



# Optimization for Deep Learning

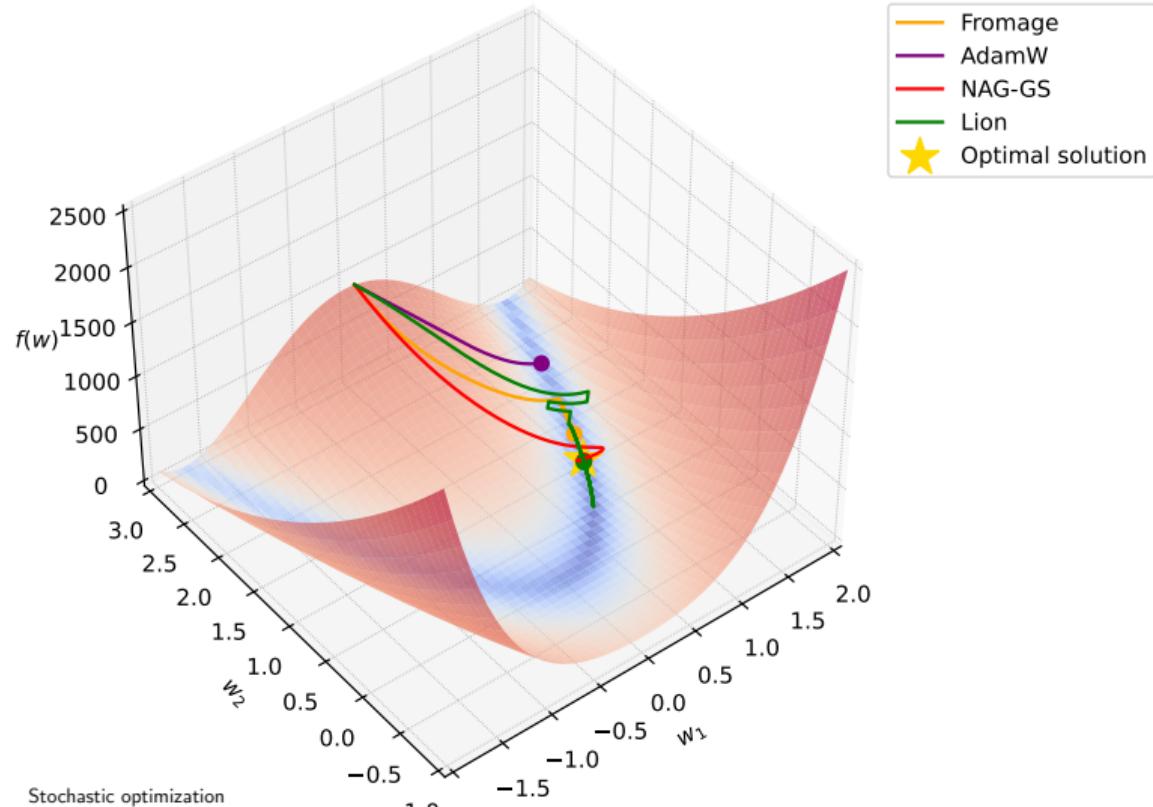
Даня Меркулов

Методы Оптимизации в Машинном Обучении. ФКН ВШЭ

## Stochastic optimization

# A lot of them

Rosenbrock Function.  
Adaptive stochastic gradient algorithms.  
Learning rate 0.003



## SGD diverges with any learning rate for LLS

## Optimization for Deep Learning from the practical perspective

## How to compare them? AlgoPerf benchmark<sup>1 2</sup>

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

## How to compare them? AlgoPerf benchmark<sup>1 2</sup>

- **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<sup>1 2</sup>

- **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<sup>1 2</sup>

- **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<sup>1 2</sup>

- **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).

---

<sup>1</sup>Benchmarking Neural Network Training Algorithms

<sup>2</sup>Accelerating neural network training: An analysis of the AlgoPerf competition

## AlgoPerf benchmark

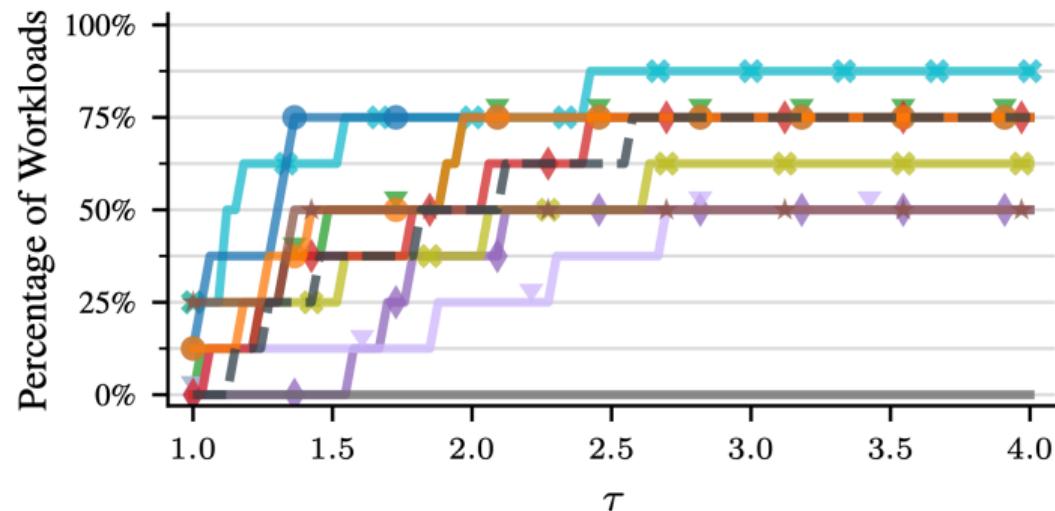
**Summary of fixed base workloads in the AlgoPerf benchmark.** Losses include cross-entropy (CE), mean absolute error (L1), and Connectionist Temporal Classification loss (CTC). Additional evaluation metrics are structural similarity index measure (SSIM), (word) error rate (ER & WER), mean average precision (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

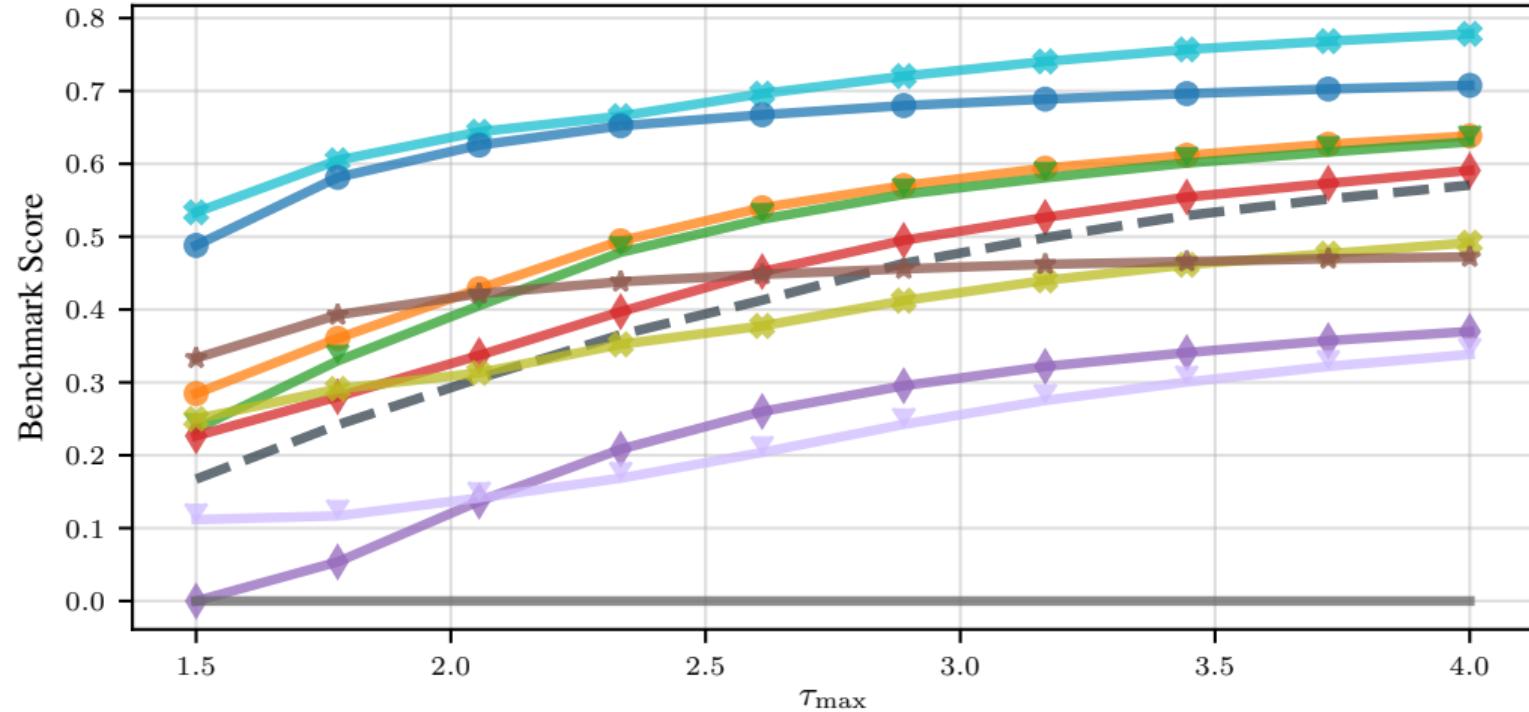
Submission	Line	Score
PYTORCH DISTRIBUTED SHAMPOO		0.7784
SCHEDULE FREE ADAMW		0.7077
GENERALIZED ADAM		0.6383
CYCLIC LR		0.6301
NADAMP		0.5909
BASELINE		0.5707
AMOS		0.4918
CASPR ADAPTIVE		0.4722
LAWA QUEUE		0.3699
LAWA EMA		0.3384
SCHEDULE FREE PRODIGY		0

(a) External tuning leaderboard



(b) External tuning performance profiles

# AlgoPerf benchmark

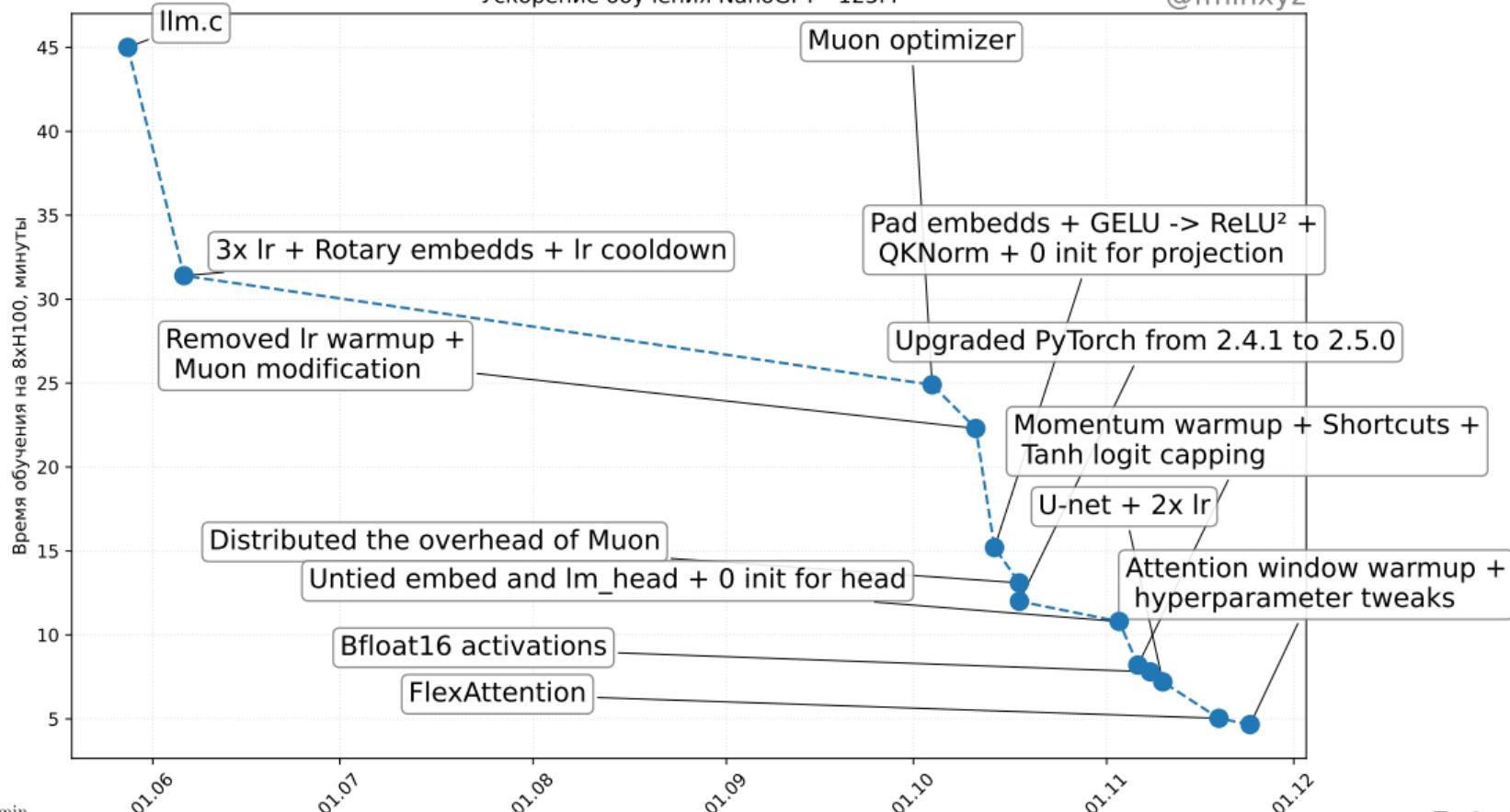


- PyTorch Distr. Shampoo
- Schedule Free AdamW
- Generalized Adam
- Cyclic LR
- NadamP
- Baseline
- Amos
- CASPR Adaptive
- Lawa Queue
- Lawa EMA
- Schedule Free Prodigy

# NanoGPT speedrun

Ускорение обучения NanoGPT - 125M

@fminxyz



## Работают ли трюки, если увеличить размер модели?

Scaling up the NanoGPT (124M) speedrun

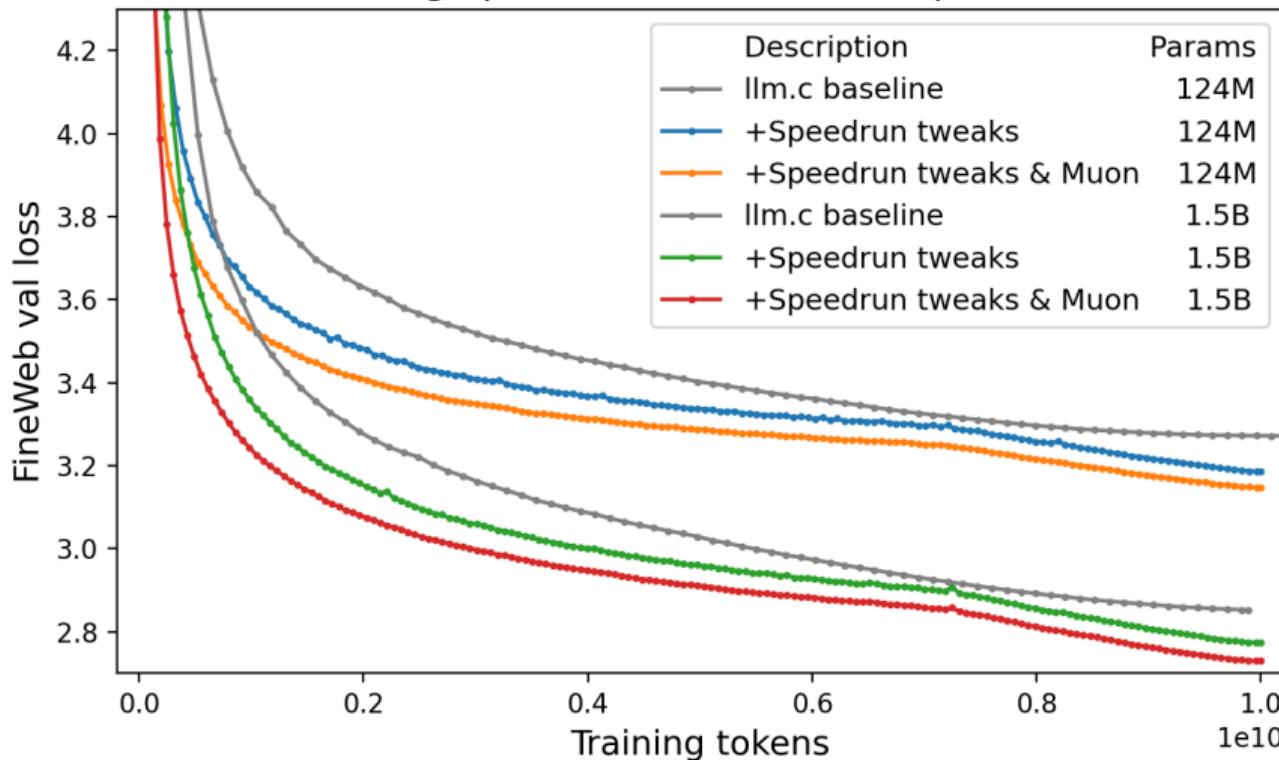


Рисунок 2: Источник

## Работают ли трюки, если увеличить размер модели?

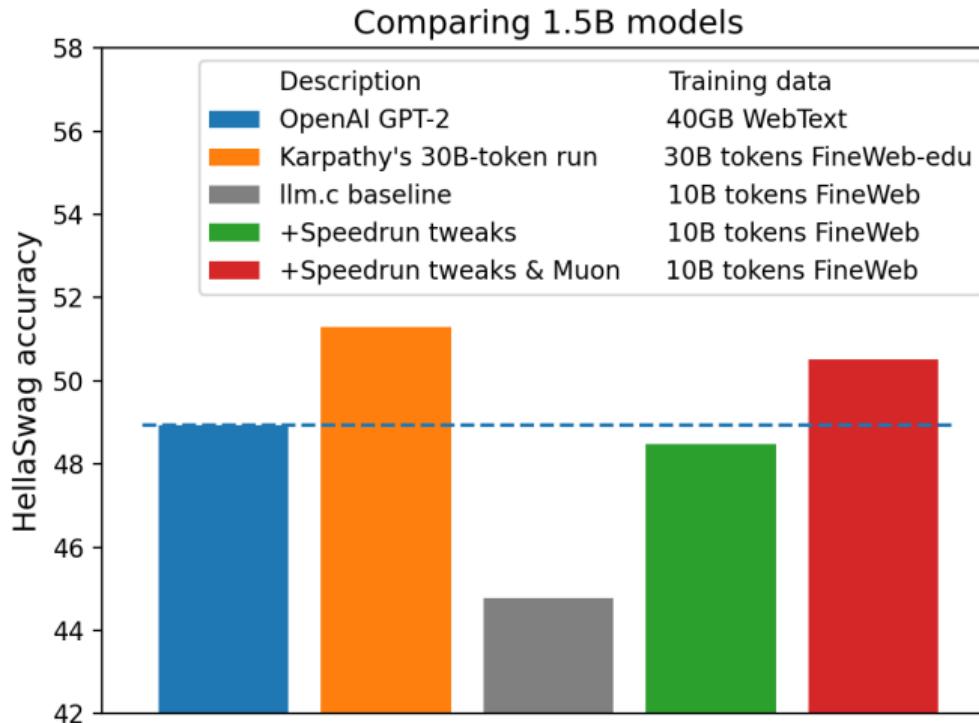


Рисунок 3: Источник

## Shampoo<sup>3</sup>

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<sup>3</sup>

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<sup>3</sup>

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<sup>3</sup>

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<sup>3</sup>

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<sup>3</sup>

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<sup>3</sup>

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<sup>3</sup>

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).

<sup>3</sup>Shampoo: Preconditioned Stochastic Tensor Optimization  
 $f \rightarrow \min_{x,y,z}$  Optimization for Deep Learning from the practical perspective

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

---

<sup>4</sup>Deriving Muon

## Unexpected stories

## Adam работает хуже для CV, чем для LLM? <sup>5</sup>

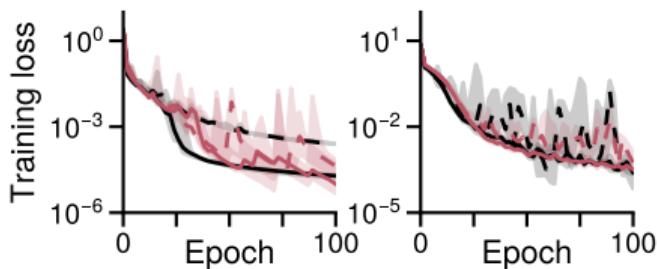


Рисунок 4: CNNs on MNIST and CIFAR10

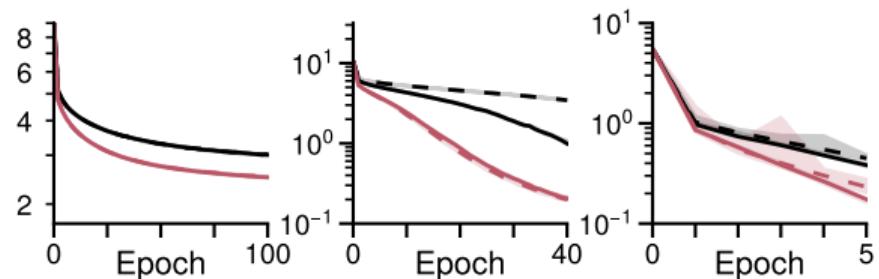


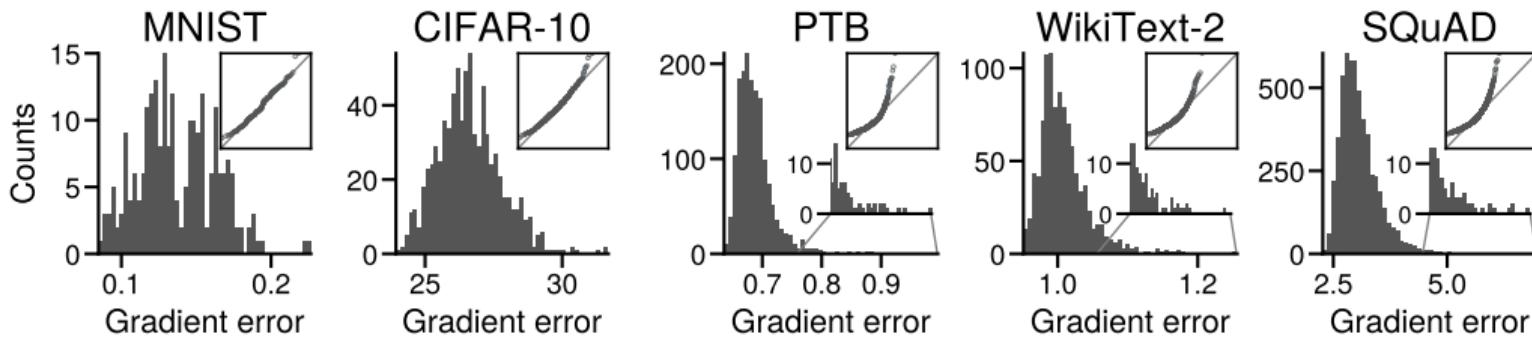
Рисунок 5: Transformers on PTB, WikiText2, and SQuAD

Черные линии - SGD; красные линии - Adam.

<sup>5</sup>Linear attention is (maybe) all you need (to understand transformer optimization)

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

Потому что шум градиентов в языковых моделях имеет тяжелые хвосты?



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

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

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

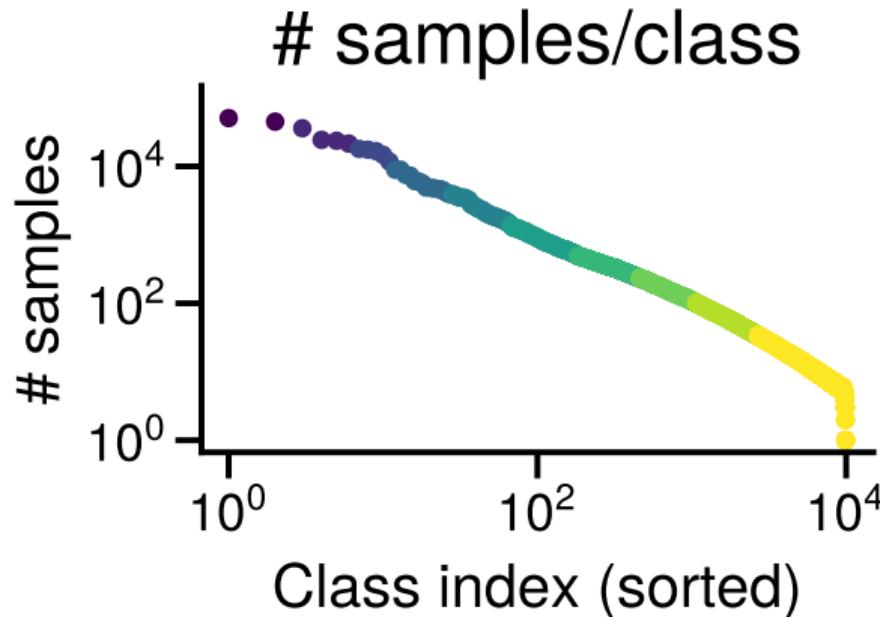
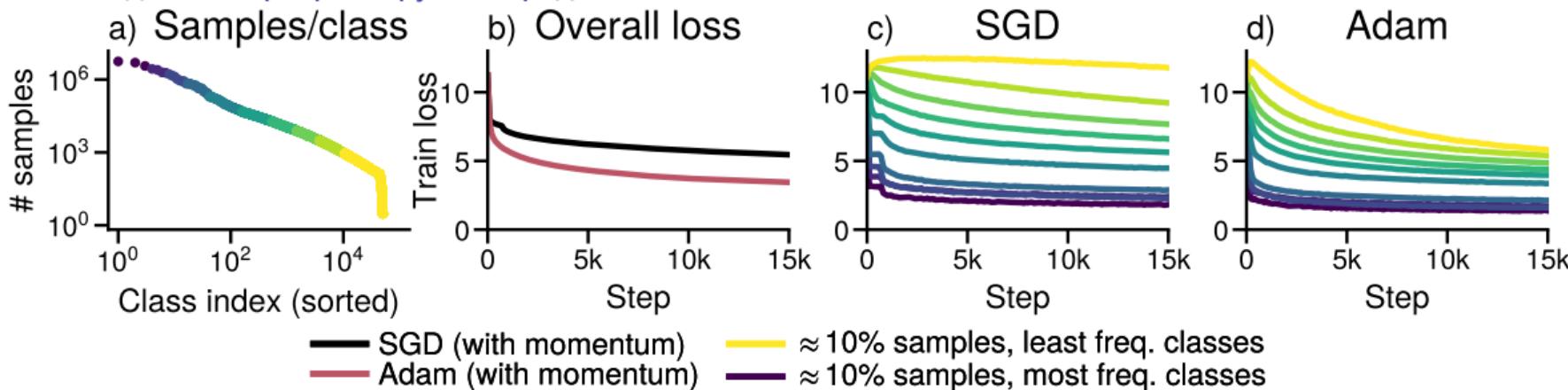


Рисунок 6: Распределение частоты токенов в PTB

<sup>7</sup>Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models

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

SGD медленно прогрессирует на редких классах



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

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

## Impact of initialization <sup>9</sup>

 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 <sup>9</sup>

💡 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 <sup>9</sup>

💡 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

<sup>9</sup>On the importance of initialization and momentum in deep learning Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton

# Влияние инициализации весов нейронной сети на сходимость методов<sup>10</sup>

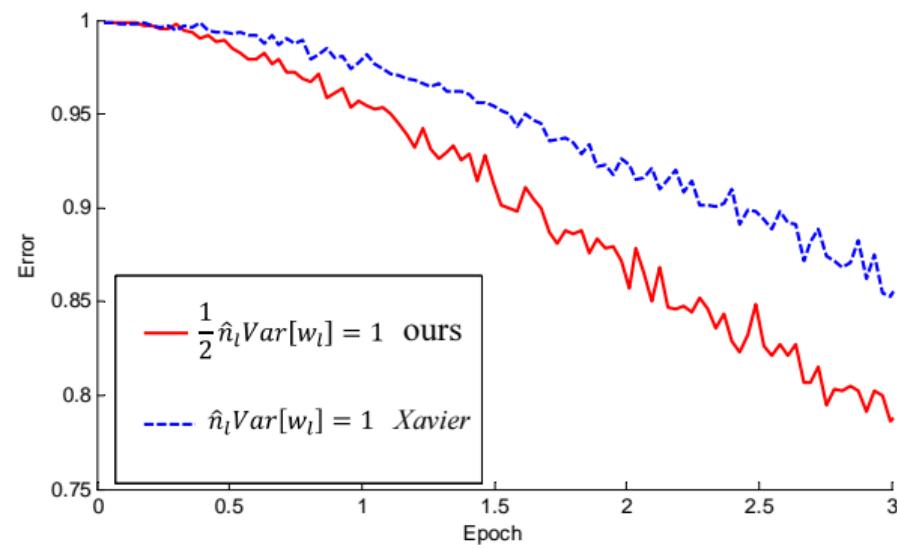


Рисунок 7: 22-layer ReLU net: good init converges faster

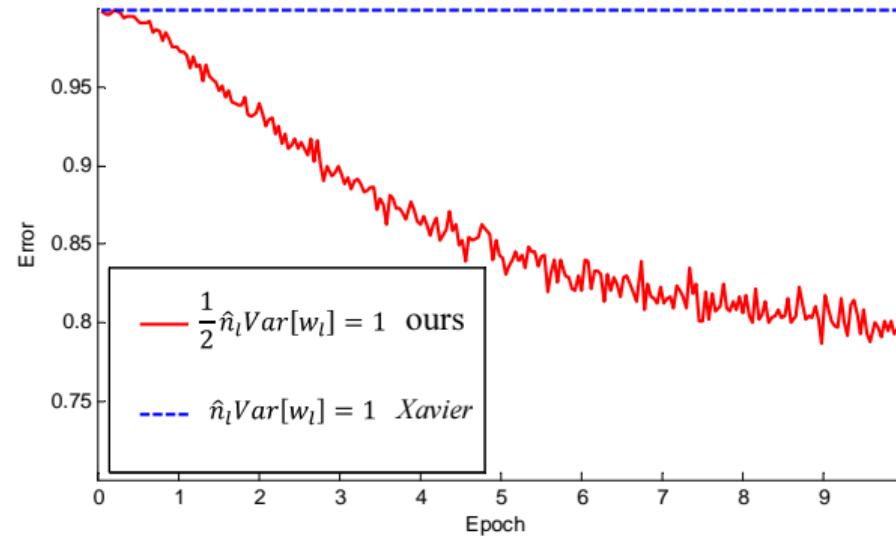
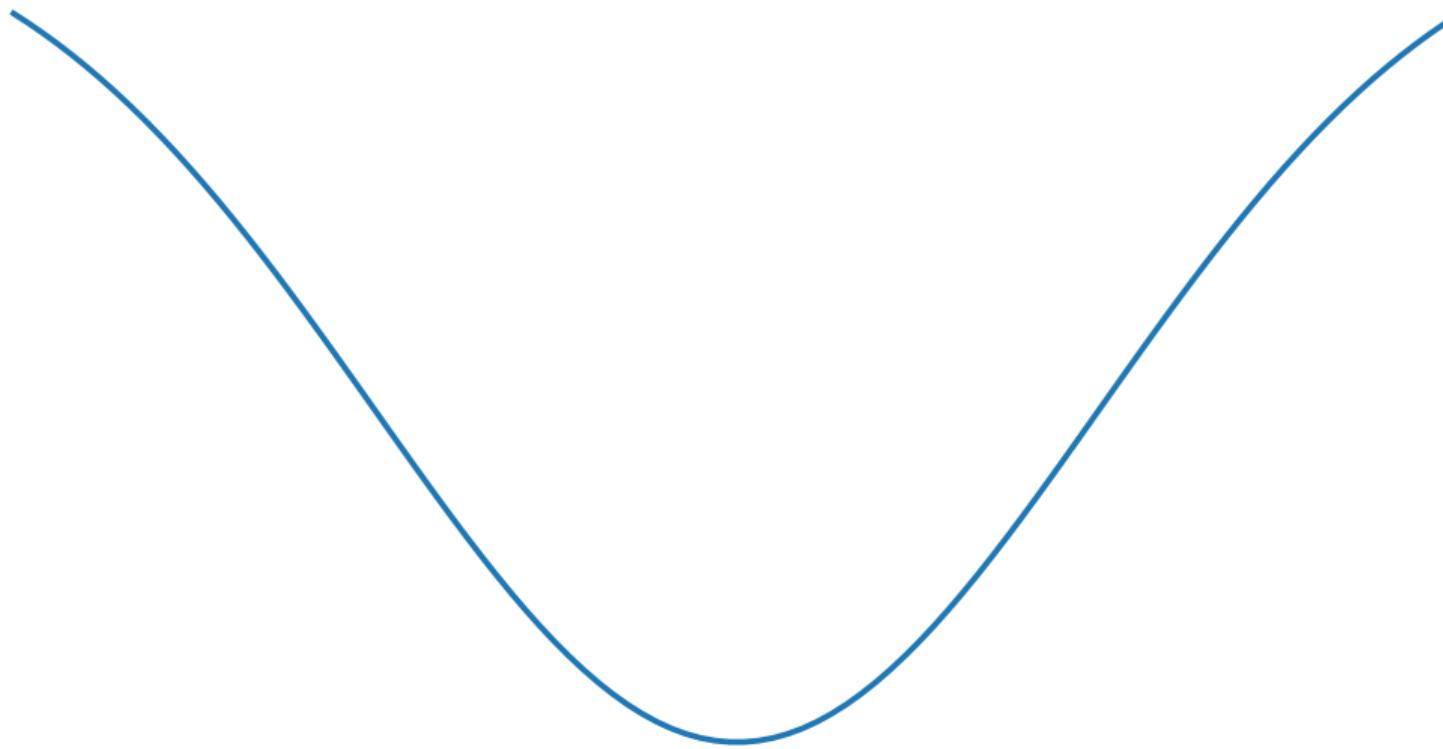


Рисунок 8: 30-layer ReLU net: good init is able to converge

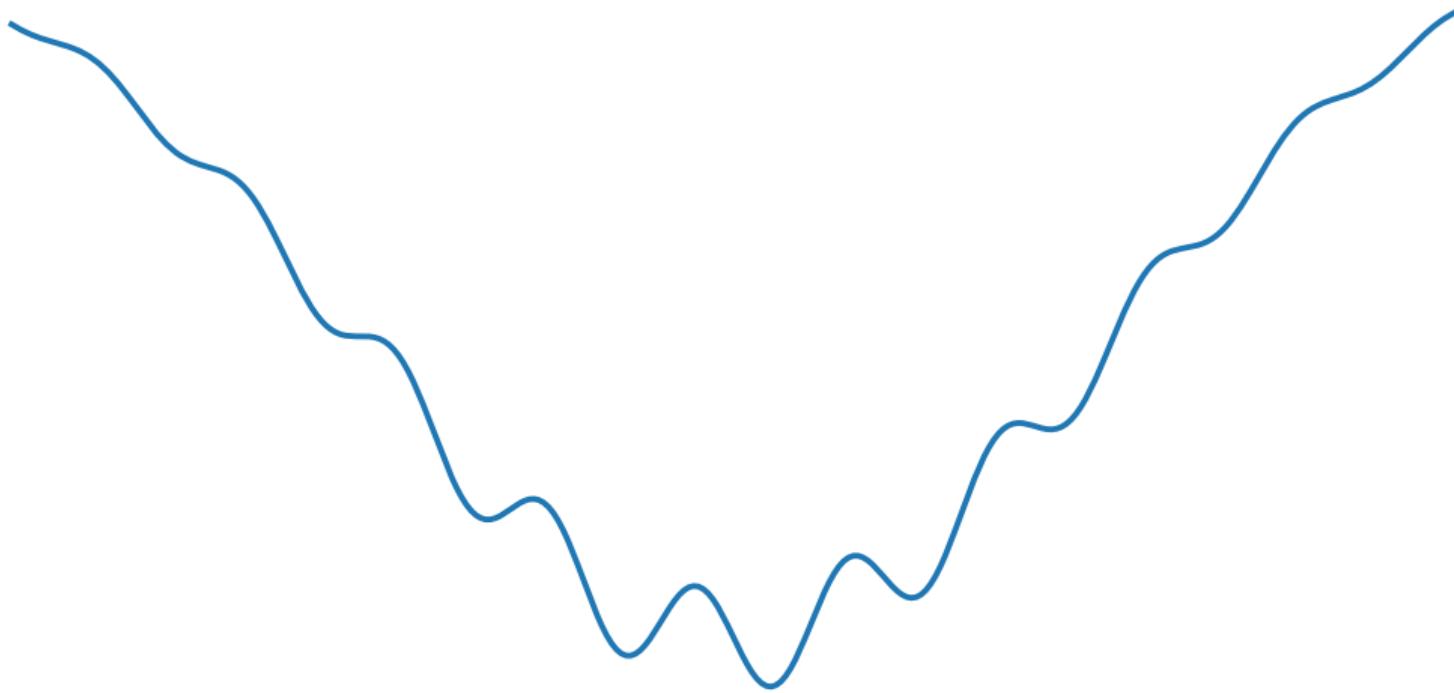
<sup>10</sup>Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

## Funny stories

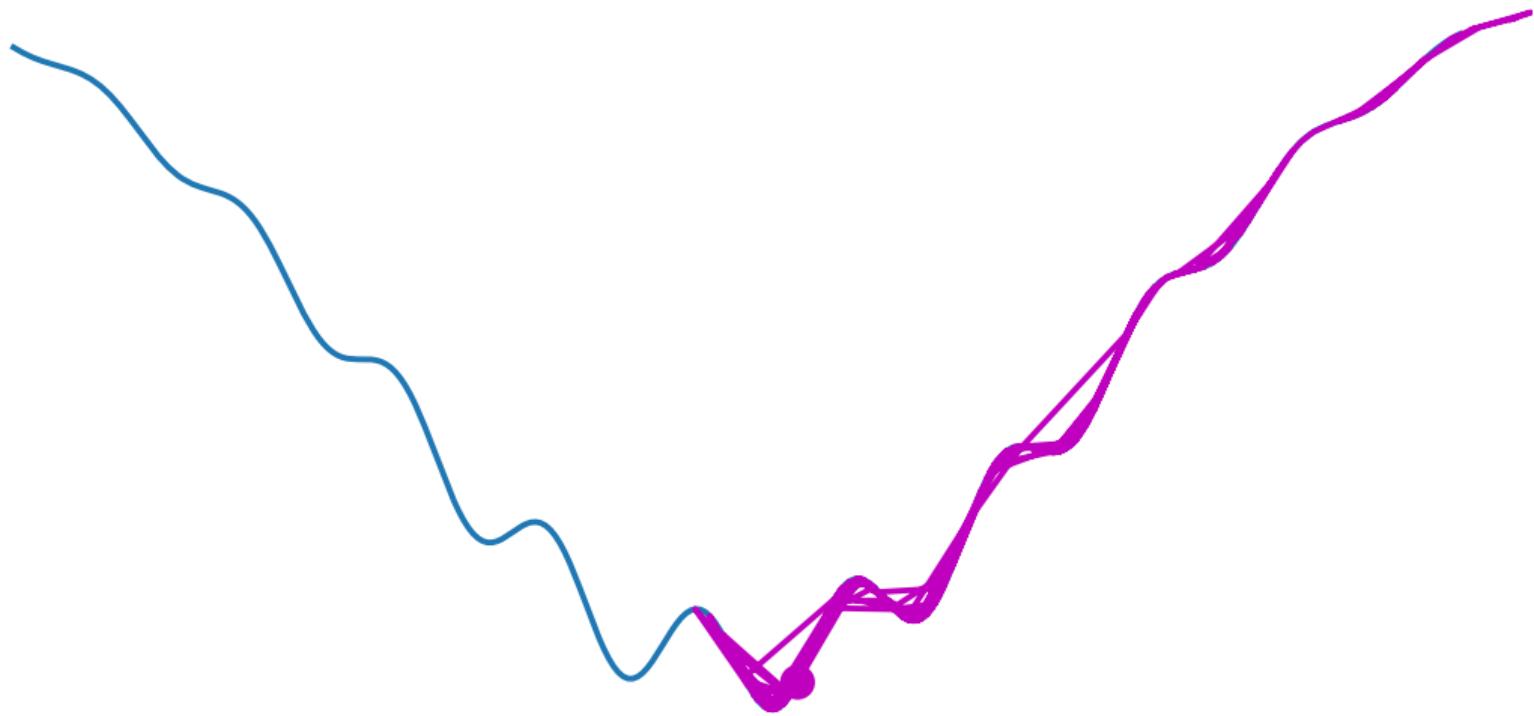
Градиентный спуск сходится к локальному минимуму



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  $\hat{w}$ .

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

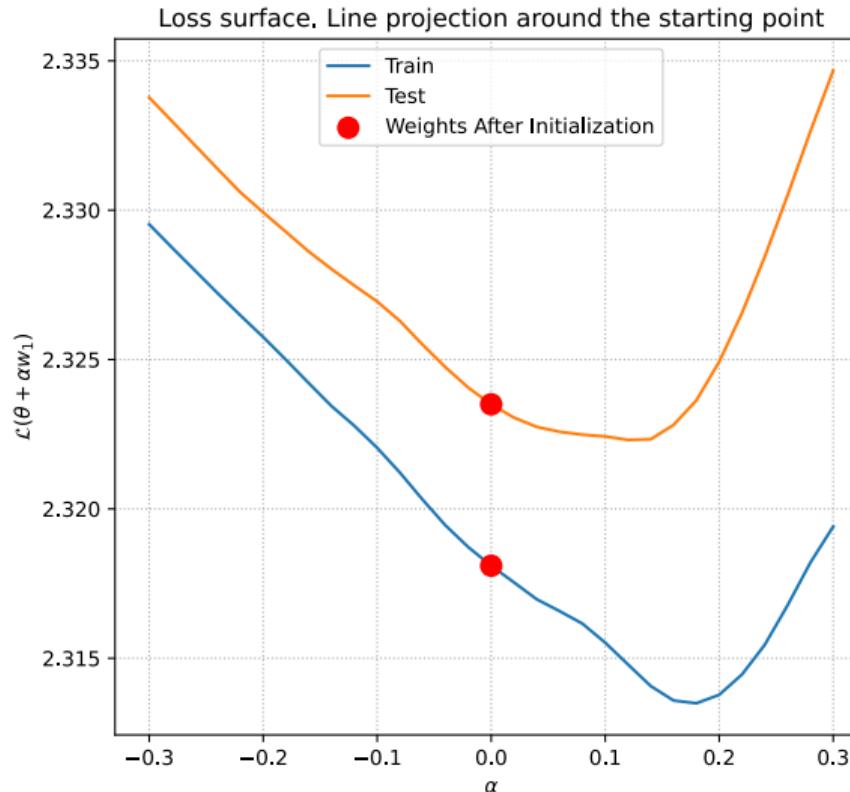
## Визуализация с помощью проекции на прямую

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

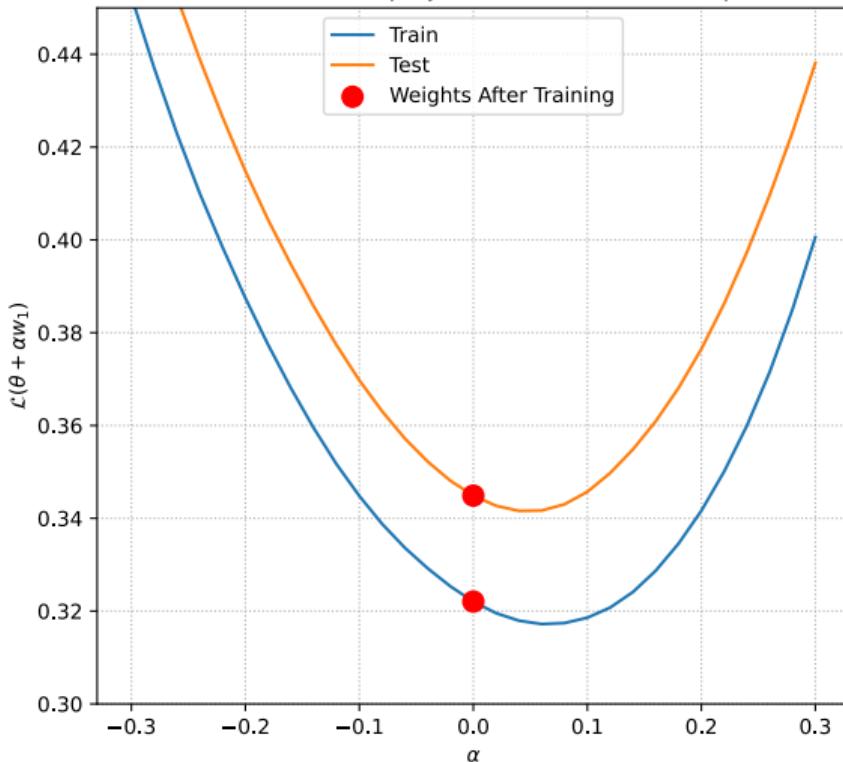
$$L(\alpha) = L(w_0 + \alpha w_1), \text{ where } \alpha \in [-b, b].$$

# Проекция функции потерь нейронной сети на прямую

No Dropout



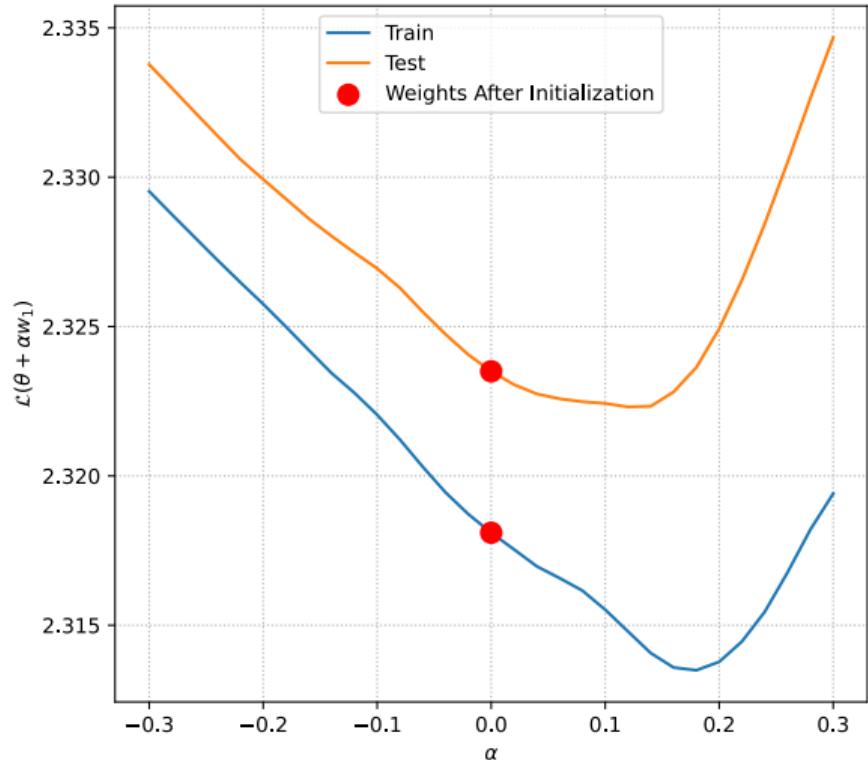
Loss surface, Line projection around the final point



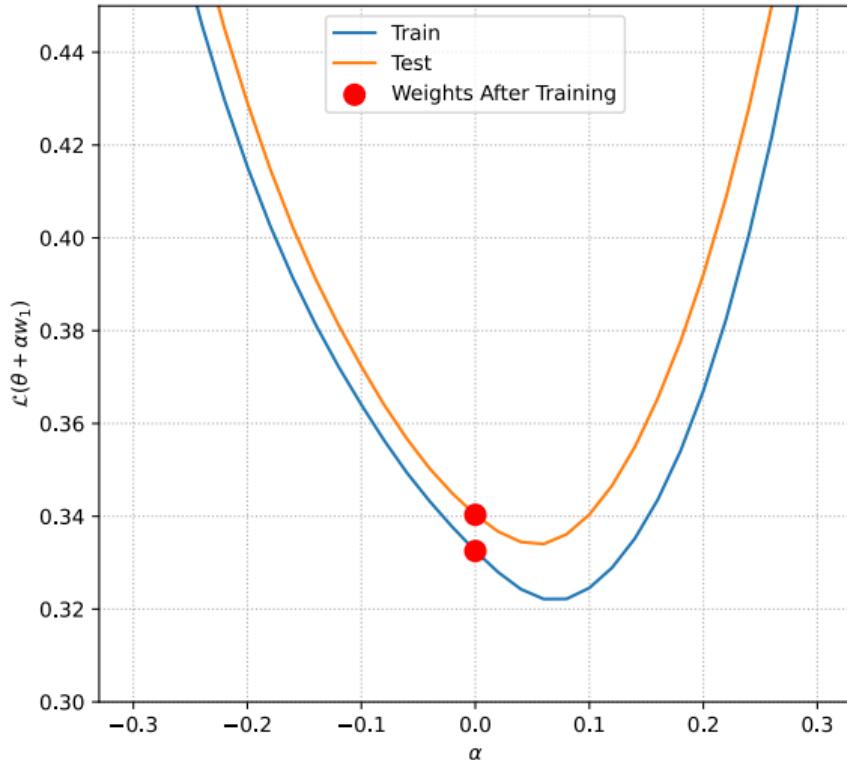
# Проекция функции потерь нейронной сети на прямую

Dropout 0.2

Loss surface, Line projection around the starting point



Loss surface, Line projection around the final point

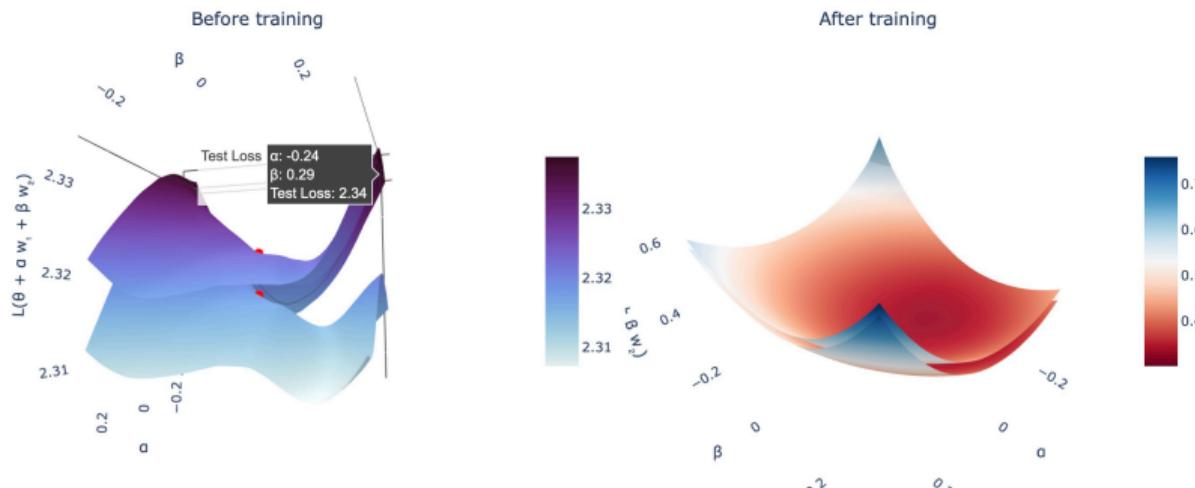


# Проекция функции потерь нейронной сети на плоскость

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

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

No Dropout. Plane projection of loss surface.

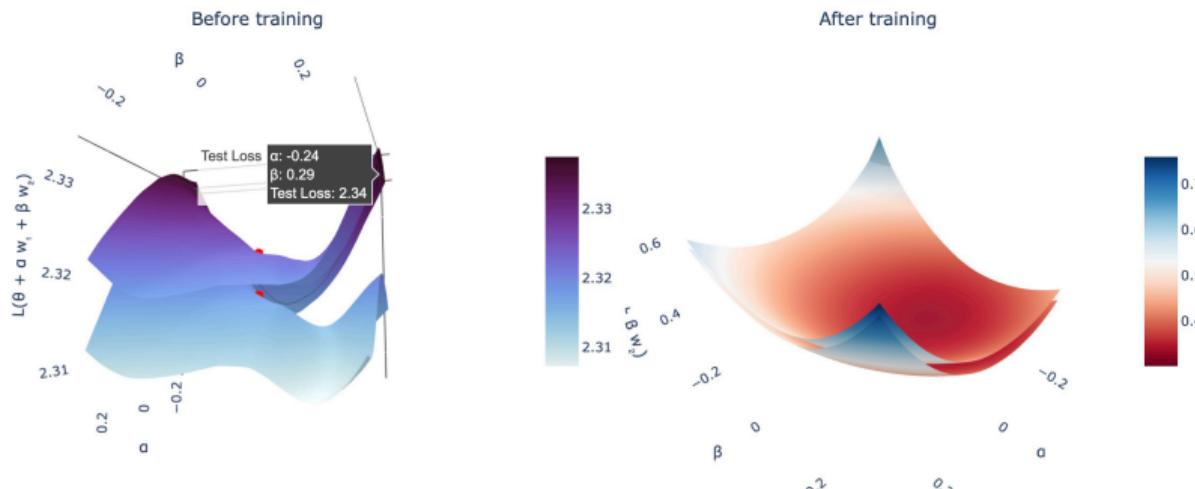


# Проекция функции потерь нейронной сети на плоскость

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

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

No Dropout. Plane projection of loss surface.



Может ли быть полезно изучение таких проекций? <sup>11</sup>

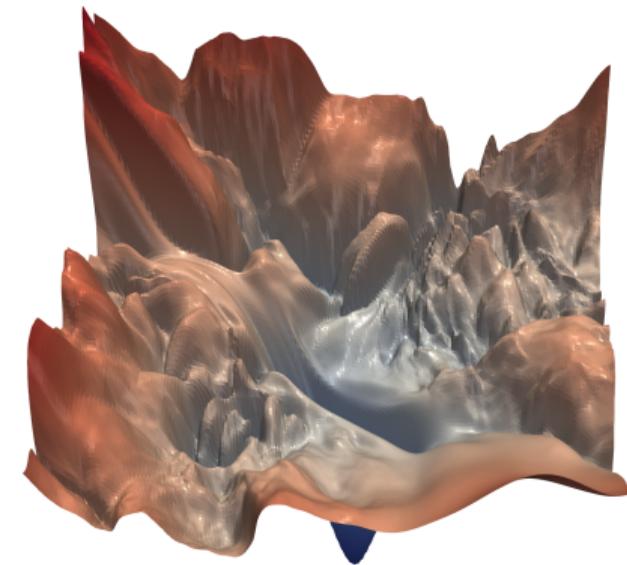


Рисунок 12: The loss surface of ResNet-56  
without skip connections

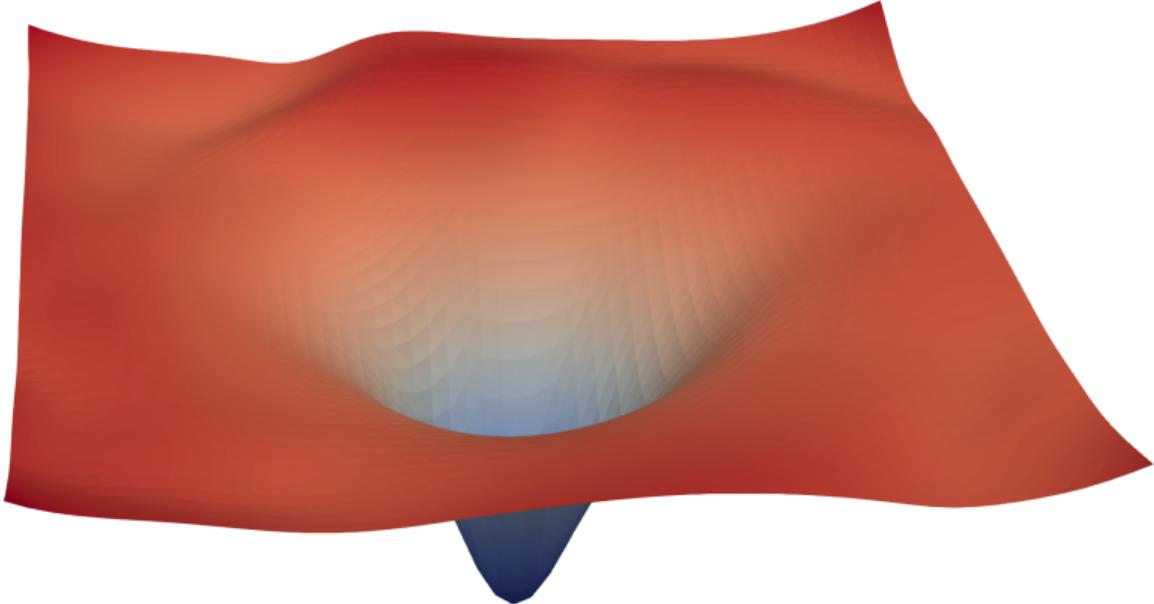


Рисунок 13: The loss surface of ResNet-56 with skip connections

<sup>11</sup>Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein

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

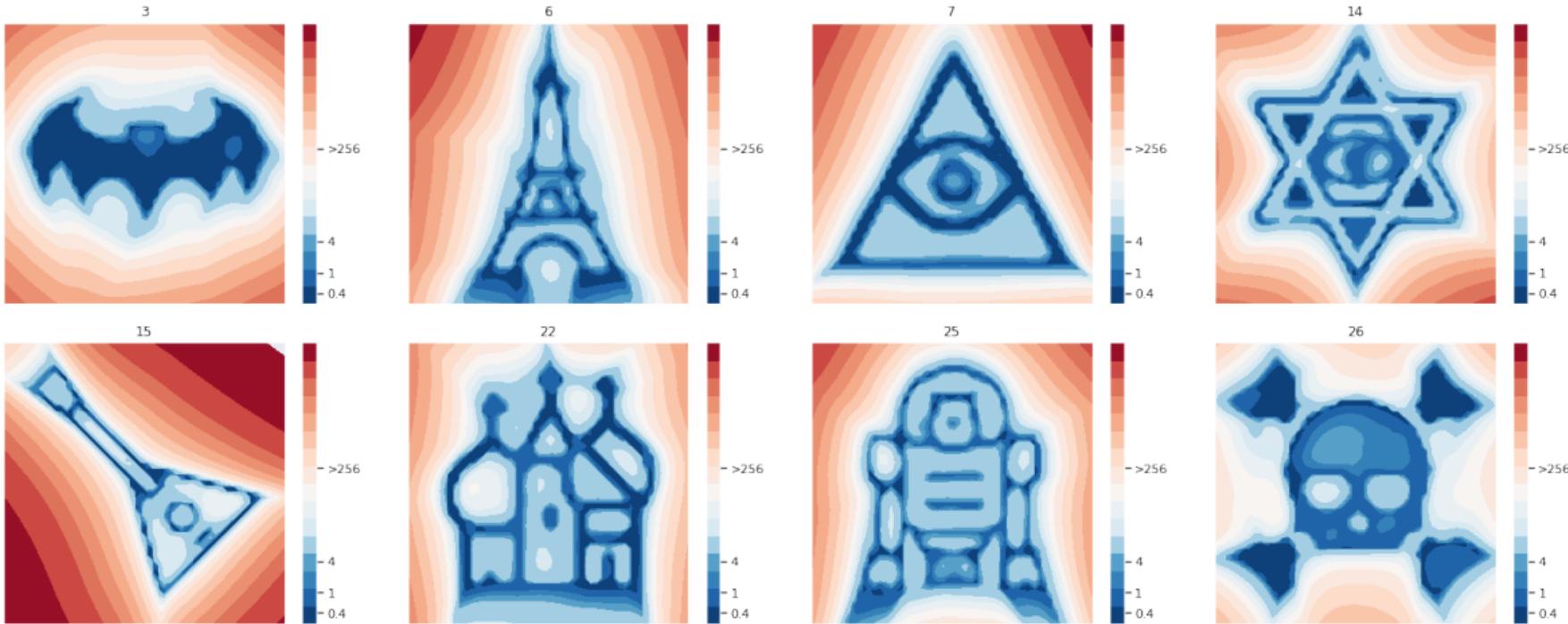
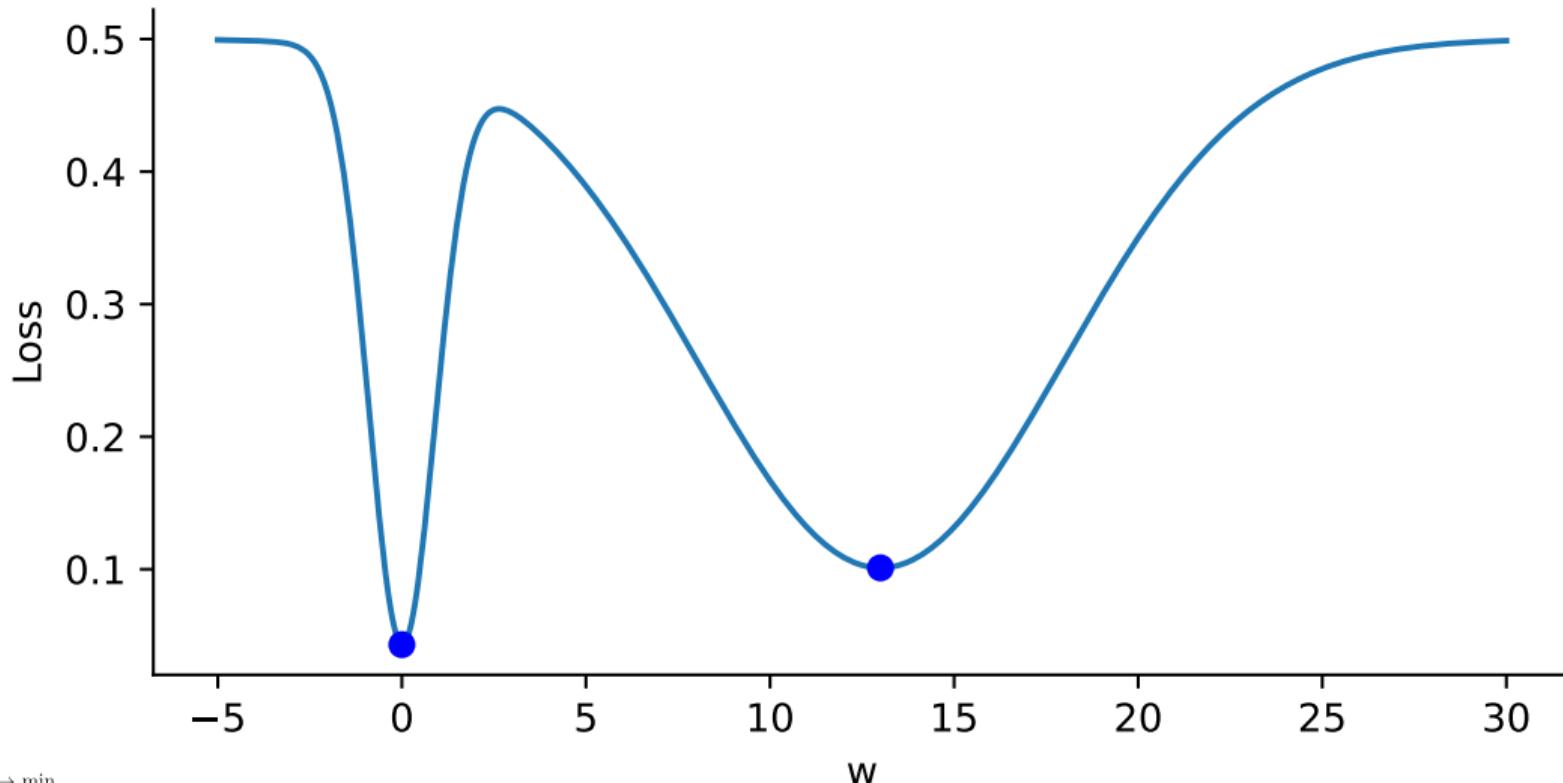


Рисунок 14: Examples of a loss landscape of a typical CNN model on FashionMNIST and CIFAR10 datasets found with MPO. Loss values are color-coded according to a logarithmic scale

<sup>12</sup>Loss Landscape Sightseeing with Multi-Point Optimization, Ivan Skorokhodov, Mikhail Burtsev  
 $f \rightarrow \min_{x,y,z}$  Funny stories

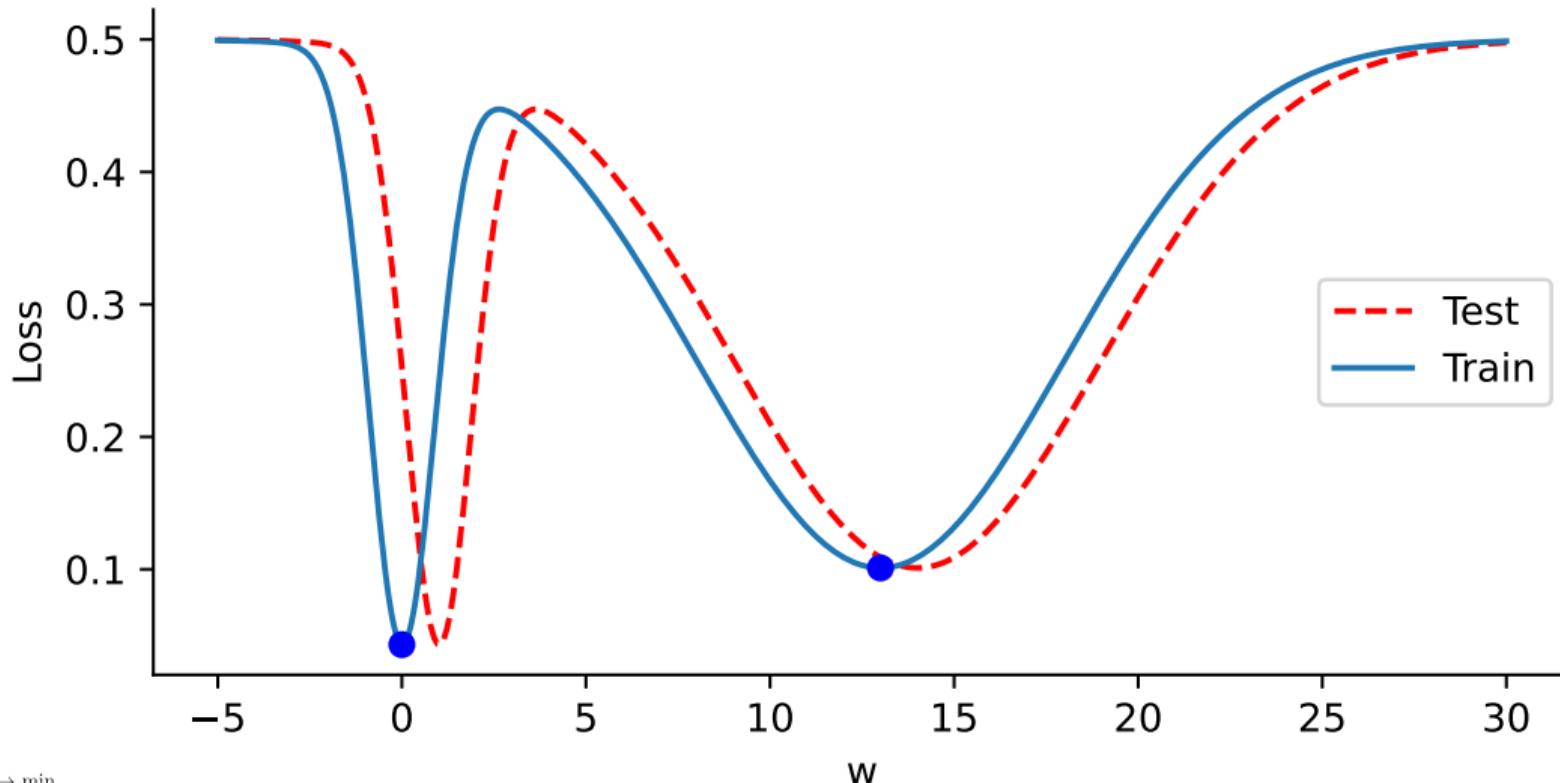
## Ширина локальных минимумов

Узкие и широкие локальные минимумы



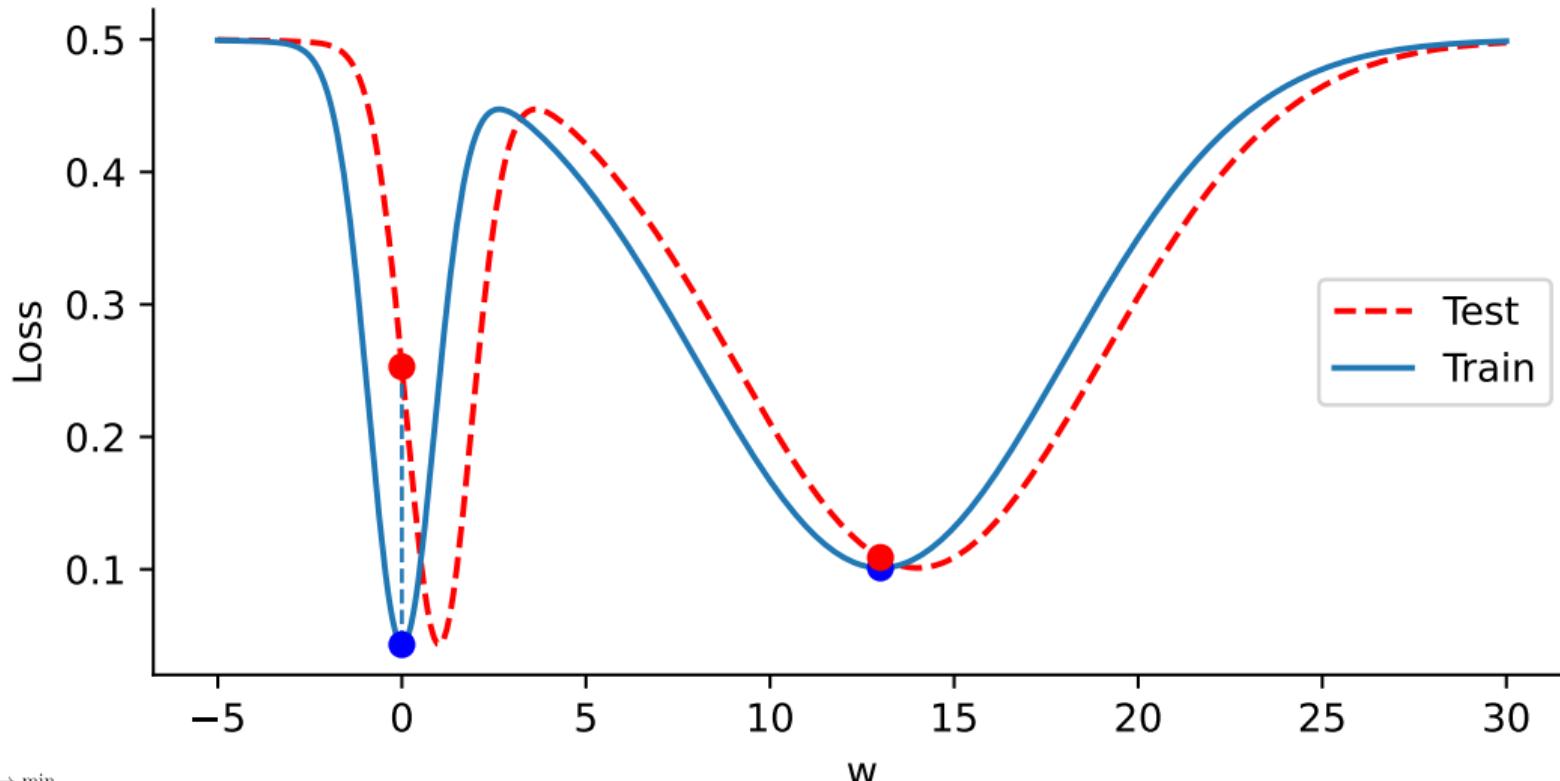
## Ширина локальных минимумов

Узкие и широкие локальные минимумы



## Ширина локальных минимумов

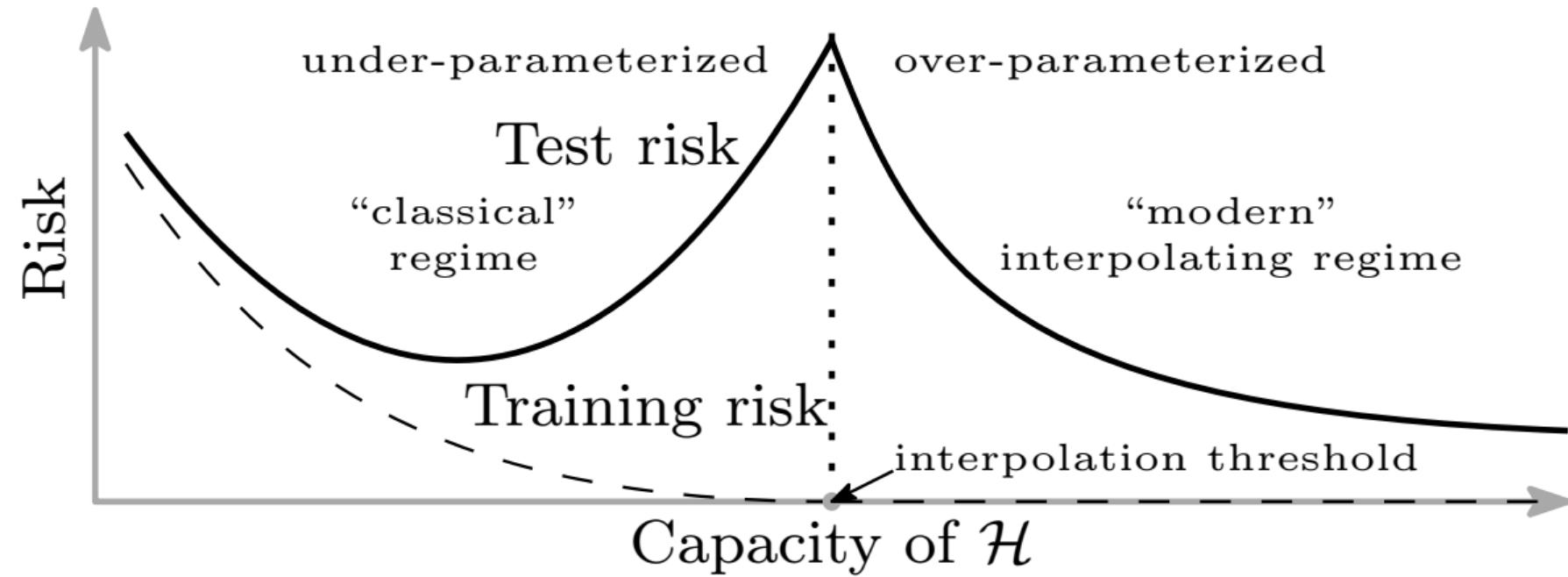
Узкие и широкие локальные минимумы



## Exponential learning rate

- Exponential Learning Rate Schedules for Deep Learning

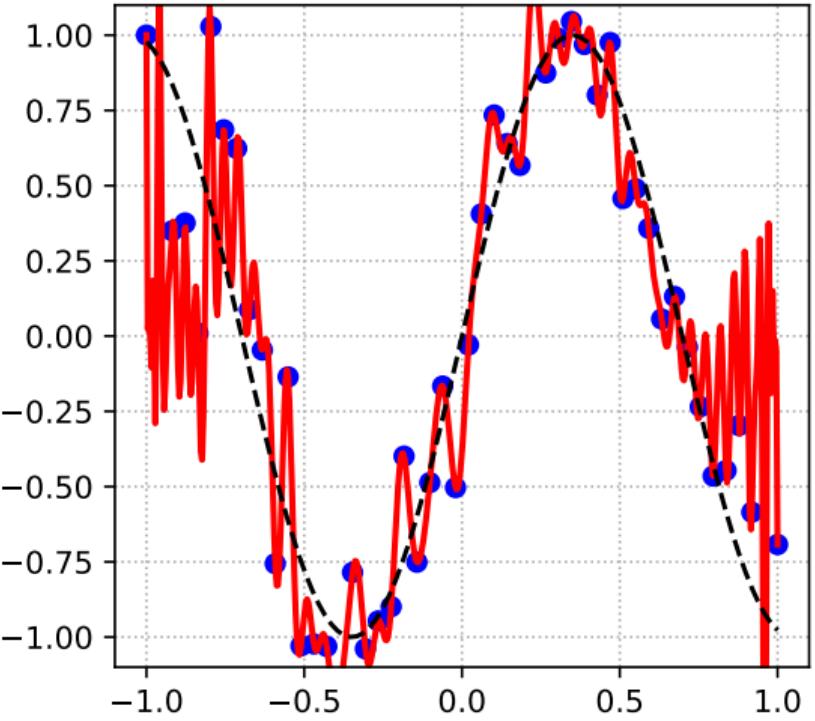
## Double Descent <sup>13</sup>



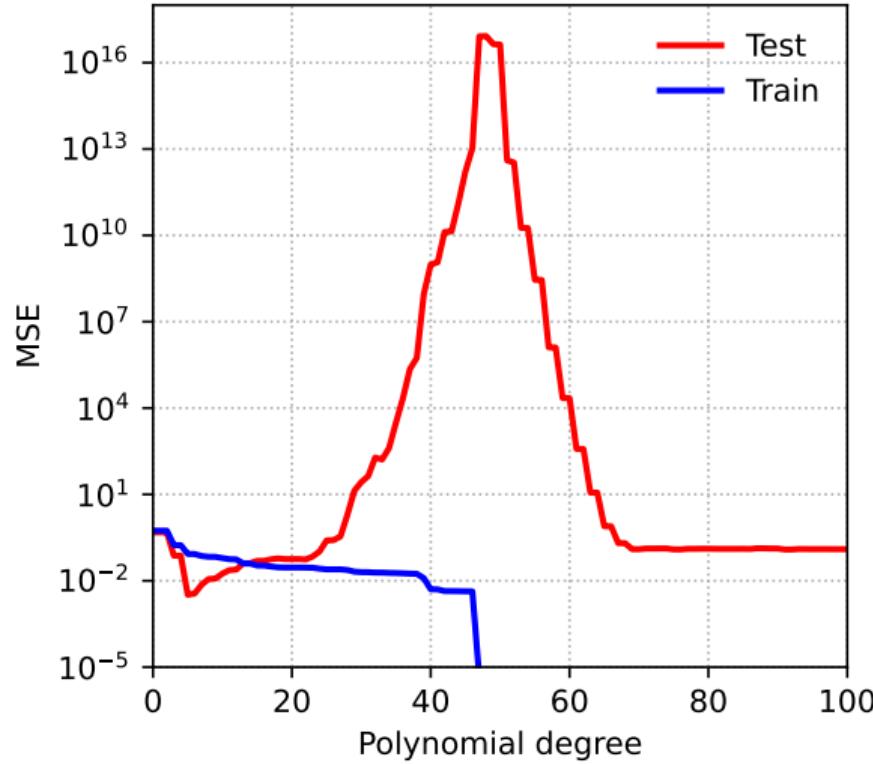
<sup>13</sup>Reconciling modern machine learning practice and the bias-variance trade-off, Mikhail Belkin, Daniel Hsu, Siyuan Ma, Soumik Mandal

## Double Descent

Polynomial Fitting



@fminxyz



Modular Division (training on 50% of data)

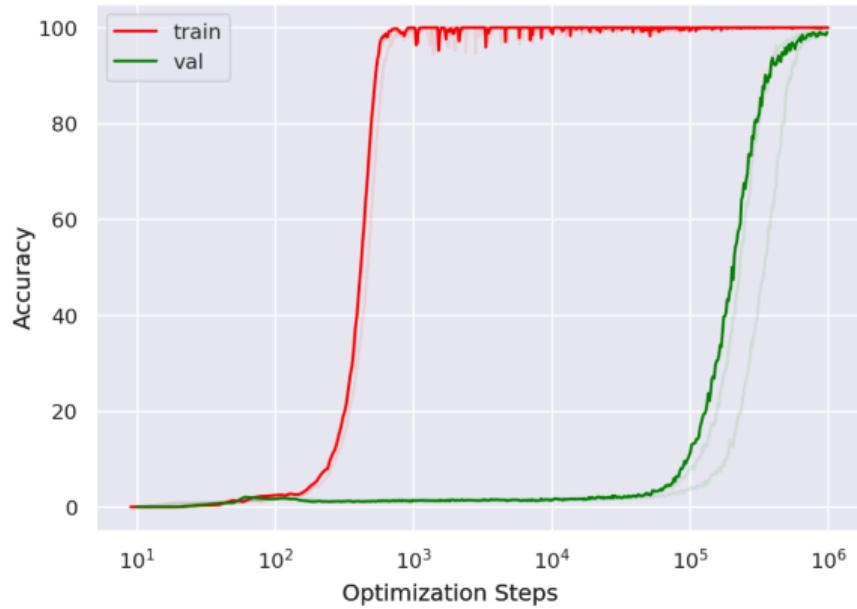


Рисунок 15: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments ( $\sim$  half an hour) is available here

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

# Grokking<sup>14</sup>

Modular Division (training on 50% of data)

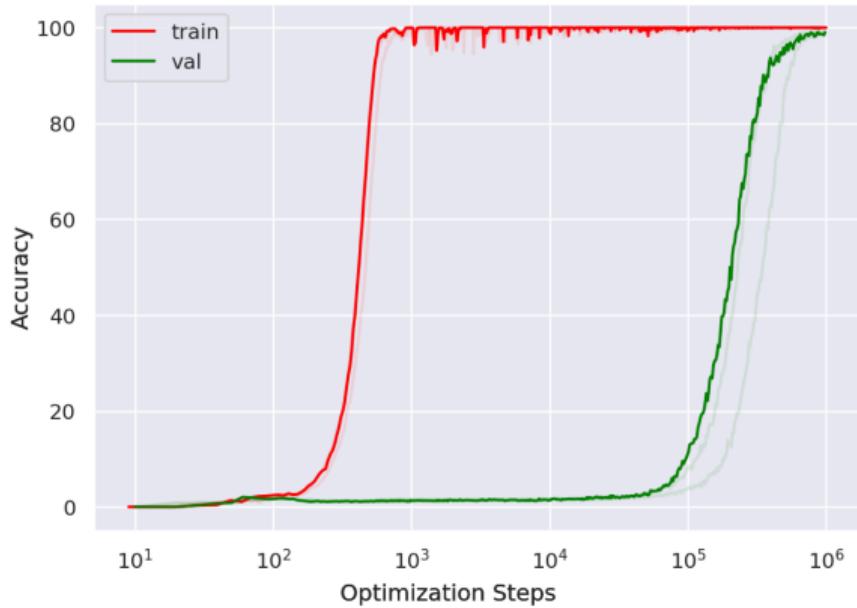


Рисунок 15: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments ( $\sim$  half an hour) is available here

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

# Grokking<sup>14</sup>

Modular Division (training on 50% of data)

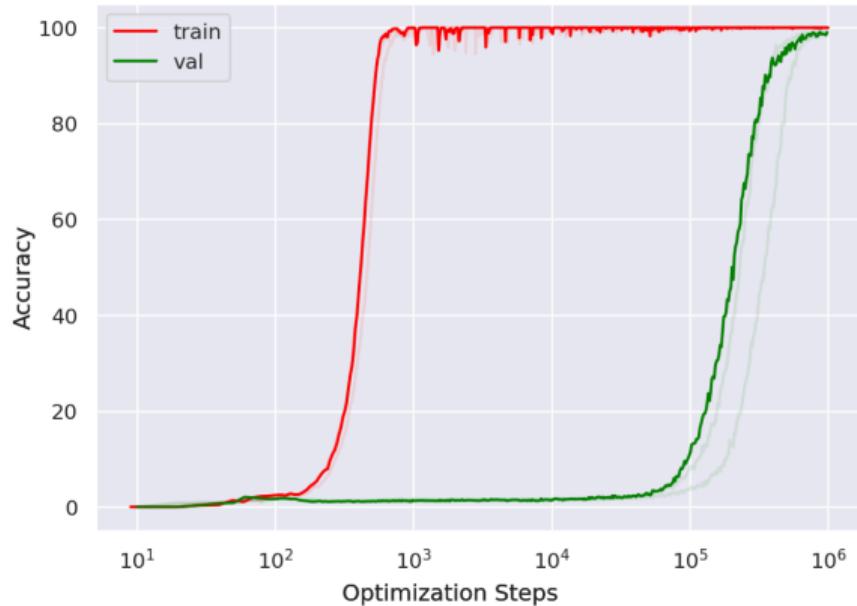


Рисунок 15: Training transformer with 2 layers, width 128, and 4 attention heads, with a total of about  $4 \cdot 10^5$  non-embedding parameters. Reproduction of experiments (~ half an hour) is available here

- 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**.

<sup>14</sup>Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, Vedant Misra, [x,y,z](#)