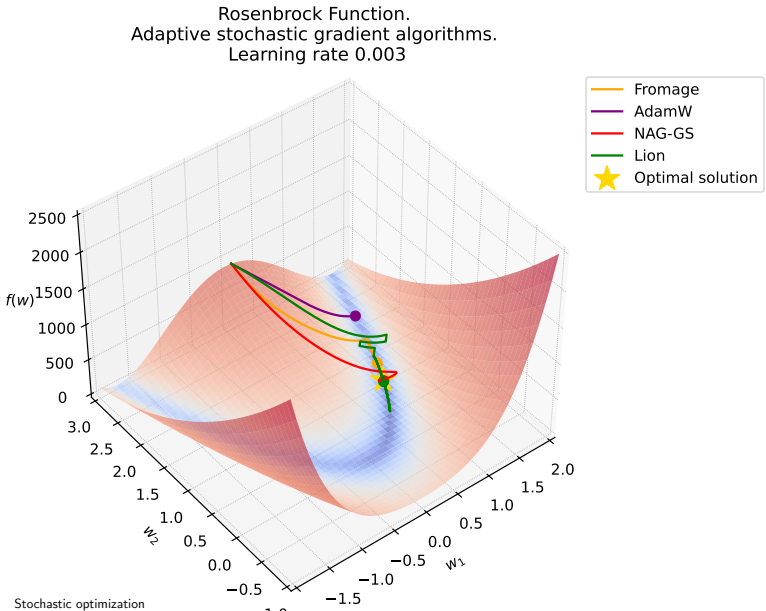
# Optimization for Deep Learning

Даня Меркулов

Методы Оптимизации в Машинном Обучении. ФКН ВШЭ

# Stochastic optimization

# A lot of them


Rosenbrock Function.
Adaptive stochastic gradient algorithms.
Learning rate 0.003
Fromage
AdamW
NAG-GS
Lion
Optimal solution
f(w)
2500
2000
1500
1000
500
0
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
w2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
w1

# SGD diverges with any learning rate for LLS

# Optimization for Deep Learning from the practical perspective

# How to compare them? AlgoPerf benchmark [1] [2]

- **AlgoPerf Benchmark:** Compares NN training algorithms with two rulesets:

# How to compare them? AlgoPerf benchmark [1] [2]

- **AlgoPerf Benchmark:** Compares NN training algorithms with two rulesets:
  - **External Tuning:** Simulates hyperparameter tuning with limited resources (5 trials, quasirandom search). Scored on median fastest time to target across 5 studies.

# How to compare them? AlgoPerf benchmark [1] [2]

- **AlgoPerf Benchmark:** Compares NN training algorithms with two rulesets:
  - **External Tuning:** Simulates hyperparameter tuning with limited resources (5 trials, quasirandom search). Scored on median fastest time to target across 5 studies.
  - **Self-Tuning:** Simulates automated tuning on a single machine (fixed/inner-loop tuning, 3x budget). Scored on median runtime across 5 studies.

# How to compare them? AlgoPerf benchmark [1] [2]

- **AlgoPerf Benchmark:** Compares NN training algorithms with two rulesets:
  - **External Tuning:** Simulates hyperparameter tuning with limited resources (5 trials, quasirandom search). Scored on median fastest time to target across 5 studies.
  - **Self-Tuning:** Simulates automated tuning on a single machine (fixed/inner-loop tuning, 3x budget). Scored on median runtime across 5 studies.
- **Scoring:** Aggregates workload scores using performance profiles. Profiles plot the fraction of workloads solved within a time factor $\tau$ relative to the fastest submission. Final score: normalized area under the profile (1.0 = fastest on all workloads).

# How to compare them? AlgoPerf benchmark [1] [2]

- **AlgoPerf Benchmark:** Compares NN training algorithms with two rulesets:
  - **External Tuning:** Simulates hyperparameter tuning with limited resources (5 trials, quasirandom search). Scored on median fastest time to target across 5 studies.
  - **Self-Tuning:** Simulates automated tuning on a single machine (fixed/inner-loop tuning, 3x budget). Scored on median runtime across 5 studies.
- **Scoring:** Aggregates workload scores using performance profiles. Profiles plot the fraction of workloads solved within a time factor $\tau$ relative to the fastest submission. Final score: normalized area under the profile (1.0 = fastest on all workloads).
- **Computational Cost:** Scoring required $\sim 49,240$ total hours on 8x NVIDIA V100 GPUs (avg. $\sim 3469$h/external, $\sim 1847$h/self-tuning submission).

[1] Benchmarking Neural Network Training Algorithms
[2] Accelerating neural network training: An analysis of the AlgoPerf competition

# AlgoPerf benchmark

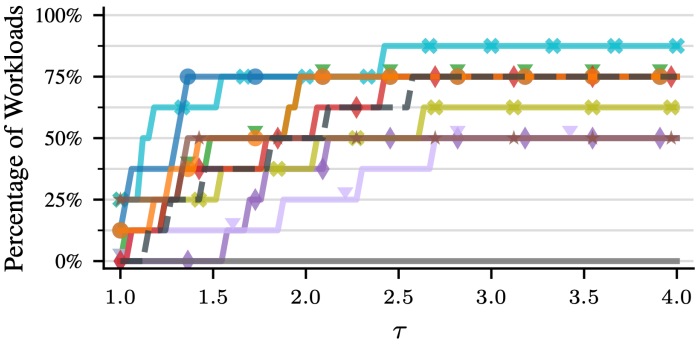
**Summary of *fixed* base workloads in the AlgoPerf benchmark.** Losses include cross-entropy (CE), mean absolute error (L1), and Connectionist Temporal Classification loss (CTC). Additional evaluation metrics are structural similarity index measure (SSIM), (word) error rate (ER & WER), mean average precision (mAP), and bilingual evaluation understudy score (BLEU). The \runtime budget is that of the external tuning ruleset, the self-tuning ruleset allows $3\times$ longer training.

| Task | Dataset | Model | Loss | Metric | Validation Target | Runtime Budget |
|---|---|---|---|---|---|---|
| Clickthrough rate prediction | CRITEO 1TB | DLRMSMALL | CE | CE | 0.123735 | 7703 |
| MRI reconstruction | FASTMRI | U-NET | L1 | SSIM | 0.7344 | 8859 |
| Image classification | IMAGENET | ResNet-50 | CE | ER | 0.22569 | 63,008 |
| | | ViT | CE | ER | 0.22691 | 77,520 |
| Speech recognition | LIBRISPEECH | Conformer | CTC | WER | 0.085884 | 61,068 |
| | | DeepSpeech | CTC | WER | 0.119936 | 55,506 |
| Molecular property prediction | OGBG | GNN | CE | mAP | 0.28098 | 18,477 |
| Translation | WMT | Transformer | CE | BLEU | 30.8491 | 48,151 |

# AlgoPerf benchmark

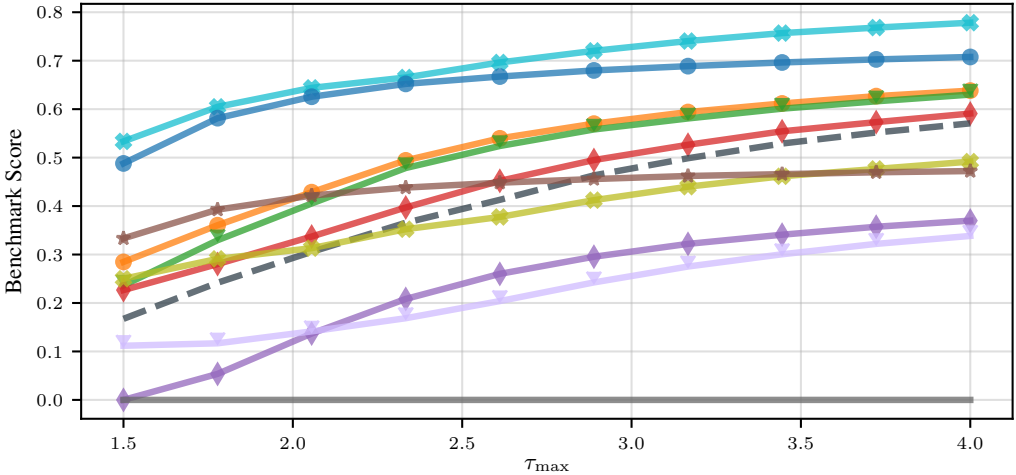
| Submission | Line | Score |
|---|---|---|
| PyTorch Distributed Shampoo | | 0.7784 |
| Schedule Free AdamW | | 0.7077 |
| Generalized Adam | | 0.6383 |
| Cyclic LR | | 0.6301 |
| NadamP | | 0.5909 |
| Baseline | | 0.5707 |
| Amos | | 0.4918 |
| CASPR Adaptive | | 0.4722 |
| LAWA Queue | | 0.3699 |
| LAWA EMA | | 0.3384 |
| Schedule Free Prodigy | | 0 |

**(a) External tuning** leaderboard

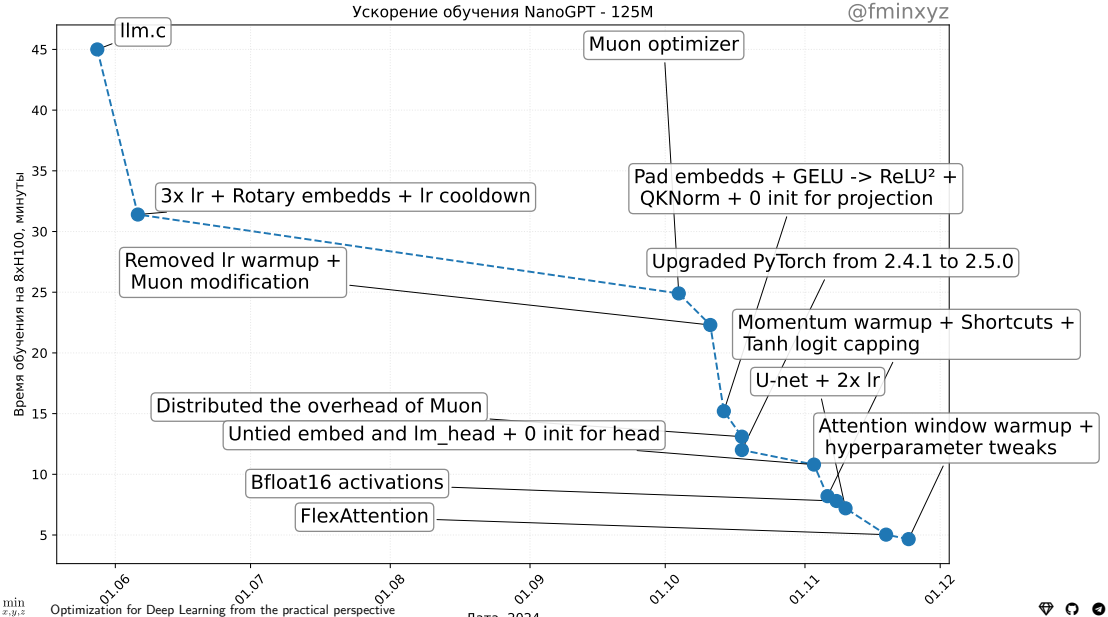



**(b) External tuning** performance profiles

# AlgoPerf benchmark



# NanoGPT speedrun



# Работают ли трюки, если увеличить размер модели?

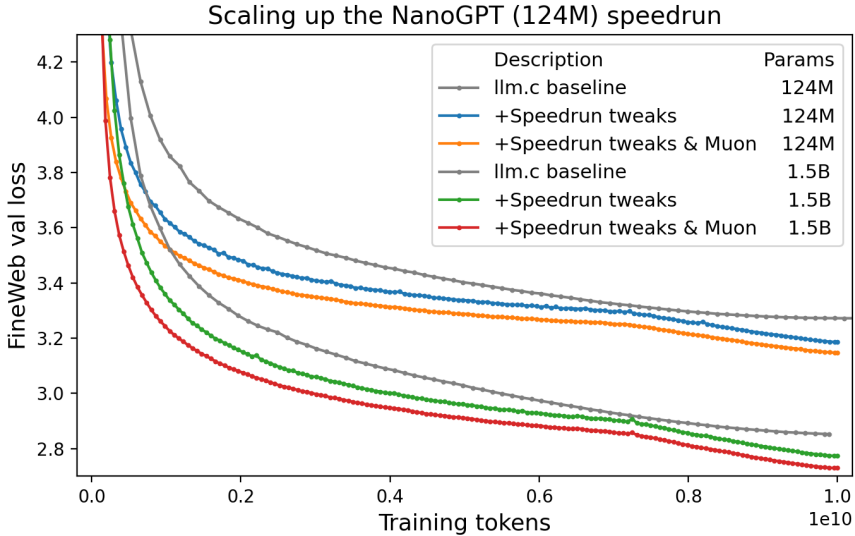



Рисунок 2: 🔗 Источник

# Работают ли трюки, если увеличить размер модели?

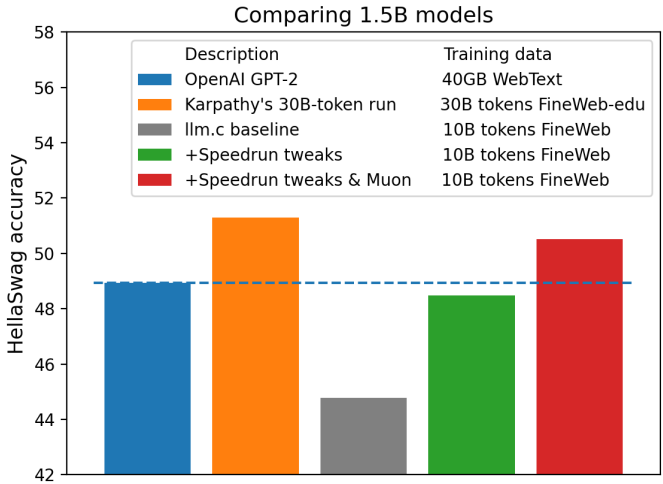



Рисунок 3: Источник

## Shampoo [3]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

**Core Idea:** Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where $G_k$ are the gradients.

Simplified concept:

1. Compute gradient $G_k$.

**Notes:**

## Shampoo [3]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

**Core Idea:** Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where $G_k$ are the gradients.

Simplified concept:

1. Compute gradient $G_k$.
2. Update statistics $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ and $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.

**Notes:**

## Shampoo [3]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

**Core Idea:** Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where $G_k$ are the gradients.

Simplified concept:

1. Compute gradient $G_k$.
2. Update statistics $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ and $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Compute preconditioners $P_L = L_k^{-1/4}$ and $P_R = R_k^{-1/4}$. (Inverse matrix root)

**Notes:**

## Shampoo [3]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

**Core Idea:** Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where $G_k$ are the gradients.

Simplified concept:

1. Compute gradient $G_k$.
2. Update statistics $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ and $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Compute preconditioners $P_L = L_k^{-1/4}$ and $P_R = R_k^{-1/4}$. (Inverse matrix root)
4. Update: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

**Notes:**

## Shampoo [3]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

**Core Idea:** Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where $G_k$ are the gradients.

Simplified concept:

1. Compute gradient $G_k$.
2. Update statistics $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ and $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Compute preconditioners $P_L = L_k^{-1/4}$ and $P_R = R_k^{-1/4}$. (Inverse matrix root)
4. Update: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

**Notes:**

- Aims to capture curvature information more effectively than first-order methods.

## Shampoo [3]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

**Core Idea:** Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where $G_k$ are the gradients.

Simplified concept:

1. Compute gradient $G_k$.
2. Update statistics $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ and $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Compute preconditioners $P_L = L_k^{-1/4}$ and $P_R = R_k^{-1/4}$. (Inverse matrix root)
4. Update: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

**Notes:**

- Aims to capture curvature information more effectively than first-order methods.
- Computationally more expensive than Adam but can converge faster or to better solutions in terms of steps.

## Shampoo [3]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

**Core Idea:** Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where $G_k$ are the gradients.

Simplified concept:

1. Compute gradient $G_k$.
2. Update statistics $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ and $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Compute preconditioners $P_L = L_k^{-1/4}$ and $P_R = R_k^{-1/4}$. (Inverse matrix root)
4. Update: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

**Notes:**

- Aims to capture curvature information more effectively than first-order methods.
- Computationally more expensive than Adam but can converge faster or to better solutions in terms of steps.
- Requires careful implementation for efficiency (e.g., efficient computation of inverse matrix roots, handling large matrices).

# Shampoo [3]

Stands for **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. It's a method inspired by second-order optimization designed for large-scale deep learning.

**Core Idea:** Approximates the full-matrix AdaGrad pre conditioner using efficient matrix structures, specifically Kronecker products.

For a weight matrix $W \in \mathbb{R}^{m \times n}$, the update involves preconditioning using approximations of the statistics matrices $L \approx \sum_k G_k G_k^T$ and $R \approx \sum_k G_k^T G_k$, where $G_k$ are the gradients.

Simplified concept:

1. Compute gradient $G_k$.
2. Update statistics $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ and $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Compute preconditioners $P_L = L_k^{-1/4}$ and $P_R = R_k^{-1/4}$. (Inverse matrix root)
4. Update: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

**Notes:**

- Aims to capture curvature information more effectively than first-order methods.
- Computationally more expensive than Adam but can converge faster or to better solutions in terms of steps.
- Requires careful implementation for efficiency (e.g., efficient computation of inverse matrix roots, handling large matrices).
- Variants exist for different tensor shapes (e.g., convolutional layers).

[3] Shampoo: Preconditioned Stochastic Tensor Optimization

# Muon [4]

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

[4] Deriving Muon

# Unexpected stories

# Adam работает хуже для CV, чем для LLM? [5]

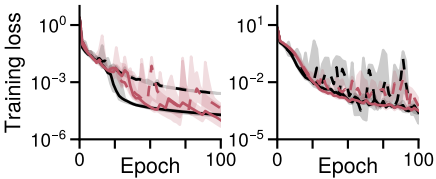



Рисунок 4: CNNs on MNIST and CIFAR10

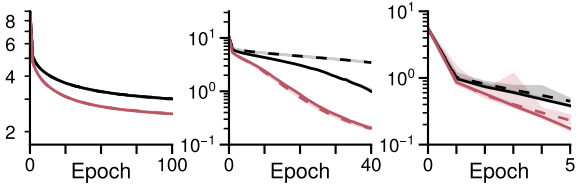


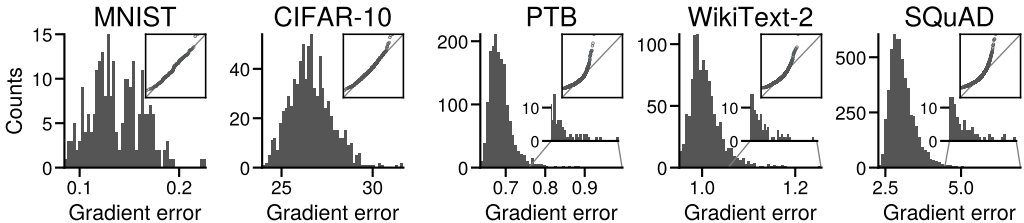

Рисунок 5: Transformers on PTB, WikiText2, and SQuAD

Черные линии - SGD; красные линии - Adam.

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

# Почему Adam работает хуже для CV, чем для LLM? [6]

Потому что шум градиентов в языковых моделях имеет тяжелые хвосты?




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

# Почему Adam работает хуже для CV, чем для LLM? [7]

Нет! Метки имеют тяжелые хвосты!

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

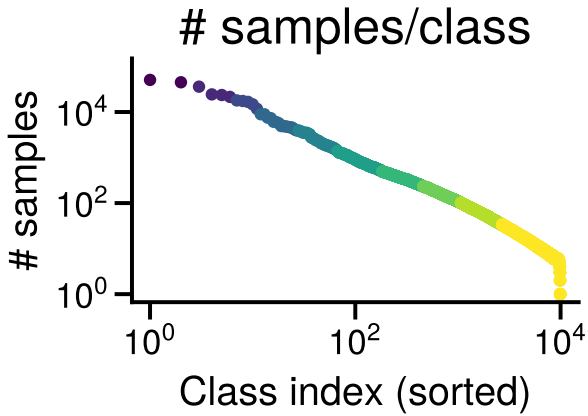
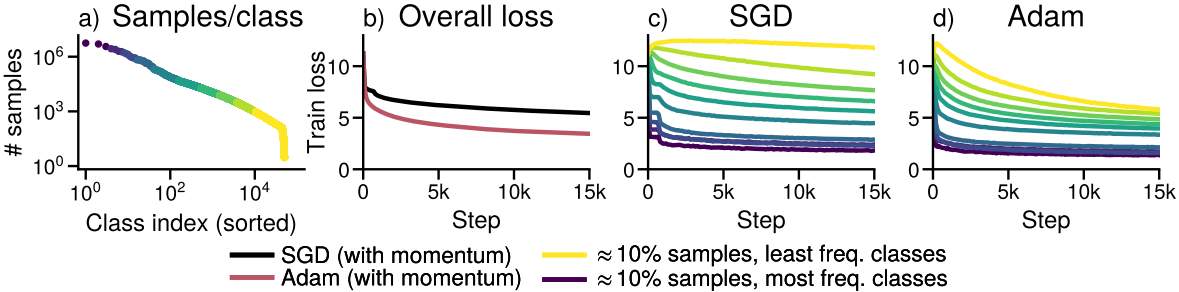



Рисунок 6: Распределение частоты токенов в PTB

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

# Почему Adam работает хуже для CV, чем для LLM? [8]

SGD медленно прогрессирует на редких классах




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

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

# Impact of initialization [9]

Properly initializing a NN important. NN loss is highly nonconvex; optimizing it to attain a "good" solution hard, requires careful tuning.

- Don't initialize all weights to be the same — why?

# Impact of initialization [9]

Properly initializing a NN important. NN loss is highly nonconvex; optimizing it to attain a "good" solution hard, requires careful tuning.

- Don't initialize all weights to be the same — why?
- Random: Initialize randomly, e.g., via the Gaussian $N(0, \sigma^2)$, where std $\sigma$ depends on the number of neurons in a given layer. *Symmetry breaking*.

# Impact of initialization [9]

Properly initializing a NN important. NN loss is highly nonconvex; optimizing it to attain a "good" solution hard, requires careful tuning.

- Don't initialize all weights to be the same — why?
- Random: Initialize randomly, e.g., via the Gaussian $N(0, \sigma^2)$, where std $\sigma$ depends on the number of neurons in a given layer. *Symmetry breaking*.
- One can find more useful advices here

[9] On the importance of initialization and momentum in deep learning Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton

# Влияние инициализации весов нейронной сети на сходимость методов [10]

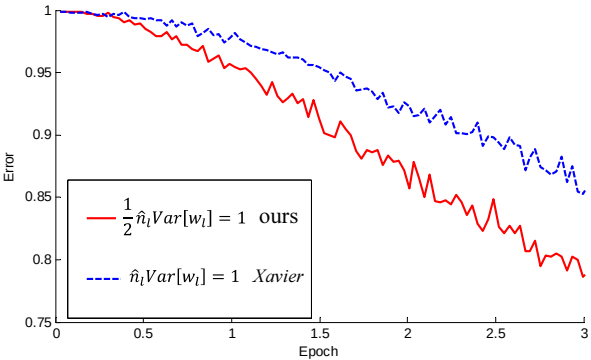



Рисунок 7: 22-layer ReLU net: good init converges faster

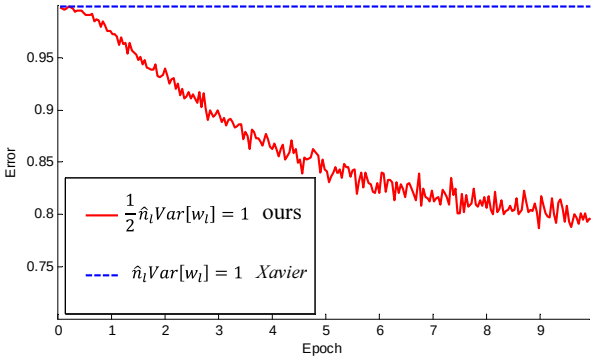
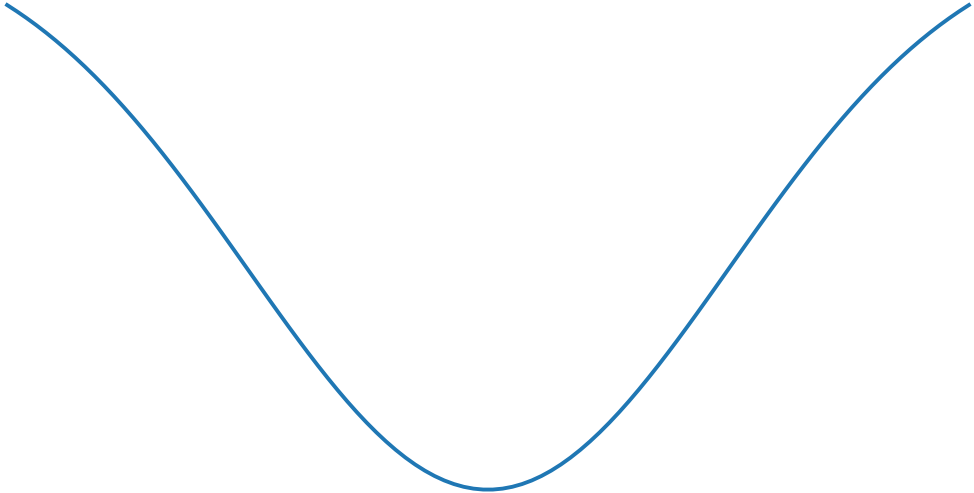



Рисунок 8: 30-layer ReLU net: good init is able to converge

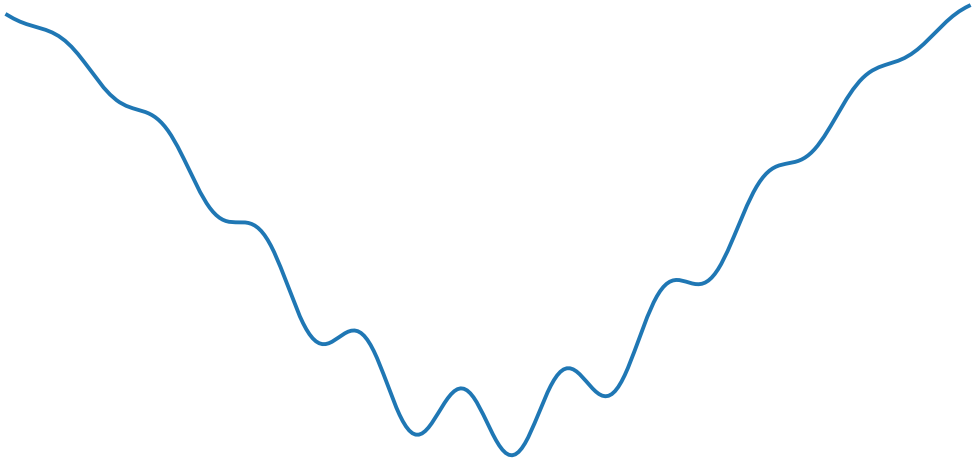
[10] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

# Funny stories

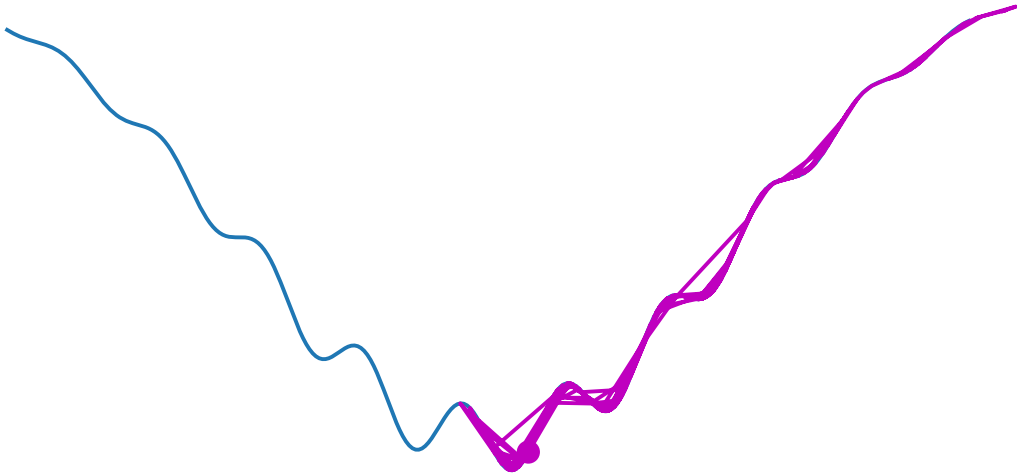
# Градиентный спуск сходится к локальному минимуму



# Градиентный спуск
# сходится к локальному минимуму



# Стохастический градиентный спуск выпрыгивает из локальных минимумов



# Визуализация с помощью проекции на прямую

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

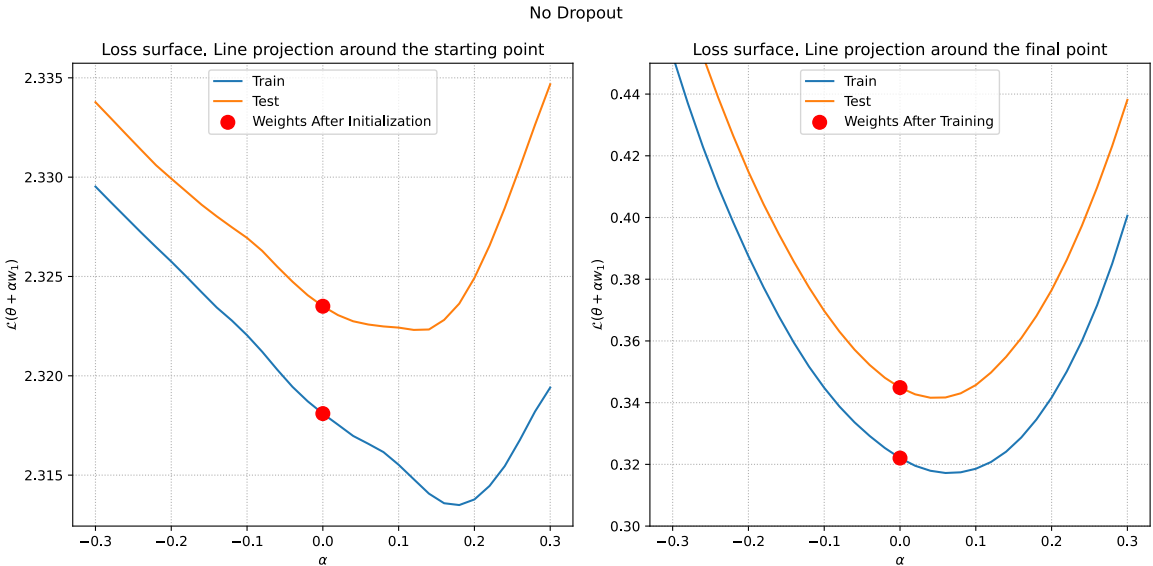
$$L(\alpha) = L(w_0 + \alpha w_1), \text{ where } \alpha \in [-b, b].$$

# Визуализация с помощью проекции на прямую

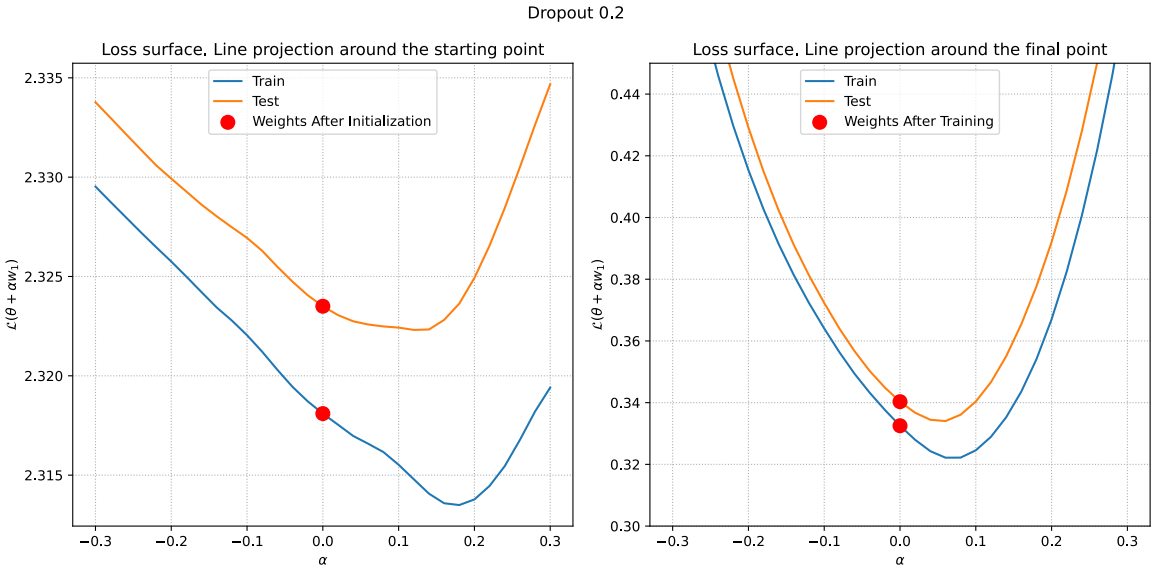
- Обозначим начальную точку как $w_0$, представляющую собой веса нейронной сети при инициализации. Веса, полученные после обучения, обозначим как $\widehat{w}$.
- Генерируем случайный вектор такой же размерности и нормы $w_1 \in \mathbb{R}^p$, затем вычисляем значение функции потерь вдоль этого вектора:

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

# Проекция функции потерь нейронной сети на прямую



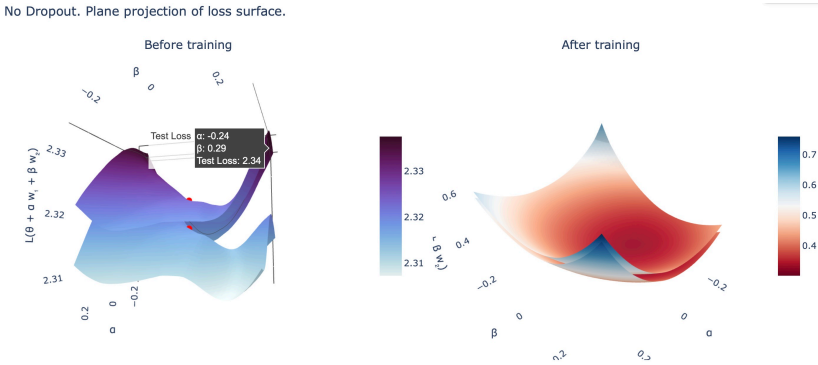
# Проекция функции потерь нейронной сети на прямую



# Проекция функции потерь нейронной сети на плоскость

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

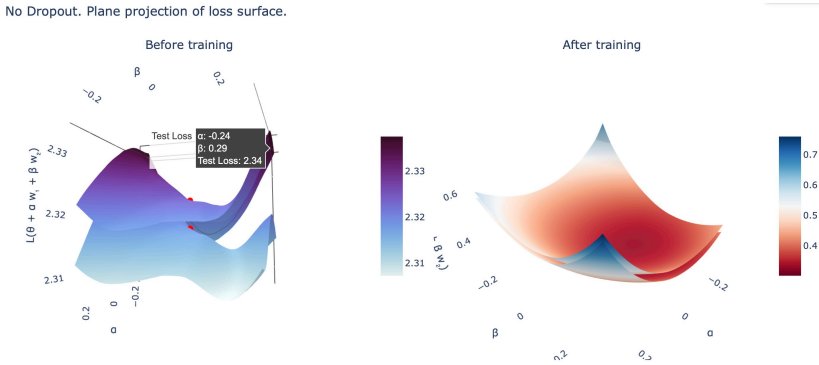
$$L(\alpha, \beta) = L(w_0 + \alpha w_1 + \beta w_2), \text{ where } \alpha, \beta \in [-b, b]^2.$$



# Проекция функции потерь нейронной сети на плоскость

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

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



# Может ли быть полезно изучение таких проекций? [11]

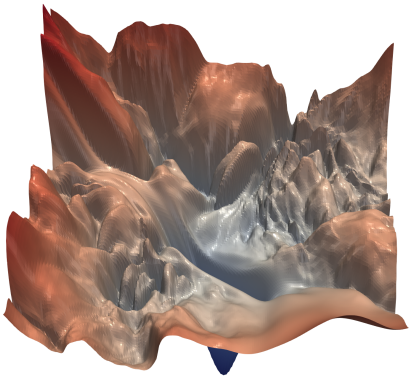


Рисунок 12: The loss surface of ResNet-56 without skip connections

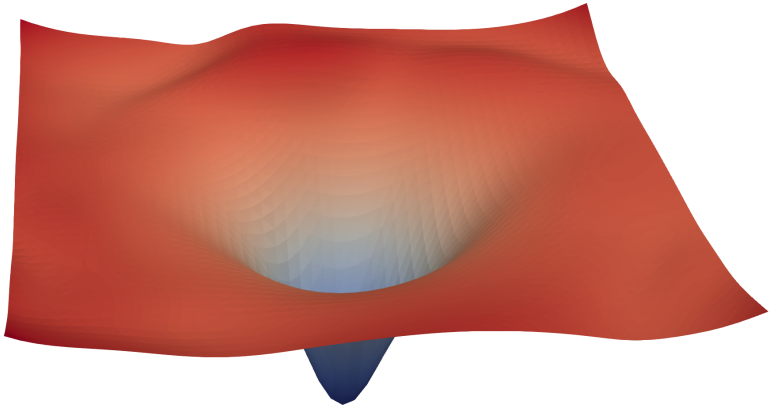


Рисунок 13: The loss surface of ResNet-56 with skip connections

[11] Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein

# Может ли быть полезно изучение таких проекций, если серьезно? [12]

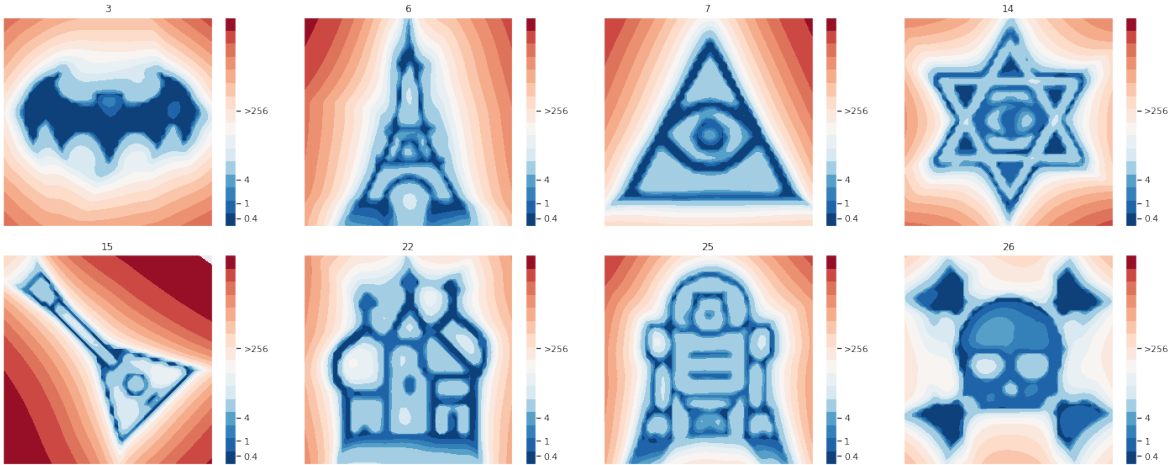
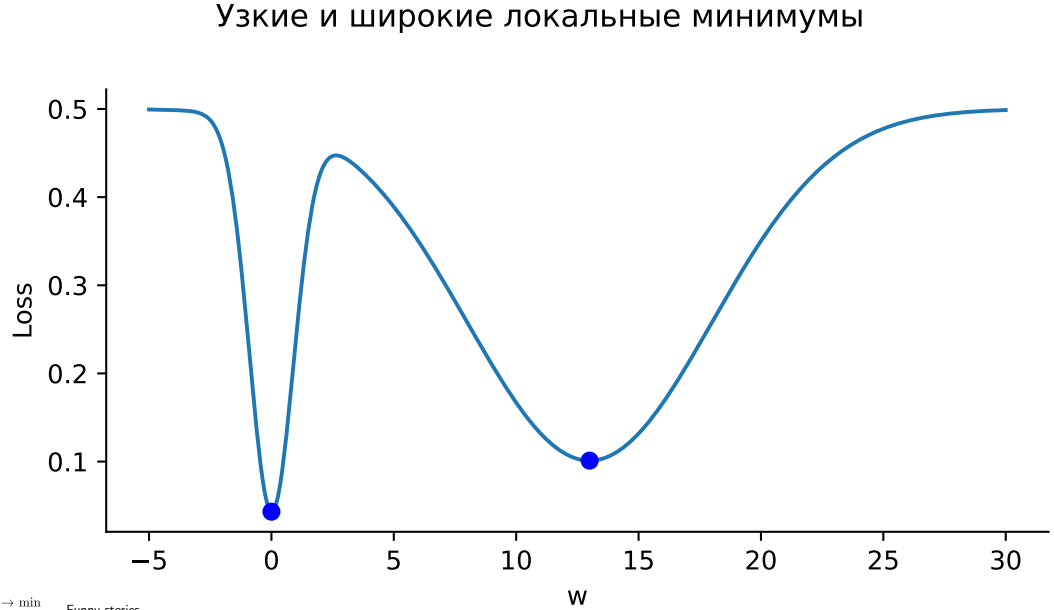


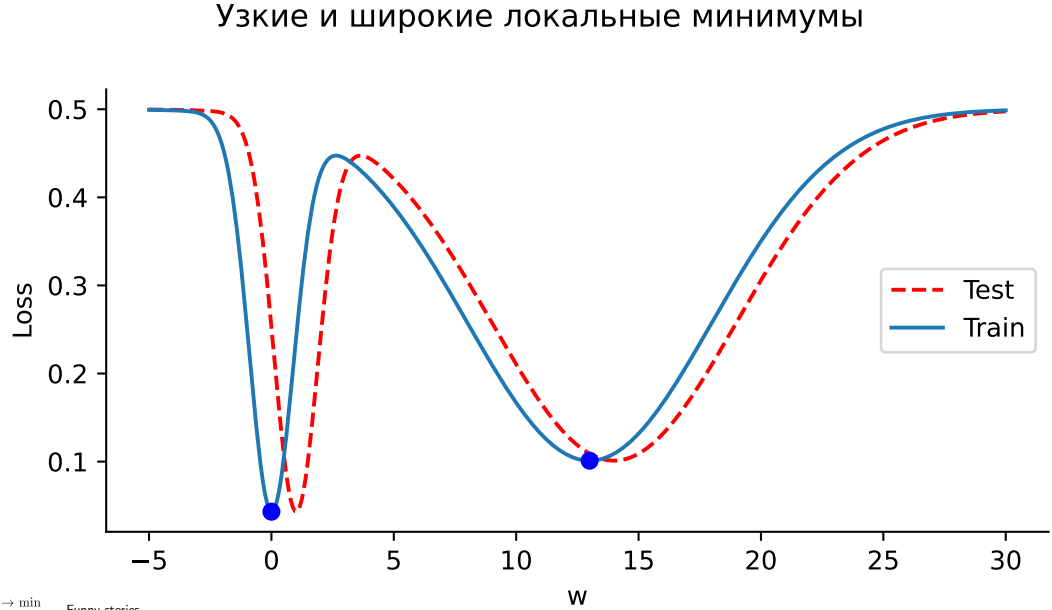

Рисунок 14: Examples of a loss landscape of a typical CNN model on FashionMNIST and CIFAR10 datasets found with MPO. Loss values are color-coded according to a logarithmic scale

[12] Loss Landscape Sightseeing with Multi-Point Optimization, Ivan Skorokhodov, Mikhail Burtsev

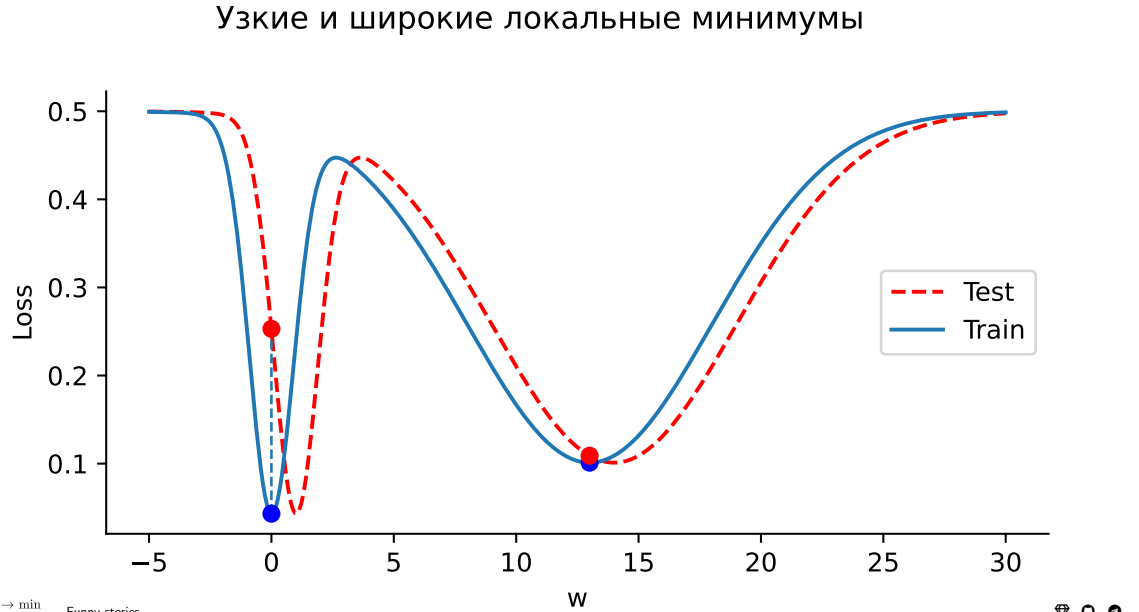
## Ширина локальных минимумов



## Ширина локальных минимумов



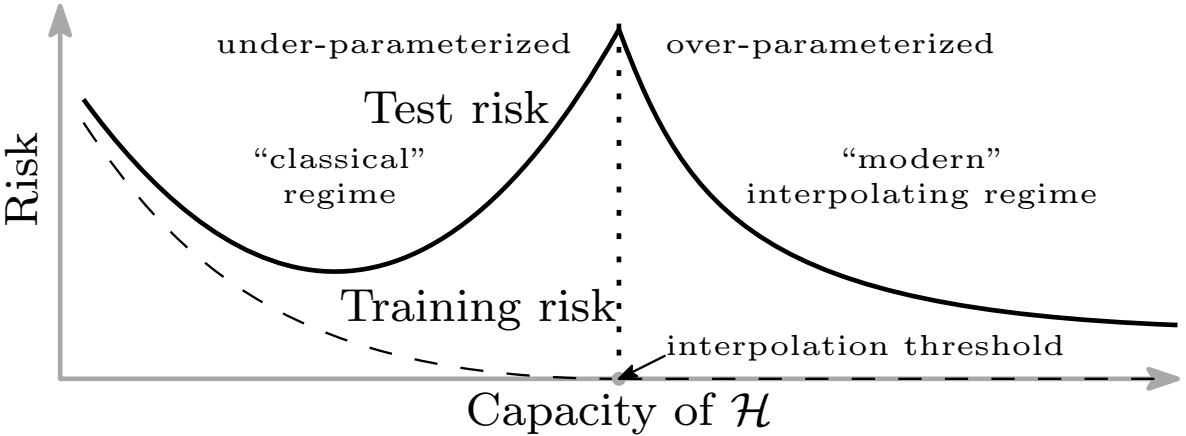
## Ширина локальных минимумов



# Exponential learning rate

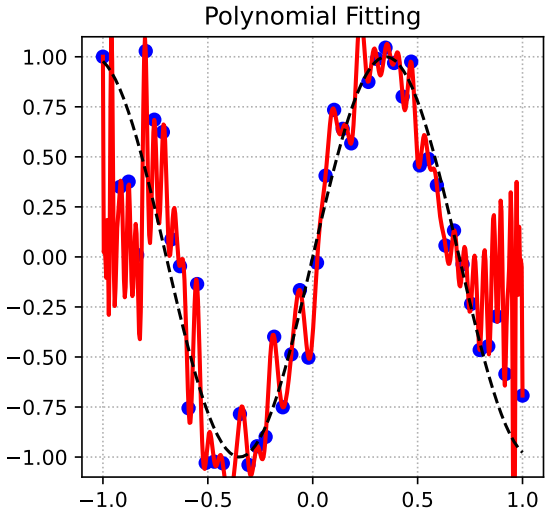
- Exponential Learning Rate Schedules for Deep Learning

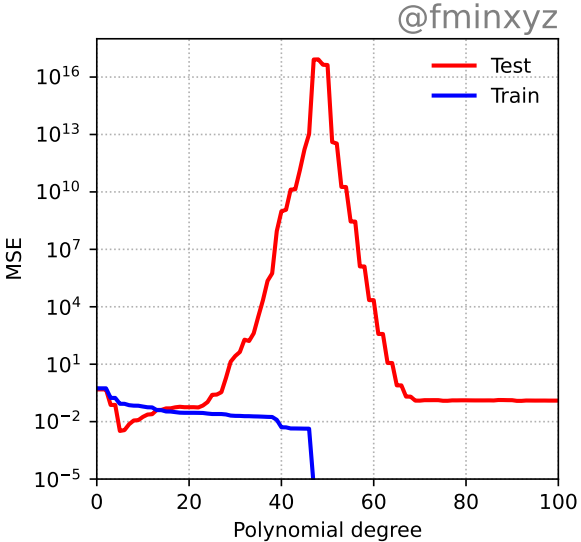
# Double Descent [13]




[13] Reconciling modern machine learning practice and the bias-variance trade-off, Mikhail Belkin, Daniel Hsu, Siyuan Ma, Soumik Mandal

# Double Descent



# Grokking [14]

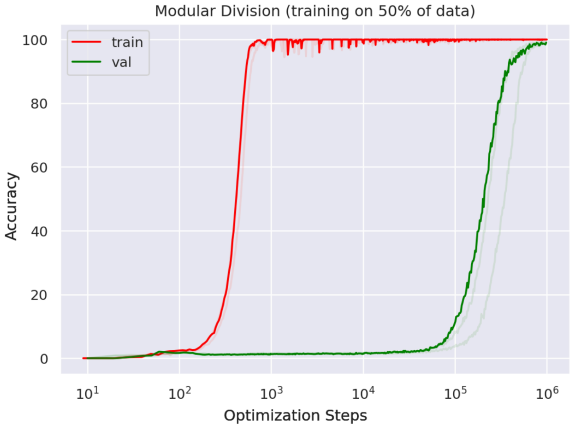



Рисунок 15: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about $4 \cdot 10^5$ non-embedding parameters. Reproduction of experiments (~ half an hour) is available here

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

# Grokking [14]

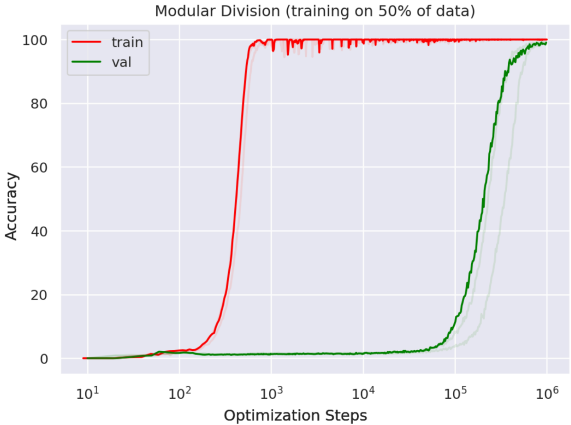



Рисунок 15: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about $4 \cdot 10^5$ non-embedding parameters. Reproduction of experiments (~ half an hour) is available here

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

# Grokking [14]

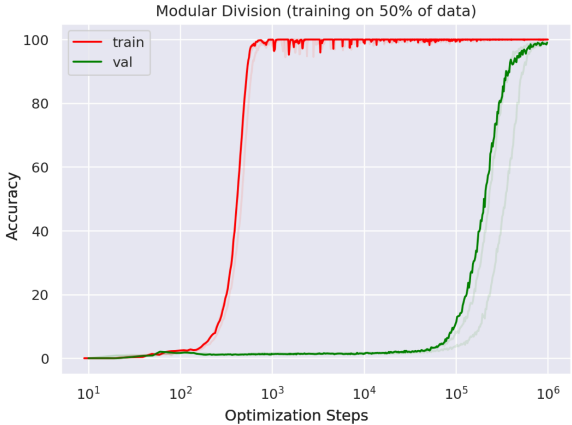



Рисунок 15: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about $4 \cdot 10^5$ non-embedding parameters. Reproduction of experiments (~ half an hour) is available here

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

[14] Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, Vedant Misra