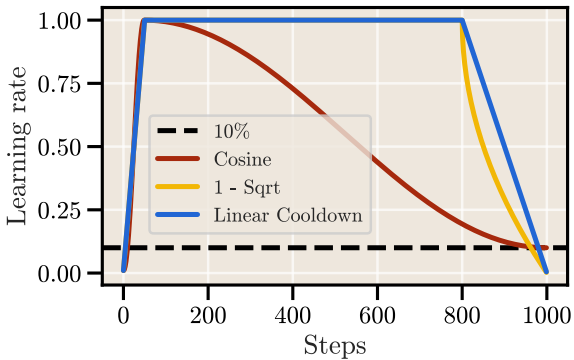
WSD (Warmup-Stable-Decay) [17]



Три фазы:

1. **Warmup**: линейный рост до $\alpha_{\max}$ за T_w шагов
2. **Stable**: постоянный lr $\alpha_{\max}$

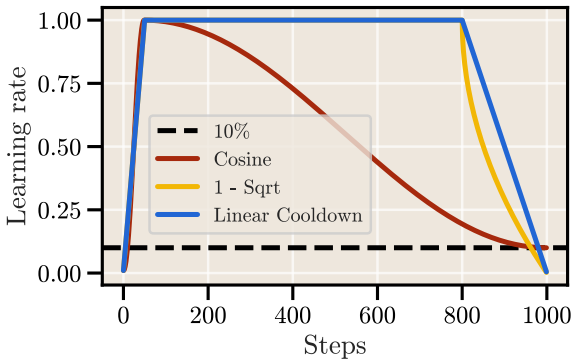
WSD (Warmup-Stable-Decay) [17]



Три фазы:

1. **Warmup**: линейный рост до $\alpha_{\max}$ за T_w шагов
2. **Stable**: постоянный lr $\alpha_{\max}$
3. **Decay**: cosine decay до $\alpha_{\min}$

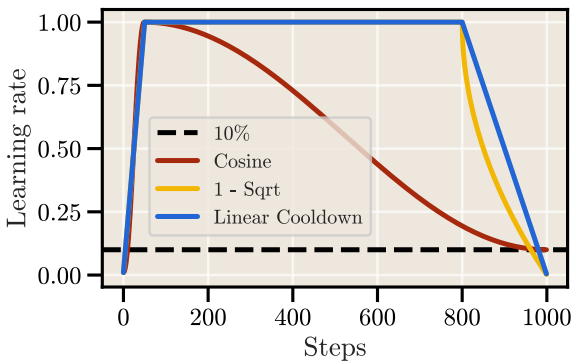
WSD (Warmup-Stable-Decay) [17]



Три фазы:

1. **Warmup**: линейный рост до $\alpha_{\max}$ за T_w шагов
2. **Stable**: постоянный lr $\alpha_{\max}$
3. **Decay**: cosine decay до $\alpha_{\min}$

WSD (Warmup-Stable-Decay) [17]

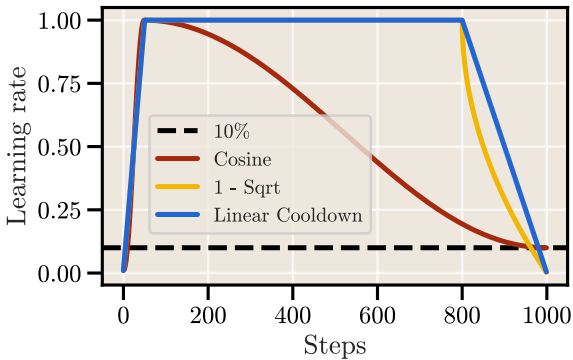


Три фазы:

1. **Warmup**: линейный рост до $\alpha_{\max}$ за T_w шагов
2. **Stable**: постоянный lr $\alpha_{\max}$
3. **Decay**: cosine decay до $\alpha_{\min}$

- Удобен, когда **бюджет обучения заранее не определён**: можно продолжать фазу stable и запускать decay когда готовы остановиться

WSD (Warmup-Stable-Decay) [17]



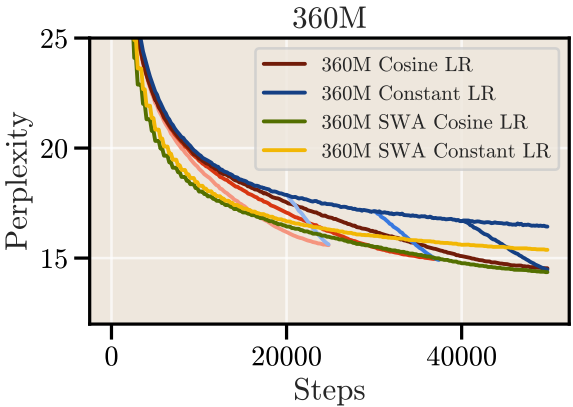
Три фазы:

1. **Warmup**: линейный рост до $\alpha_{\max}$ за T_w шагов
2. **Stable**: постоянный lr $\alpha_{\max}$
3. **Decay**: cosine decay до $\alpha_{\min}$

- Удобен, когда **бюджет обучения заранее не определён**: можно продолжать фазу stable и запускать decay когда готовы остановиться
- Используется в DeepSeek, OLMo и других проектах с variable training duration

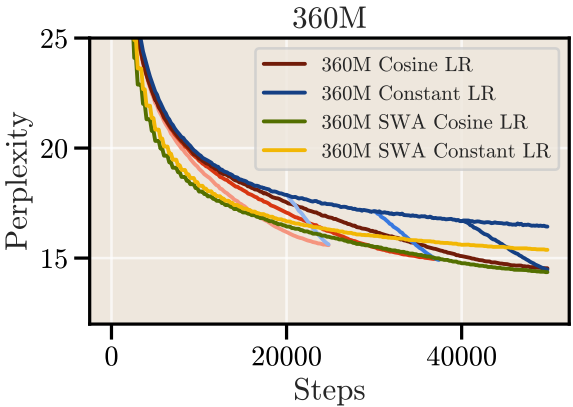
[17] Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations

Cooldown [18]



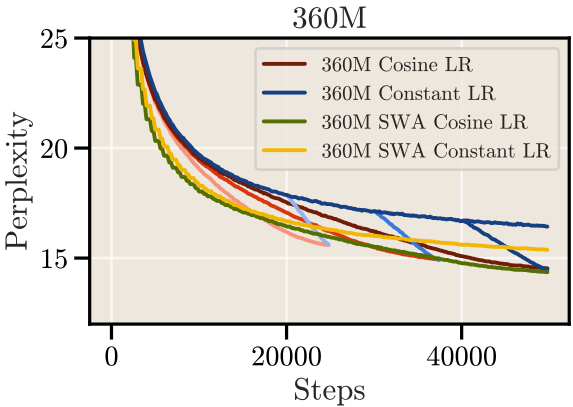
- **Cooldown** — финальная фаза убывания lr после основного обучения

Cooldown [18]



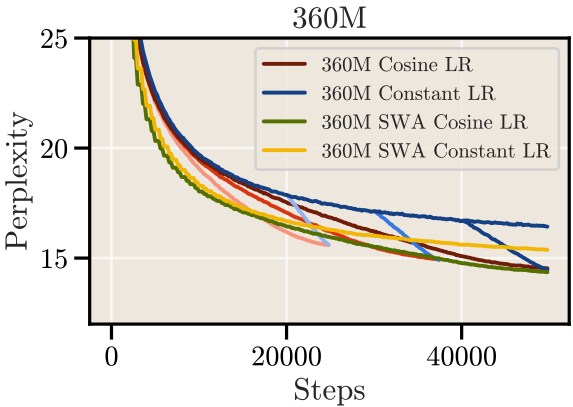
- **Cooldown** — финальная фаза убывания lr после основного обучения
- Можно обучать с высоким lr дольше, затем быстро «охладить»

Cooldown [18]



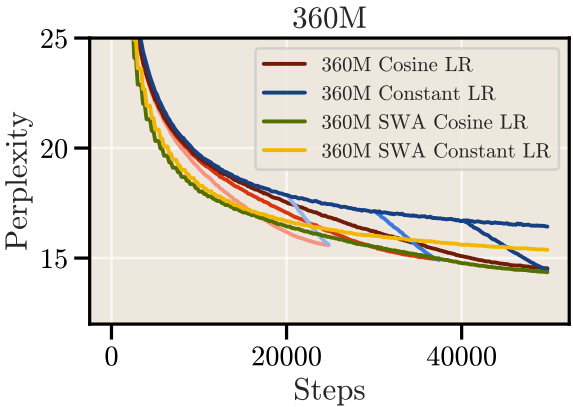
- **Cooldown** — финальная фаза убывания lr после основного обучения
- Можно обучать с высоким lr дольше, затем быстро «охладить»
- Позволяет **переиспользовать чекпоинты**: обучили модель → сохранили → можно продолжить обучение на новых данных, завершив cooldown позже

Cooldown [18]



- **Cooldown** — финальная фаза убывания lr после основного обучения
- Можно обучать с высоким lr дольше, затем быстро «охладить»
- Позволяет **переиспользовать чекпоинты**: обучили модель → сохранили → можно продолжить обучение на новых данных, завершив cooldown позже

Cooldown [18]



- **Cooldown** — финальная фаза убывания lr после основного обучения
- Можно обучать с высоким lr дольше, затем быстро «охладить»
- Позволяет **переиспользовать чекпоинты**: обучили модель → сохранили → можно продолжить обучение на новых данных, завершив cooldown позже
- Эмпирически: модели с cooldown достигают **лучшей** val perplexity, чем с cosine decay той же длительности

[18] Scaling Vision Transformers

размер батча ↑↑ в 100 раз

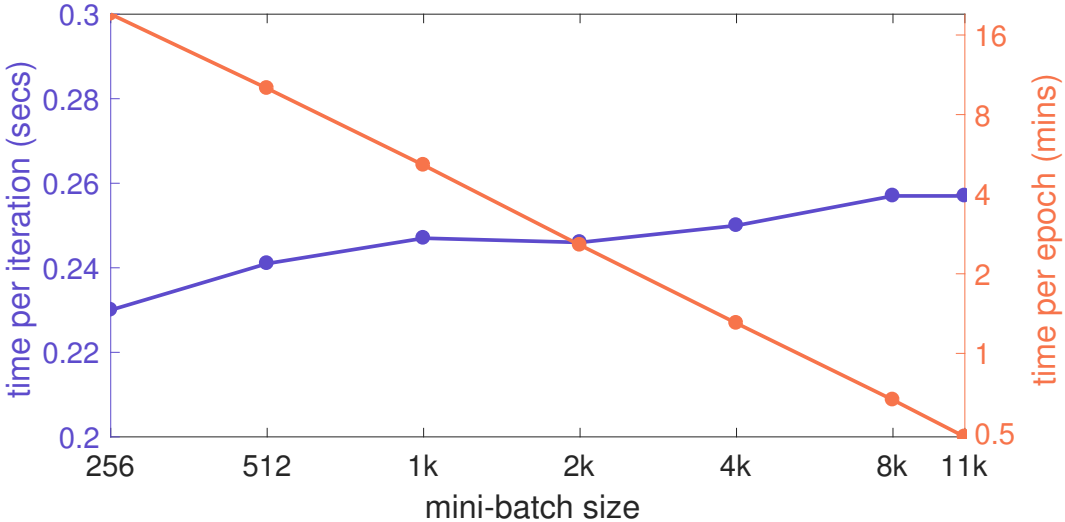
α ↑↓

Large batch training

$B = 1$ $\alpha = \alpha_0$ 100 шагов $\sim 100\alpha_0$

$B = 100$ $\alpha = \alpha_0$ 1 шаг $\sim 100 \cdot \alpha_0$

Large batch training [19]



[19] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Линейное и корневое правила масштабирования

При обучении с большими батчами learning rate нужно скорректировать. **Линейное правило масштабирования**[20]:

SGD →

$$\alpha_{\text{new}} = \alpha_{\text{base}} \cdot \frac{B_{\text{new}}}{B_{\text{base}}}$$

linear scaling rule

Правило корня[21]:

$$\alpha_{\text{new}} = \alpha_{\text{base}} \cdot \sqrt{\frac{B_{\text{new}}}{B_{\text{base}}}}$$

square root scaling rule

для адаптивных методов (Adam, RmsProp ...)

[20] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

[21] Learning Rates as a Function of Batch Size: A Random Matrix Theory Approach to Neural Network Training

Линейное и корневое правила масштабирования

При обучении с большими батчами learning rate нужно скорректировать. **Линейное правило масштабирования**[20]:

$$\alpha_{\text{new}} = \alpha_{\text{base}} \cdot \frac{B_{\text{new}}}{B_{\text{base}}}$$

Правило корня[21]:

$$\alpha_{\text{new}} = \alpha_{\text{base}} \cdot \sqrt{\frac{B_{\text{new}}}{B_{\text{base}}}}$$

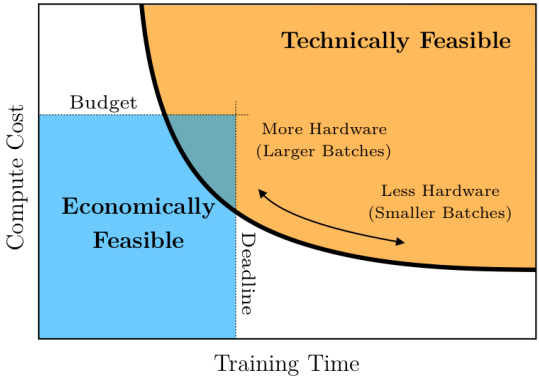
Effective batch size (kn)	α	top-1 error (%)
256	0.10	23.60 ± 0.12
8k	$0.10 \cdot 32$	**23.74 ± 0.09**
8k	0.10	41.67 ± 0.10
8k	$0.10 \cdot \sqrt{32}$	26.22 ± 0.03

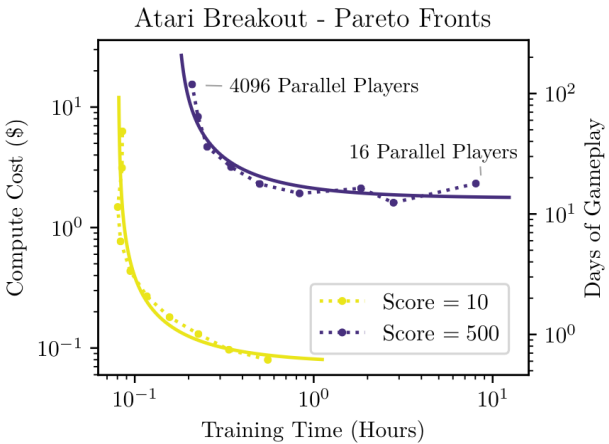
Линейное правило хорошо работает для SGD; для Adam авторы рекомендуют корневое.

[20] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

[21] Learning Rates as a Function of Batch Size: A Random Matrix Theory Approach to Neural Network Training

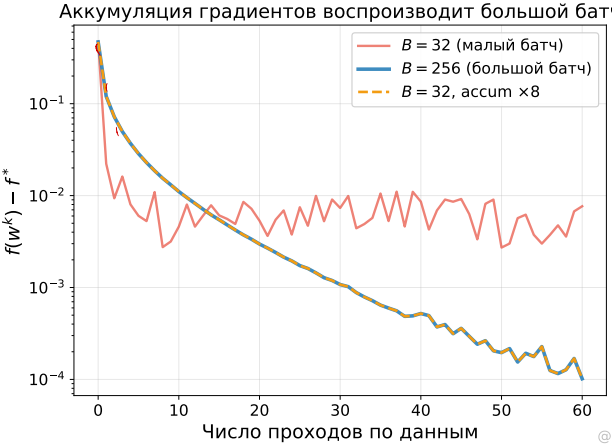
Масштабирование: критический размер батча [22]





[22] An Empirical Model of Large-Batch Training

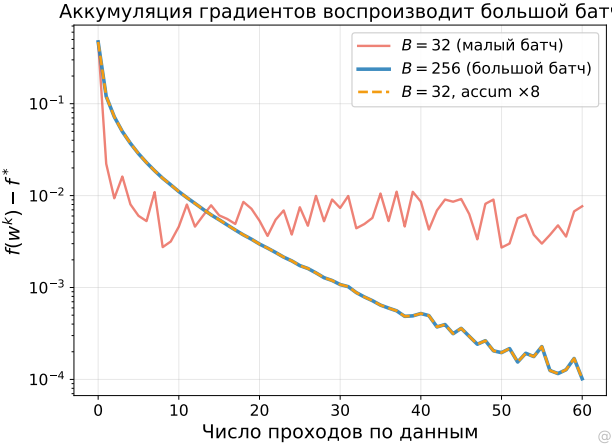
Аккумуляция градиентов = большой батч (численный эксперимент)



- **Аккумуляция** k **микро-батчей** размера b с корректной нормализацией loss даёт **тот же** шаг, что один батч kb.

Figure 9: Логистическая регрессия на реальных данных (рукописные цифры `sklearn`, чёт/нечет), одна инициализация. Кривая «$B{=}32$, accum$\times 8$» **точно** совпадает с «$B{=}256$» (расхождение $\sim 10^{-17}$) — аккумуляция воспроизводит большой батч. Маленький батч $B{=}32$ шумнее и застревает на полке $\sim 50\times$ выше.

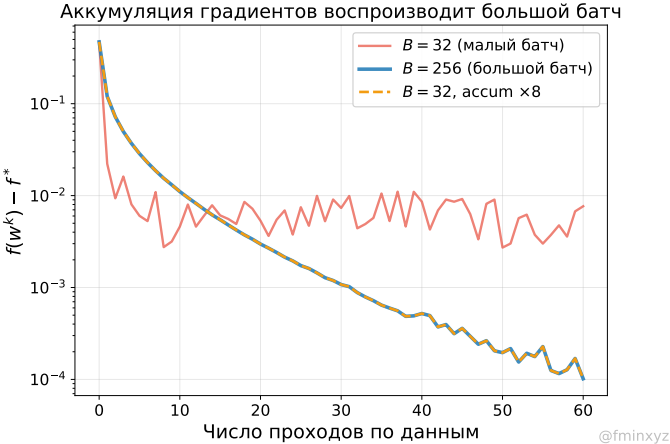
Аккумуляция градиентов = большой батч (численный эксперимент)



- **Аккумуляция** k **микро-батчей** размера b с корректной нормализацией loss даёт **тот же** шаг, что один батч kb.
- Поэтому кривые $B{=}256$ и $B{=}32{\times}8$ ложатся друг на друга с точностью до численной ошибки.

Figure 9: Логистическая регрессия на реальных данных (рукописные цифры `sklearn`, чёт/нечет), одна инициализация. Кривая «$B{=}32$, accum$\times 8$» **точно** совпадает с «$B{=}256$» (расхождение $\sim 10^{-17}$) — аккумуляция воспроизводит большой батч. Маленький батч $B{=}32$ шумнее и застревает на полке $\sim 50\times$ выше.

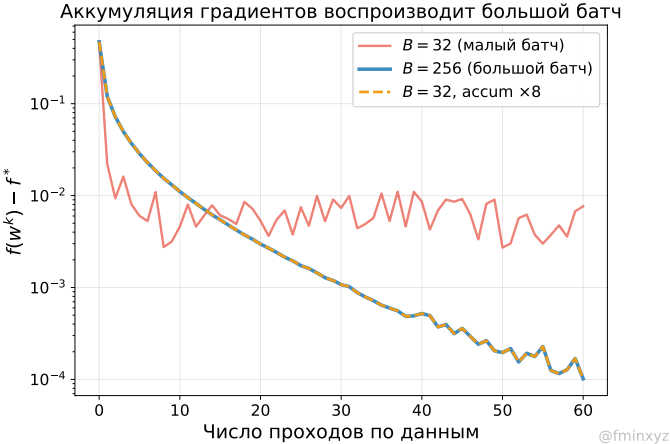
Аккумуляция градиентов = большой батч (численный эксперимент)



- **Аккумуляция k микро-батчей** размера b с корректной нормализацией loss даёт **тот же** шаг, что один батч kb.
- Поэтому кривые $B{=}256$ и $B{=}32{\times}8$ ложатся друг на друга с точностью до численной ошибки.
- Большой батч $\Rightarrow$ меньше шум градиента $\Rightarrow$ ниже «полка» (noise floor).

Figure 9: Логистическая регрессия на реальных данных (рукописные цифры `sklearn`, чёт/нечет), одна инициализация. Кривая «$B{=}32$, accum$\times 8$» **точно** совпадает с «$B{=}256$» (расхождение $\sim 10^{-17}$) — аккумуляция воспроизводит большой батч. Маленький батч $B{=}32$ шумнее и застревает на полке $\sim 50\times$ выше.

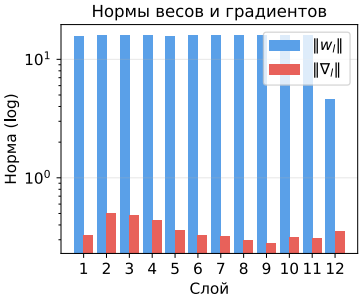
Аккумуляция градиентов = большой батч (численный эксперимент)

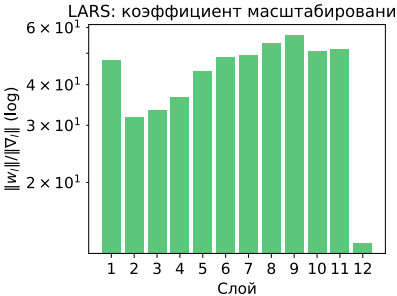


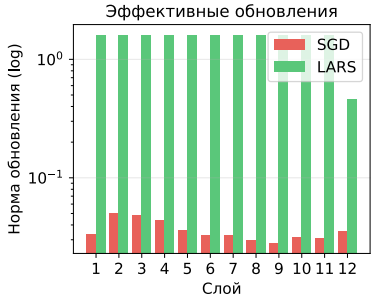
- **Аккумуляция k микро-батчей** размера b с корректной нормализацией loss даёт **тот же** шаг, что один батч kb.
- Поэтому кривые $B{=}256$ и $B{=}32{\times}8$ ложатся друг на друга с точностью до численной ошибки.
- Большой батч $\Rightarrow$ меньше шум градиента $\Rightarrow$ ниже «полка» (noise floor).
- Цена аккумуляции — лишние forward/backward, но **память не растёт**.

Figure 9: Логистическая регрессия на реальных данных (рукописные цифры `sklearn`, чёт/нечет), одна инициализация. Кривая «$B{=}32$, accum$\times 8$» **точно** совпадает с «$B{=}256$» (расхождение $\sim 10^{-17}$) — аккумуляция воспроизводит большой батч. Маленький батч $B{=}32$ шумнее и застревает на полке $\sim 50\times$ выше.

LARS (Layer-wise Adaptive Rate Scaling) [23]







[23] Large Batch Training of Convolutional Networks

LARS: алгоритм

Проблема: в глубоких сетях отношение $\|\nabla_l\|/\|w_l\|$ может различаться на порядки между слоями. Глобальный lr приводит к тому, что одни слои обновляются слишком сильно, другие — слишком слабо.

LARS: алгоритм

Проблема: в глубоких сетях отношение $\|\nabla_l\|/\|w_l\|$ может различаться на порядки между слоями. Глобальный lr приводит к тому, что одни слои обновляются слишком сильно, другие — слишком слабо.

Решение LARS: послойное масштабирование lr:

$$\Delta w_l = -\alpha \cdot \underbrace{\frac{\|w_l\|}{\|\nabla_l\| + \lambda\|w_l\|}}_{\text{local lr } \phi_l} \cdot (\nabla_l + \lambda w_l)$$

- Каждый слой получает свой эффективный lr, пропорциональный отношению норм

LARS: алгоритм

Проблема: в глубоких сетях отношение $\|\nabla_l\|/\|w_l\|$ может различаться на порядки между слоями. Глобальный lr приводит к тому, что одни слои обновляются слишком сильно, другие — слишком слабо.

Решение LARS: послойное масштабирование lr:

$$\Delta w_l = -\alpha \cdot \underbrace{\frac{\|w_l\|}{\|\nabla_l\| + \lambda\|w_l\|}}_{\text{local lr } \phi_l} \cdot (\nabla_l + \lambda w_l)$$

- Каждый слой получает свой эффективный lr, пропорциональный отношению норм
- Параметр λ — коэффициент weight decay

LARS: алгоритм

Проблема: в глубоких сетях отношение $\|\nabla_l\|/\|w_l\|$ может различаться на порядки между слоями. Глобальный lr приводит к тому, что одни слои обновляются слишком сильно, другие — слишком слабо.

Решение LARS: послойное масштабирование lr:

$$\Delta w_l = -\alpha \cdot \underbrace{\frac{\|w_l\|}{\|\nabla_l\| + \lambda\|w_l\|}}_{\text{local lr } \phi_l} \cdot (\nabla_l + \lambda w_l)$$

- Каждый слой получает свой эффективный lr, пропорциональный отношению норм
- Параметр λ — коэффициент weight decay
- Позволяет обучать ResNet-50 на ImageNet с **батчем до 32K** без потери качества (обычный SGD ломается при $B > 8\text{K}$)

LAMB (Layer-wise Adaptive Moments for Batch training) [24]

LAMB = LARS + Adam. Послойное масштабирование применяется к Adam-обновлению:

$$r_l = \frac{\hat{m}_l}{\sqrt{\hat{v}_l} + \varepsilon} + \lambda w_l$$

$$\Delta w_l = -\alpha \cdot \frac{\|w_l\|}{\|r_l\|} \cdot r_l$$

LAMB (Layer-wise Adaptive Moments for Batch training) [24]

LAMB = LARS + Adam. Послойное масштабирование применяется к Adam-обновлению:

$$r_l = \frac{\hat{m}_l}{\sqrt{\hat{v}_l} + \varepsilon} + \lambda w_l$$

$$\Delta w_l = -\alpha \cdot \frac{\|w_l\|}{\|r_l\|} \cdot r_l$$

- LARS — для SGD + моментум, LAMB — для Adam/AdamW

LAMB (Layer-wise Adaptive Moments for Batch training) [24]

LAMB = LARS + Adam. Послойное масштабирование применяется к Adam-обновлению:

$$r_l = \frac{\hat{m}_l}{\sqrt{\hat{v}_l} + \varepsilon} + \lambda w_l$$

$$\Delta w_l = -\alpha \cdot \frac{\|w_l\|}{\|r_l\|} \cdot r_l$$

- LARS — для SGD + моментум, LAMB — для Adam/AdamW
- LAMB позволяет обучать **BERT** с батчем до **64K** (vs 256 в оригинале) — обучение за **76 минут** вместо 3 дней

LAMB (Layer-wise Adaptive Moments for Batch training) [24]

LAMB = LARS + Adam. Послойное масштабирование применяется к Adam-обновлению:

$$r_l = \frac{\hat{m}_l}{\sqrt{\hat{v}_l} + \varepsilon} + \lambda w_l$$

$$\Delta w_l = -\alpha \cdot \frac{\|w_l\|}{\|r_l\|} \cdot r_l$$

- LARS — для SGD + моментум, LAMB — для Adam/AdamW
- LAMB позволяет обучать **BERT** с батчем до **64K** (vs 256 в оригинале) — обучение за **76 минут** вместо 3 дней
- Практически все крупномасштабные обучения LLM используют LARS/LAMB или их вариации

[24] Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes

Gradient accumulation (аккумуляция градиентов)

Позволяет увеличить эффективный размер батча без увеличения потребления памяти:

Без аккумуляции

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

Gradient accumulation (аккумуляция градиентов)

Позволяет увеличить эффективный размер батча без увеличения потребления памяти:

Без аккумуляции

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

С аккумуляцией

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    if (i+1) % accumulation_steps == 0:
        optimizer.step()
        optimizer.zero_grad()
```

Gradient accumulation (аккумуляция градиентов)

Позволяет увеличить эффективный размер батча без увеличения потребления памяти:

Без аккумуляции

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

С аккумуляцией

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    if (i+1) % accumulation_steps == 0:
        optimizer.step()
        optimizer.zero_grad()
```

- Эффективный батч $= B\times$ `accumulation_steps`

Gradient accumulation (аккумуляция градиентов)

Позволяет увеличить эффективный размер батча без увеличения потребления памяти:

Без аккумуляции

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

С аккумуляцией

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    if (i+1) % accumulation_steps == 0:
        optimizer.step()
        optimizer.zero_grad()
```

- Эффективный батч $= B\times$ `accumulation_steps`
- **Идентично** обучению с большим батчем (при корректной нормализации loss)

Gradient accumulation (аккумуляция градиентов)

Позволяет увеличить эффективный размер батча без увеличения потребления памяти:

Без аккумуляции

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

С аккумуляцией

```
for i, (inputs, targets) in enumerate(data):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    if (i+1) % accumulation_steps == 0:
        optimizer.step()
        optimizer.zero_grad()
```

- Эффективный батч = $B\times$ `accumulation_steps`
- **Идентично** обучению с большим батчем (при корректной нормализации loss)
- Единственный overhead: лишние forward pass (память не растёт)

MultiGPU training

Data Parallel training

1. Параметрический сервер копирует модель на каждое устройство

Data Parallel training

1. Параметрический сервер копирует модель на каждое устройство
2. Каждое устройство выполняет forward и backward pass на своей части данных

Data Parallel training

1. Параметрический сервер копирует модель на каждое устройство
2. Каждое устройство выполняет forward и backward pass на своей части данных
3. Параметрический сервер собирает градиенты

Data Parallel training

1. Параметрический сервер копирует модель на каждое устройство
2. Каждое устройство выполняет forward и backward pass на своей части данных
3. Параметрический сервер собирает градиенты
4. Параметрический сервер обновляет модель

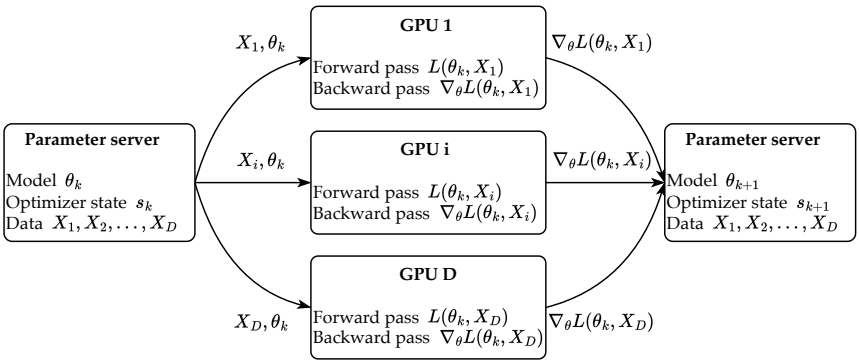
Data Parallel training

1. Параметрический сервер копирует модель на каждое устройство
2. Каждое устройство выполняет forward и backward pass на своей части данных
3. Параметрический сервер собирает градиенты
4. Параметрический сервер обновляет модель

Data Parallel training

1. Параметрический сервер копирует модель на каждое устройство
2. Каждое устройство выполняет forward и backward pass на своей части данных
3. Параметрический сервер собирает градиенты
4. Параметрический сервер обновляет модель

Батч на устройство: b. Общий батч: Db. Data parallelism подразумевает разделение данных между GPU, каждый с копией модели. Градиенты усредняются и веса обновляются синхронно:



Distributed Data Parallel training

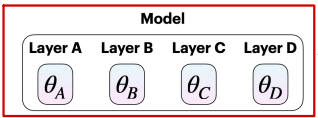
Distributed Data Parallel (DDP) [25] расширяет data parallelism на несколько узлов. Каждый узел вычисляет градиенты локально, затем синхронизирует с остальными. Это метод по умолчанию в Accelerate library.

DataParallel	DistributedDataParallel
Больше overhead; модель копируется на каждом forward pass	Модель копируется только один раз
Только single-node	Масштабируется на несколько машин
Медленнее; multithreading + GIL	Быстрее; multiprocessing, без GIL

[25] Getting Started with Distributed Data Parallel

Наивный параллелизм модели

Модель делится между GPU — каждый обрабатывает подмножество слоёв, снижая потребление памяти на каждом GPU. Позволяет работать с моделями, не помещающимися на одном GPU.



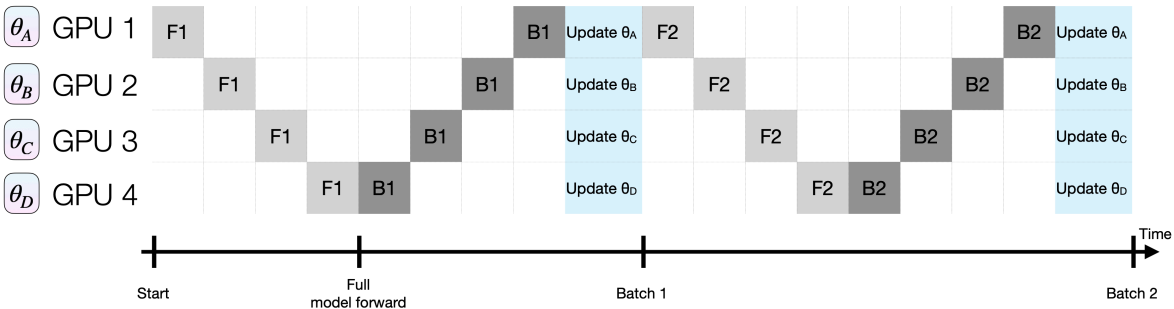
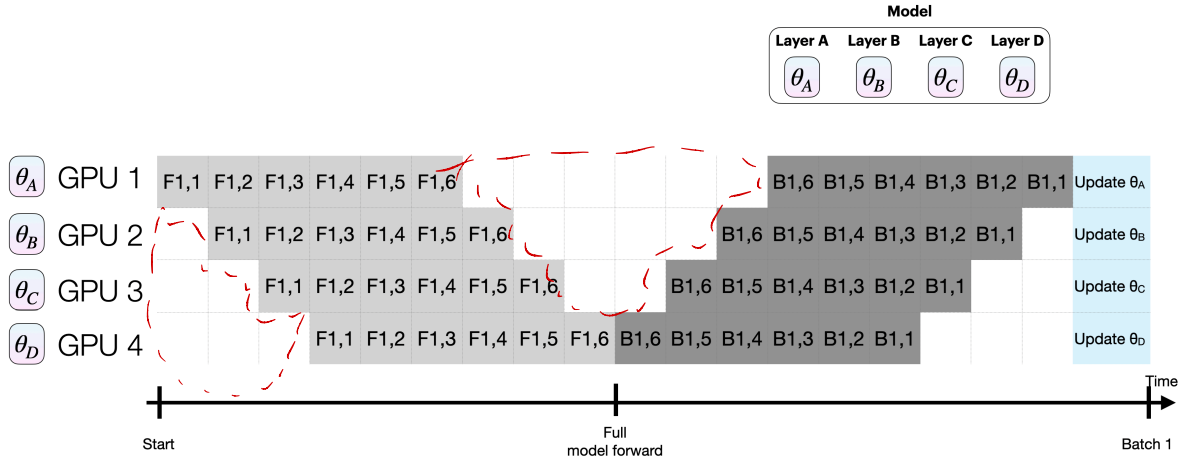


Figure 11: Параллелизм модели

Pipeline parallelism: GPipe [26]

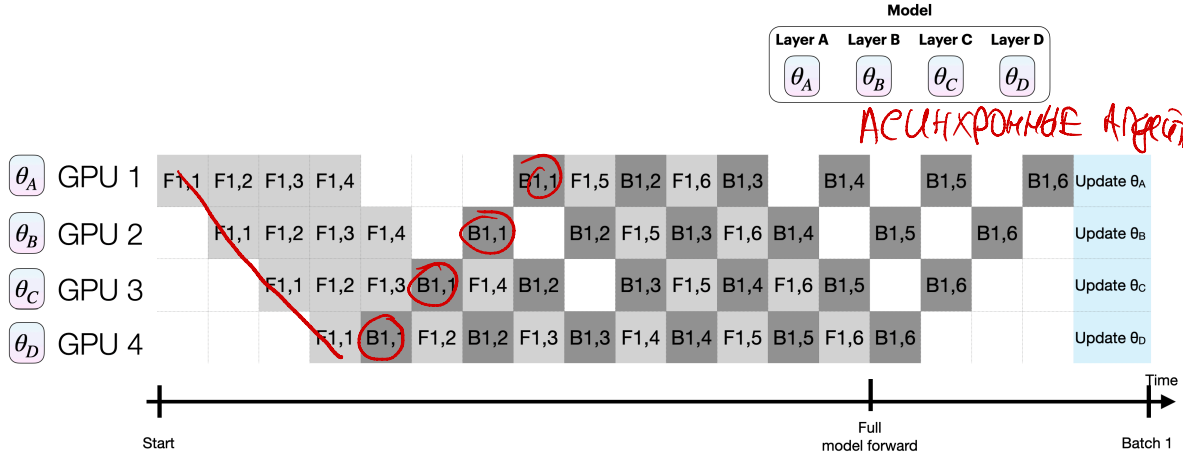
GPipe разделяет модель на стадии. Микробатчи проходят по конвейеру, обеспечивая перекрытие вычислений и коммуникаций:



[26] GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Pipeline parallelism: PipeDream [27]

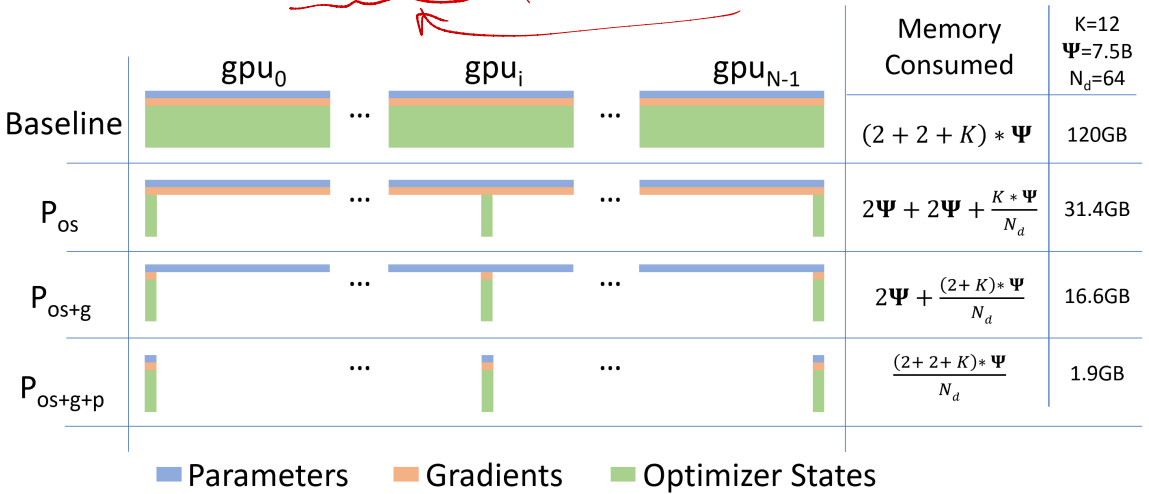
PipeDream использует асинхронный pipeline parallelism, чередуя forward и backward проходы между стадиями для максимальной утилизации:



[27] PipeDream: Generalized Pipeline Parallelism for DNN Training

ZeRO [28]

FSDP (PyTorch)



[28] ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

ZeRO: три стадии

Стадия	Что шардируется	Память на GPU	Коммуникации
ZeRO-1	Состояние оптимизатора (m, v)	$4N + \frac{12N}{D}$	= DDP
ZeRO-2	+ Градиенты	$4N + \frac{8N+4N}{D}$	= DDP
ZeRO-3	+ Параметры модели	$\frac{16N}{D}$	$1.5\times$ DDP

где N — число параметров, D — число GPU.

ZeRO: три стадии

Стадия	Что шардируется	Память на GPU	Коммуникации
ZeRO-1	Состояние оптимизатора (m, v)	$4N + \frac{12N}{D}$	= DDP
ZeRO-2	+ Градиенты	$4N + \frac{8N+4N}{D}$	= DDP
ZeRO-3	+ Параметры модели	$\frac{16N}{D}$	$1.5\times$ DDP

где N — число параметров, D — число GPU.

- **ZeRO-3** с 64 GPU: каждый GPU хранит лишь $\frac{1}{64}$ модели

ZeRO: три стадии

Стадия	Что шардируется	Память на GPU	Коммуникации
ZeRO-1	Состояние оптимизатора (m, v)	$4N + \frac{12N}{D}$	= DDP
ZeRO-2	+ Градиенты	$4N + \frac{8N+4N}{D}$	= DDP
ZeRO-3	+ Параметры модели	$\frac{16N}{D}$	$1.5\times$ DDP

где N — число параметров, D — число GPU.

- **ZeRO-3** с 64 GPU: каждый GPU хранит лишь $\frac{1}{64}$ модели
- **FSDP** (Fully Sharded Data Parallel) в PyTorch — реализация ZeRO-3

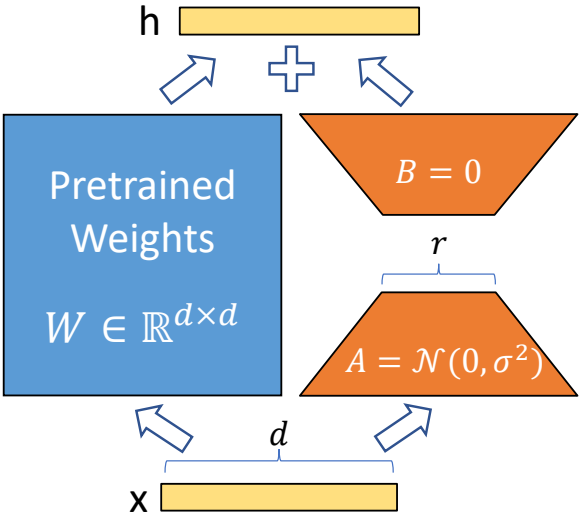
ZeRO: три стадии

Стадия	Что шардируется	Память на GPU	Коммуникации
ZeRO-1	Состояние оптимизатора (m, v)	$4N + \frac{12N}{D}$	= DDP
ZeRO-2	+ Градиенты	$4N + \frac{8N+4N}{D}$	= DDP
ZeRO-3	+ Параметры модели	$\frac{16N}{D}$	$1.5\times$ DDP

где N — число параметров, D — число GPU.

- **ZeRO-3** с 64 GPU: каждый GPU хранит лишь $\frac{1}{64}$ модели
- **FSDP** (Fully Sharded Data Parallel) в PyTorch — реализация ZeRO-3
- Можно комбинировать с pipeline и tensor parallelism

LoRA [29]



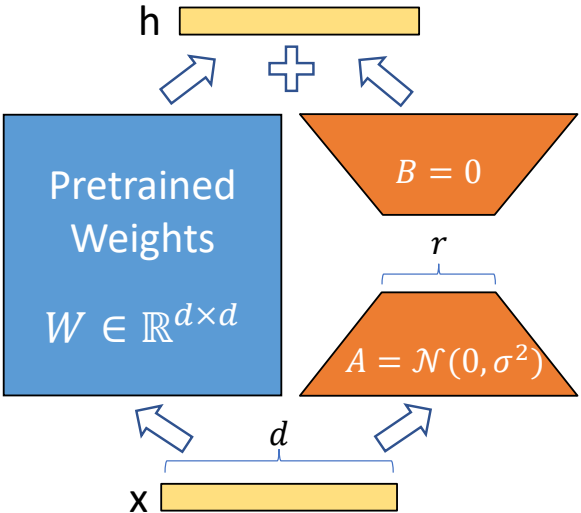
LoRA аппроксимирует обновление весов низкоранговым разложением:

$$W_{\text{new}} = W + \Delta W, \qquad \Delta W = AB^T$$

где $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{n \times r}$, $r \ll \min(m, n)$.

- A инициализируется случайно, $B = 0$ (стартуем с тождественного отображения)

LoRA [29]



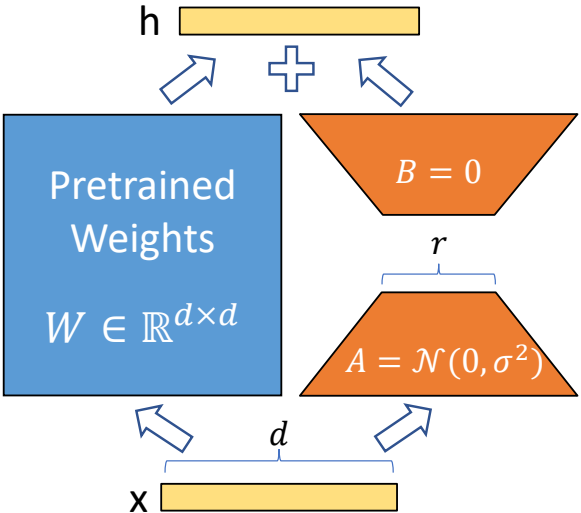
LoRA аппроксимирует обновление весов низкоранговым разложением:

$$W_{\text{new}} = W + \Delta W, \qquad \Delta W = AB^T$$

где $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{n \times r}$, $r \ll \min(m, n)$.

- A инициализируется случайно, $B = 0$ (стартуем с тождественного отображения)
- r обычно от 2 до 64

LoRA [29]



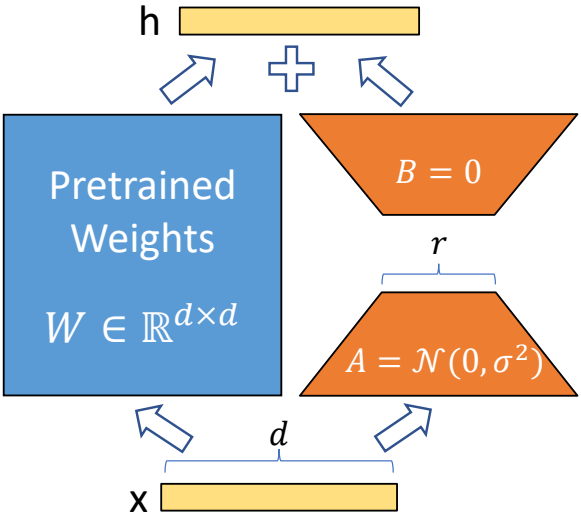
LoRA аппроксимирует обновление весов низкоранговым разложением:

$$W_{\text{new}} = W + \Delta W, \qquad \Delta W = AB^T$$

где $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{n \times r}$, $r \ll \min(m, n)$.

- A инициализируется случайно, $B = 0$ (стартуем с тождественного отображения)
- r обычно от 2 до 64
- Обычно применяется к attention-слоям

LoRA [29]



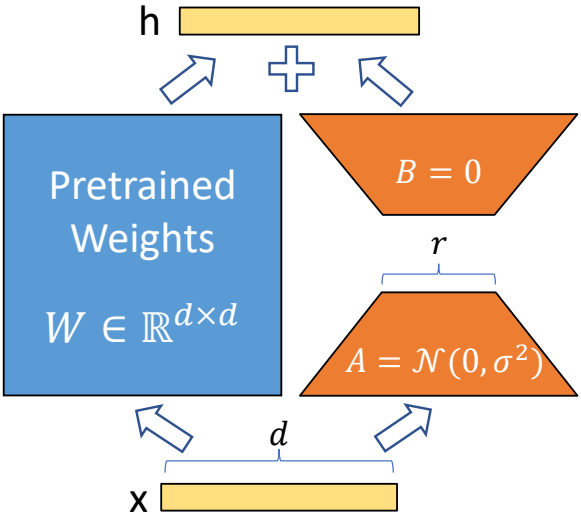
LoRA аппроксимирует обновление весов низкоранговым разложением:

$$W_{\text{new}} = W + \Delta W, \qquad \Delta W = AB^T$$

где $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{n \times r}$, $r \ll \min(m, n)$.

- A инициализируется случайно, $B = 0$ (стартуем с тождественного отображения)
- r обычно от 2 до 64
- Обычно применяется к attention-слоям

LoRA [29]



LoRA аппроксимирует обновление весов низкоранговым разложением:

$$W_{\text{new}} = W + \Delta W, \qquad \Delta W = AB^T$$

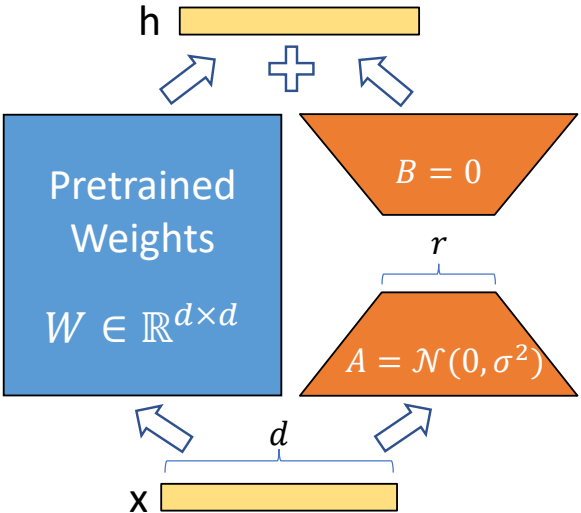
где $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{n \times r}$, $r \ll \min(m, n)$.

- A инициализируется случайно, $B = 0$ (стартуем с тождественного отображения)
- r обычно от 2 до 64
- Обычно применяется к attention-слоям

$$h = W_{\text{new}}x = Wx + AB^Tx$$

- Обучаем только A и B: $(m + n) \cdot r$ параметров вместо mn

LoRA [29]



LoRA аппроксимирует обновление весов низкоранговым разложением:

$$W_{\text{new}} = W + \Delta W, \qquad \Delta W = AB^T$$

где $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{n \times r}$, $r \ll \min(m, n)$.

- A инициализируется случайно, $B = 0$ (стартуем с тождественного отображения)
- r обычно от 2 до 64
- Обычно применяется к attention-слоям

$$h = W_{\text{new}}x = Wx + AB^Tx$$

- Обучаем только A и B: $(m + n) \cdot r$ параметров вместо mn
- При $r = 16$, $m = n = 4096$: **0.8%** от полного числа параметров

[29] LoRA: Low-Rank Adaptation of Large Language Models

Квантизация весов с помощью представления Кашина [30]

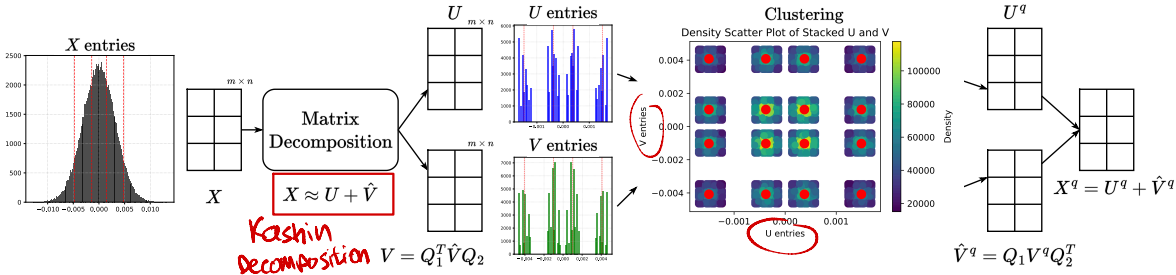


Figure 12: Схема алгоритма квантизации весов с помощью матричного разложения

[30] Quantization of Large Language Models with an Overdetermined Basis

Эта задача оптимизации сложнее, чем кажется

Влияние инициализации [31]

Правильная инициализация нейронной сети критически важна. Функция потерь сильно невыпукла; нахождение «хорошего» решения требует аккуратной настройки.

- Нельзя инициализировать все веса одинаково — почему?

Влияние инициализации [31]

Правильная инициализация нейронной сети критически важна. Функция потерь сильно невыпукла; нахождение «хорошего» решения требует аккуратной настройки.

- Нельзя инициализировать все веса одинаково — почему?
- Случайная инициализация: $W \sim N(0, \sigma^2)$, где σ зависит от числа нейронов в слое. *Нарушение симметрии.*

[31] On the importance of initialization and momentum in deep learning

Влияние инициализации весов на сходимость [32]

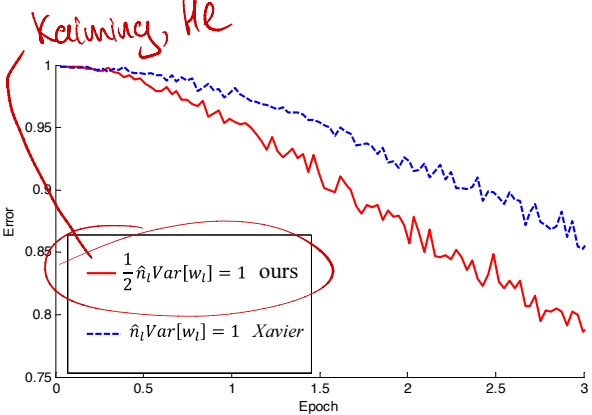


Figure 13: 22-слойная ReLU сеть: хорошая инициализация сходится быстрее

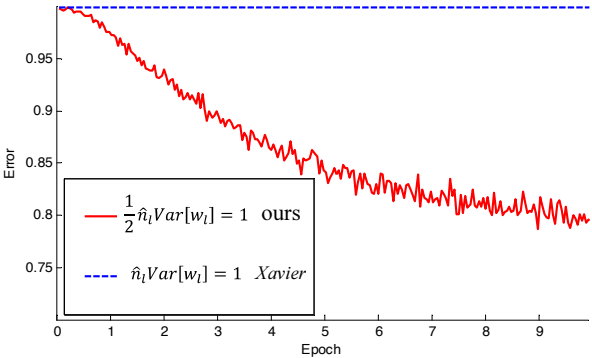


Figure 14: 30-слойная ReLU сеть: хорошая инициализация позволяет сойтись

[32] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Методы уменьшения дисперсии: почему не работают на глубоких сетях? 33

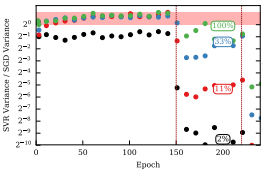


Figure 15: DenseNet

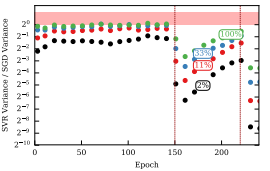


Figure 16: Small ResNet

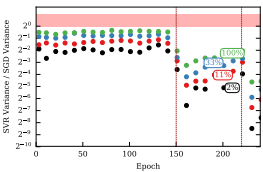


Figure 17: LeNet-5

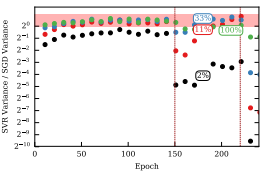


Figure 18: ResNet-110

- **SVRG / SAG** дают убедительные выигрыши в выпуклых задачах, но на CIFAR-10 (LeNet-5) и ImageNet (ResNet-18) не опережают обычный SGD.

Методы уменьшения дисперсии: почему не работают на глубоких сетях? 33

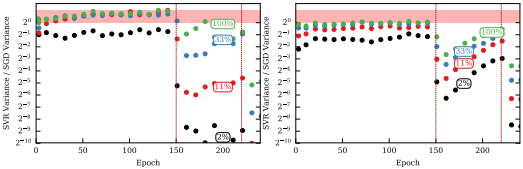


Figure 15: DenseNet

Figure 16: Small ResNet

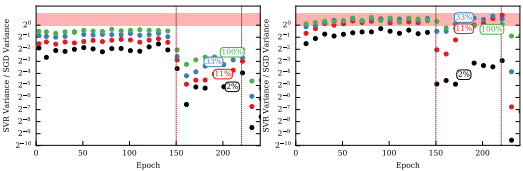


Figure 17: LeNet-5

Figure 18: ResNet-110

- **SVRG / SAG** дают убедительные выигрыши в выпуклых задачах, но на CIFAR-10 (LeNet-5) и ImageNet (ResNet-18) не опережают обычный SGD.
- Измеренное отношение «дисперсия SGD / дисперсия SVRG» остаётся $\lesssim 2$ для большинства слоёв — реальное снижение шума минимально.

Методы уменьшения дисперсии: почему не работают на глубоких сетях? 33

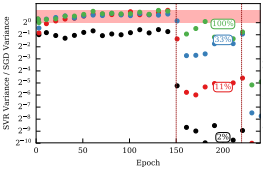


Figure 15: DenseNet

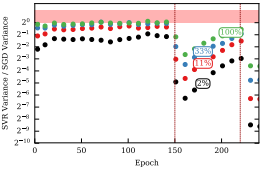


Figure 16: Small ResNet

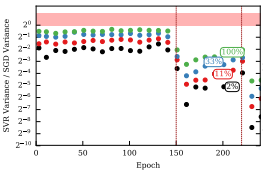


Figure 17: LeNet-5

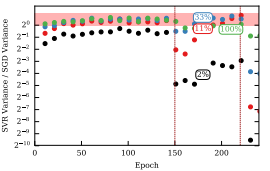


Figure 18: ResNet-110

- **SVRG / SAG** дают убедительные выигрыши в выпуклых задачах, но на CIFAR-10 (LeNet-5) и ImageNet (ResNet-18) не опережают обычный SGD.
- Измеренное отношение «дисперсия SGD / дисперсия SVRG» остаётся $\lesssim 2$ для большинства слоёв — реальное снижение шума минимально.
- Возможные причины:

Методы уменьшения дисперсии: почему не работают на глубоких сетях? 33

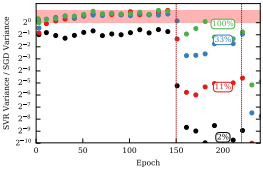


Figure 15: DenseNet

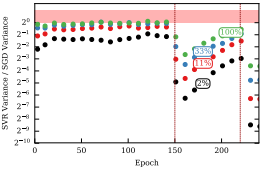


Figure 16: Small ResNet

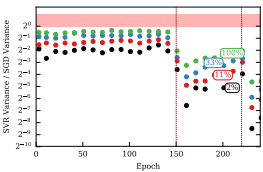


Figure 17: LeNet-5

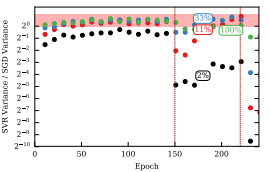


Figure 18: ResNet-110

- **SVRG / SAG** дают убедительные выигрыши в выпуклых задачах, но на CIFAR-10 (LeNet-5) и ImageNet (ResNet-18) не опережают обычный SGD.
- Измеренное отношение «дисперсия SGD / дисперсия SVRG» остаётся $\lesssim 2$ для большинства слоёв — реальное снижение шума минимально.
- Возможные причины:
  - **Аугментация данных** делает опорный градиент g_{ref} устаревшим уже после пары мини-батчей.

Методы уменьшения дисперсии: почему не работают на глубоких сетях? 33

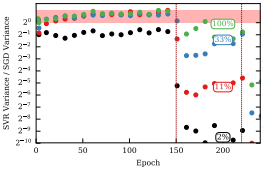


Figure 15: DenseNet

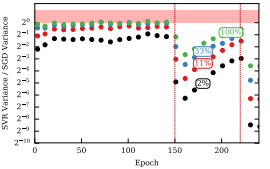


Figure 16: Small ResNet

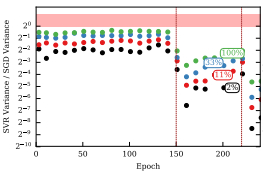


Figure 17: LeNet-5

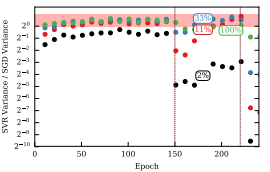


Figure 18: ResNet-110

- **SVRG / SAG** дают убедительные выигрыши в выпуклых задачах, но на CIFAR-10 (LeNet-5) и ImageNet (ResNet-18) не опережают обычный SGD.
- Измеренное отношение «дисперсия SGD / дисперсия SVRG» остаётся $\lesssim 2$ для большинства слоёв — реальное снижение шума минимально.
- Возможные причины:
  - **Аугментация данных** делает опорный градиент g_{ref} устаревшим уже после пары мини-батчей.
  - **BatchNorm** и **Dropout** добавляют внутреннюю стохастичность, которую невозможно компенсировать прошлым g_{ref}.

Методы уменьшения дисперсии: почему не работают на глубоких сетях? 33

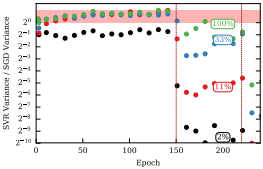


Figure 15: DenseNet

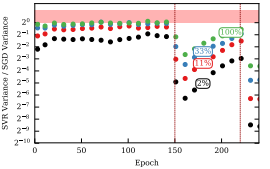


Figure 16: Small ResNet

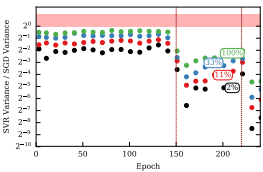


Figure 17: LeNet-5

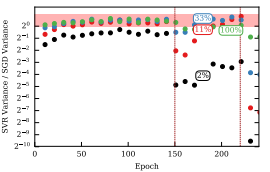


Figure 18: ResNet-110

- **SVRG / SAG** дают убедительные выигрыши в выпуклых задачах, но на CIFAR-10 (LeNet-5) и ImageNet (ResNet-18) не опережают обычный SGD.
- Измеренное отношение «дисперсия SGD / дисперсия SVRG» остаётся $\lesssim 2$ для большинства слоёв — реальное снижение шума минимально.
- Возможные причины:
  - **Аугментация данных** делает опорный градиент g_{ref} устаревшим уже после пары мини-батчей.
  - **BatchNorm** и **Dropout** добавляют внутреннюю стохастичность, которую невозможно компенсировать прошлым g_{ref}.
  - Дополнительный полный проход по датасету съедает потенциальную экономию итераций.

Методы уменьшения дисперсии: почему не работают на глубоких сетях? [33]

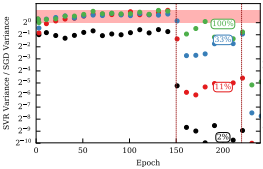


Figure 15: DenseNet

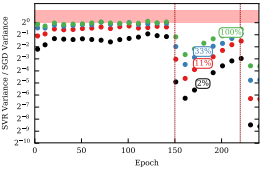


Figure 16: Small ResNet

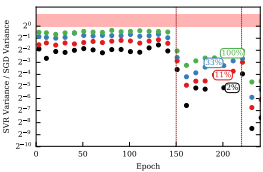


Figure 17: LeNet-5

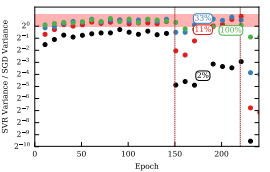


Figure 18: ResNet-110

- **SVRG / SAG** дают убедительные выигрыши в выпуклых задачах, но на CIFAR-10 (LeNet-5) и ImageNet (ResNet-18) не опережают обычный SGD.
- Измеренное отношение «дисперсия SGD / дисперсия SVRG» остаётся $\lesssim 2$ для большинства слоёв — реальное снижение шума минимально.
- Возможные причины:
  - **Аугментация данных** делает опорный градиент g_{ref} устаревшим уже после пары мини-батчей.
  - **BatchNorm** и **Dropout** добавляют внутреннюю стохастичность, которую невозможно компенсировать прошлым g_{ref}.
  - Дополнительный полный проход по датасету съедает потенциальную экономию итераций.
- **Вывод**: существующие методы уменьшения дисперсии непрактичны для современных глубоких сетей; будущие решения должны учитывать стохастичность архитектуры и данных.

[33] On the Ineffectiveness of Variance Reduced Optimization for Deep Learning

Adam работает хуже для CV, чем для LLM? [34]

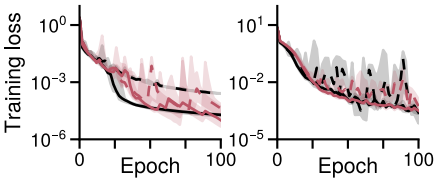


Figure 19: CNN на MNIST и CIFAR10

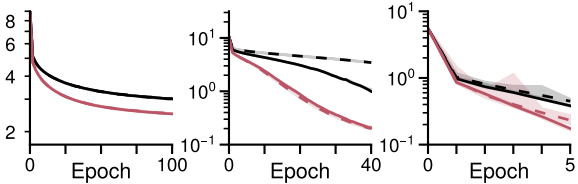


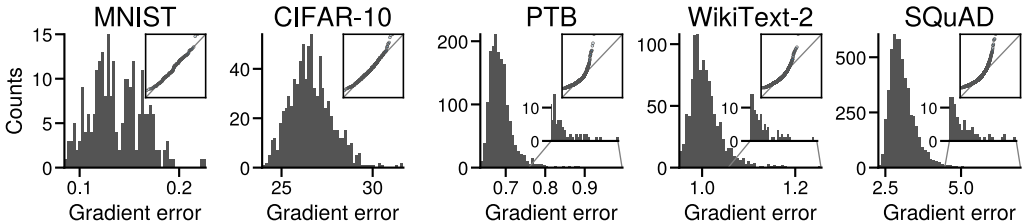
Figure 20: Трансформеры на PTB, WikiText2, SQuAD

Чёрные линии — SGD; красные линии — Adam.

[34] Linear attention is (maybe) all you need (to understand transformer optimization)

Почему Adam работает хуже для CV, чем для LLM? [35]

Потому что шум градиентов в языковых моделях имеет тяжёлые хвосты?



[35] Linear attention is (maybe) all you need (to understand transformer optimization)

Почему Adam работает хуже для CV, чем для LLM? [36]

Нет! Метки имеют 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

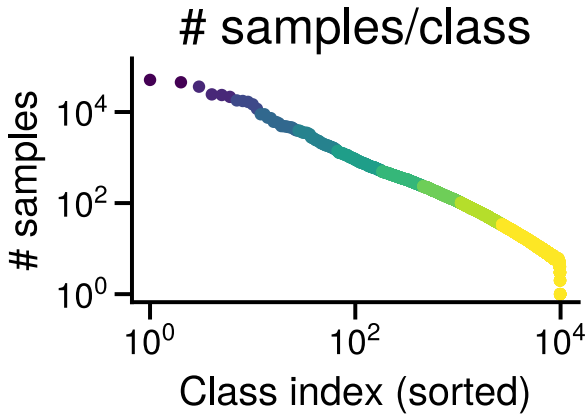
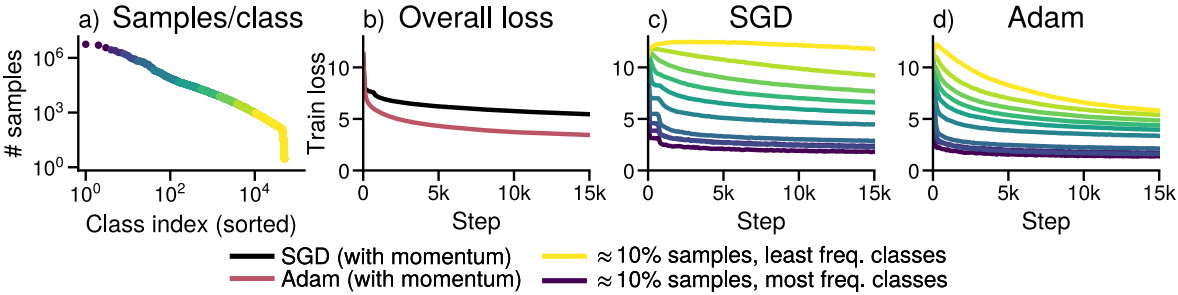


Figure 21: Распределение частоты токенов в PTB

[36] Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

SGD медленно прогрессирует на редких классах [37]



SGD не добивается прогресса на низкочастотных классах, в то время как Adam добивается. Обучение GPT-2 S на WikiText-103. (a) Распределение классов. (b) Общий train loss. (c, d) Train loss для каждой группы при использовании SGD и Adam.

[37] Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

Визуализация функции потерь: проекция на прямую

- Начальные веса w_0 и обученные $\hat{w}$. Генерируем случайный вектор $w_1 \in \mathbb{R}^p$ той же нормы:

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

$W = np.linalg.randn(T)$

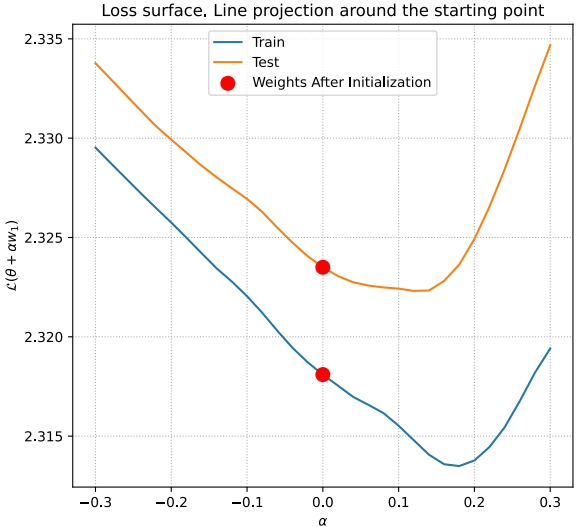
$W_0 \in R^{1T}$, $W_1 \in R^{1T}$

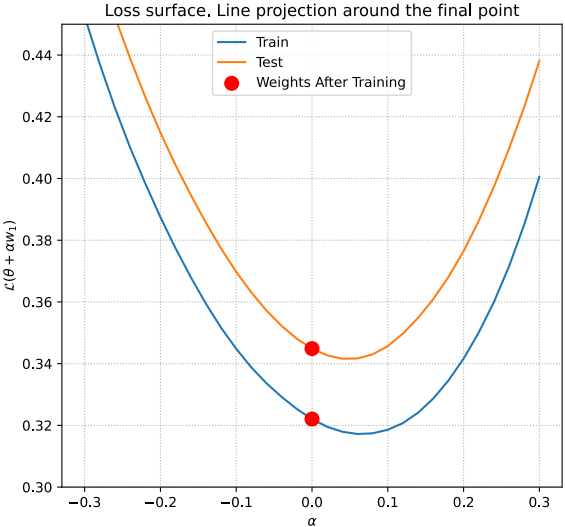
$\tilde{W} = W(\alpha) = W_0 + \alpha W_1$

Проекция функции потерь нейронной сети на прямую

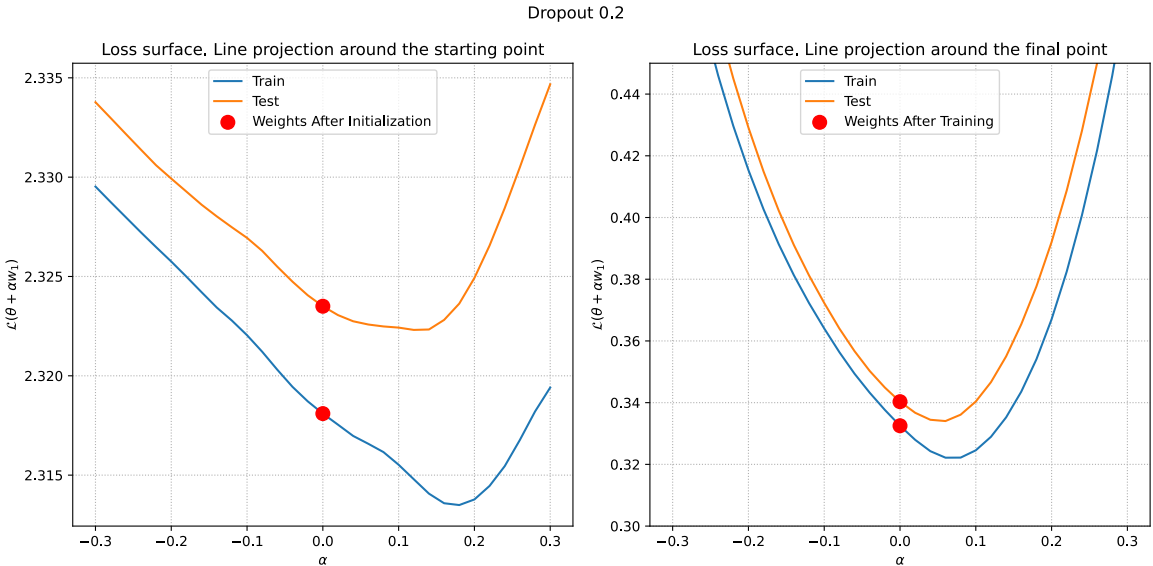
w_0 сразу! No Dropout

$w + dw_1$





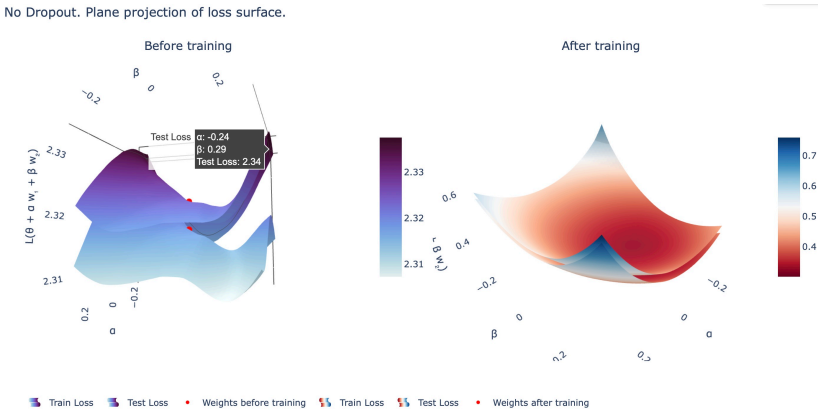
Проекция функции потерь нейронной сети на прямую



Проекция функции потерь нейронной сети на плоскость

- Расширяем идею: проекция поверхности потерь на плоскость, заданную 2 случайными векторами.

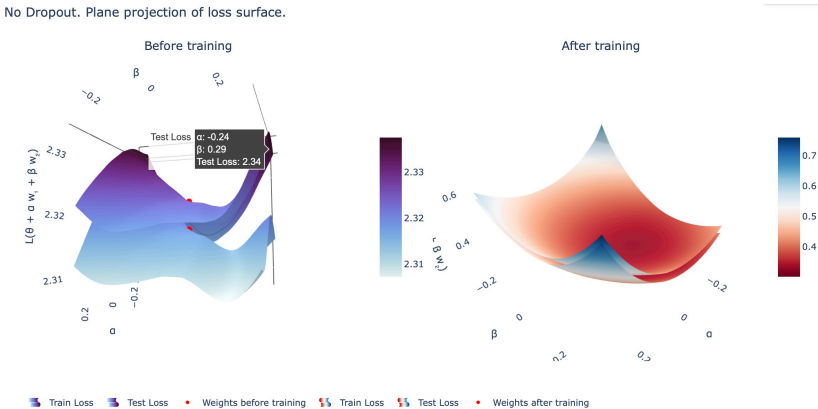
$$L(\alpha, \beta) = L(w_0 + \alpha w_1 + \beta w_2), \text{ где } \alpha, \beta \in [-b, b]^2.$$



Проекция функции потерь нейронной сети на плоскость

- Расширяем идею: проекция поверхности потерь на плоскость, заданную 2 случайными векторами.
- Два случайных гауссовых вектора в пространстве большой размерности с высокой вероятностью ортогональны.

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



Полезна ли визуализация потерь? [38]

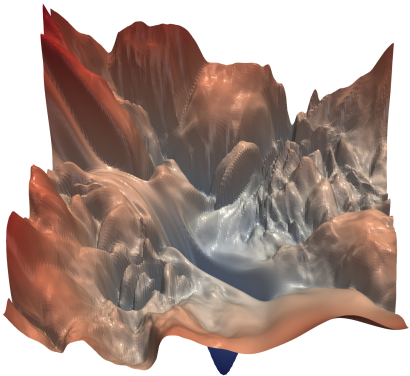


Figure 25: ResNet-56 без skip connections

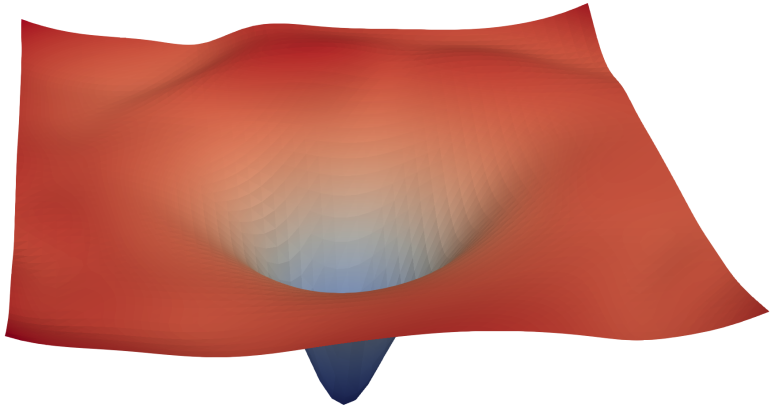


Figure 26: ResNet-56 co skip connections

[38] Visualizing the Loss Landscape of Neural Nets

Серьёзные визуализации [39]

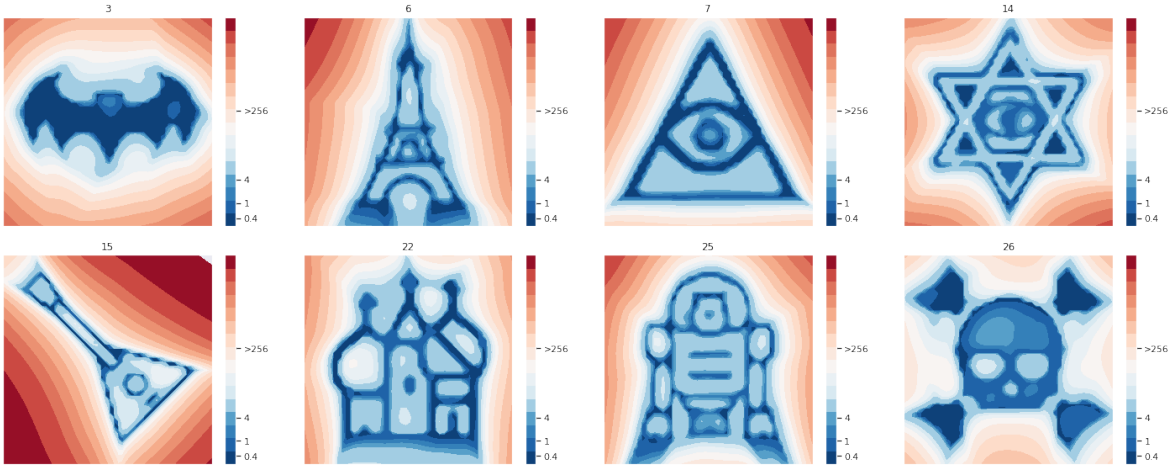
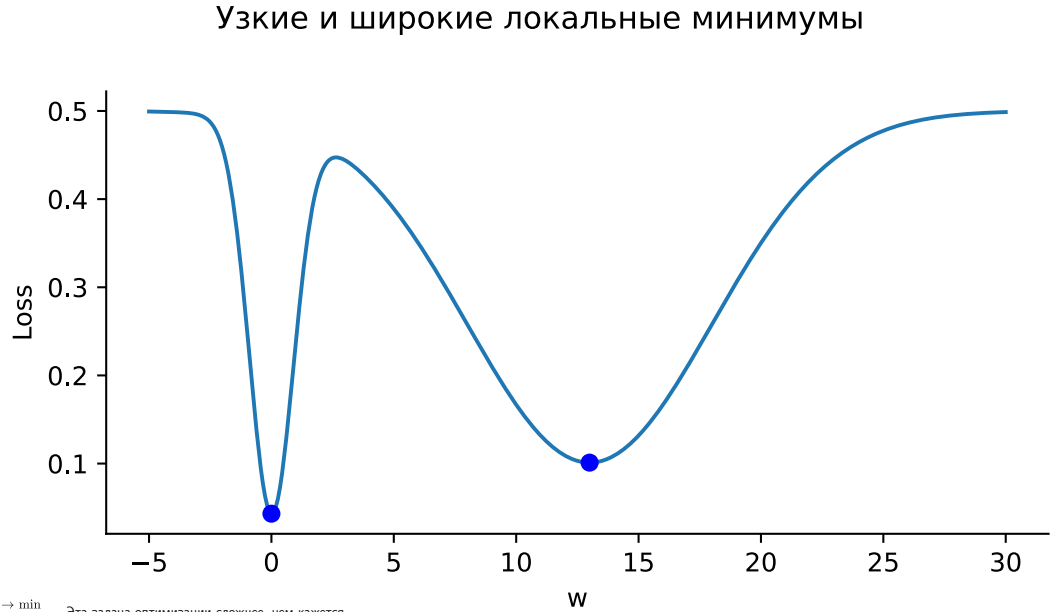


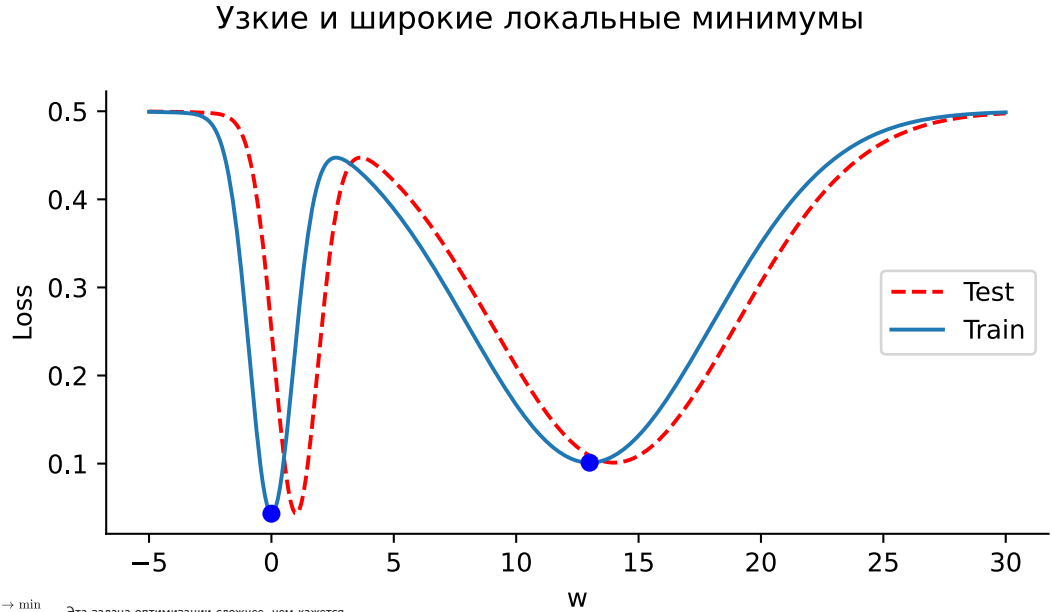
Figure 27: Примеры поверхности потерь типичной CNN на FashionMNIST и CIFAR10, найденные с помощью MPO. Значения потерь кодированы цветом по логарифмической шкале.

[39] Loss Landscape Sightseeing with Multi-Point Optimization

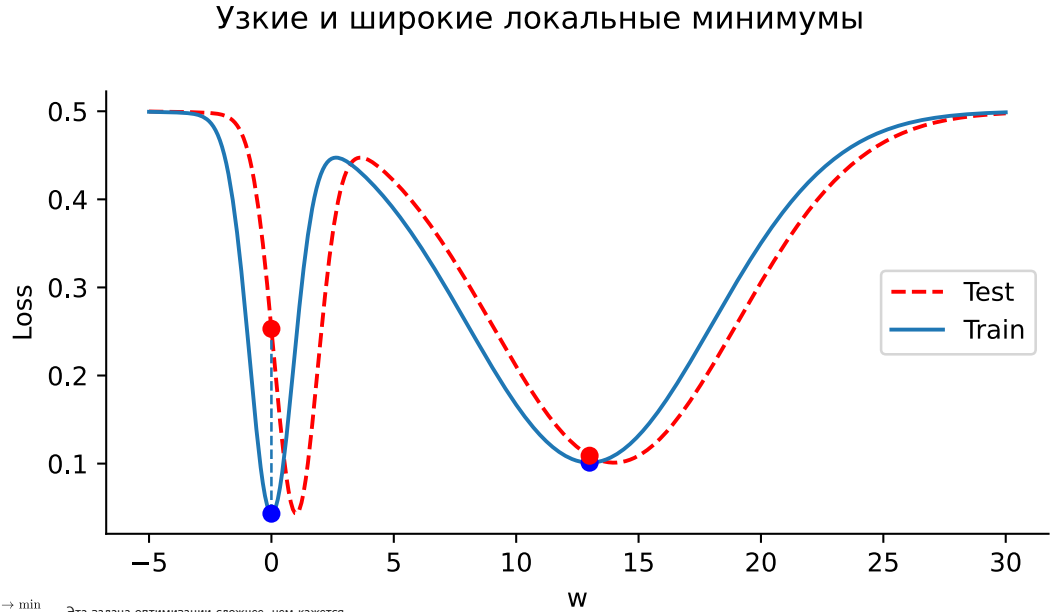
Ширина локальных минимумов



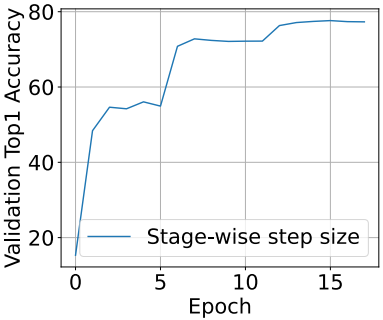
Ширина локальных минимумов

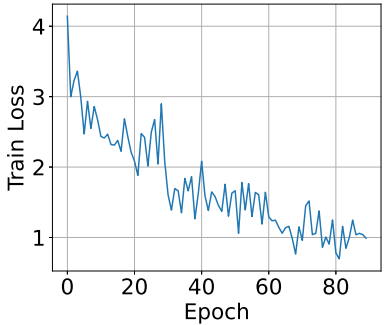


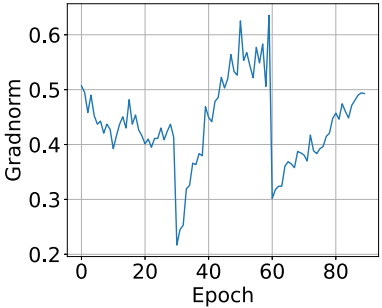
Ширина локальных минимумов



Модели не сходятся к стационарным точкам, но это не страшно [40]







[40] NN Weights Do Not Converge to Stationary Points

Grokking [41]

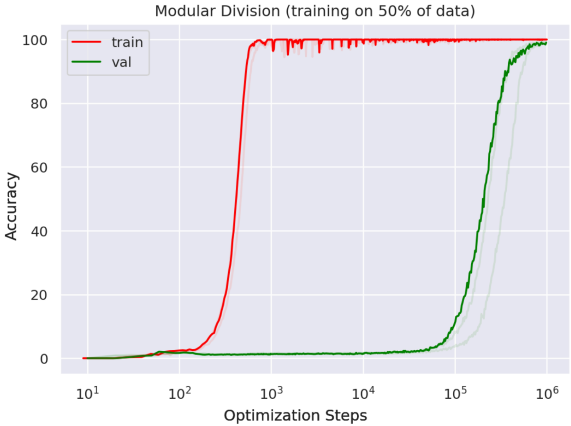


Figure 28: Обучение трансформера с 2 слоями, шириной 128 и 4 головами внимания ($\sim 4 \cdot 10^5$ необеднённых параметров). Воспроизведение ($\sim$ полчаса) здесь

- Рекомендую лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети**. видео, Презентация

Grokking [41]

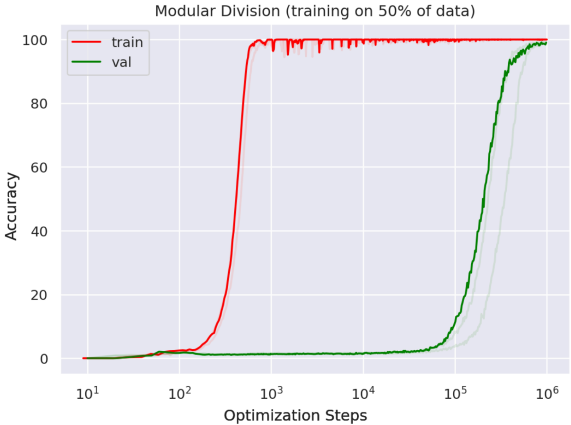


Figure 28: Обучение трансформера с 2 слоями, шириной 128 и 4 головами внимания ($\sim 4 \cdot 10^5$ необеднённых параметров). Воспроизведение ($\sim$ полчаса) здесь

- Рекомендую лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети**. видео, Презентация
- Свидетели Градиента — интересные наблюдения и эксперименты про гроккинг.

Grokking [41]

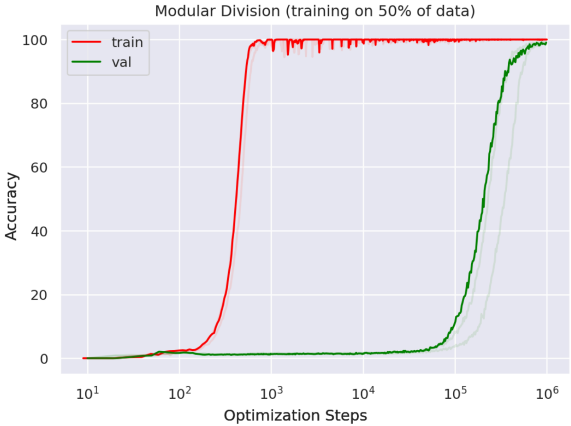
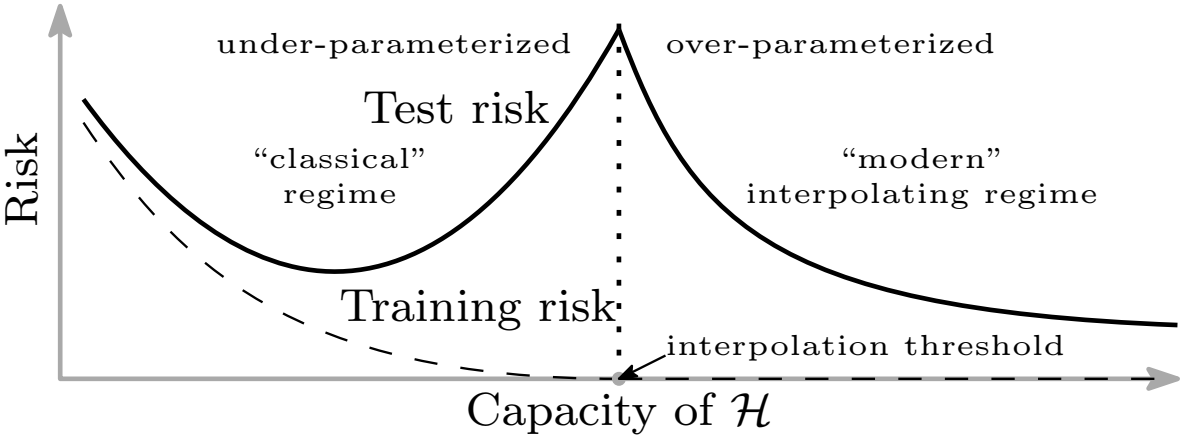


Figure 28: Обучение трансформера с 2 слоями, шириной 128 и 4 головами внимания ($\sim 4 \cdot 10^5$ необеднённых параметров). Воспроизведение ($\sim$ полчаса) здесь

- Рекомендую лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети**. видео, Презентация
- Свидетели Градиента — интересные наблюдения и эксперименты про гроккинг.
- Доклад **Чем не является гроккинг**.

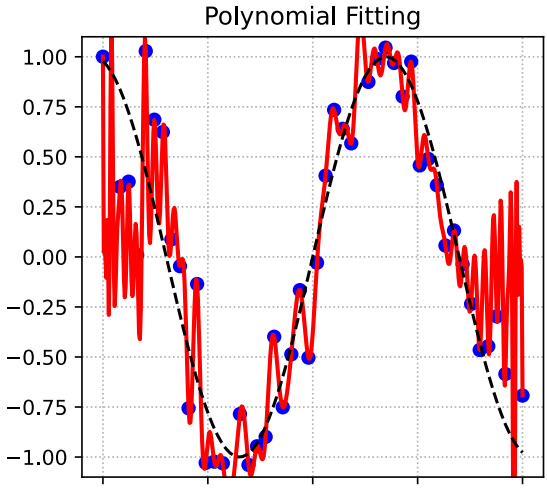
[41] Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets

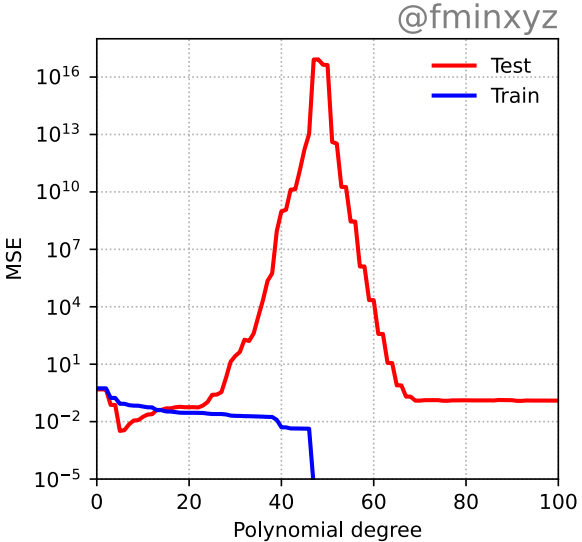
Double Descent [42]



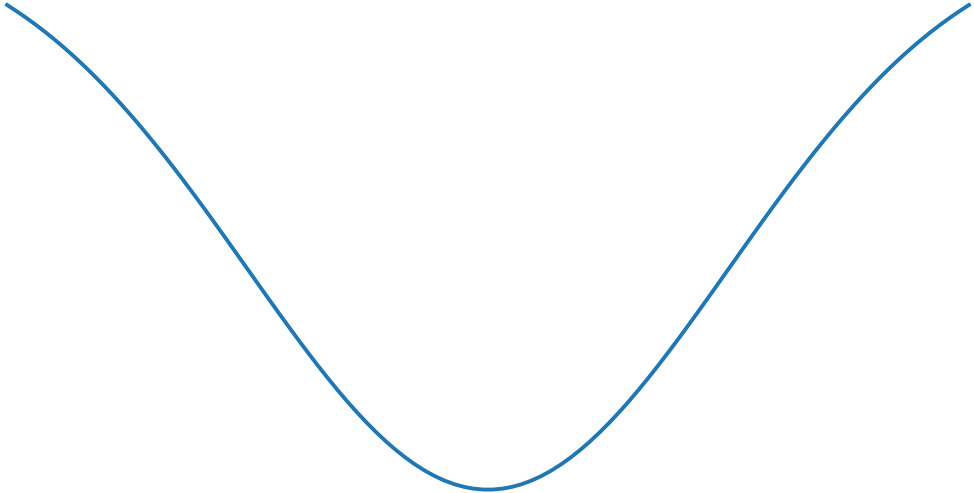
[42] Reconciling modern machine learning practice and the bias-variance trade-off

Double Descent

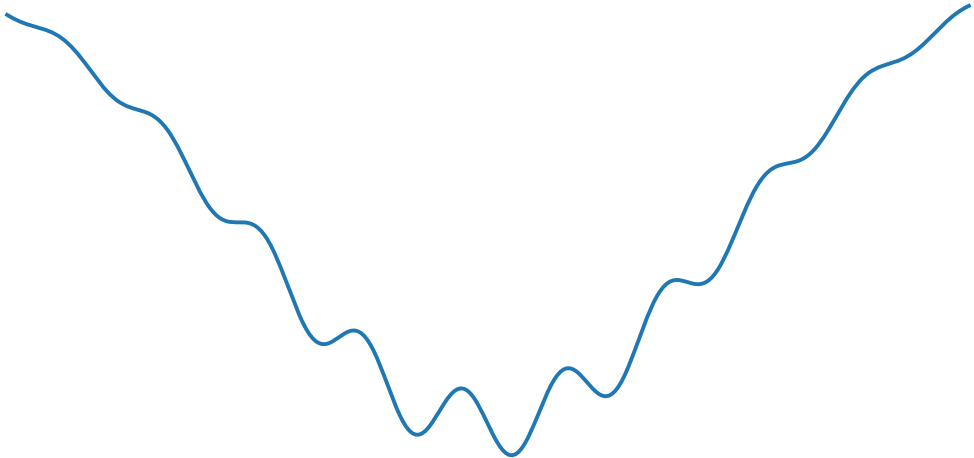




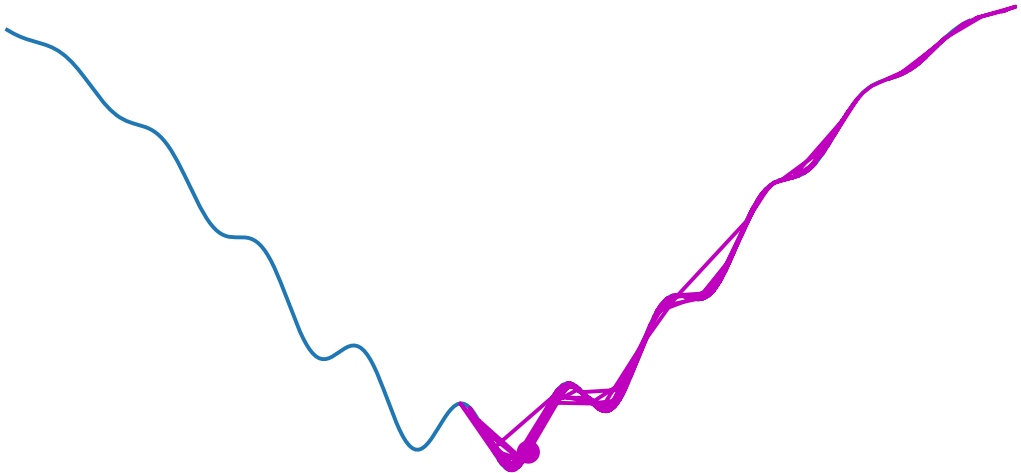
Градиентный спуск сходится к локальному минимуму



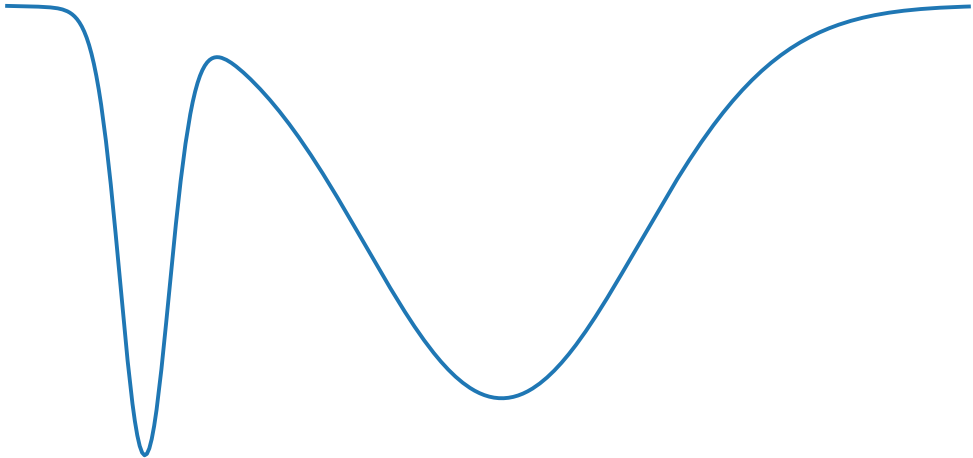
Градиентный спуск сходится к локальному минимуму



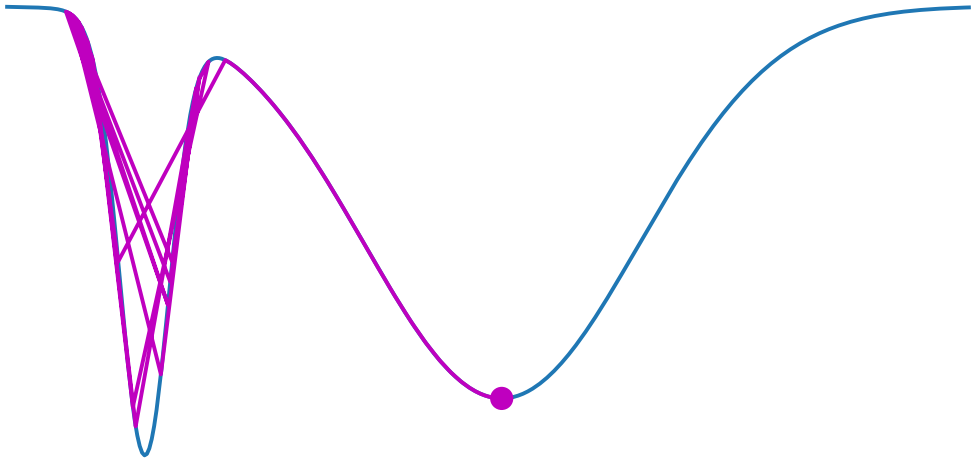
Стохастический градиентный спуск выпрыгивает из локальных минимумов



Градиентный спуск с маленьким шагом сходится в узкий локальный минимум



Градиентный спуск с большим шагом избегает узкого локального минимума



Тренды

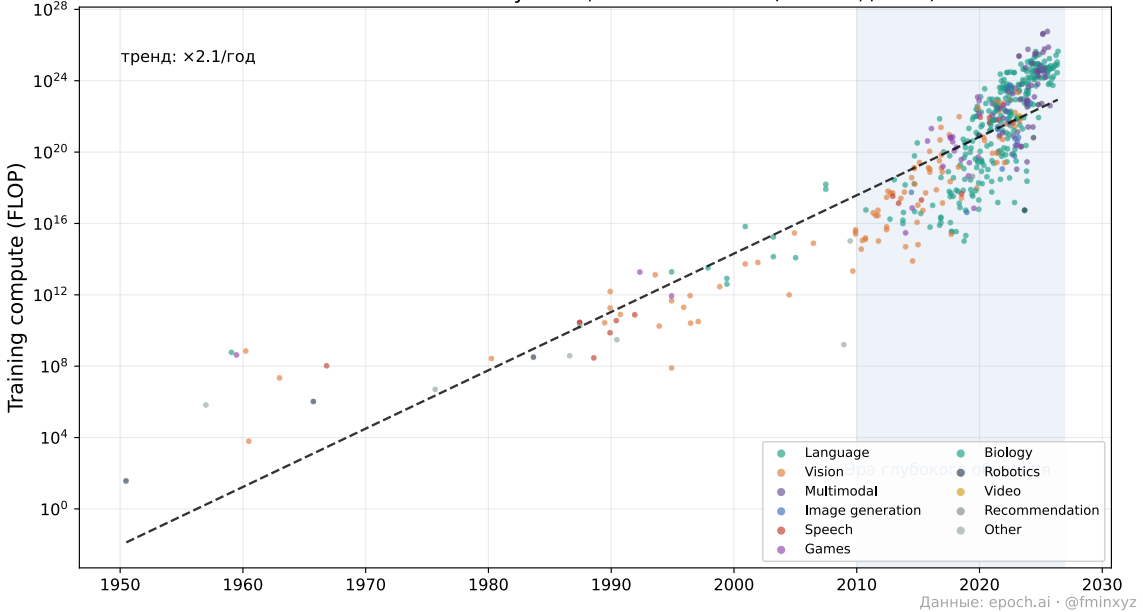


Figure 29: Динамика вычислений, необходимых для обучения моделей. Источник

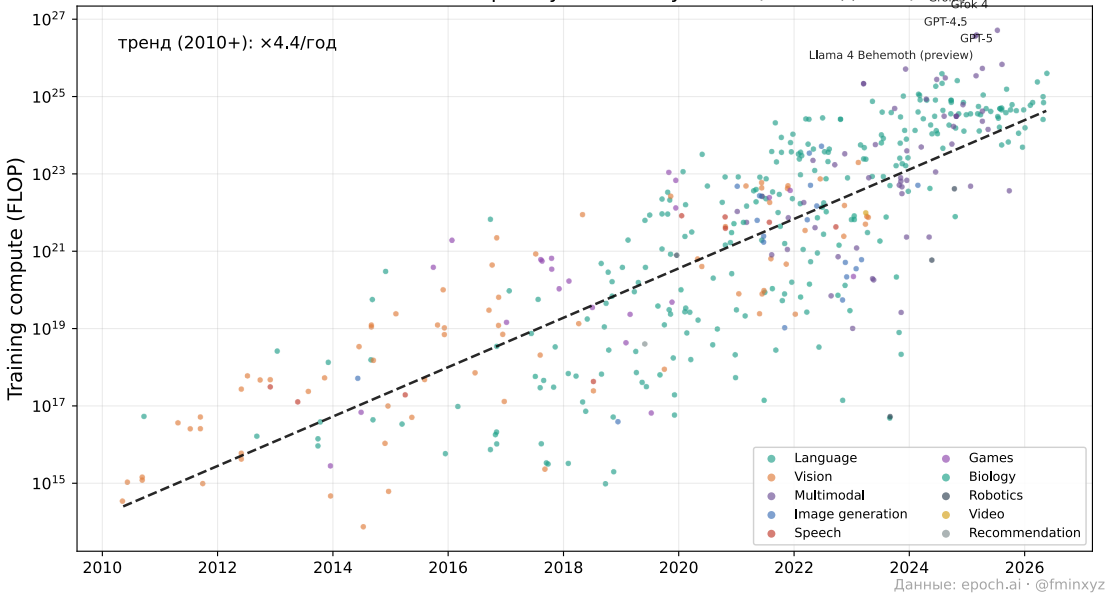


Figure 30: Динамика вычислений, необходимых для обучения нейросетевых моделей. Источник

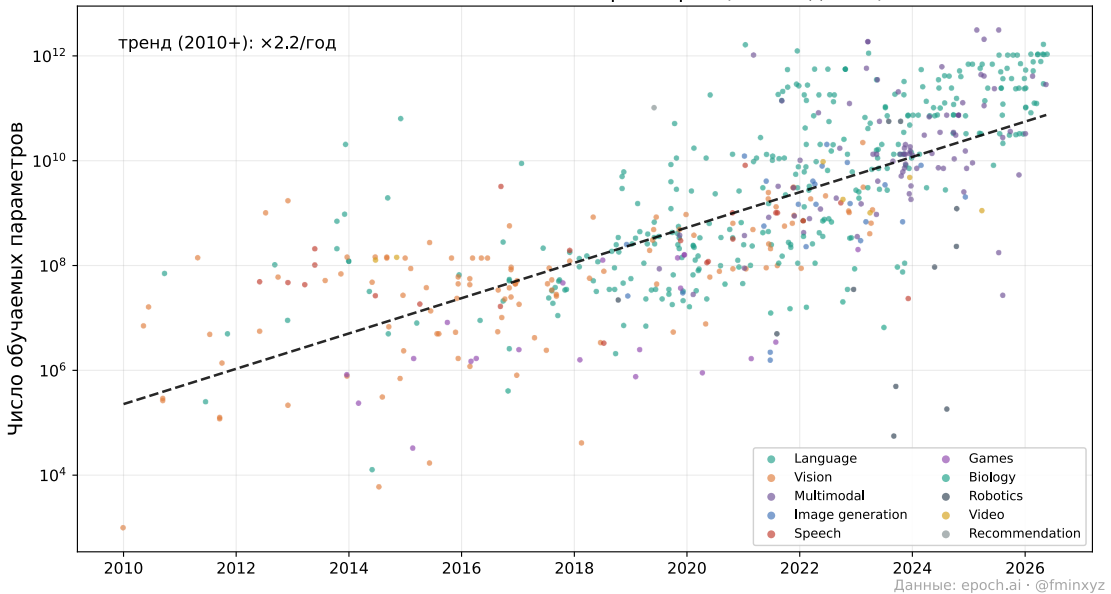
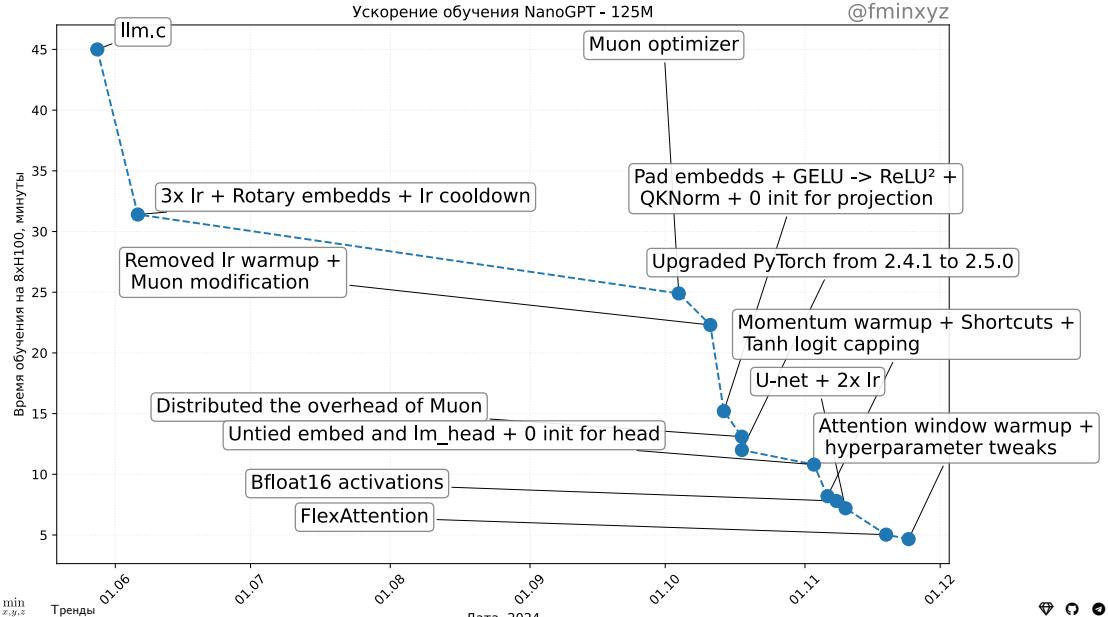


Figure 31: Динамика количества обучаемых параметров нейросетевых моделей. Источник

NanoGPT speedrun



Работают ли трюки, если увеличить размер модели?

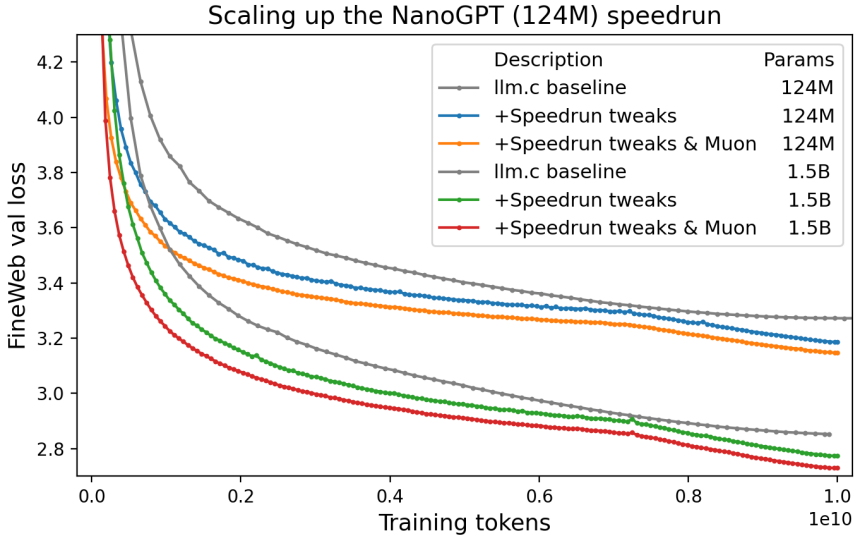


Figure 33: 🔗 Источник

Работают ли трюки, если увеличить размер модели?

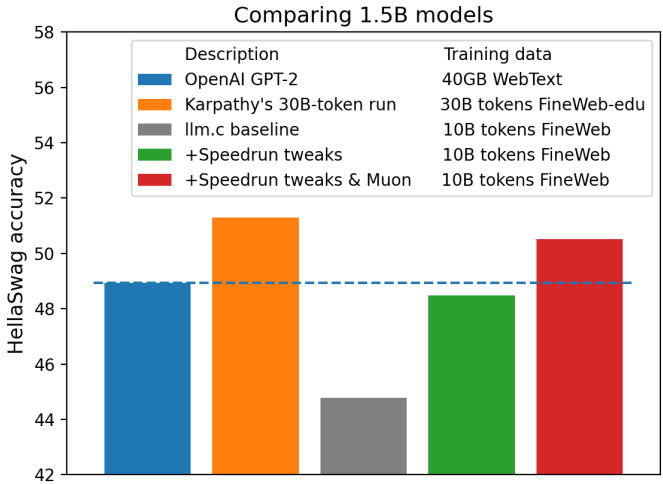


Figure 34: Источник

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)
- Kumar et al. "Scaling Laws for Precision" (2024)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)
- Kumar et al. "Scaling Laws for Precision" (2024)
- Micikevicius et al. "Mixed Precision Training" (2017)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)
- Kumar et al. "Scaling Laws for Precision" (2024)
- Micikevicius et al. "Mixed Precision Training" (2017)
- Chen et al. "Training Deep Nets with Sublinear Memory Cost" (2016)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)
- Kumar et al. "Scaling Laws for Precision" (2024)
- Micikevicius et al. "Mixed Precision Training" (2017)
- Chen et al. "Training Deep Nets with Sublinear Memory Cost" (2016)
- Goyal et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour" (2017)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)
- Kumar et al. "Scaling Laws for Precision" (2024)
- Micikevicius et al. "Mixed Precision Training" (2017)
- Chen et al. "Training Deep Nets with Sublinear Memory Cost" (2016)
- Goyal et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour" (2017)
- You et al. "Large Batch Training of Convolutional Networks" (LARS, 2017)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)
- Kumar et al. "Scaling Laws for Precision" (2024)
- Micikevicius et al. "Mixed Precision Training" (2017)
- Chen et al. "Training Deep Nets with Sublinear Memory Cost" (2016)
- Goyal et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour" (2017)
- You et al. "Large Batch Training of Convolutional Networks" (LARS, 2017)
- You et al. "Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes" (LAMB, 2019)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)
- Kumar et al. "Scaling Laws for Precision" (2024)
- Micikevicius et al. "Mixed Precision Training" (2017)
- Chen et al. "Training Deep Nets with Sublinear Memory Cost" (2016)
- Goyal et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour" (2017)
- You et al. "Large Batch Training of Convolutional Networks" (LARS, 2017)
- You et al. "Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes" (LAMB, 2019)
- Rajbhandari et al. "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models" (2019)

Дополнительные материалы

- Kaplan et al. "Scaling Laws for Neural Language Models" (2020)
- Hoffmann et al. "Training Compute-Optimal Large Language Models" (Chinchilla, 2022)
- Yang & Hu et al. "Tuning Large NNs via Zero-Shot Hyperparameter Transfer" (μP/muTransfer, 2022)
- Blake et al. "u-μP: Unit-Scaled Maximal Update Parametrization" (2024)
- DeepSeek-AI "DeepSeek LLM: Scaling Open-Source Language Models" (HP scaling laws, 2024)
- "Optimal Hyperparameter Scaling Law in LLM Pre-training" (Step Law, 2025)
- Sardana et al. "Beyond Chinchilla-Optimal: Accounting for Inference" (2023)
- Kumar et al. "Scaling Laws for Precision" (2024)
- Micikevicius et al. "Mixed Precision Training" (2017)
- Chen et al. "Training Deep Nets with Sublinear Memory Cost" (2016)
- Goyal et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour" (2017)
- You et al. "Large Batch Training of Convolutional Networks" (LARS, 2017)
- You et al. "Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes" (LAMB, 2019)
- Rajbhandari et al. "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models" (2019)
- Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models" (2021)