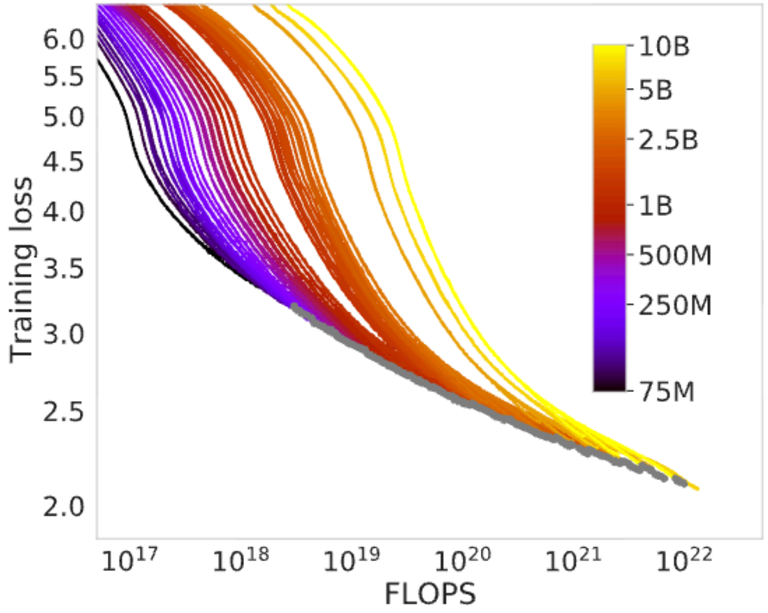
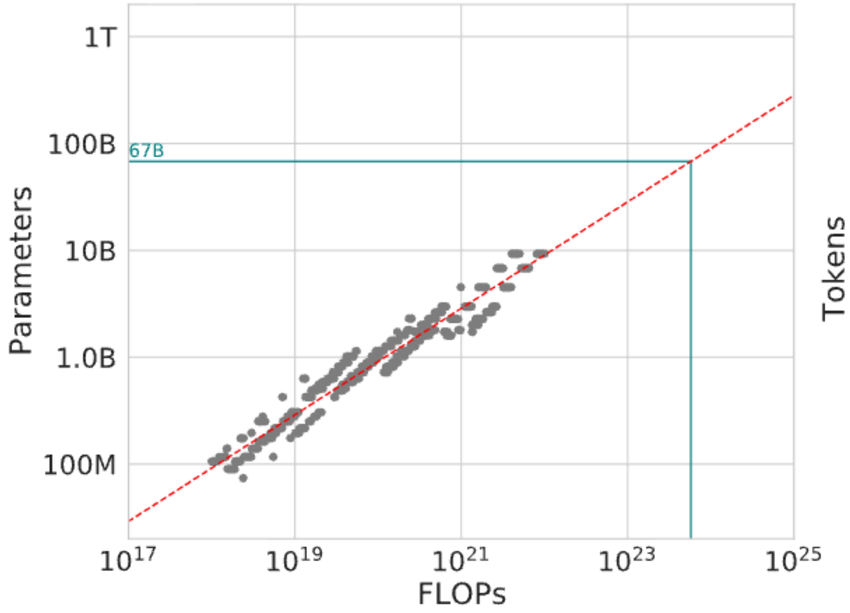
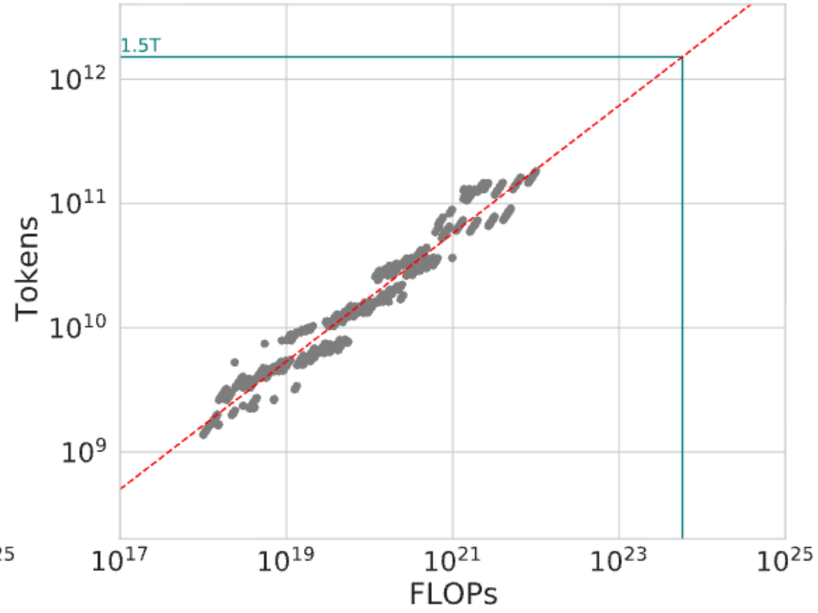
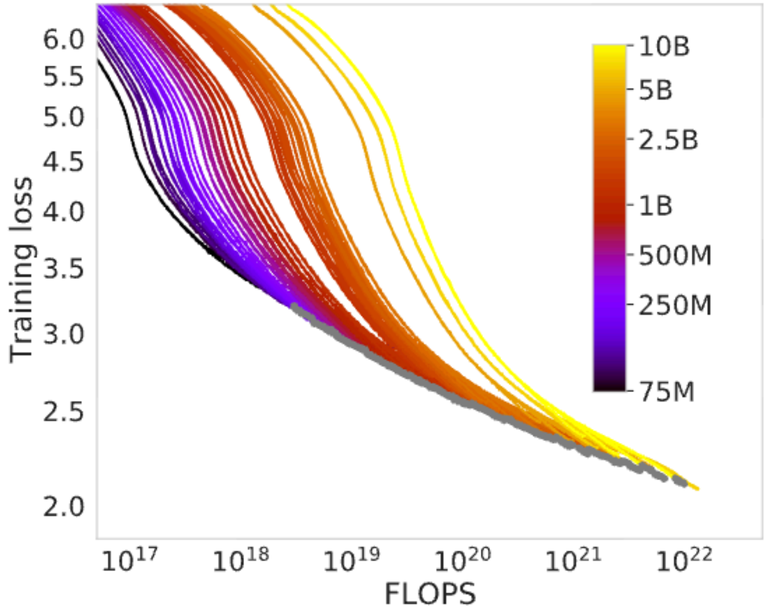
Training LLMs: do we understand our Optimizers?

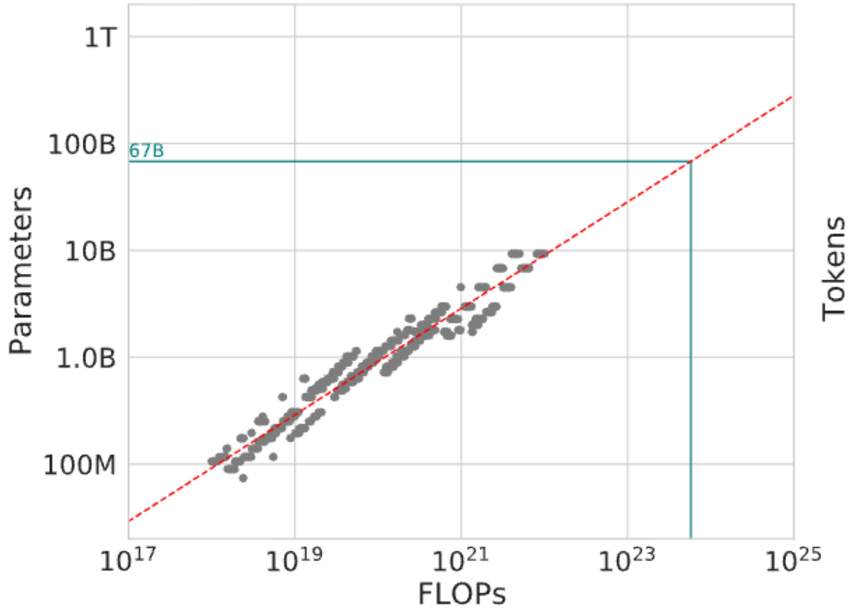
Antonio Orvieto, 2025

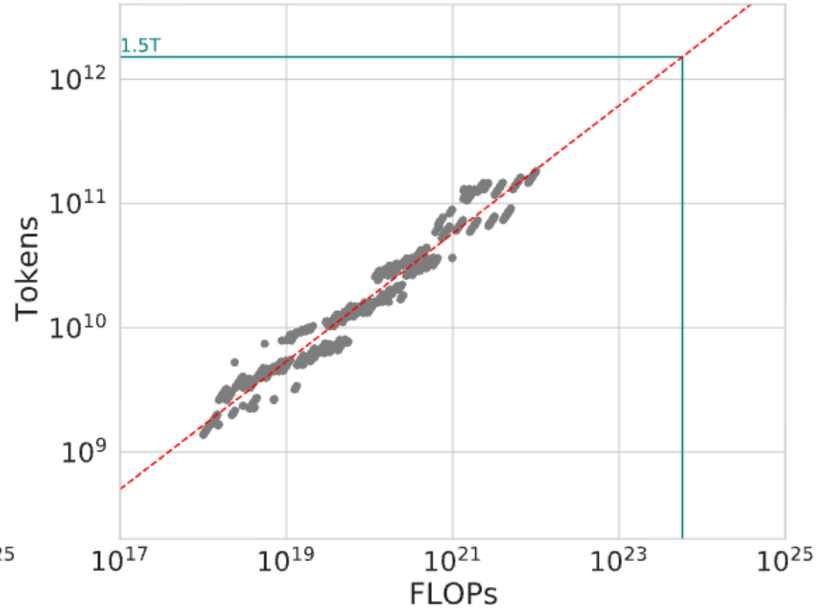

Training loss
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
10B
5B
2.5B
1B
500M
250M
75M
10^{17}
10^{18}
10^{19}
10^{20}
10^{21}
10^{22}
FLOPS


Parameters
1T
100B
67B
10B
1.0B
100M
10^{17}
10^{19}
10^{21}
10^{23}
10^{25}
FLOPs


Tokens
1.5T
10^{12}
10^{11}
10^{10}
10^{9}
10^{17}
10^{19}
10^{21}
10^{23}
10^{25}
FLOPs







DeepMind

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

$$\hat{L}(N, D) \triangleq E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}.$$

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_T\} \sim p(\mathcal{X})$ be a token sequence from the true data generating distribution $p(\mathcal{X})$ where $x_t \in \mathbb{V}$. Let $\mathcal{X}_{\leq t} = \{x_1, x_2, \ldots, x_t\}$.

A **language model** learns the factorization

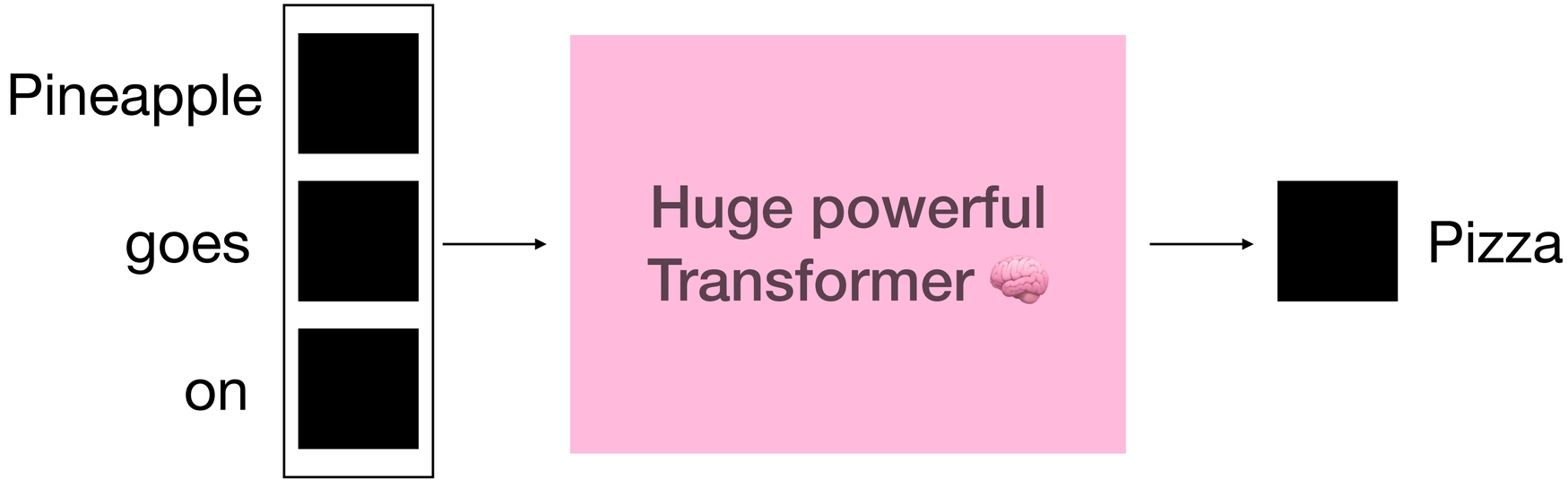
$$p(\mathcal{X}; \theta) = p(x_1) \prod_{t=1}^{T-1} p(x_{t+1} \mid \mathcal{X}_{\leq t}; \theta)$$

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_T\} \sim p(\mathcal{X})$ be a token sequence from the true data generating distribution $p(\mathcal{X})$ where $x_t \in \mathbb{V}$. Let $\mathcal{X}_{\leq t} = \{x_1, x_2, \ldots, x_t\}$.

A **language model** learns the factorization

Probability of the next word given previous

$$p(\mathcal{X}; \theta) = p(x_1) \prod_{t=1}^{T-1} p(x_{t+1} \mid \mathcal{X}_{\leq t}; \theta)$$



Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*

Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry

Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan

Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter

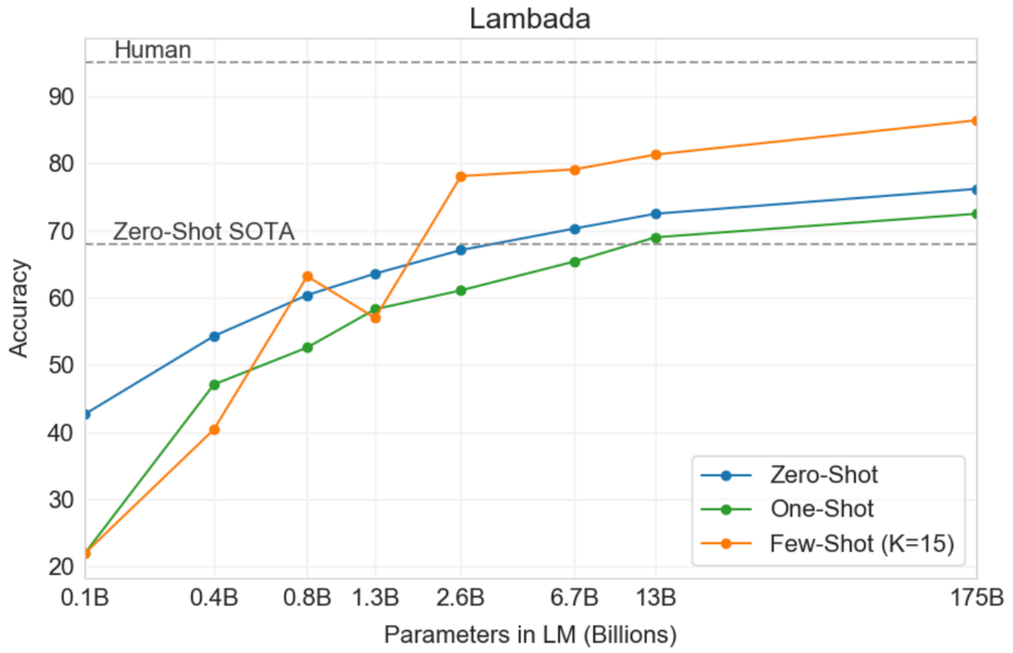
Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray

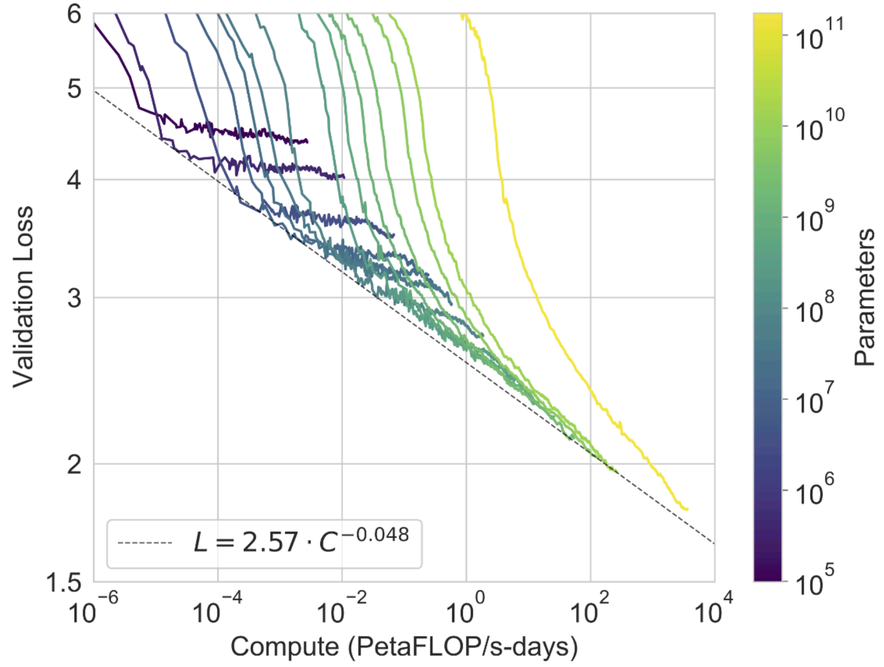
Benjamin Chess Jack Clark Christopher Berner

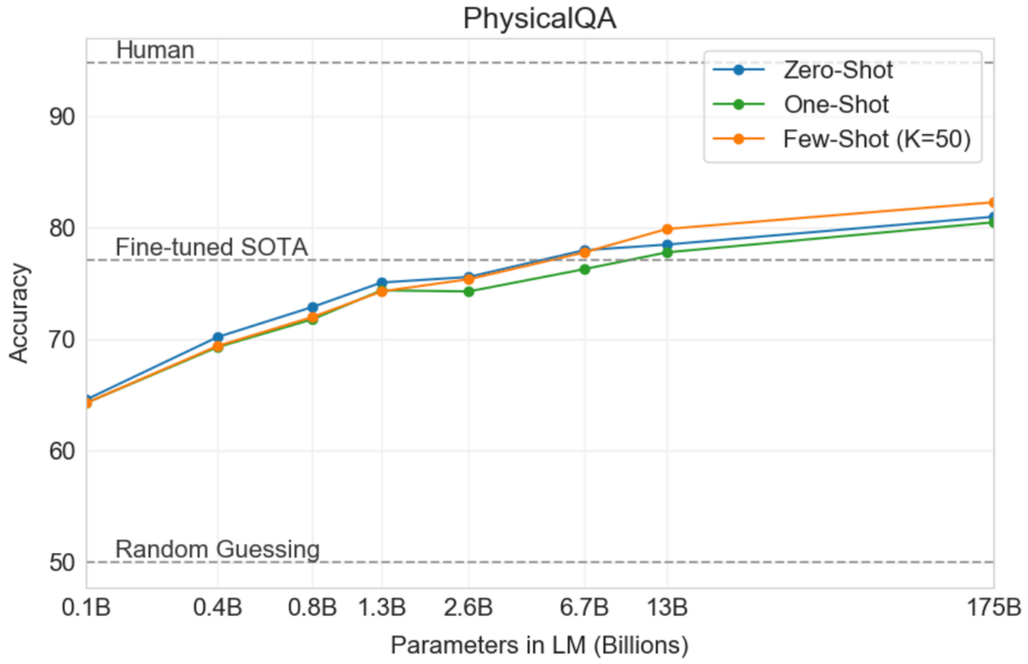
Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

OpenAI

2020





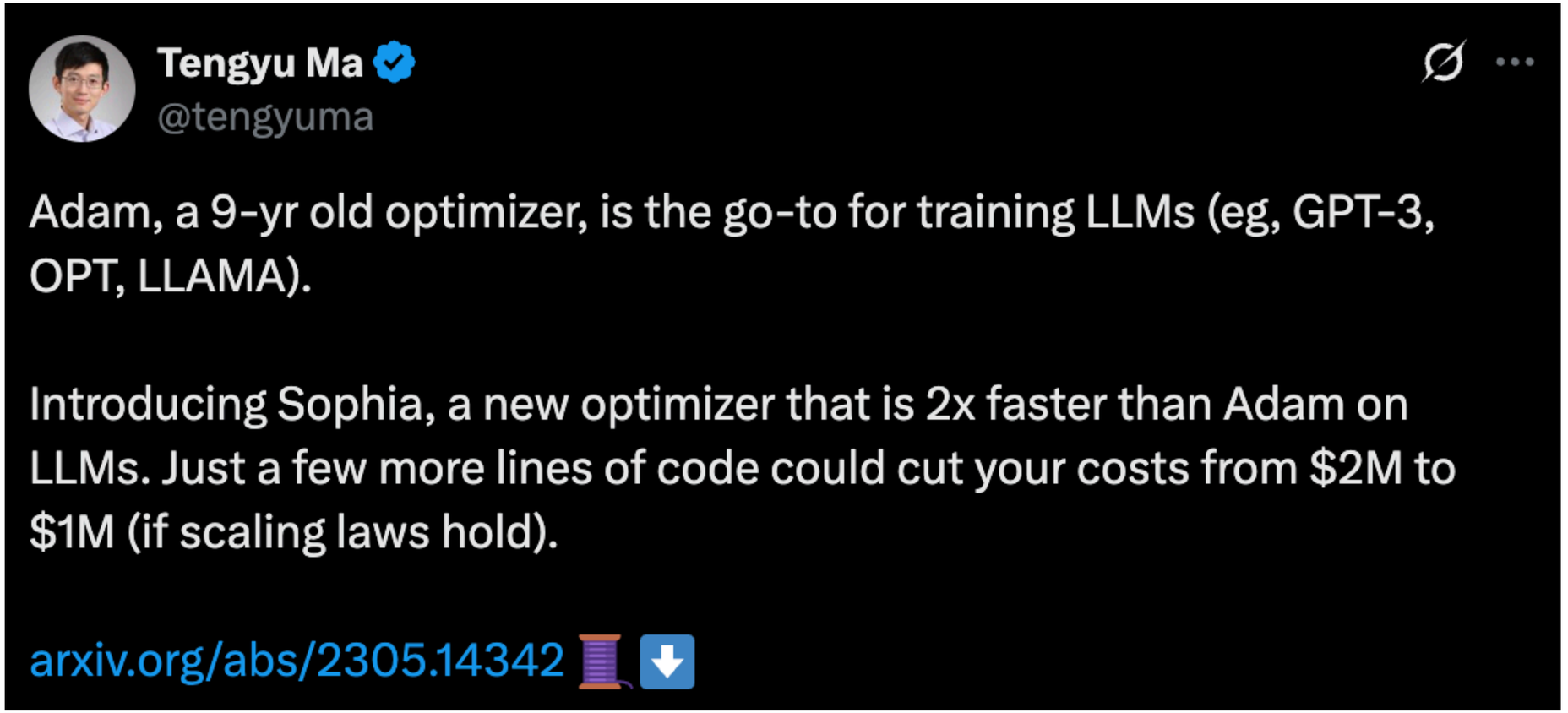


Recent estimates and public claims

Here are some of the more recent reported / estimated costs:

Model / claim	Estimated cost (compute only, often)	Notes / caveats
GPT-4	$78 million (compute) Visual Capitalist	A widely circulated figure.
GPT-4	$80M to $100M TechRadar	Some media estimates including additional overhead.
GPT-4 (or "the cost")	"more than $100M" (Sam Altman) Wikipedia +1	OpenAI has suggested a figure above $100M.
Gemini (Google's newer models)	$191 million (for Gemini Ultra, claimed) About Chromebo...	From a media/AI cost summary.
DeepSeek R1 (Chinese model, more efficient)	$294,000 Reuters	A lower cost example, though likely for a more optimized or smaller run.

But…



Spoiler: this was 2.5 years ago, we still use Adam

What does theory say? (Jiang et al., 2025)

Under Assumptions 1-4a, it is known that SGD, with an appropriately chosen step size, can find a point $\hat{\boldsymbol{w}}$ such that $\mathbb{E}\left[\|\nabla F(\hat{\boldsymbol{w}})\|_2^2\right] \leq \epsilon^2$ after at most $\mathcal{O}\left(\frac{L(F(\boldsymbol{w}_1)-F^*)\sigma^2}{\epsilon^4} + \frac{L(F(\boldsymbol{w}_1)-F^*)}{\epsilon^2}\right)$ iterations (Ghadimi and Lan, 2013; Bottou et al., 2018). Moreover, this complexity matches the lower bound for any first-order method up to an absolute constant, as shown by Arjevani et al. (2023).

According to this classical convergence theory, SGD is the optimal first-order method in this setting in the worst-case sense, leaving no room for further improvement.

* this is a bit of a simplification, see e.g. (L_0, L_1) smoothness, and Jiang et. al 25

Outline

- **Introduction**: what is Adam?
- **Introduction 2**: History class: why do we use Adam over SGD?
  + a tiny bit of Muon for the fans.
- **Why** is this understanding *optimizer gaps* **important** for LLMs future.
- **Core**:
  - Part 1: large ablation of Optimizer Gaps on LMs.
  - Part 2: insights and new interpretations of Adam.
  - A surprise, for everyone (especially me, shock).

REVISITING ASSOCIATIVE RECALL IN MODERN RECURRENT MODELS

Destiny Okpekpe & Antonio Orvieto
Max Planck Institute for Intelligent Systems
ELLIS Institute Tübingen
Tübingen AI Center
{destiny,antonio}@tue.ellis.eu

Motivation: why are we not satisfied with Adam

In Search of Adam's Secret Sauce

Antonio Orvieto *
ELLIS Institute Tübingen, MPI-IS
Tübingen AI Center, Germany

Robert Gower
Flatiron Institute
New York, US

My obsession this spring

Is your batch size the problem? Revisiting the Adam-SGD gap in language modeling

Teodora Srećković,* **Jonas Geiping,** **Antonio Orvieto**
Max Planck Institute for Intelligent Systems
ELLIS Institute Tübingen, Tübingen AI Center

The surprise

```
optimizer = torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08)
```

Published as a conference paper at ICLR 2015

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

Cited > 220k times!

ICLR
International Conference On Learning Representations

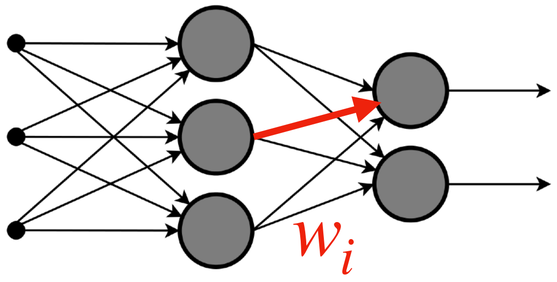
Test of Time

Adam: A Method for Stochastic Optimization
Diederik P. Kingma, Jimmy Ba
https://arxiv.org/abs/1412.6980

As one of the most widely adopted optimization algorithms in deep learning, Adam revolutionized neural network training, enabling significantly faster convergence and more stable training across a wide variety of architectures and tasks. The algorithm automatically adjusts parameter-specific learning rates based on first and second moments of gradients, handling sparse gradients and non-stationary objectives. Adam's practical success has made it the default optimizer for countless state-of-the-art models, from computer vision and natural language processing to reinforcement learning, demonstrating remarkable versatility across problem domains and neural network architectures.

Quote from 'The Young Pope' by Sorrentino, Episode 2

```
optimizer = torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08)
```



Update of i-th weight

$$w_i^{k+1} = w_i^k - \frac{\eta}{\sqrt{v_i^k + \epsilon}} m_i^k$$

`lr=0.001`

`eps=1e-08`

← SGD with momentum and scaled stepsize

$$m_i^{k+1} = \beta_1 \, m_i^k + (1 - \beta_1) \nabla_i L(w^{k+1})$$

← Momentum update

$$v_i^{k+1} = \beta_2 \, v_i^k + (1 - \beta_2) \nabla_i L(w^{k+1})^2$$

← Scaling factor update

`beta_1=0.9`

`beta_2=0.999`

Usually, update is written in vector form, with element-wise divisions and multiplications.

$$m_i^0 = \nabla_i L(w^0)$$
$$v_i^0 = \nabla_i L(w^0)^2$$

In a nutshell..

$$m_i^k = \text{EMA}_{\beta_1}\left[\nabla_i L(w^k)\right], \quad v_i^k = \text{EMA}_{\beta_2}\left[\nabla_i L(w^k)^2\right]$$

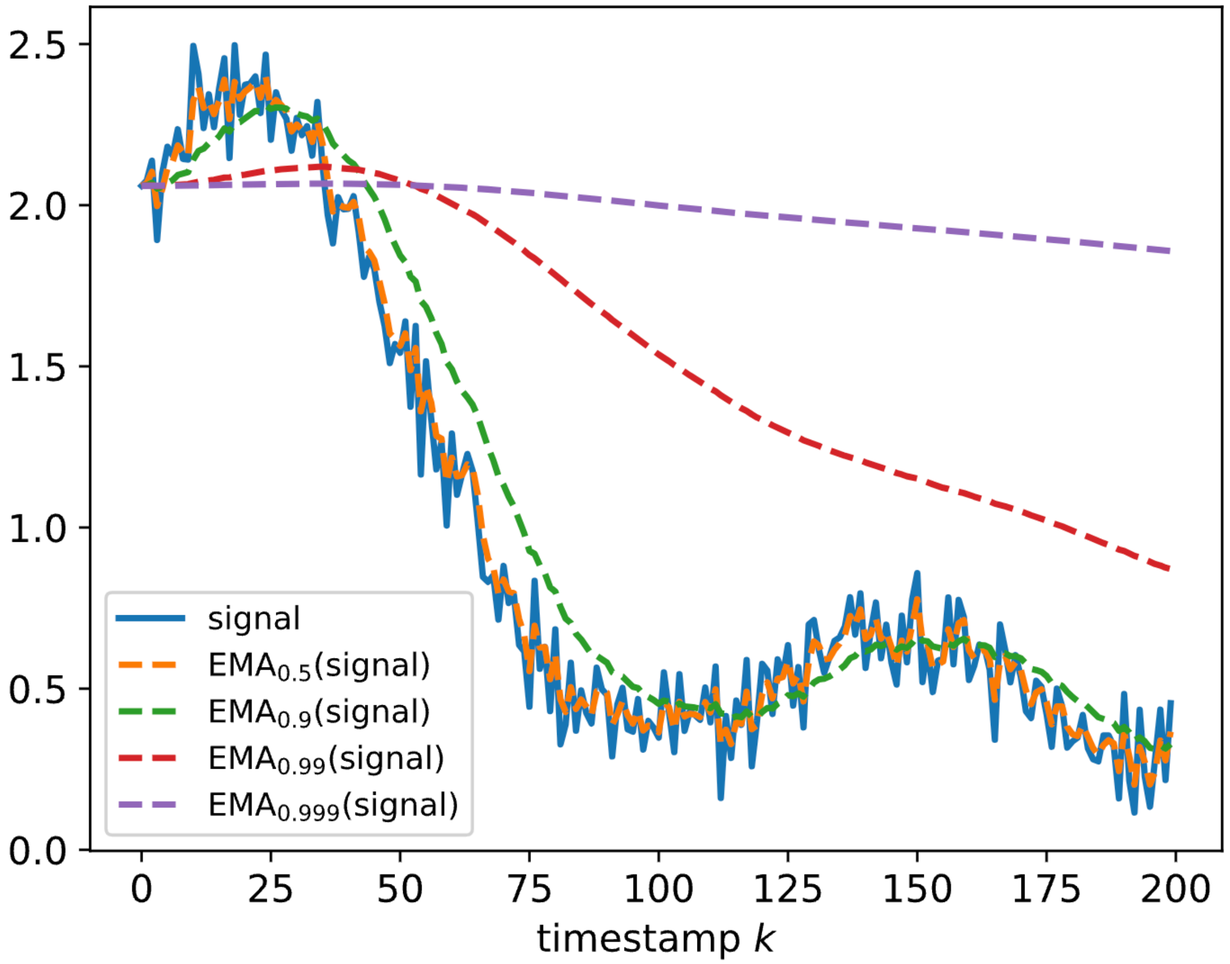
$$w_i^{k+1} = w_i^k - \frac{\eta}{\sqrt{v_i^k + \epsilon}} m_i^k$$

EMA: exponential moving average

$$w_i^{k+1} = w_i^k - \frac{\eta}{\sqrt{v_i^k + \epsilon}} \nabla_i L(w_i^k)$$

Setting β_1 to 0 (no momentum)
we get **RMSprop**
(Tieleman, Hinton, 2012)

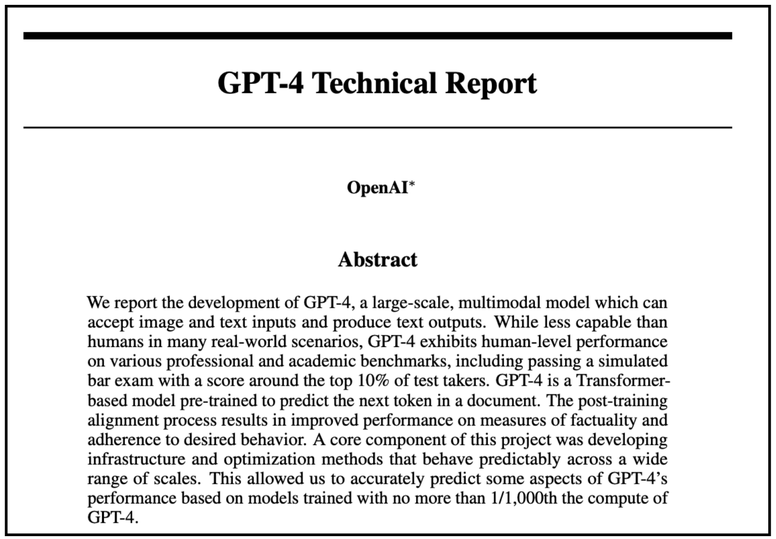
Effect of EMA



- Moving averages are slower as β parameters increase.
- $\beta \sim 0.5$ acts here as denoising.
- Higher β produce more delay but smoother output signal.
- Very high βs yield extremely resilient variation in output.
- Ultimately, all of this depends on the speed at which input signal changes.

Why is it particularly crucial?

Attention is dominating everything!



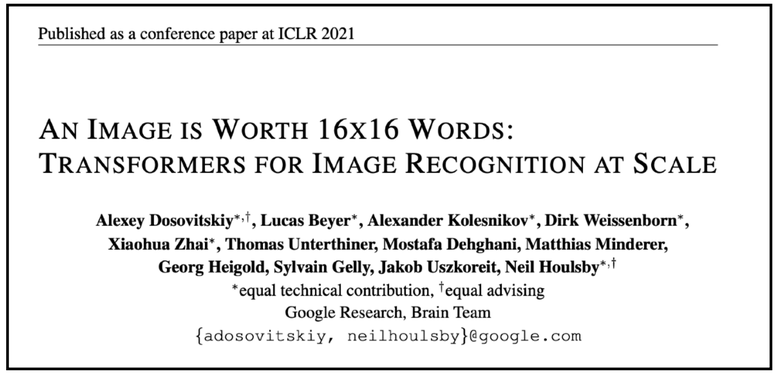
GPT-4 Technical Report

OpenAI*

Abstract

We report the development of GPT-4, a large-scale, multimodal model which can accept image and text inputs and produce text outputs. While less capable than humans in many real-world scenarios, GPT-4 exhibits human-level performance on various professional and academic benchmarks, including passing a simulated bar exam with a score around the top 10% of test takers. GPT-4 is a Transformer-based model pre-trained to predict the next token in a document. The post-training alignment process results in improved performance on measures of factuality and adherence to desired behavior. A core component of this project was developing infrastructure and optimization methods that behave predictably across a wide range of scales. This allowed us to accurately predict some aspects of GPT-4's performance based on models trained with no more than 1/1,000th the compute of GPT-4.

Text: sequence of words



Published as a conference paper at ICLR 2021

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

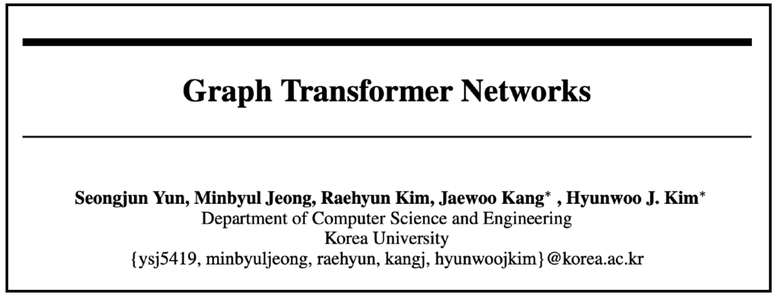
Alexey Dosovitskiy*,†, Lucas Beyer*, Alexander Kolesnikov*, Dirk Weissenborn*, Xiaohua Zhai*, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby*,†

*equal technical contribution, †equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

Image: “sequence” of pixel patches.



Graph Transformer Networks

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang* , Hyunwoo J. Kim*

Department of Computer Science and Engineering

Korea University

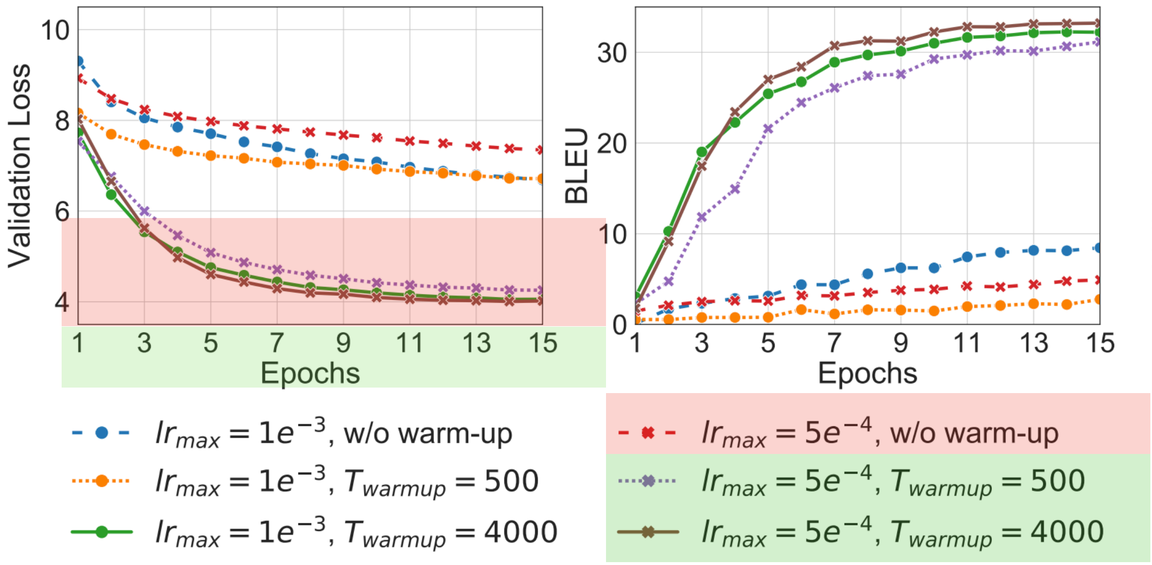
{ysj5419, minbyuljeong, raehyun, kangj, hyunwoojkim}@korea.ac.kr

Graph: “sequence” of nodes.

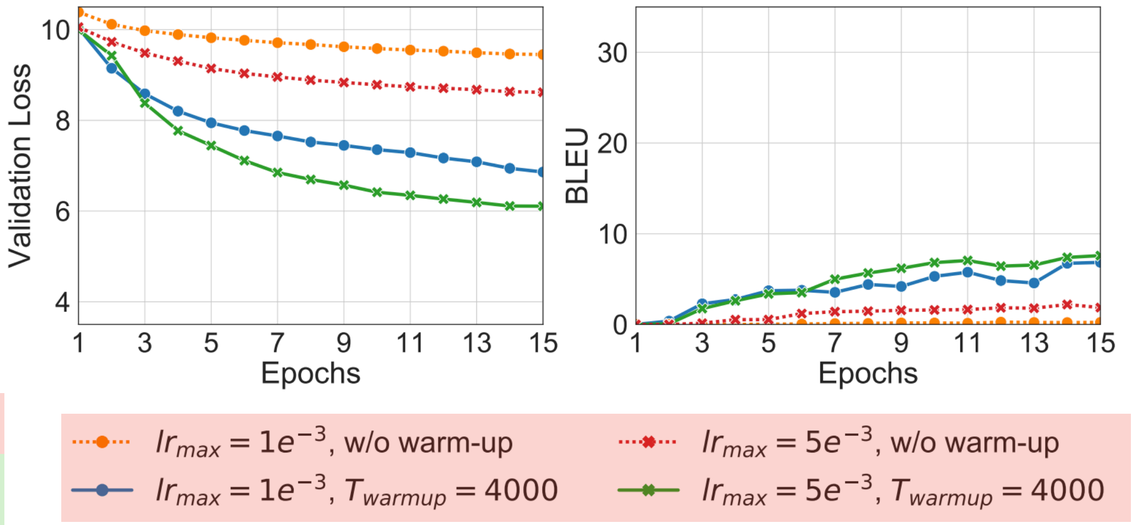
On Layer Normalization in the Transformer Architecture

Ruibin Xiong[†*1,2] Yunchang Yang[*3] Di He[4,5] Kai Zheng[4] Shuxin Zheng[5] Chen Xing[6] Huishuai Zhang[5]
Yanyan Lan[1,2] Liwei Wang[4,3] Tie-Yan Liu[5]

~2020



(a) Loss/BLEU on the IWSLT14 De-En task (Adam)



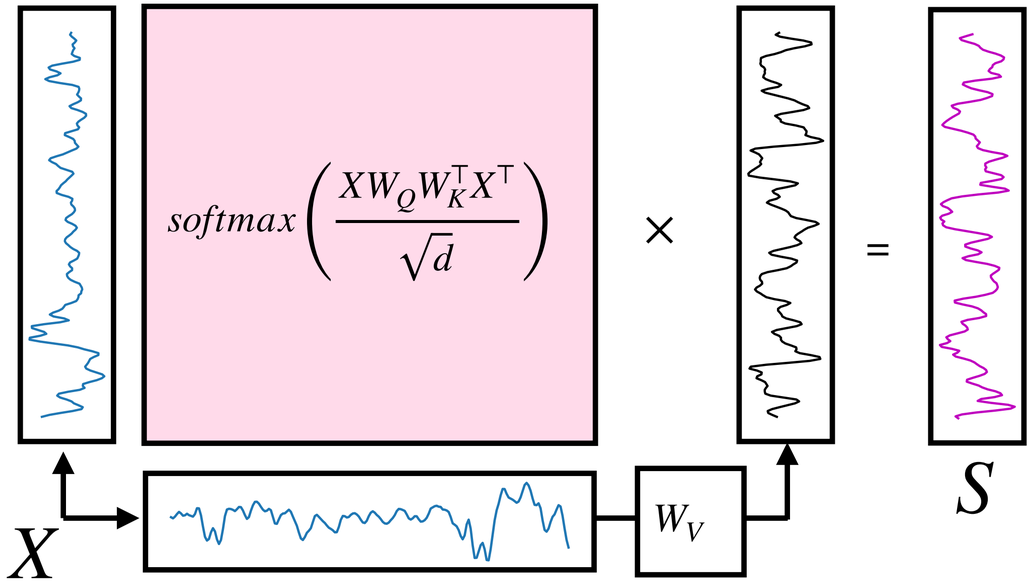
(b) Loss/BLEU on the IWSLT14 De-En task (SGD)

- PostLN transformers are hard to train, you need warmup
- .. and Adam! **No Adam no party**.

The Attention Mechanism (layer ℓ)

$$
\begin{cases}
A^{\ell} = \text{softmax}\left(\frac{1}{\sqrt{d}} X^{\ell} W^{Q,\ell} \left(X^{\ell} W^{K,\ell}\right)^{\top}\right) \\
Z^{\ell} = A^{\ell} X^{\ell} W^{V,\ell} \\
Y^{\ell} = \sigma(Z^{\ell} W^{F_1,\ell}) W^{F_2,\ell}
\end{cases}
$$

Keys, queries, keys dimensions



W^Q, W^K have a **role completely different** from W^V, W^{F_1}, W^{F_2}

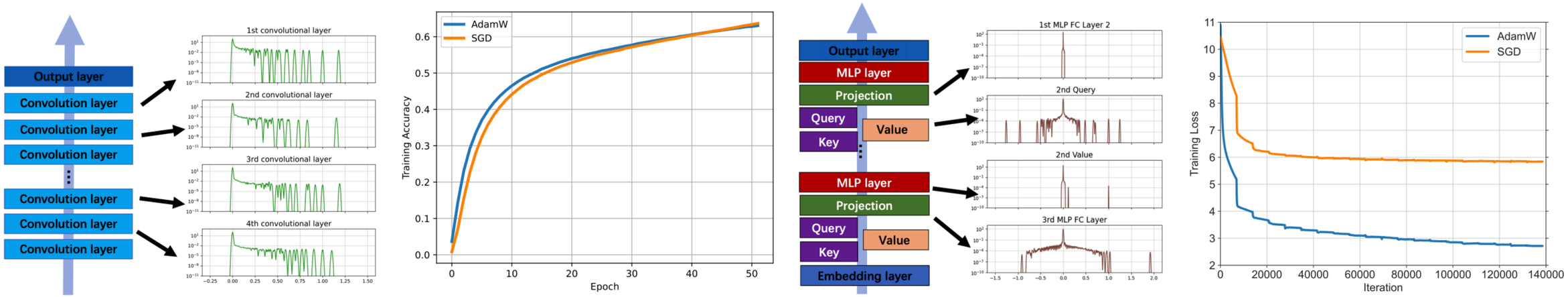
- In pink: token mixing parameters
- In yellow: MLP mixing parameters

In transformers, Adam adapts to **different needs (curvatures)** of parameter groups:

Why Transformers Need Adam: A Hessian Perspective

Yushun Zhang[12], Congliang Chen[12], Tian Ding[2], Ziniu Li[12], Ruoyu Sun[12*], Zhi-Quan Luo[12]
[1]The Chinese University of Hong Kong, Shenzhen, China
[2]Shenzhen Research Institute of Big Data
{yushunzhang,congliangchen,ziniuli}@link.cuhk.edu.cn
dingtian@sribd.cn, sunruoyu@cuhk.edu.cn, luozq@cuhk.edu.cn

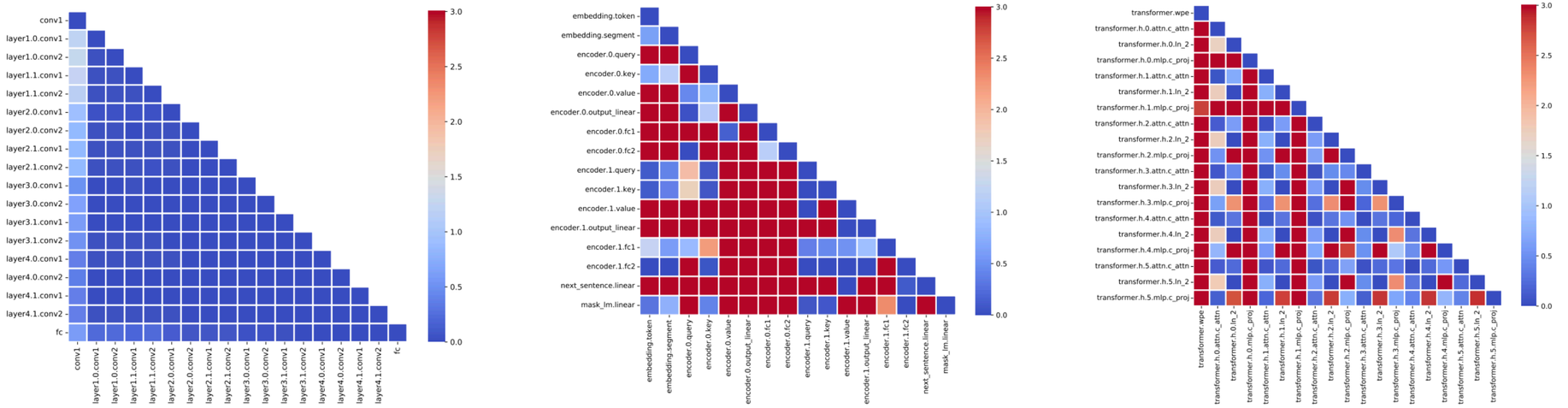
Good modern paper! (2024)



(a) VGG16 (b) VGG16 (c) BERT (d) BERT

Figure 3: **(a) (c)**: The blockwise Hessian spectra of VGG16 (CNN) and BERT (Transformer) at initialization. The x-axis records the eigenvalues and the y-axis records the frequency in the log scale. To allow comparison in the same figure, we sample 4 blocks in each model. The plotted spectra are normalized by their 10th largest eigenvalues. The spectra are similar among blocks for VGG and differ significantly across blocks for BERT. **(b) (d)** Adam v.s. SGD for training VGG16 and BERT.

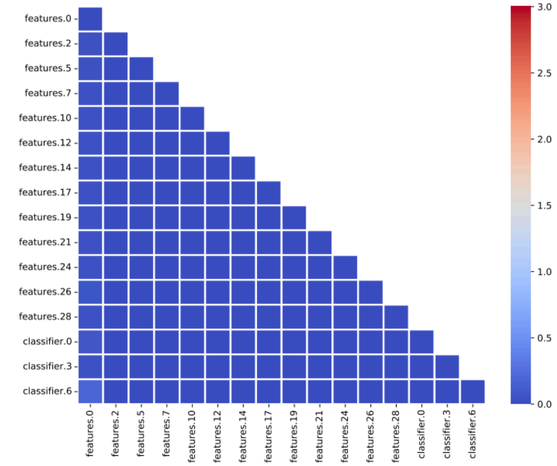
Observation 1: For all Transformers we checked, the blockwise Hessian spectra are largely *different* from each other. In contrast, the blockwise Hessian spectra of CNNs are *similar*.



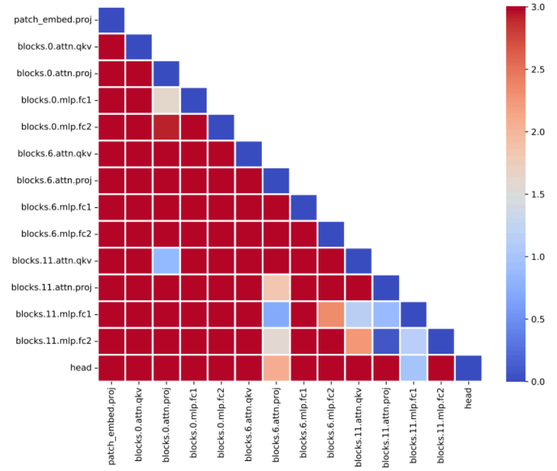
(a) ResNet18

(b) BERT

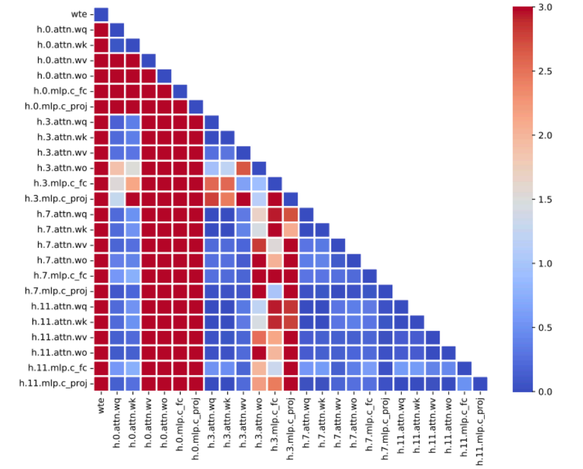
(c) GPT2-nano



(d) VGG16



(e) ViT-base



(f) GPT2

Figure 4: The JS distance among blockwise Hessian spectra at initialization. We find that the JS distance of blockwise spectra in CNNs is significantly smaller than that in Transformers.

Noise Is Not the Main Factor Behind the Gap Between SGD and Adam on Transformers, But Sign Descent Might Be

Frederik Kunstner, Jacques Chen, J. Wilder Lavington & Mark Schmidt[†]
University of British Columbia, Canada CIFAR AI Chair (Amii)[†]
{kunstner,jola2372,schmidtm}@cs.ubc.ca
jacquesc@students.cs.ubc.ca

Why are Adaptive Methods Good for Attention Models?

Jingzhao Zhang
MIT
jzhzhang@mit.edu

Sai Praneeth Karimireddy
EPFL
sai.karimireddy@epfl.ch

Andreas Veit
Google Research
aveit@google.com

Seungyeon Kim
Google Research
seungyeonk@google.com

Sashank Reddi
Google Research
sashank@google.com

Sanjiv Kumar
Google Research
sanjivk@google.com

Suvrit Sra
MIT
suvrit@mit.edu

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

Frederik Kunstner [1] **Robin Yadav** [1] **Alan Milligan** [1] **Mark Schmidt** [1 2] **Alberto Bietti** [3]

Why Transformers Need Adam: A Hessian Perspective

Yushun Zhang[12], **Congliang Chen**[12], **Tian Ding**[2], **Ziniu Li**[12], **Ruoyu Sun**[12*], **Zhi-Quan Luo**[12]
[1]The Chinese University of Hong Kong, Shenzhen, China
[2]Shenzhen Research Institute of Big Data
{yushunzhang,congliangchen,ziniuli}@link.cuhk.edu.cn
dingtian@sribd.cn, sunruoyu@cuhk.edu.cn, luozq@cuhk.edu.cn

Toward Understanding Why Adam Converges Faster Than SGD for Transformers

Yan Pan YPAN2@ANDREW.CMU.EDU
Yuanzhi Li YUANZHIL@ANDREW.CMU.EDU
Carnegie Mellon University

Just to name a few..

However, If you look back to 2017..

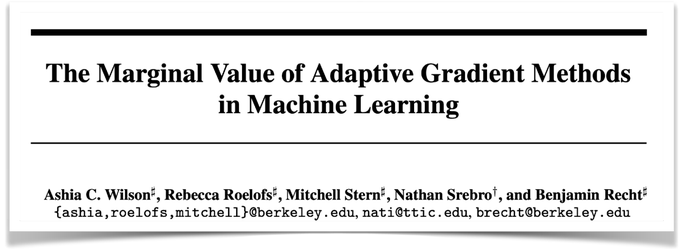
The Marginal Value of Adaptive Gradient Methods in Machine Learning

Ashia C. Wilson[♯], **Rebecca Roelofs**[♯], **Mitchell Stern**[♯], **Nathan Srebro**[†], **and Benjamin Recht**[♯]
{ashia,roelofs,mitchell}@berkeley.edu, nati@ttic.edu, brecht@berkeley.edu

Abstract

Adaptive optimization methods, which perform local optimization with a metric constructed from the history of iterates, are becoming increasingly popular for training deep neural networks. Examples include AdaGrad, RMSProp, and Adam. We show that for simple overparameterized problems, adaptive methods often find drastically different solutions than gradient descent (GD) or stochastic gradient descent (SGD). We construct an illustrative binary classification problem where the data is linearly separable, GD and SGD achieve zero test error, and AdaGrad, Adam, and RMSProp attain test errors arbitrarily close to half. We additionally study the empirical generalization capability of adaptive methods on several state-of-the-art deep learning models. We observe that the solutions found by adaptive methods generalize worse (often *significantly* worse) than SGD, even when these solutions have better training performance. These results suggest that practitioners should reconsider the use of adaptive methods to train neural networks.

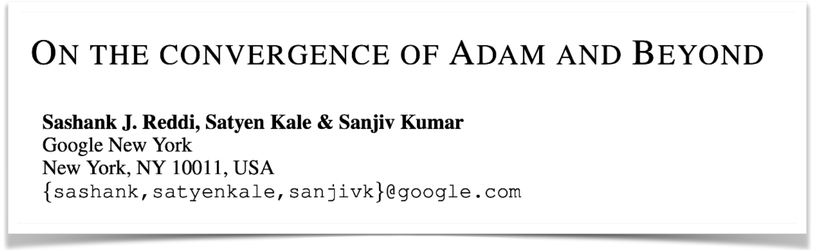
1) 2015-2018: Vision people trusted SGD with momentum, and thought **Adam was fast but worse at test accuracy**



The Marginal Value of Adaptive Gradient Methods in Machine Learning

Ashia C. Wilson[♯], Rebecca Roelofs[♯], Mitchell Stern[♯], Nathan Srebro[†], and Benjamin Recht[♯]
{ashia,roelofs,mitchell}@berkeley.edu, nati@ttic.edu, brecht@berkeley.edu

2) At the same time: optimization people started to think of Adam as a bad optimizer - it **does not converge** in some settings.



ON THE CONVERGENCE OF ADAM AND BEYOND

Sashank J. Reddi, Satyen Kale & Sanjiv Kumar
Google New York
New York, NY 10011, USA
{sashank,satyenkale,sanjivk}@google.com

3) 2018: Loshchilov & Hutter discovered Adam's implementation of L2 regularization was wrong. **Adam(W) :** fast and generalizes well.

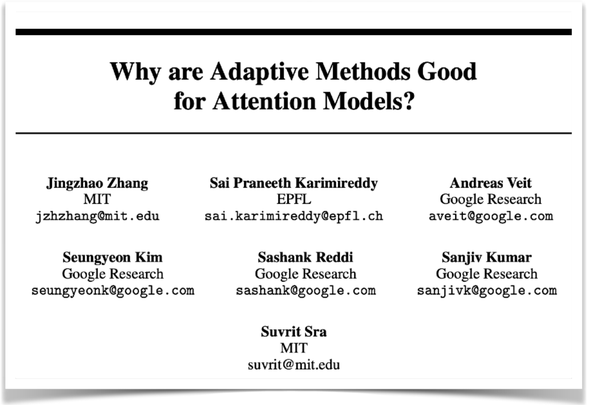
DECOUPLED WEIGHT DECAY REGULARIZATION

Ilya Loshchilov & Frank Hutter
University of Freiburg
Freiburg, Germany,
{ilya,fh}@cs.uni-freiburg.de

4) At the same time: **Transformers** were born, and Adam was picked as optimizer of choice.

5) **Transformers could be only trained with Adam**, yet this information on **NLP was not too interesting for optimization researchers**

6) In 2020 some papers came out claiming **heavy tail noise** in NLP tasks was at the root of success of Adam. For many, case was closed.



Why are Adaptive Methods Good for Attention Models?

Jingzhao Zhang
MIT
jzhzhang@mit.edu

Sai Praneeth Karimireddy
EPFL
sai.karimireddy@epfl.ch

Andreas Veit
Google Research
aveit@google.com

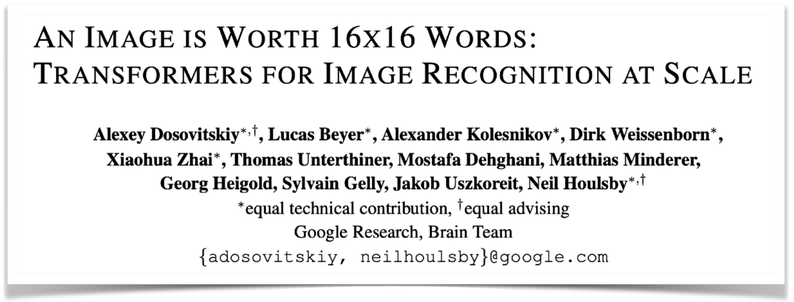
Seungyeon Kim
Google Research
seungyeonk@google.com

Sashank Reddi
Google Research
sashank@google.com

Sanjiv Kumar
Google Research
sanjivk@google.com

Suvrit Sra
MIT
suvrit@mit.edu

7) Around 2021, **ViTs** took over vision: it was then clear : efficient training of transformers is crucial.



AN IMAGE IS WORTH 16X16 WORDS:
TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy*,†, Lucas Beyer*, Alexander Kolesnikov*, Dirk Weissenborn*, Xiaohua Zhai*, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby*,†
*equal technical contribution, †equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

But why is it important to go beyond Adam?

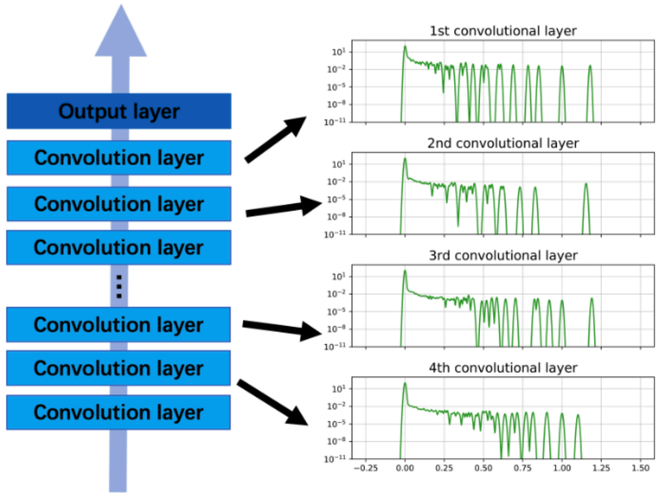
Motivation 1: Adam-SGD gap

Some architectures like **Transformers** are only trainable with **adaptive methods** (Adam, Muon, Scion, etc..)

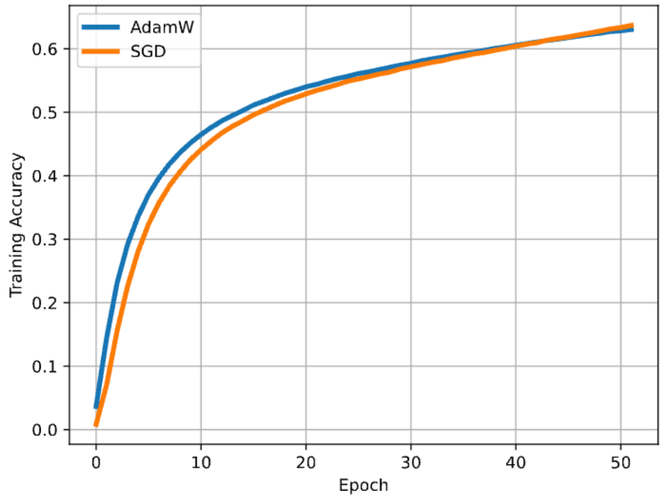
What makes such architectures/ data unique? **What can we learn from this?**

Why Transformers Need Adam: A Hessian Perspective

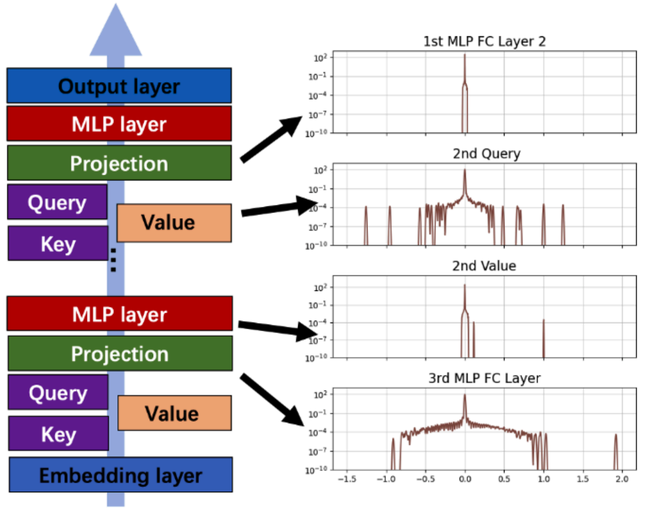
Yushun Zhang[12], **Congliang Chen**[12], **Tian Ding**[2], **Ziniu Li**[12], **Ruoyu Sun**[12*], **Zhi-Quan Luo**[12]
[1]The Chinese University of Hong Kong, Shenzhen, China
[2]Shenzhen Research Institute of Big Data
{yushunzhang,congliangchen,ziniuli}@link.cuhk.edu.cn
dingtian@sribd.cn, sunruoyu@cuhk.edu.cn, luozq@cuhk.edu.cn



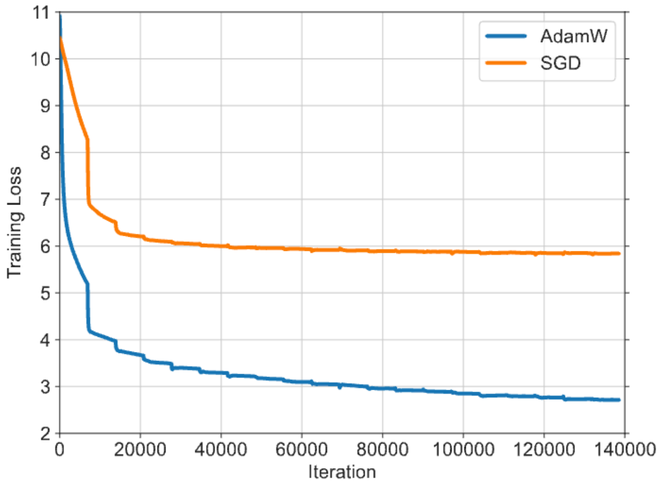
(a) VGG16



(b) VGG16

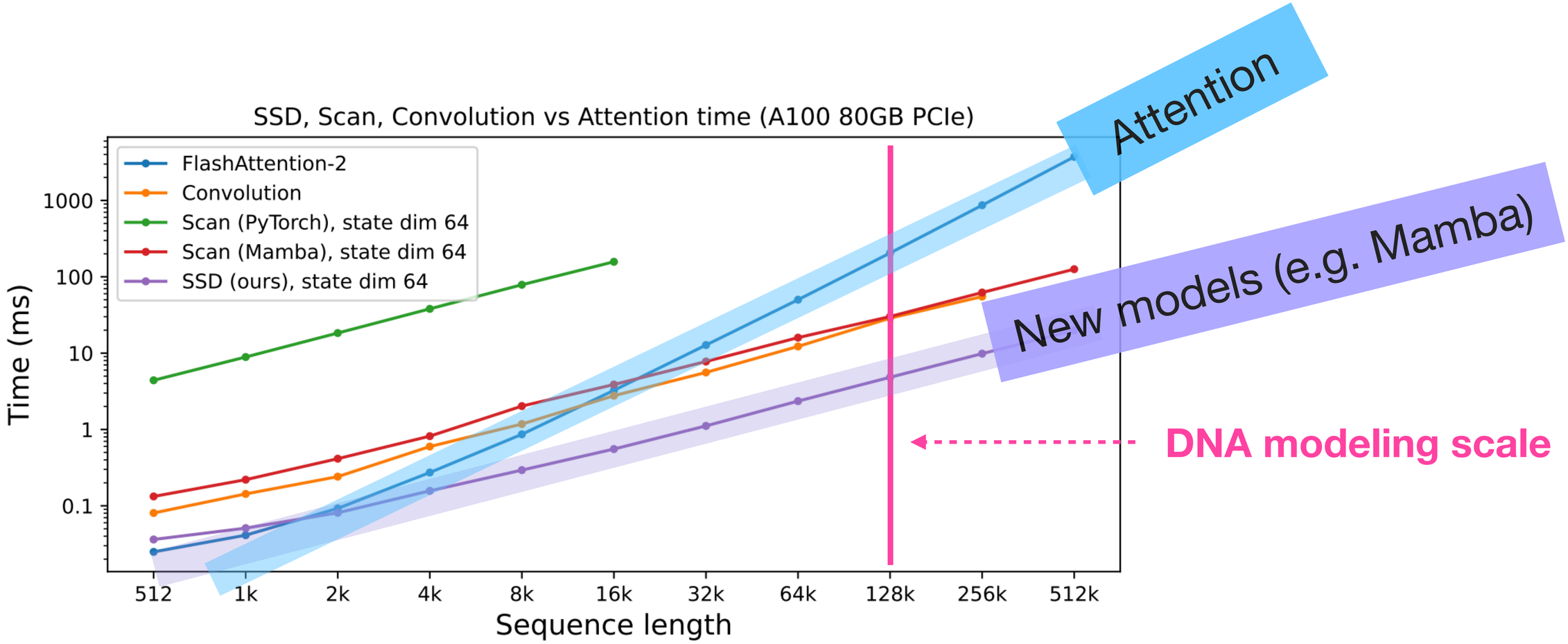


(c) BERT



(d) BERT

Motivation 2: New capabilities



An Empirical Study of Mamba-based Language Models

Roger Waleffe[1,2*] Wonmin Byeon[1] Duncan Riach[1] Brandon Norick[1†]
Vijay Korthikanti[1] Tri Dao[3,4] Albert Gu[5,6] Ali Hatamizadeh[1] Sudhakar Singh[1]
Deepak Narayanan[1] Garvit Kulshreshtha[1] Vartika Singh[1] Jared Casper[1]
Jan Kautz[1] Mohammad Shoeybi[1] Bryan Catanzaro[1]

[1]NVIDIA [2]University of Wisconsin-Madison [3]Princeton University
[4]Together AI [5]Carnegie Mellon University [6]Cartesia AI

*Our results show that while pure SSM-based models match or exceed Transformers on many tasks, both Mamba and Mamba-2 models lag behind Transformer models on tasks which require **strong copying or in-context learning abilities** (e.g., five-shot MMLU, Phonebook Lookup) or long-context reasoning.*

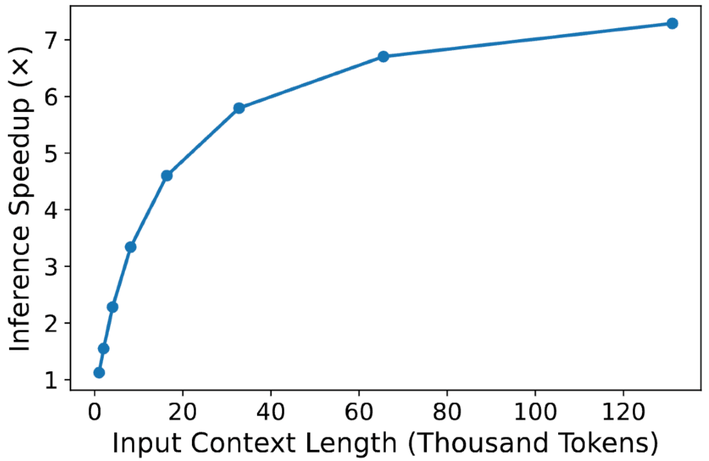


Figure 5: Predicted speedup to generate one token for an 8B-parameter Mamba-2-Hybrid model compared to a Transformer.

Table 2: Evaluation results for 8B-parameter models trained on 1.1T tokens. Pure SSM models (Mamba and Mamba-2) match or exceed Transformers on many natural language tasks, but fall short on others (e.g., MMLU) (see Section 3.3).

Model	WinoGrande	PIQA	HellaSwag	ARC-E	ARC-C	MMLU 0-Shot	MMLU 5-Shot	Avg. w/o MMLU	Avg
Transformer	69.22	78.29	75.6	73.15	**43.09**	**38.32**	**46.28**	67.87	**60.56**
Mamba	68.27	**78.89**	**75.63**	**75.42**	42.15	28.63	28.00	68.07	56.71
Mamba-2	**70.8**	78.35	75.54	75.13	43.00	28.94	29.19	**68.56**	57.28

Table 7: Detailed evaluation results on 12 common natural language tasks comparing an 8B-parameter hybrid model (Mamba-2-Hybrid) with the pure Mamba-2 SSM and Transformer models from Section 3.3 when training for 3.5T tokens. The Mamba-2-Hybrid model achieves the highest overall accuracy and is 2.65 points better than the Transformer on average.

Model	WG	PIQA	HellaSwag	ARC-E	ARC-C	MMLU 0-Shot	MMLU 5-Shot	OpenBook	TruthFul	PubMed	RACE	NQ	SquadV2	Avg
Transformer	69.14	78.62	75.89	73.27	43.77	45.69	50.07	42.00	35.48	69.20	39.52	15.15	53.4	53.17
Mamba-2	**71.59**	**79.82**	**77.69**	75.93	**48.12**	47.25	48.7	**44.2**	35.66	**75.2**	37.7	17.17	51.9	54.69
Mamba-2-Hybrid	71.27	79.65	**77.68**	**77.23**	47.7	**51.46**	**53.60**	42.80	**38.72**	69.80	**39.71**	**17.34**	**58.67**	**55.82**

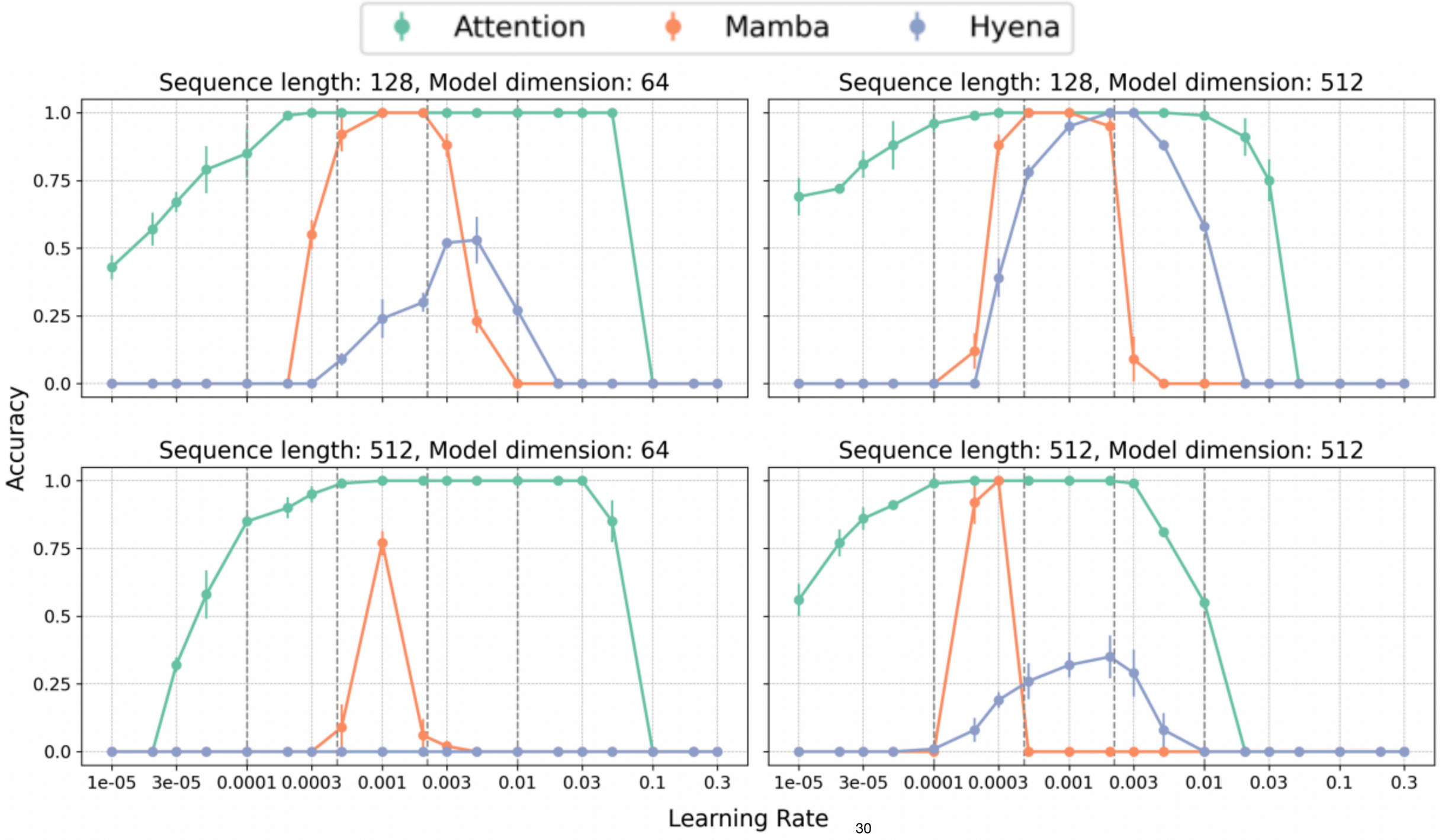
REVISITING ASSOCIATIVE RECALL IN MODERN RECURRENT MODELS

Destiny Okpekpe & Antonio Orvieto
MPI-IS and ELLIS Institute Tuebingen

*"**Hakuna Matata** means **no worries** for the rest of your days. **Hakuna Matata** means ..."*

$$A\ 6\ I\ 9\ \mathbf{C}\ \mathbf{7}\ P\ 1\ S\ 4\ D\ 2\ \mathbf{C} \rightarrow\ ?$$

TLDR: SSMs can totally do this as good as attention.
they are just super hard and unstable to train!!!


Attention
Mamba
Hyena
Sequence length: 128, Model dimension: 64
Sequence length: 128, Model dimension: 512
Sequence length: 512, Model dimension: 64
Sequence length: 512, Model dimension: 512
Accuracy
1.0
0.75
0.5
0.25
0.0
1e-05
3e-05
0.0001
0.0003
0.001
0.003
0.01
0.03
0.1
0.3
Learning Rate

It was all about optimization from the start!

On the difficulty of training Recurrent Neural Networks

Razvan Pascanu PASCANUR@IRO.UMONTREAL.CA
Universite de Montreal

Tomas Mikolov T.MIKOLOV@GMAIL.COM
Brno University

Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Universite de Montreal

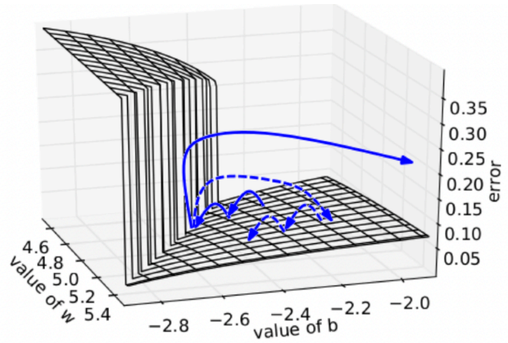


Figure 6. We plot the error surface of a single hidden unit recurrent network, highlighting the existence of high curvature walls. The solid lines depicts standard trajectories that gradient descent might follow. Using dashed arrow the diagram shows what would happen if the gradients is rescaled to a fixed size when its norm is above a threshold.

So what about understanding how to optimize better?

Recall: SSMs are super fast at inference, we really want these

Perhaps Muon? 👀

MUON IS SCALABLE FOR LLM TRAINING

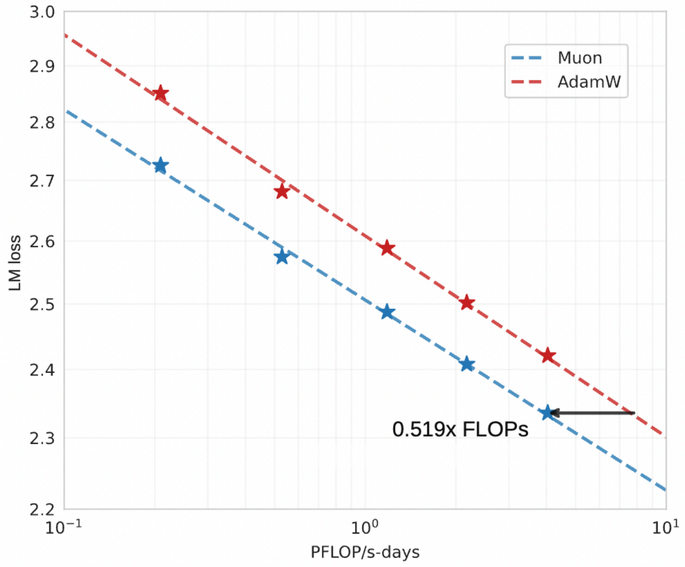
TECHNICAL REPORT

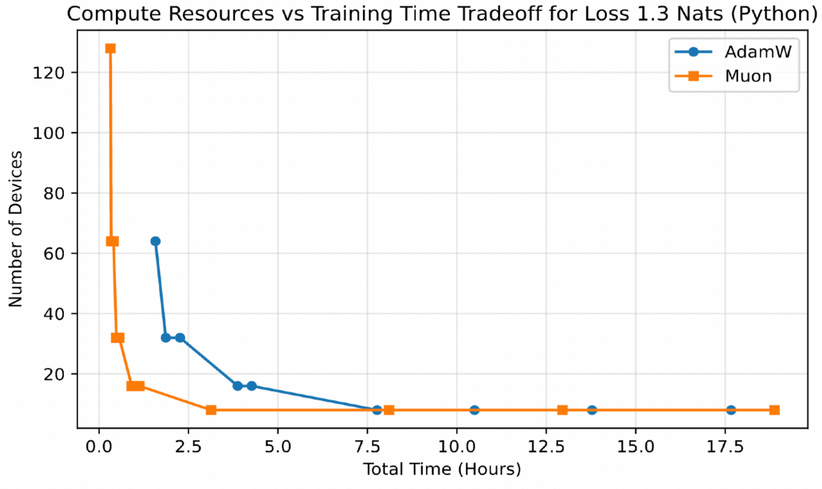
Jingyuan Liu[1] **Jianlin Su**[1] **Xingcheng Yao**[2] **Zhejun Jiang**[1] **Guokun Lai**[1] **Yulun Du**[1]
Yidao Qin[1] **Weixin Xu**[1] **Enzhe Lu**[1] **Junjie Yan**[1] **Yanru Chen**[1] **Huabin Zheng**[1]
Yibo Liu[1] **Shaowei Liu**[1] **Bohong Yin**[1] **Weiran He**[1] **Han Zhu**[1] **Yuzhi Wang**[1]
Jianzhou Wang[1] **Mengnan Dong**[1] **Zheng Zhang**[1] **Yongsheng Kang**[1] **Hao Zhang**[1]
Xinran Xu[1] **Yutao Zhang**[1] **Yuxin Wu**[1] **Xinyu Zhou**[1] * **Zhilin Yang**[1]

[1] Moonshot AI [2] UCLA

Practical Efficiency of Muon for Pretraining

Essential AI
San Francisco, CA
research@essential.ai





Perhaps Muon/Scion? 👀

MUON IS SCALABLE FOR LLM TRAINING

TECHNICAL REPORT

Jingyuan Liu[1] Jianlin Su[1] Xingcheng Yao[2] Zhejun Jiang[1] Guokun Lai[1] Yulun Du[1]
Yidao Qin[1] Weixin Xu[1] Enzhe Lu[1] Junjie Yan[1] Yanru Chen[1] Huabin Zheng[1]
Yibo Liu[1] Shaowei Liu[1] Bohong Yin[1] W
Jianzhou Wang[1] Mengnan Dong[1] Zheng Zhan
Xinran Xu[1] Yutao Zhang[1] Yuxin Wu[1]

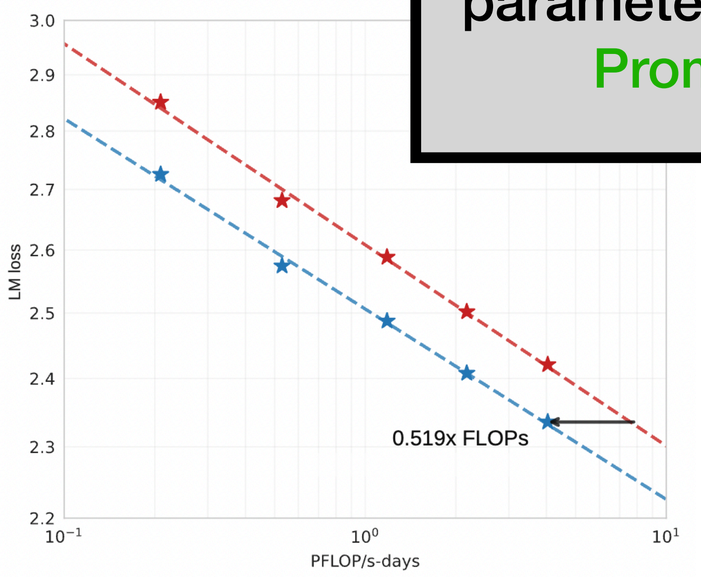
[1] Moonshot AI [2]

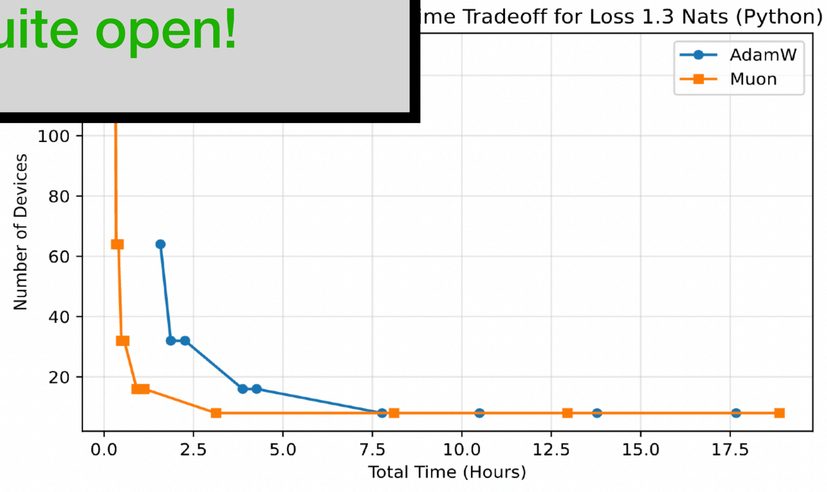
Practical Efficiency of Muon for Pretraining

Essential AI
Francisco, CA
ch@essential.ai

Yet Muon implementations reduce to Adam on many crucial network parameters… + other obscure things

Promising, but quite open!





MUON IS SCALABLE FOR LLM TRAINING

TECHNICAL REPORT

Jingyuan Liu[1] **Jianlin Su**[1] **Xingcheng Yao**[2] **Zhejun Jiang**[1] **Guokun Lai**[1] **Yulun Du**[1]
Yidao Qin[1] **Weixin Xu**[1] **Enzhe Lu**[1] **Junjie Yan**[1] **Yanru Chen**[1] **Huabin Zheng**[1]
Yibo Liu[1] **Shaowei Liu**[1] **Bohong Yin**[1] **Weiran He**[1] **Han Zhu**[1] **Yuzhi Wang**[1]
Jianzhou Wang[1] **Mengnan Dong**[1] **Zheng Zhang**[1] **Yongsheng Kang**[1] **Hao Zhang**[1]
Xinran Xu[1] **Yutao Zhang**[1] **Yuxin Wu**[1] **Xinyu Zhou**[1] * **Zhilin Yang**[1]

[1] Moonshot AI [2] UCLA

Matching update RMS of AdamW Muon is designed to update matrix-based parameters. In practice, AdamW is used in couple with Muon to handle non-matrix based parameters, like RMSNorm, LM head, and embedding parameters. We would like the optimizer hyper-parameters (learning rate η, weight decay λ) to be shared among matrix and non-matrix parameters.

We propose to match Muon's update RMS to be similar to that of AdamW. From empirical observations, AdamW's update RMS is usually around 0.2 to 0.4. Therefore, we scale Muon's update RMS to this range by the following adjustment:

$$\mathbf{W}_t = \mathbf{W}_{t-1} - \eta_t(0.2 \cdot \mathbf{O}_t \cdot \sqrt{\max(A, B)} + \lambda \mathbf{W}_{t-1}) \tag{4}$$

We validated this choice with empirical results (see Appendix A for details). Moreover, we highlighted that with this adjustment, Muon can directly **reuse** the learning rate and weight decay tuned for AdamW.

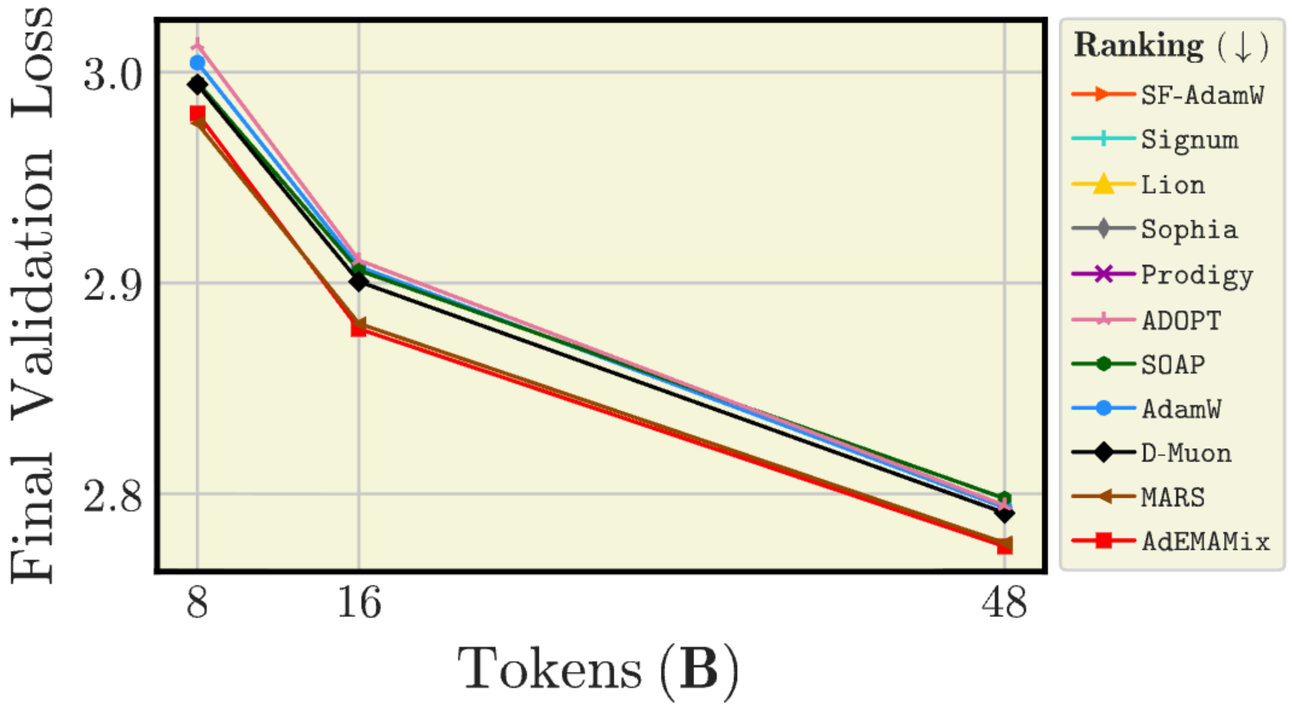
Sept 1st, 2025

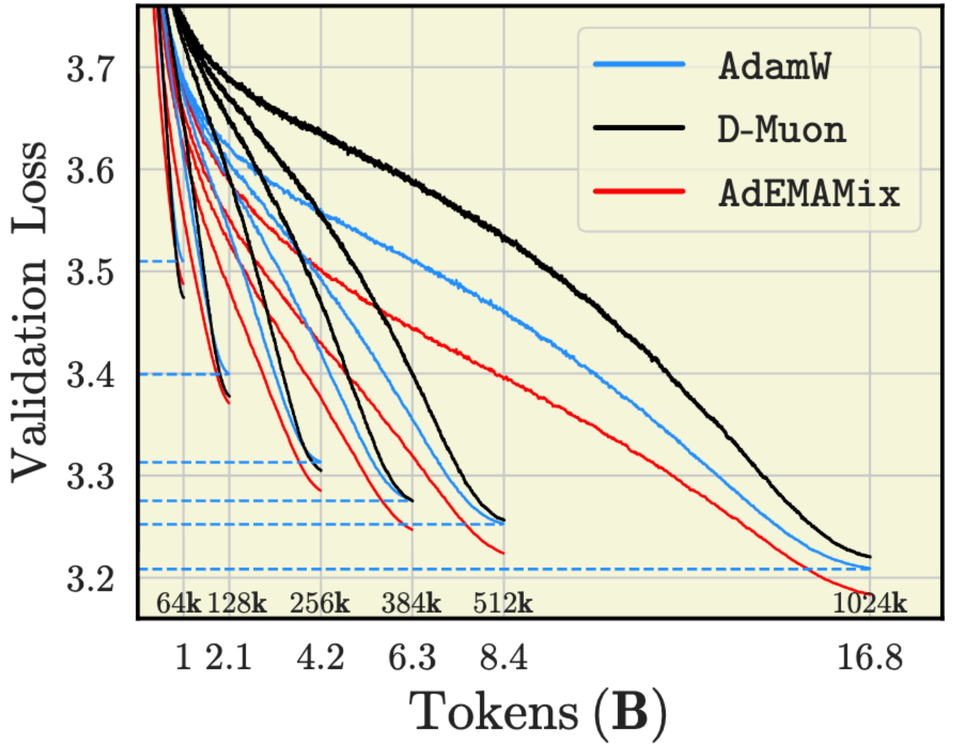
Benchmarking Optimizers for Large Language Model Pretraining

Andrei Semenov
EPFL
andrii.semenov@epfl.ch

Matteo Pagliardini
EPFL
matteo.pagliardini@epfl.ch

Martin Jaggi
EPFL
martin.jaggi@epfl.ch





Fantastic Pretraining Optimizers and Where to Find Them

Kaiyue Wen
Stanford University
kaiyuew@stanford.edu

David Hall
Stanford University
dlwh@cs.stanford.edu

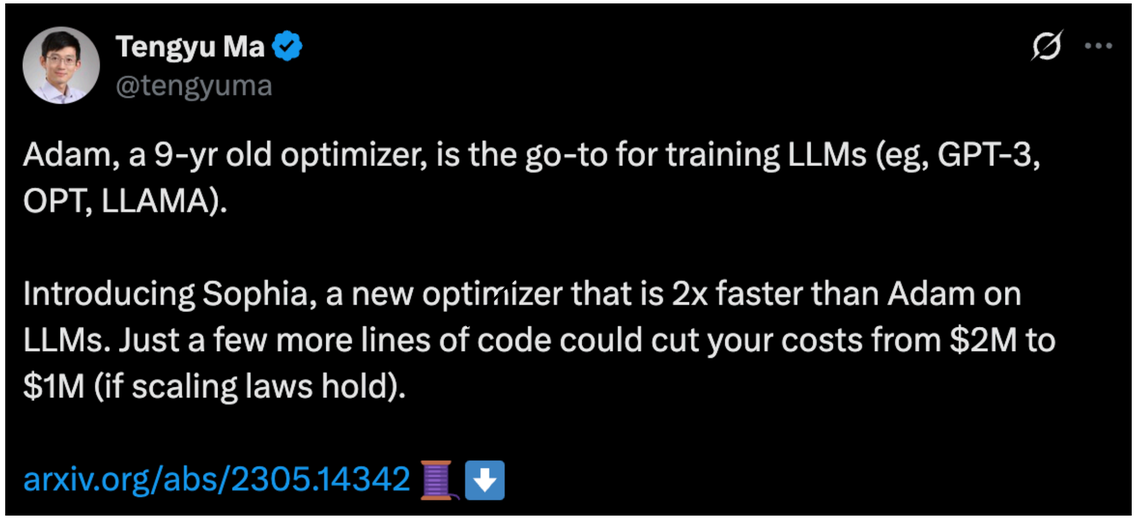
Tengyu Ma
Stanford University
tengyuma@stanford.edu

Percy Liang
Stanford University
pliang@cs.stanford.edu

September 8, 2025

AdamW has long been the dominant optimizer in language model pretraining, despite numerous claims that alternative optimizers offer 1.4 to 2× speedup. We posit that two methodological shortcomings have obscured fair comparisons and hindered practical adoption: (i) unequal hyperparameter tuning and (ii) limited or misleading evaluation setups. To address these two issues, we conduct a systematic study of ten deep learning optimizers across four model scales (0.1B–1.2B parameters) and data-to-model ratios (1–8× the Chinchilla optimum). We find that fair and informative comparisons require rigorous hyperparameter tuning and evaluations across a range of model scales and data-to-model ratios, performed at the end of training. First, optimal hyperparameters for one optimizer may be suboptimal for another, making blind hyperparameter transfer unfair. Second, the actual speedup of many proposed optimizers over well-tuned baselines is lower than claimed and decreases with model size to only 1.1× for 1.2B parameter models. Thirdly, comparing intermediate checkpoints before reaching the target training budgets can be misleading, as rankings between two optimizers can flip during training due to learning rate decay. Through our thorough investigation, we find that all the fastest optimizers such as Muon and Soap, use matrices as preconditioners — multiplying gradients with matrices rather than entry-wise scalars. However, the speedup of matrix-based optimizers is inversely proportional to model scale, decreasing from 1.4× over AdamW for 0.1B parameter models to merely 1.1× for 1.2B parameter models.

Btw,
this is actually
a very good
paper..



…. Same story again? Maybe, let’s hope not.

Ok! How do we start?

Simplified Models

$$m_i^k = \text{EMA}_{\beta_1}\left[\nabla_i L(w^k)\right], \qquad v_i^k = \text{EMA}_{\beta_2}\left[\nabla_i L(w^k)^2\right]$$

$\beta_1 = 0$

$$w_i^{k+1} = w_i^k - \frac{\eta}{\sqrt{v_i^k + \epsilon}} m_i^k$$

Adam

$\beta_2, \epsilon = 0$

$$w_i^{k+1} = w_i^k - \frac{\eta}{\sqrt{v_i^k + \epsilon}} \nabla_i L(w_i^k)$$

RMSprop

$$w_i^{k+1} = w_i^k - \eta \text{sign}(\nabla_i L(w_i^k))$$

SignSGD

Momentum reintroduced

$$w_i^{k+1} = w_i^k - \eta \text{sign}(m_i^k)$$

Signum

Published as a conference paper at ICLR 2020

WHY GRADIENT CLIPPING ACCELERATES TRAINING: A THEORETICAL JUSTIFICATION FOR ADAPTIVITY

Jingzhao Zhang, Tianxing He, Suvrit Sra & Ali Jadbabaie
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{jzhzhang, tianxing, suvrit, jadbabai}@mit.edu

Published as a conference paper at ICLR 2023

NOISE IS NOT THE MAIN FACTOR BEHIND THE GAP BETWEEN SGD AND ADAM ON TRANSFORMERS, BUT SIGN DESCENT MIGHT BE

Frederik Kunstner, Jacques Chen, J. Wilder Lavington & Mark Schmidt[†]
University of British Columbia, Canada CIFAR AI Chair (Amii)[†]
{kunstner,jola2372,schmidtm}@cs.ubc.ca
jacquesc@students.cs.ubc.ca

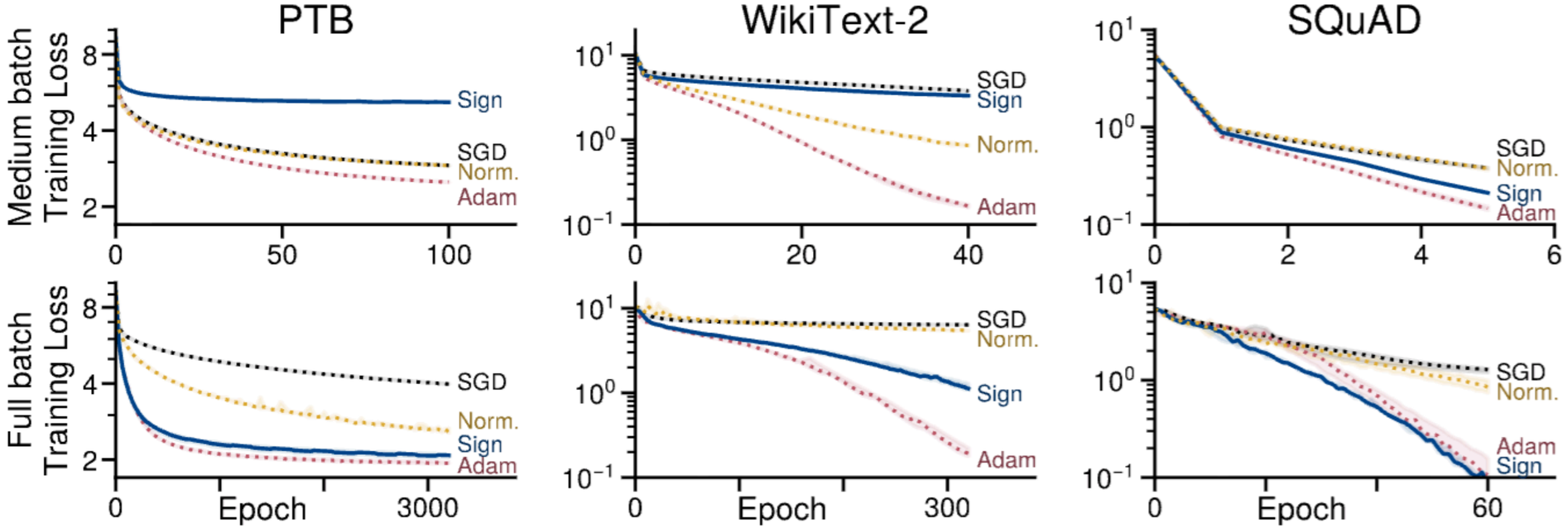


Figure 7: **Sign descent can close most of the gap between GD and Adam in full batch.** At small

Published as a conference paper at ICLR 2020

WHY GRADIENT CLIPPING ACCELERATES TRAINING: A THEORETICAL JUSTIFICATION FOR ADAPTIVITY

Jingzhao Zhang, Tianxing He, Suvrit Sra & Ali Jadbabaie
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{jzhzhang, tianxing, s

Published as a conference paper at ICLR 2023

NOISE IS NOT THE MAIN FACTOR BEHIND THE GAP BETWEEN SGD AND ADAM ON TRANSFORMERS, BUT SIGN DESCENT MIGHT BE

Frederik Kunstner, Jacques Chen, J. Wilder Lavington & Mark Schmidt[†]
University of British Columbia, Canada CIFAR AI Chair (Amii)[†]
{kunstner,jola2372,schmidtm}@cs.ubc.ca

Quite small experiments, but theory is good!

1) **How big** is the Signum-Adam gap in standard LM training setups?

2) **Is that all?** Is there more to understand? And if so, can we have a simplified model.

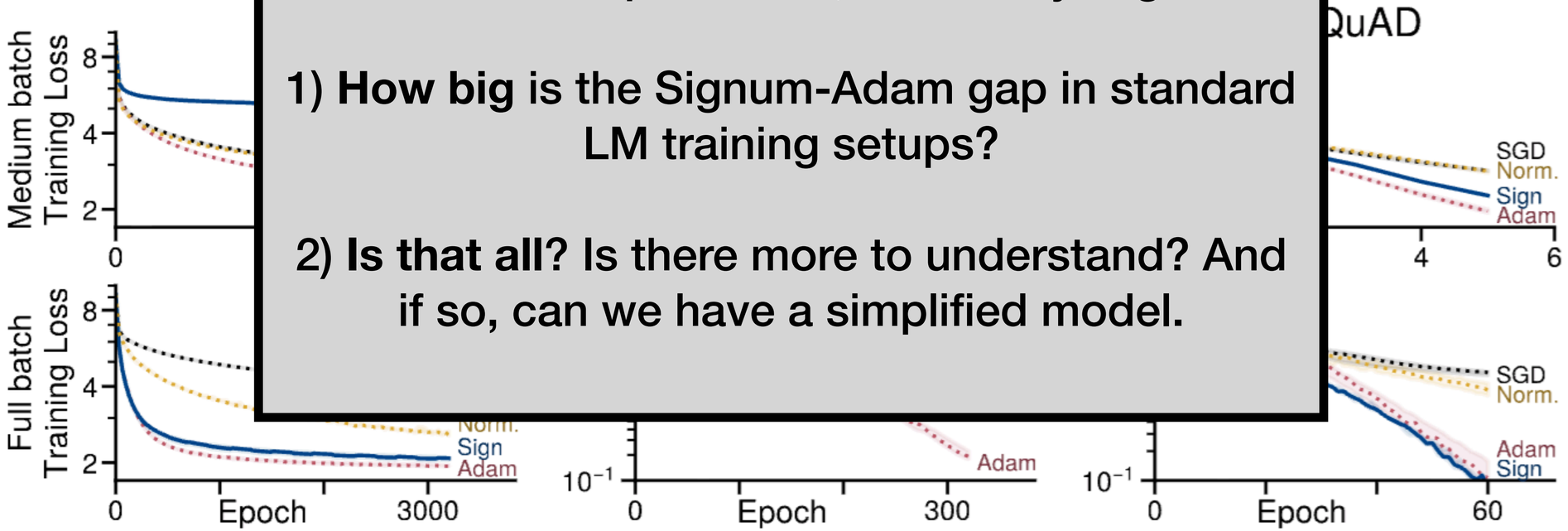


Figure 7: **Sign descent can close most of the gap between GD and Adam in full batch.** At small

I got a bit obsessed.. so I did spend months in my office tuning optimizers in LMs..

(> 3000 runs)

Implementation & Enhancements

- **Base**: nanoGPT
- **Additions**: RoPE, RMSNorm (Pre), SwiGLU
- **Tokenizer**: GPT-NeoX (vocab = 50,280)

Training Recipe

- Precision: **bfloat16** (FP16 for inference)
- **LR schedule**: warm-up (10%) → cosine decay to 1e–5
- **Gradient clipping**: norm > 1
- **Validation**: 100M tokens
- **No weight tying**

Compute

- **160M model**
  - 12 layers, 12 heads, d=768, FlashAttention
  - 1× A100-80GB → **~5.8h/run**
- **410M model**
  - 24 layers, 16 heads, d=1024, FlashAttention
  - 8× A100-80GB → **~4.8h/run**

Other Settings

- Pre-LN backbone with skip connections and small init
- RoPE on 25% dims
- Dropout: disabled
- Layer-wise scaled initialization

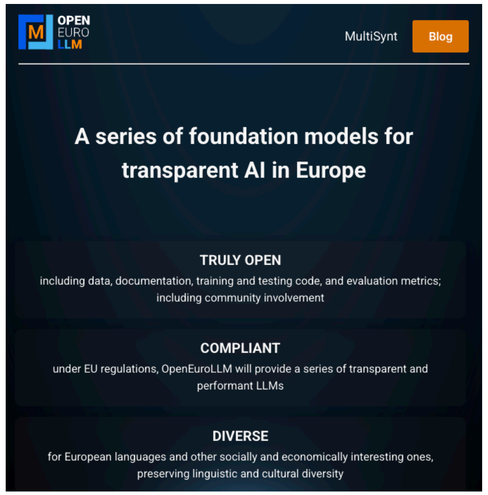
Pretrain a Transformer on Language Modeling

A minimal yet efficient implementation of causal language modeling in PyTorch.

It features a custom torch-compilable Transformer model implementation supporting RoPE, GLU, and RMSNorm. It supports distributed training via Distributed Data Parallel (DDP).

A dedicated script is included for downloading, tokenizing, and chunking data, making data preparation seamless.

https://github.com/Niccolo-Ajroldi/plainLM



Want to contribute to open-source LMs?

We are hiring interns and software engineers!!

https://institute-tue.ellis.eu/en/jobs/openeurollm

We do heavy tuning for each method, e.g. RMSprop:

$$w_i^{k+1} = w_i^k - \text{schedule}_k \cdot \frac{\eta}{\sqrt{\text{EMA}_{\beta_2}(g_i^k) + \epsilon}} \nabla_i L(w_i^k)$$

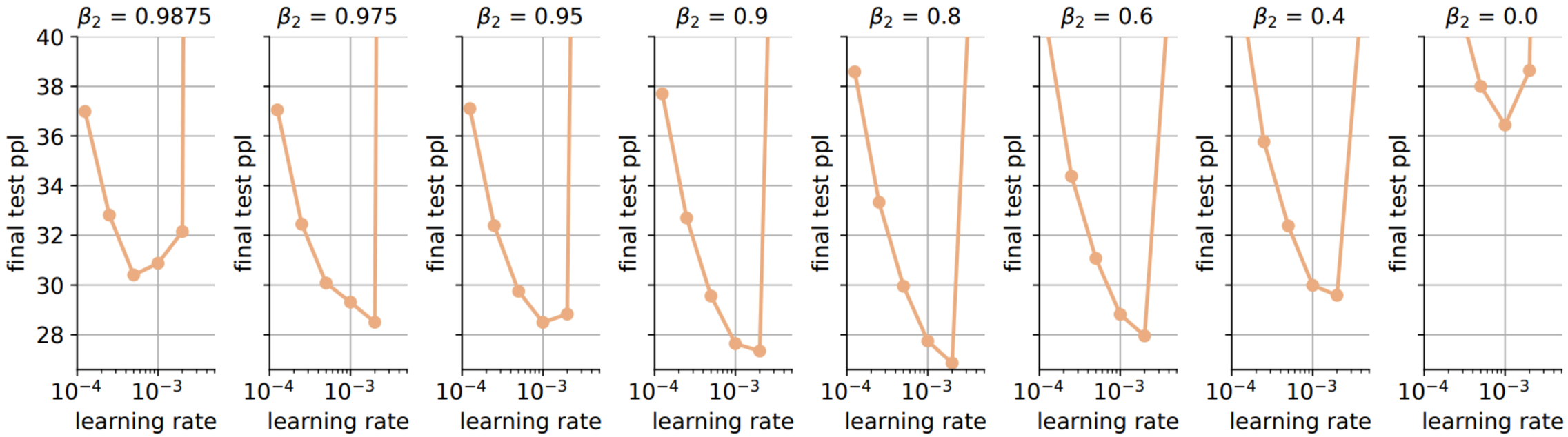


Figure 10: ***RMSprop with decoupled weight decay*** *0.1. Implemented with Pytorch AdamW setting* $\beta_1 = 0$.

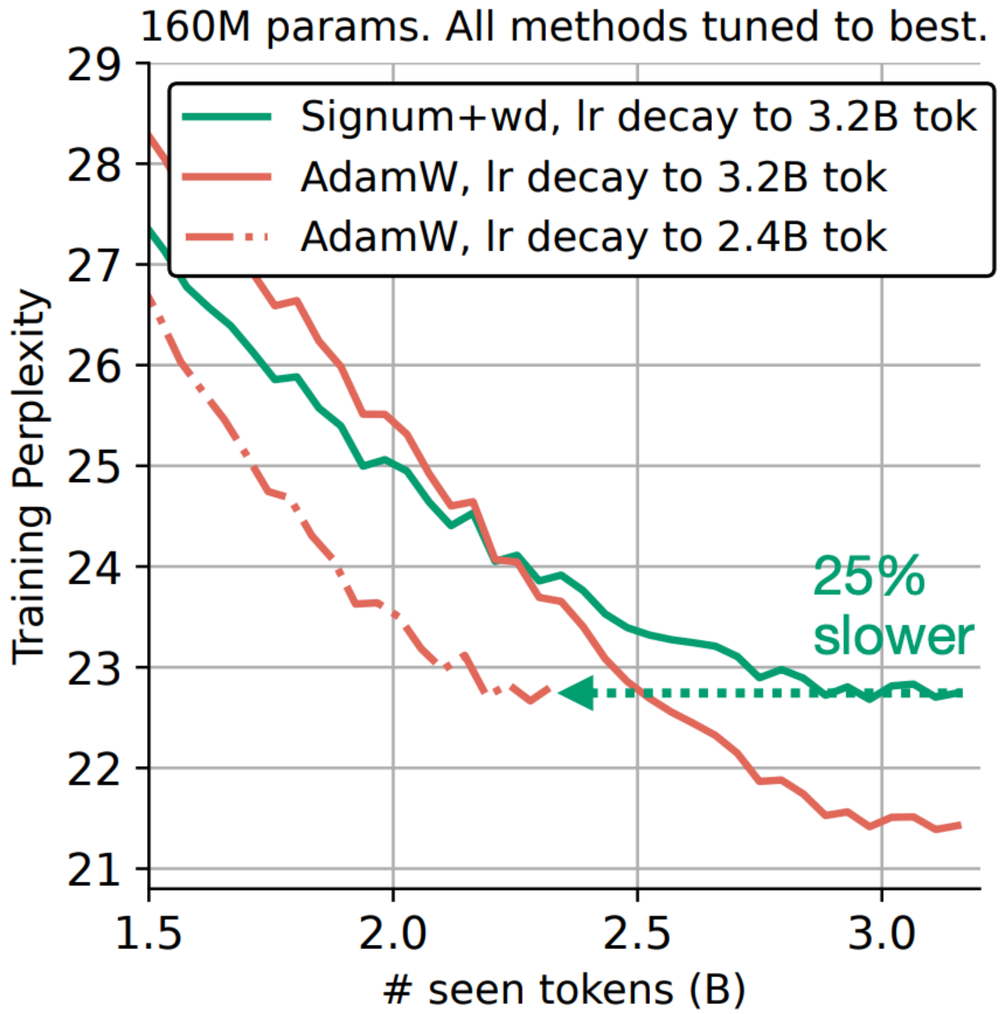
We heavily tune all methods claiming a connection to Adam. SignSGD + momentum closes 96% gap

Table 1: ***(Signum closes 96% of the perplexity gap between Adam and SGD)*** *Validation perplexity comparison of widely used optimizers that interpolate between* `SGD` *and* `Adam`*, evaluated on a language modeling task (160M parameters, 3.2B SlimPajama tokens, sequence length 2048, batch size 256 – Chinchilla optimal). We report the mean and 2-sigma interval of validation perplexity (on 100M held-out tokens) across 3 initialization seeds. Weight decay is always decoupled [Loshchilov and Hutter, 2019] and set to* 0.1 *[Biderman et al., 2023, Liu et al., 2024] except for* `SGD` *where we further tune (§B).* `RMSprop` *does not use momentum, and Gclip is global norm clipping to* 1 *(before applying momentum), Cclip is coordinate-wise clipping (after applying momentum). Other hyperparameters, for all other methods, are carefully tuned, see e.g. Figure 2 and §3.*
To optimally tune hyperparameters (e.g. Figure 2), we performed a total of 582 full training runs.

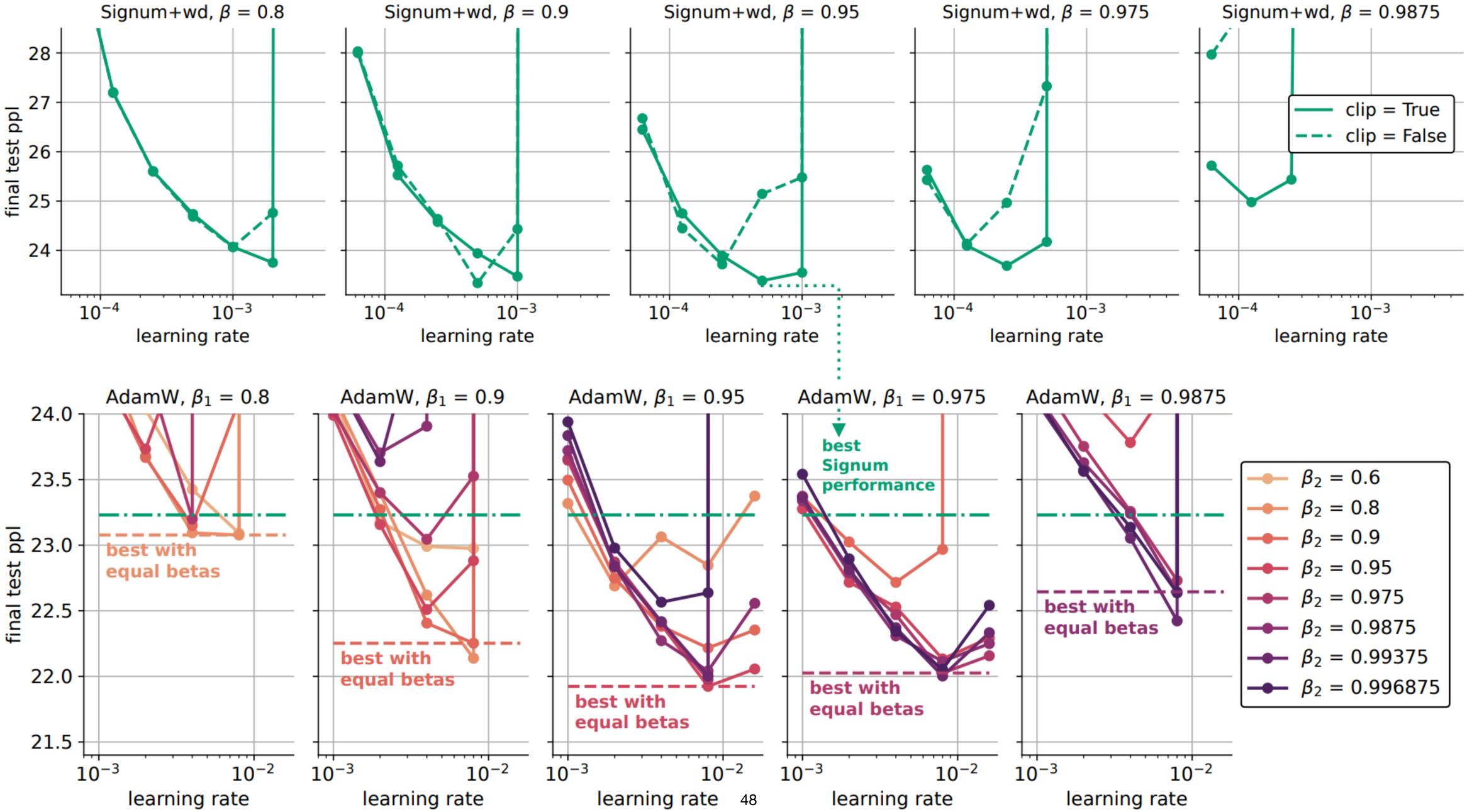
	Adam	Signum	RMSprop	SGD+Cclip	SignSGD	SGD+Gclip	SGD
Val ppl.	**21.86**± **0.21**	23.23± 0.16	27.04± 0.34	33.40± 0.39	36.78± 0.57	37.76± 0.61	53.62± 5.14

Signum is a good model but not a good method. It is 25% slower at optimal tuning!

Figure 1: *Pretraining on SlimPajama with Chinchilla-optimal [Hoffmann et al., 2022] scaling. Both momentum and learning rates for* ***Signum*** *are extensively tuned (§3). While* ***Signum*** *closes 96% of the perplexity gap between* ***Adam*** *and* ***SGD*** *with momentum (Table 1), still results in a 25% slowdown : Adam achieves the same performance with 3/4 of the budget.*



Actually, $\beta_1 = \beta_2$ works very well in Adam!



410M parameters, Chinchilla-optimal

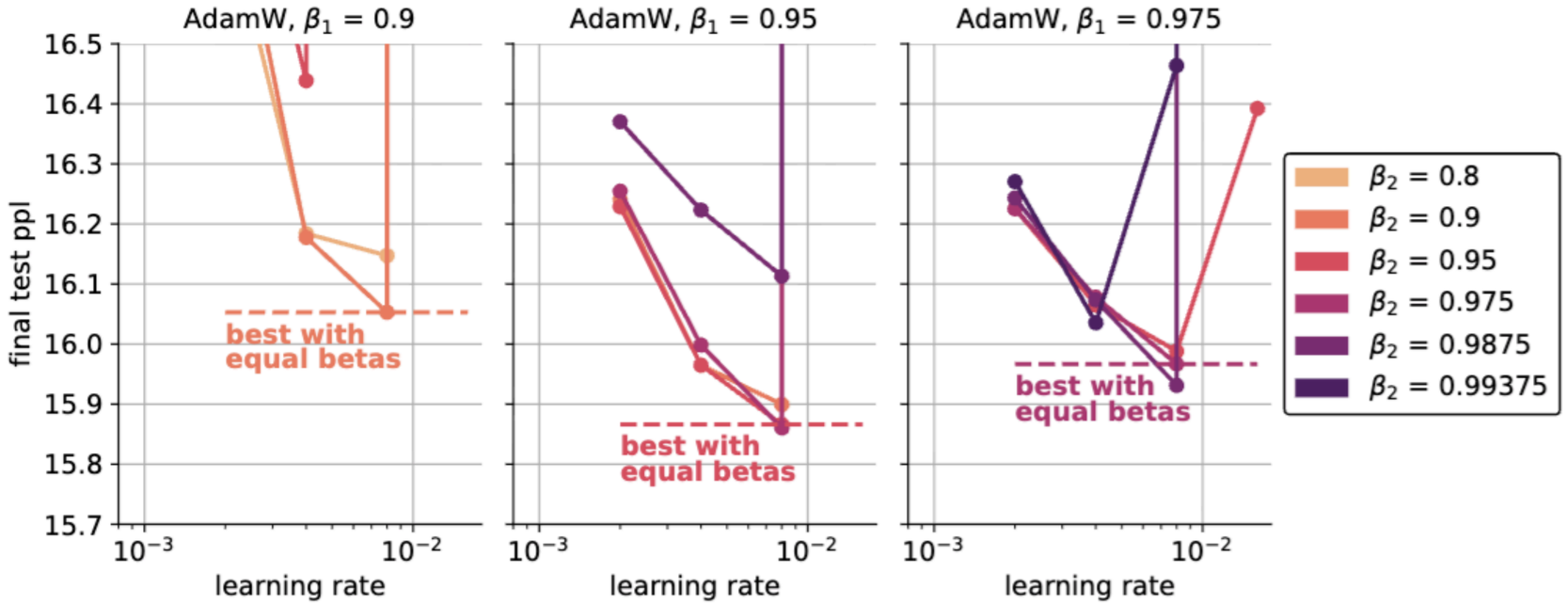


Figure 4: *The final validation performance (100M held-out tokens) for 44 trained LMs with 420M parameters trained on 8.2 B SlimPajama tokens (Chinchilla-optimal).* ***Equal betas yields near-optimal performance****. We use gradient clipping and a batch size of 512 (scaled by 2 compared to Figure 2, as suggested by Zhang et al. [2025]). Sequence length is 2048, weight decay is* 0.1*. Note that the standard setting* $(0.9, 0.95)$ *is quite suboptimal here.*

Different batch sizes? (Training for 2.5B tokens)

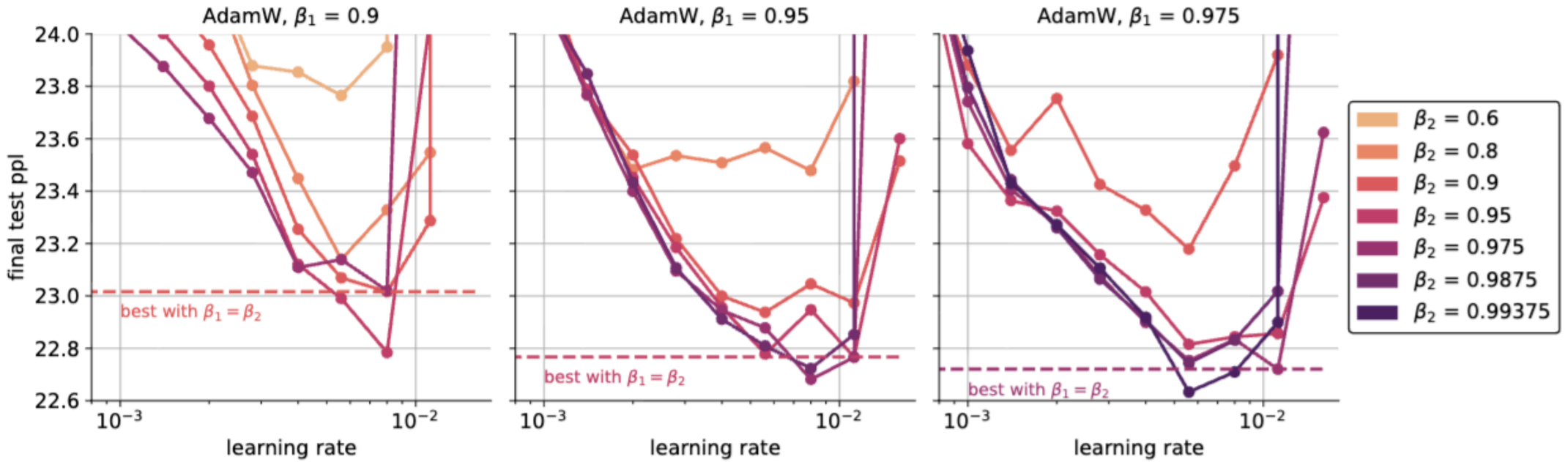


Figure 15: ***Adam, batch size 128*** *trained for 2.5B tokens. Other settings are same setting as Figure 2.*

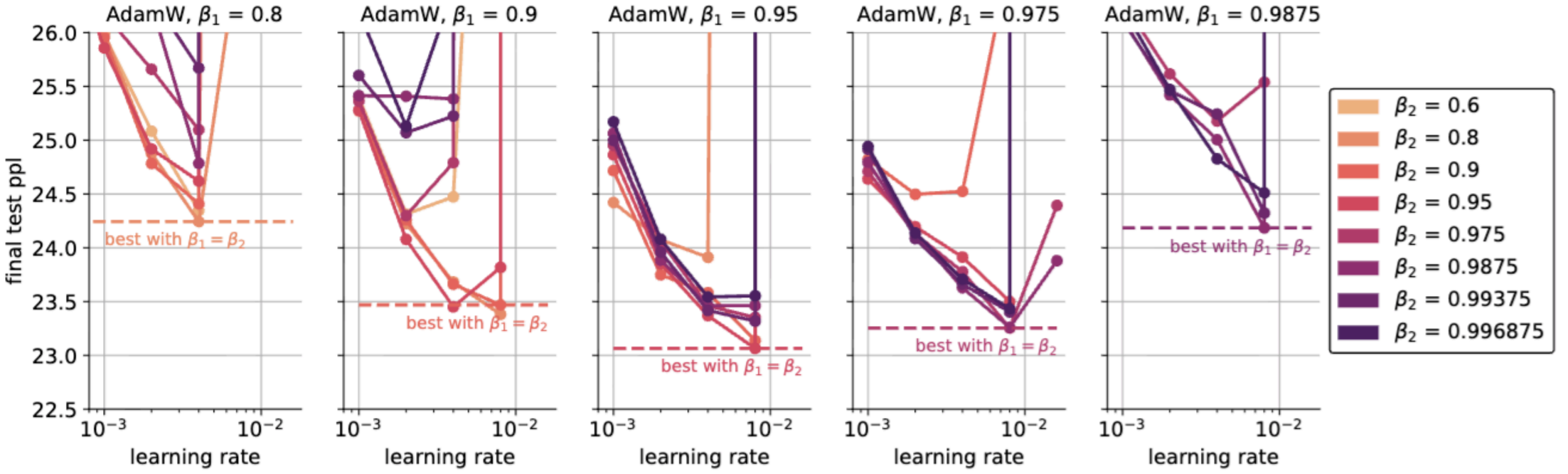


Figure 13: ***Adam, batch size 256*** *trained for 2.5B tokens. Other settings are same setting as Figure 2.*

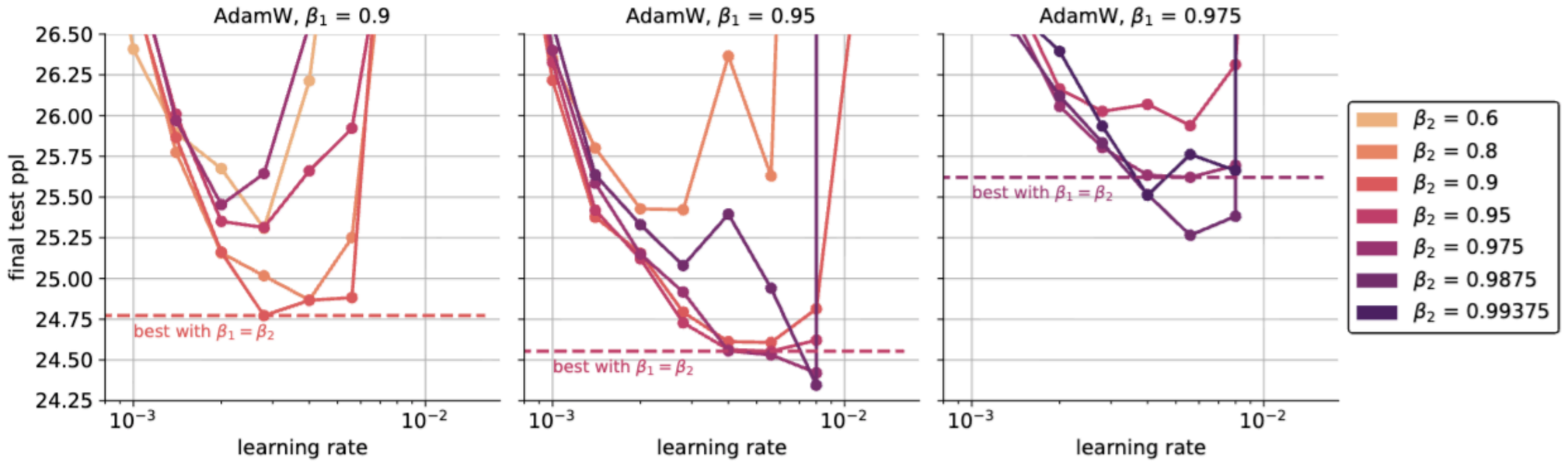


Figure 14: ***Adam, batch size 512*** *trained for 2.5B tokens. Other settings are same setting as Figure 2.*

More Data? 2x the tokens (160M)

β_1 / β_2	0.8	0.9	0.95	0.975	0.9875	0.99375
0.9	20.15	***20.00***	20.00	19.94	20.14	20.8
0.95	20.14	19.93	***19.87***	19.93	19.88	20.24
0.975	262.12	20.01	19.80	***19.73***	19.72	19.74
0.9875	5797	189	20.40	19.88	***19.81***	19.87

Can we simplify Adam given this empirical insight? **Can we ground it in theory?**

With $\beta_1 = \beta_2 = \beta$ and dropping ϵ (minor impact on performance) Adam reads

$$d_k = \frac{\mathsf{EMA}_\beta[g_k]}{\sqrt{\mathsf{EMA}_\beta[g_k^2]}}$$

Proposition. Recall that $m_k := \mathsf{EMA}_\beta[g_k]$. Then if $\beta_1 = \beta_2 = \beta$,

$$d_k = \frac{m_k}{\sqrt{m_k^2 + \beta\,\mathsf{EMA}_\beta[(m_{k-1} - g_k)^2]}}$$

Secret Sauce

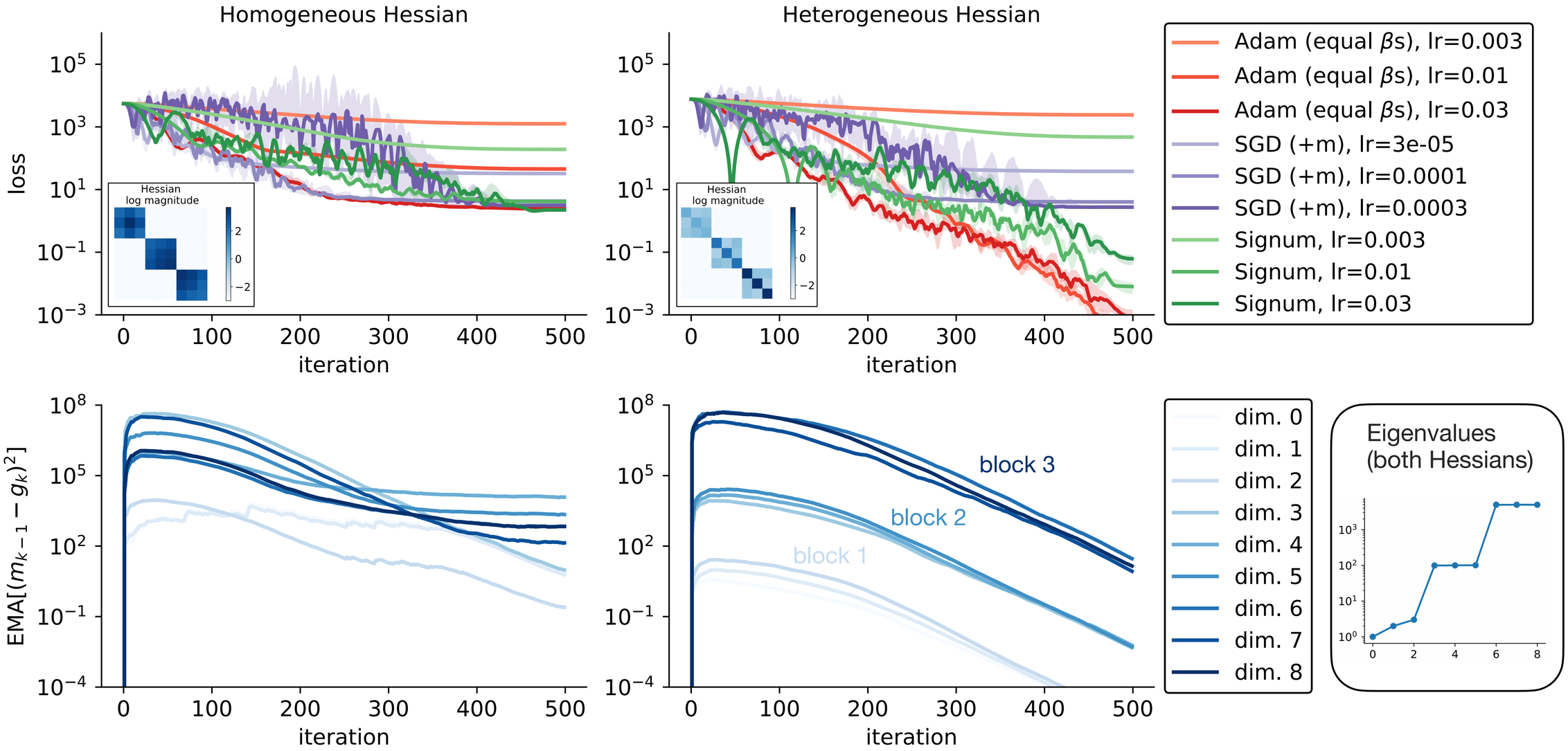
So, if $\sigma_k^2 := \beta\,\mathsf{EMA}_\beta[(m_{k-1} - g_k)^2]$, we get

$$d_k = \frac{m_k}{\sqrt{m_k^2 + \sigma_k^2}} = \frac{1}{\sqrt{1 + \sigma_k^2/m_k^2}} \cdot \mathrm{sign}(m_k)$$

Toy Model

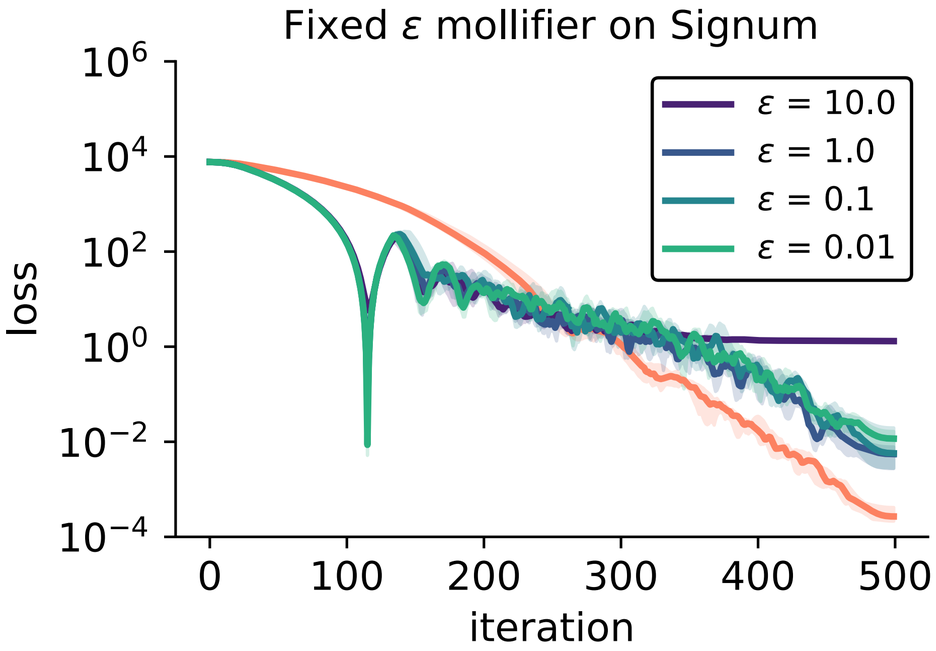
Why Transformers Need Adam: A Hessian Perspective

Yushun Zhang[12], Congliang Chen[12], Tian Ding[2], Ziniu Li[12], Ruoyu Sun[12*], Zhi-Quan Luo[12]
[1]The Chinese University of Hong Kong, Shenzhen, China
[2]Shenzhen Research Institute of Big Data
{yushunzhang,congliangchen,ziniuli}@link.cuhk.edu.cn
dingtian@sribd.cn, sunruoyu@cuhk.edu.cn, luozq@cuhk.edu.cn

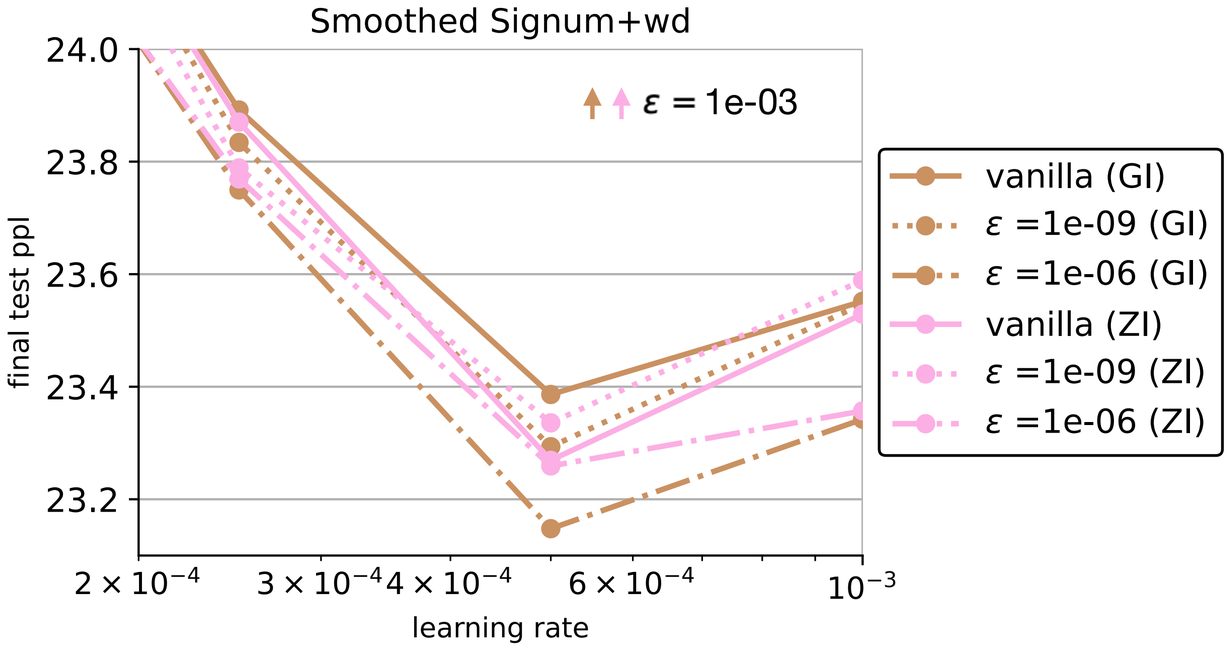


What about just adding an epsilon?

Toy quadratics



LLMs



Why $m_k := \text{EMA}_\beta[g_k]$ and $\sigma_k^2 := \beta\,\text{EMA}_\beta[(m_{k-1} - g_k)^2]$?
They look like mean and variance of gradients, but can we be more precise?

Consider the following **online variational inference** model:

- At each iteration we observe a new gradient g_{k+1}.
- Assume uniformly in time $g_{k+1} \sim \mathcal{N}(m, \sigma)$, where both m, σ are unknown.
- We want to update our estimate (m_{k+1}, σ_{k+1}) of (m, σ) such that
  A. It becomes likely that $g_{k+1} \sim \mathcal{N}(m_{k+1}, \sigma_{k+1})$
  B. We do not move much from previous distribution $\mathcal{N}(m_k, \sigma_k)$

Idea: Simultaneous estimation of mean and variance in the above model solves

$$(m_{m+1}, \sigma_{k+1}) = \text{argmin}_{m,\sigma\geq 0} \left[-\log p(g_{k+1}\,|\,m, \sigma) + \frac{1}{\lambda}\text{KL}\left(\mathcal{N}(m_k, \sigma_k)\,\|\,\mathcal{N}(m, \sigma)\right)\right]$$

$$\min_{m,\sigma\geq 0}\ \left[-\log p(g_{k+1}\,|\,m,\sigma)+\ \frac{1}{\lambda}\mathrm{KL}\left(\mathcal{N}(m_k,\sigma_k)\,\|\,\mathcal{N}(m,\sigma)\right)\right]$$

Theorem. Let $\beta=(1+\lambda)^{-1}$, then the solution to the problem above is

$$m_{k+1}=\beta m_k+(1-\beta)g_{k+1}=\mathsf{EMA}_\beta[g_{k+1}]$$

$$\sigma_{k+1}=\beta\sigma_k+\beta(1-\beta)(m_k-g_{k+1})^2=\beta\,\mathsf{EMA}_\beta\left[(m_k-g_{k+1}]^2\right)$$

Proof. Recall some well-known formulas for Gaussians

$$-\log p(g_{k+1}\,|\,m,\sigma)=\frac{1}{2}\log\sigma+\frac{1}{2\sigma}(g_{k+1}-m)^2$$

$$\mathrm{KL}\left(\mathcal{N}(m_k,\sigma_k)\,\|\,\mathcal{N}(m,\sigma)\right)=\frac{1}{2}\left[\frac{\sigma_k}{\sigma}+\frac{(m_k-m)^2}{\sigma}-1-\log\left(\frac{\sigma_k}{\sigma}\right)\right]$$

After some manipulations, we get:

$$\min_{m,\sigma\geq 0} F(m,\sigma) = \frac{1}{2}\frac{1+\lambda}{\lambda}\log(\sigma) + \frac{1}{2\sigma}\left[(g-m)^2 + \frac{1}{\lambda}\left(\sigma_k + (m_k - m)^2\right)\right] + \text{const}$$

$$\frac{\partial F}{\partial m} = 0 \quad \Longrightarrow \quad m = \frac{\lambda g + m_k}{1+\lambda}$$

$$\frac{\partial F}{\partial \sigma} = 0 \quad \Longrightarrow \quad \sigma = \frac{\lambda(g-m)^2 + \left[\sigma_k + (m_k - m)^2\right]}{1+\lambda}$$

Now m is super simple: since $\beta = (1+\lambda)^{-1}$,

$$m = \frac{1}{1+\lambda} m_k + \frac{\lambda}{1+\lambda} g \quad \Longrightarrow \quad m = \beta m_k + (1-\beta) g$$

For σ we need a bit more work, since the solution depends on the new estimate m

$$\sigma = \frac{\lambda(g-m)^2 + \left[\sigma_k + (m_k - m)^2\right]}{1+\lambda}$$

Recall that $m = \dfrac{\lambda g + m_k}{1+\lambda}$, so we have

$$(g-m)^2 = \frac{(g-m_k)^2}{(1+\lambda)^2}, \quad (m_k - m)^2 = \frac{\lambda^2 (g-m_k)^2}{(1+\lambda)^2}$$

Therefore,

$$\sigma = \frac{\lambda(g-m_k)^2}{(1+\lambda)^2} + \frac{\sigma_k}{1+\lambda} \quad \Longrightarrow \quad \sigma = \frac{\sigma_k}{1+\lambda} + \frac{\lambda(g-m_k)^2}{(1+\lambda)^2}$$

Which implies, under $\beta = (1+\lambda)^{-1}$,

$$\sigma_{k+1} = \beta\sigma_k + \beta(1-\beta)(m_k - g_{k+1})^2 = \beta\,\mathsf{EMA}_\beta\left[(m_k - g_{k+1}]^2\right)$$

Sounds familiar? Yes! Was already done in 2018!

Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients

Lukas Balles [1] Philipp Hennig [1]

2018!!

$$\frac{m_t}{\sqrt{v_t}} = \frac{\operatorname{sign}(m_t)}{\sqrt{\frac{v_t}{m_t^2}}} = \sqrt{\frac{1}{1+\frac{v_t-m_t^2}{m_t^2}}} \odot \operatorname{sign}(m_t)$$

The missing piece in Balles and Hennig (2018) was to show when and if the term $\sigma_k^2 := v_k - m_k^2$ is a measure of variance.

We show: $v_k - m_k^2$ only has a precise variance interpretation for the case $\beta_1 = \beta_2$.

Proposition: Adam's update d_k can be represented as

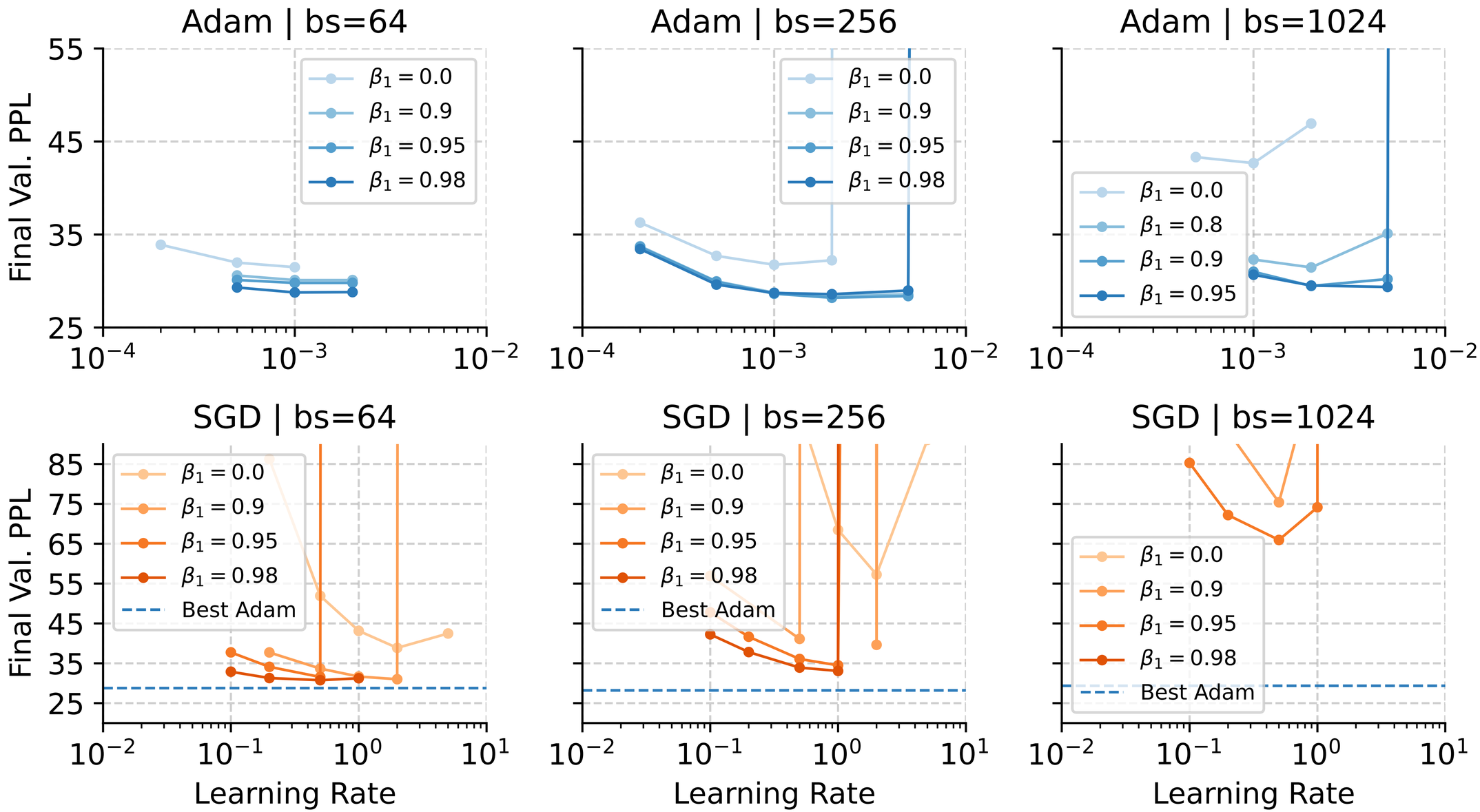
$$d_k = \frac{m_k}{\sqrt{m_k^2 + \gamma\,\mathsf{EMA}_\tau[(am_{k-1} - bg_k)^2]}}$$

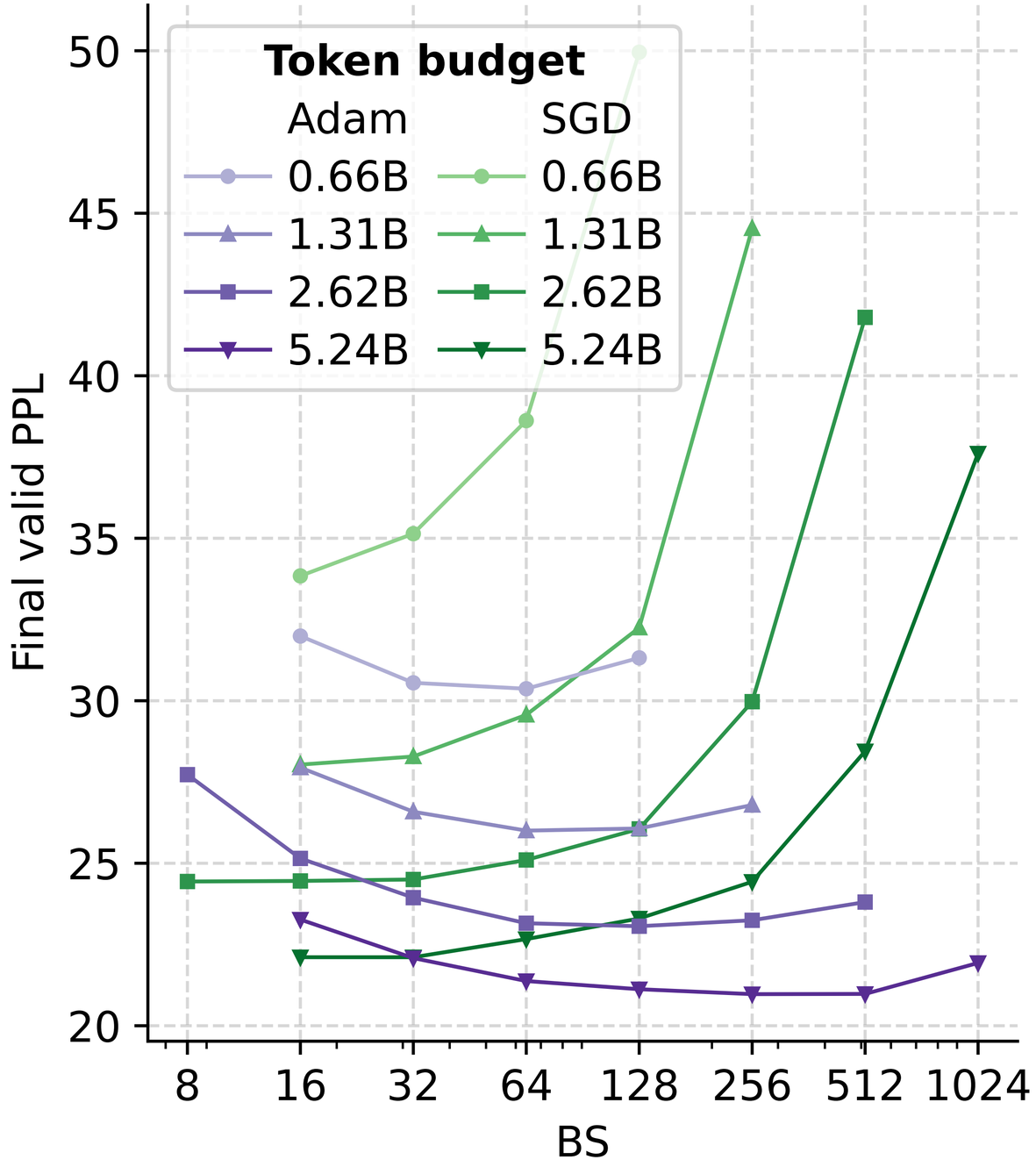
for some $a, b, \gamma \in \mathbb{R}$ and $\tau \in (0,1)$ if and only if $\beta_1 = \beta_2$.

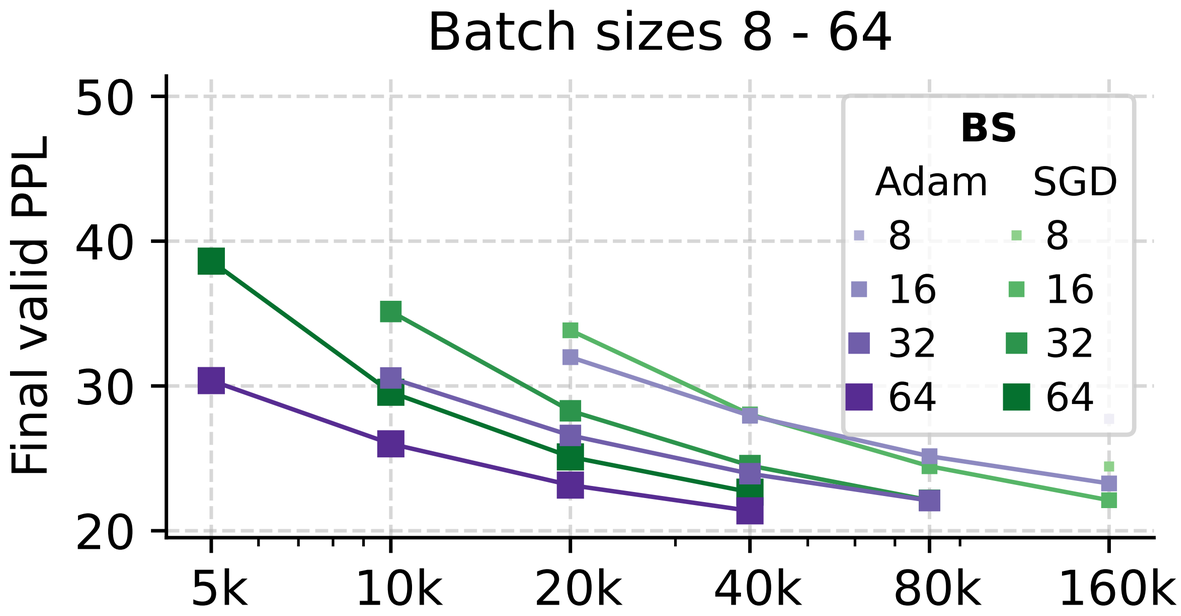
Sometimes though… reality is just much simpler.

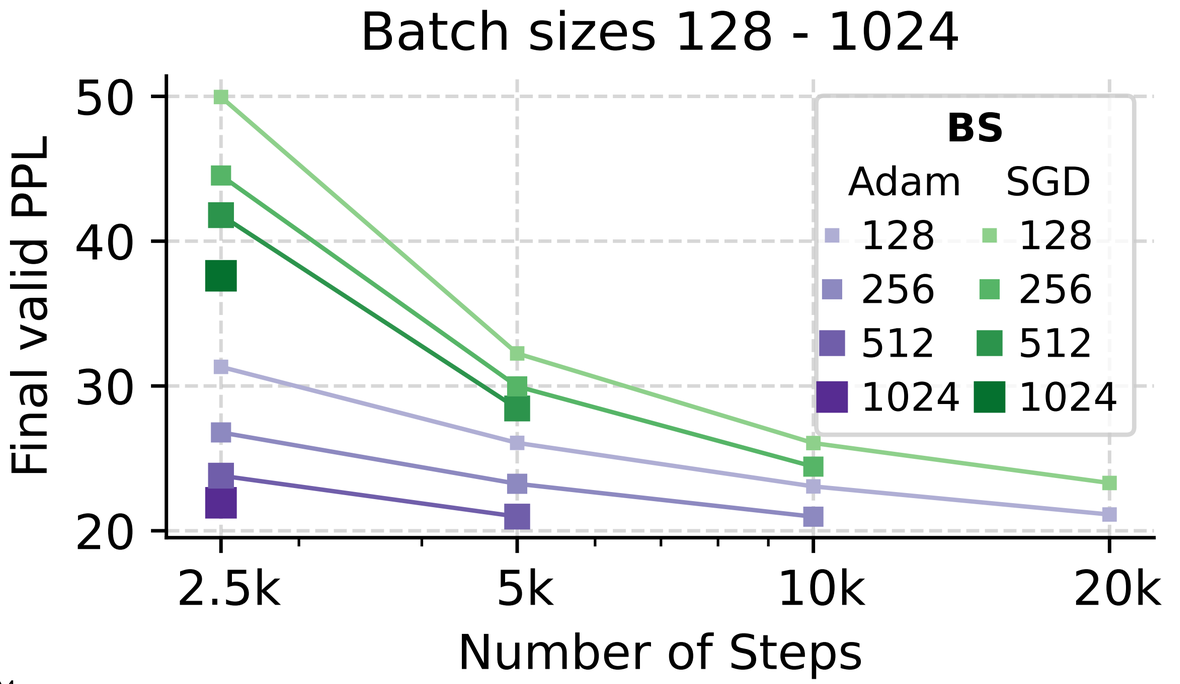
I claim the literature (including me) **missed a crucial observation.**

Adam vs. SGD training a 160M parameter transformer (1.2 B tokens budget)

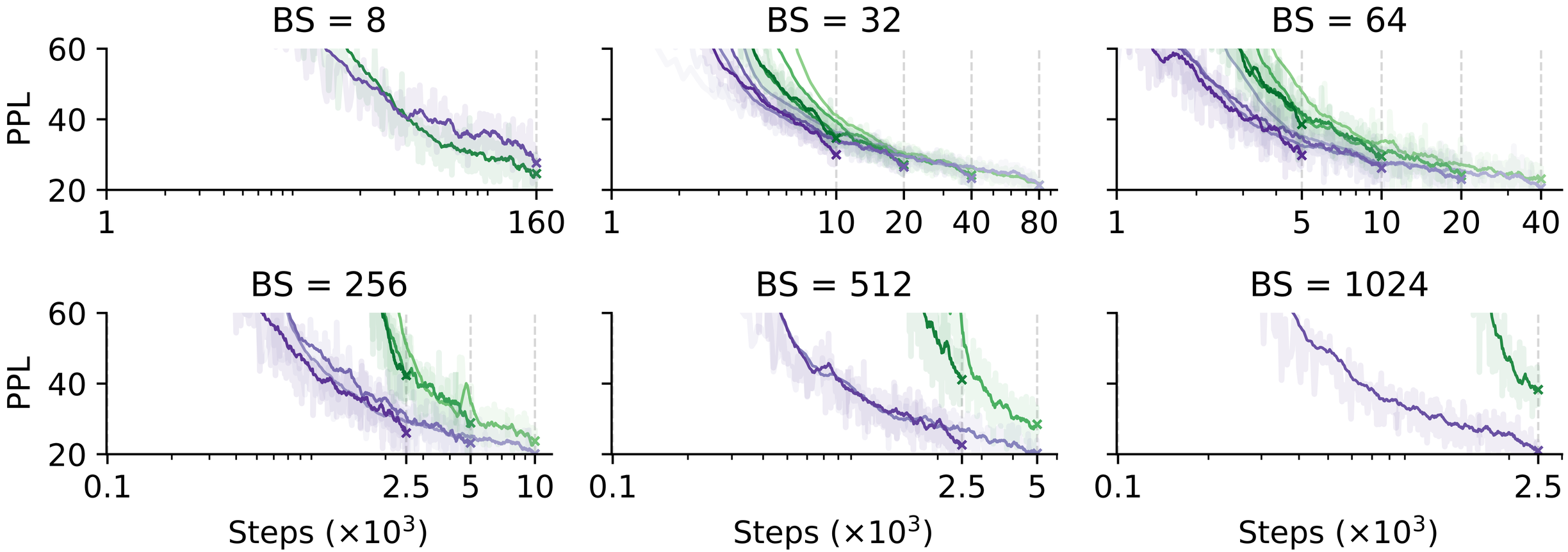



Token budget
Adam
SGD
0.66B
1.31B
2.62B
5.24B
Final valid PPL
BS
8
16
32
64
128
256
512
1024
20
25
30
35
40
45
50

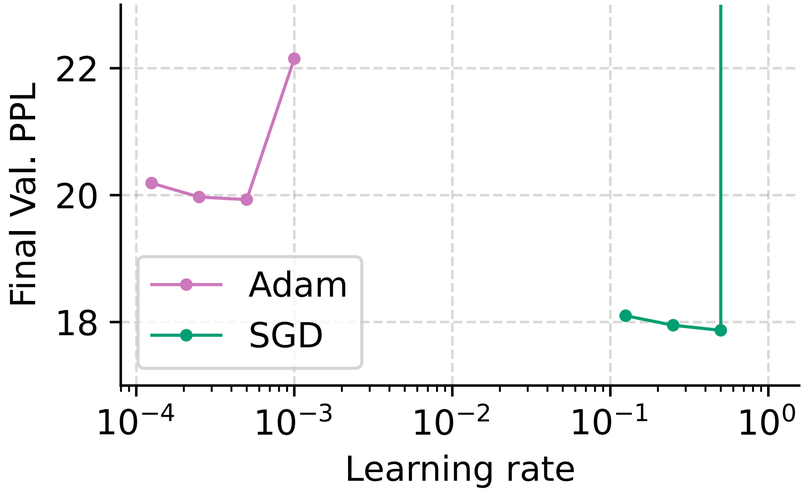

Batch sizes 8 - 64
BS
Adam
SGD
8
16
32
64
Final valid PPL
5k
10k
20k
40k
80k
160k
20
30
40
50

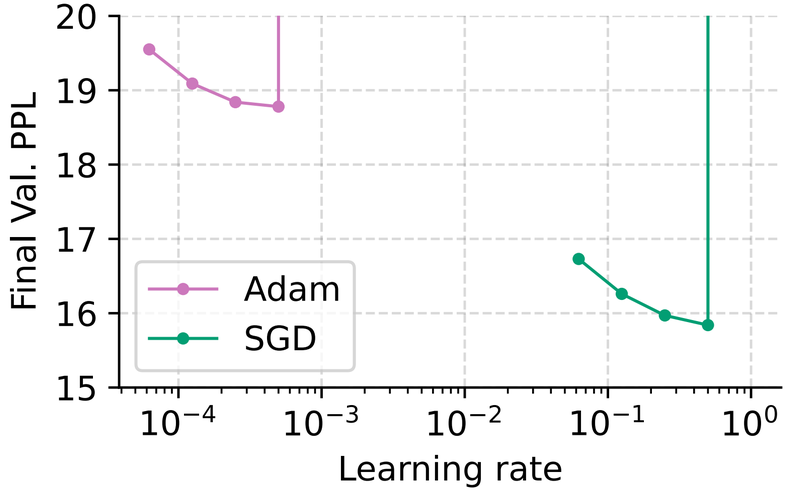

Batch sizes 128 - 1024
BS
Adam
SGD
128
256
512
1024
Final valid PPL
2.5k
5k
10k
20k
Number of Steps
20
30
40
50

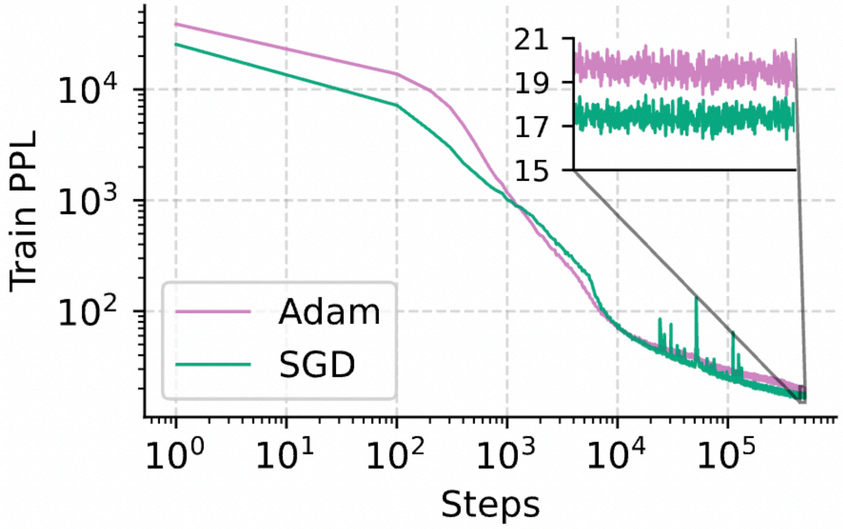
Adam vs. **SGD** training a 160M parameter transformer (different token budgets)

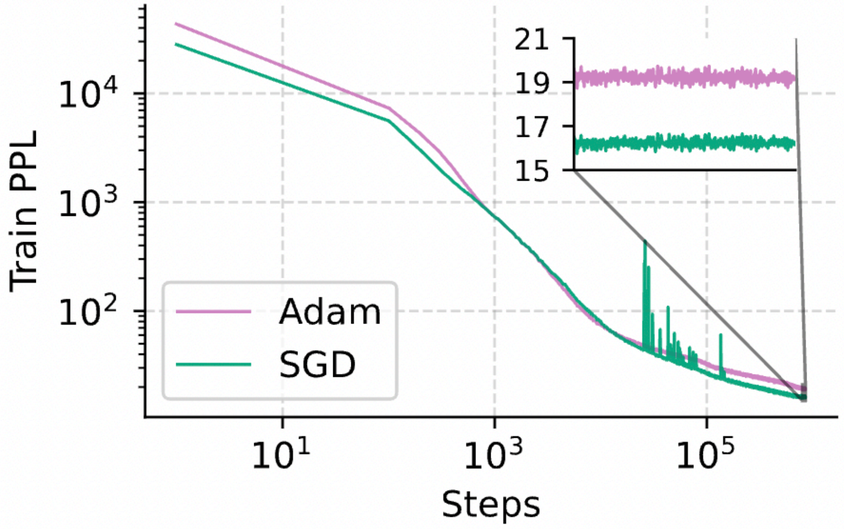


Adam < SGD at very low batch sizes, even at larger scales (tuned) !









(a) 410M model on SlimPajama (seq. length 2048, batch size 8, 500k steps) – 1.5 days of training.

(b) 1B model on FineWeb (seq. length 1024, batch size 16, 850k steps) – 5 days of training.

Another paper independently confirmed this!

Small Batch Size Training for Language Models: When Vanilla SGD Works, and Why Gradient Accumulation Is Wasteful

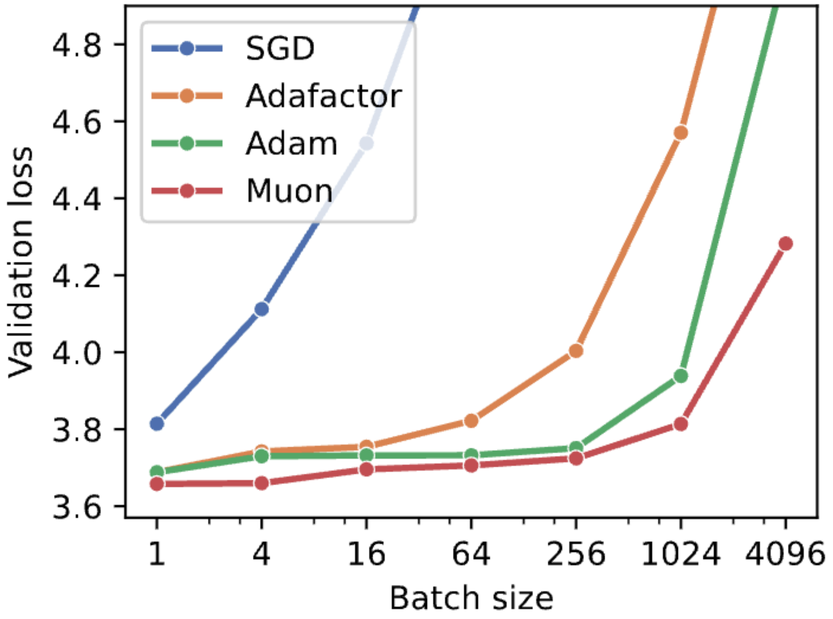
Martin Marek
New York University
martin.m@nyu.edu

Sanae Lotfi
New York University

Aditya Somasundaram
Columbia University

Andrew Gordon Wilson
New York University

Micah Goldblum
Columbia University



(a) Large batches require more sophisticated optimizers

So, how can we understand this?

It is actually super simple!!!

Assumption: gradient noise is i.i.d. with constant 1-sample covariance Σ

For **SGD**, the following is a weak-first-order-approx. :

drift diffusion

$$dX_t = -\nabla f(X_t)dt + \sqrt{\frac{\eta\Sigma}{B}}dW_t$$

For **SignSGD**, the following (Compagnoni et al. 24) is a weak-first-order-approx. :

drift diffusion

$$dX_t = -\operatorname{erf}\left(\sqrt{\frac{B}{2}}\Sigma^{-\frac{1}{2}}\nabla f(X_t)\right)dt + \sqrt{\eta}\left[I_d - \operatorname{diag}\left(\operatorname{erf}\left(\frac{\sqrt{B}\Sigma^{-\frac{1}{2}}\nabla f(X_t)}{\sqrt{2}}\right)\right)^2\right]^{1/2}dW_t$$

* i.e., algorithms follow flow for small η

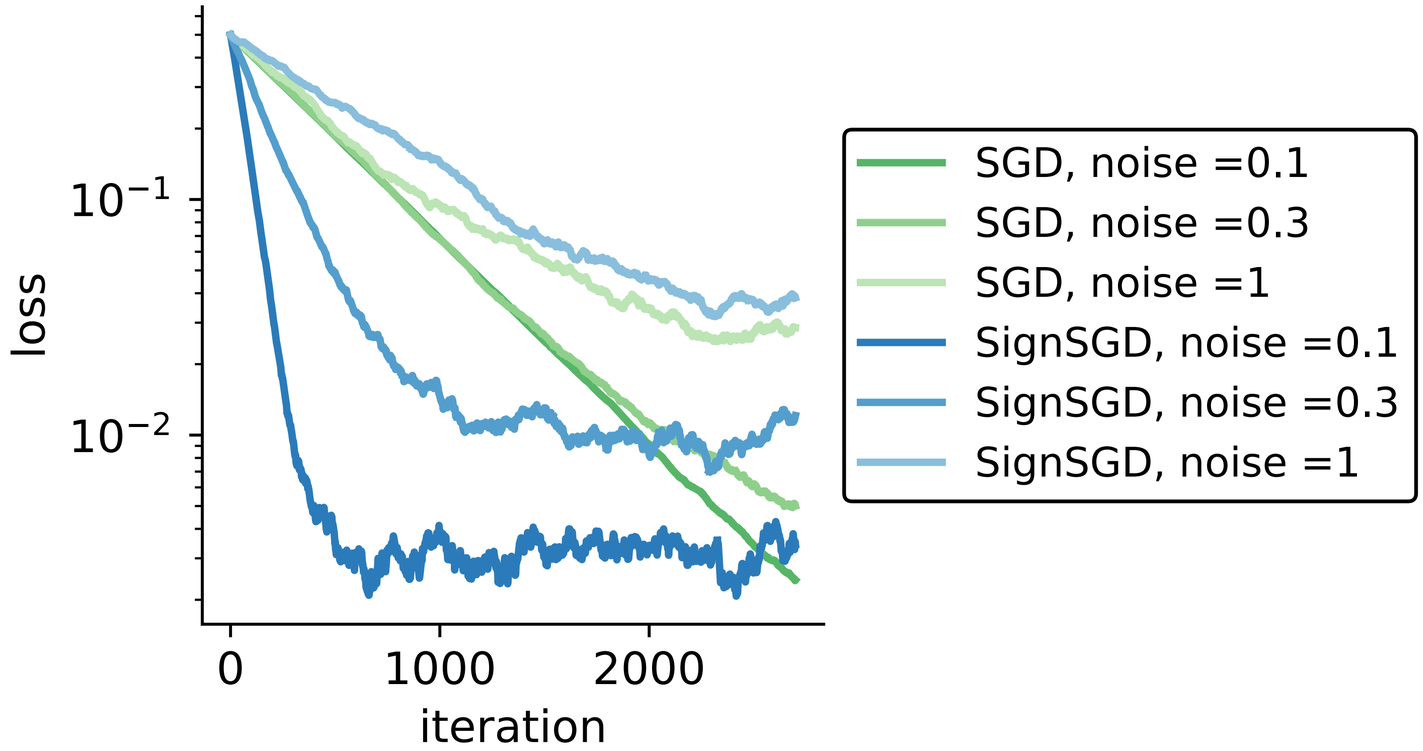
SGD drift

$$\nabla f(X_t)$$

SignSGD drift

$$\text{erf}\left(\sqrt{\frac{B}{2}}\Sigma^{-\frac{1}{2}}\nabla f(X_t)\right)$$

Example: Optimization of $f(x) = \|x\|^2/2$



Drift: models **early-stage** dynamics, before signal-to noise ratio becomes small.

- For SGD: this is B independent, For SignSGD: bigger with B (until saturation)
- Early progress in SGD is dominated by # steps: critical batch size is 1
- Early progress in SignSGD improves with batch, until saturation of erf (critical batch)

Proof (sketch, Compagnoni et at. 25). $m(x)$ is an estimate of the gradient, we assume is has Gaussian distribution centered around the full-batch gradient $\nabla f(x)$:

$$\text{sign}(m(x)), \qquad m(x) \sim \mathcal{N}(\nabla f(x), \sigma^2/B)$$

$$\mathbb{E}[\text{sign}(m(x))]$$

$$= \mathbb{P}[\text{sign}(m(x)) = \text{sign}(\nabla f(x))] \cdot \text{sign}(\nabla f(x)) - \mathbb{P}[\text{sign}(m(x)) \neq \text{sign}(\nabla f(x))] \cdot \text{sign}(\nabla f(x))$$

$$= (2\mathbb{P}[\text{sign}(m(x)) = \text{sign}(\nabla f(x))] - 1) \cdot \text{sign}(\nabla f(x))$$

Recall basics : if $Z \sim \mathcal{N}(\mu, \varsigma^2)$, then if $\ell > \mu$, we have $\mathbb{P}[Z \leq \ell] = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{\ell - \mu}{\sqrt{2\varsigma^2}}\right)$

$\Longrightarrow$ For $\mu > 0$, $\mathbb{P}[Z \geq 0] = \mathbb{P}[\text{sign}(Z) = \text{sign}(\mu)] = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{\mu}{\sqrt{2\varsigma^2}}\right)$

$\Longrightarrow$ If $\nabla f(x) > 0$, $\mathbb{P}[\text{sign}(m(x)) = \text{sign}(\nabla f(x))] = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{\nabla f(x)\sqrt{B}}{\sqrt{2\sigma^2}}\right)$ – coord.wise

$$\implies \text{If } \nabla f(x) > 0,\ \mathbb{P}[\text{sign}(m(x)) = \text{sign}(\nabla f(x))] = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{\nabla f(x)\sqrt{B}}{\sqrt{2\sigma^2}}\right) - \text{coord.wise}$$

Same holds for the negative case.

$$\mathbb{E}[\text{sign}(m(x))]$$

$$= (2\mathbb{P}[\text{sign}(m(x)) = \text{sign}(\nabla f(x))] - 1) \cdot \text{sign}(\nabla f(x))$$

$$= \text{erf}\left(\sqrt{\frac{B}{2}}(\sigma^2)^{-\frac{1}{2}}\nabla f(x)\right)$$

$$\Downarrow$$

$$dX_t = -\text{erf}\left(\sqrt{\frac{B}{2}}\Sigma^{-\frac{1}{2}}\nabla f(X_t)\right)dt + \sqrt{\eta}\left[I_d - \text{diag}\left(\text{erf}\left(\frac{\sqrt{B}\Sigma^{-\frac{1}{2}}\nabla f(X_t)}{\sqrt{2}}\right)\right)^2\right]^{1/2} dW_t$$

**Gaussianity is not strictly needed* for our insights on batch-size acceleration to hold. As discussed by Compagnoni et al. (25) and clear from the argument above on the cumulative distribution, a similar expression can hold even for distributions with heavier tails.

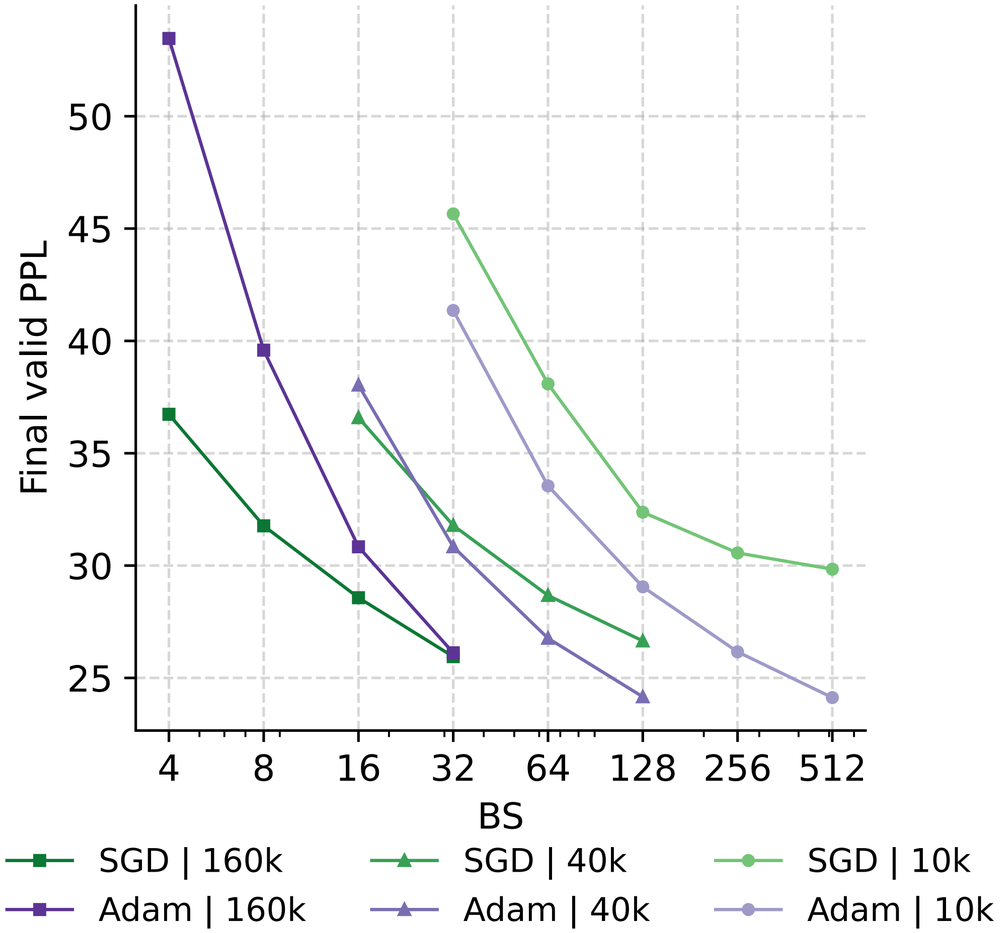
Thank you!!! 🇵🇱

We have not succeeded in answering all our problems.
The answers we have found only serve to raise a whole set
of new questions. In some ways we feel we are as confused
as ever, but we believe we are confused on a higher level
and about more important things.

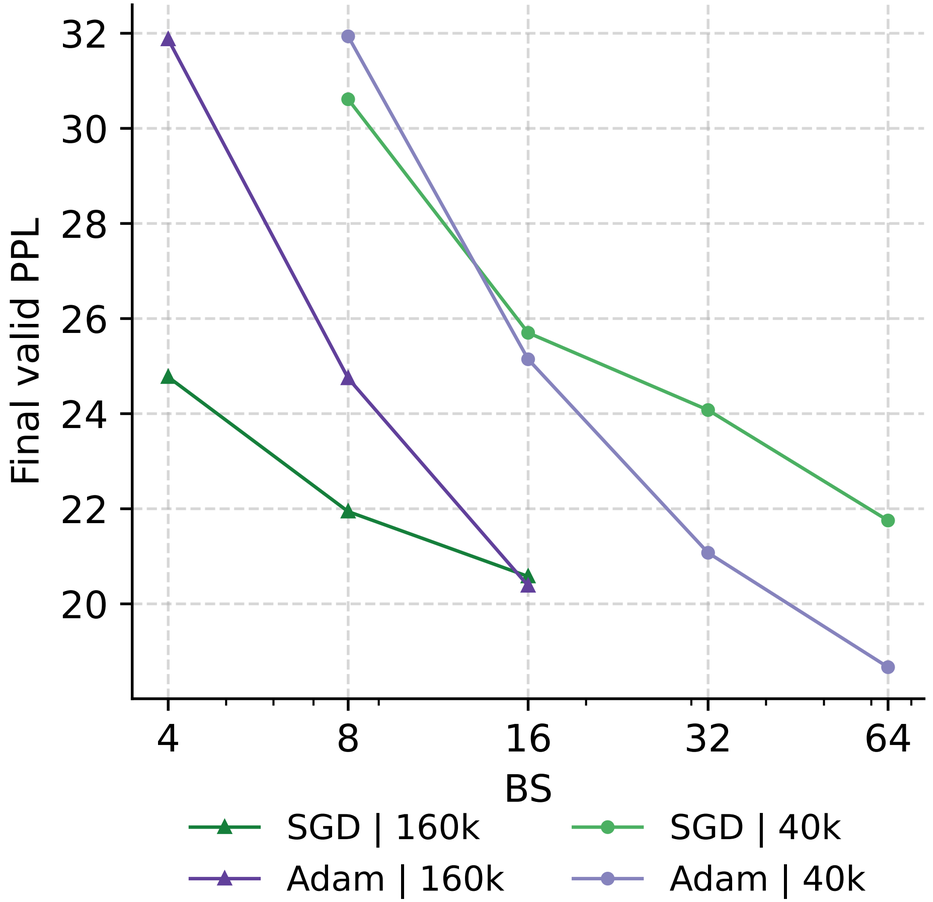
Posted outside the mathematics reading room –Tromsø University

(Quote stolen from Bernt Øksendal masterpiece book)

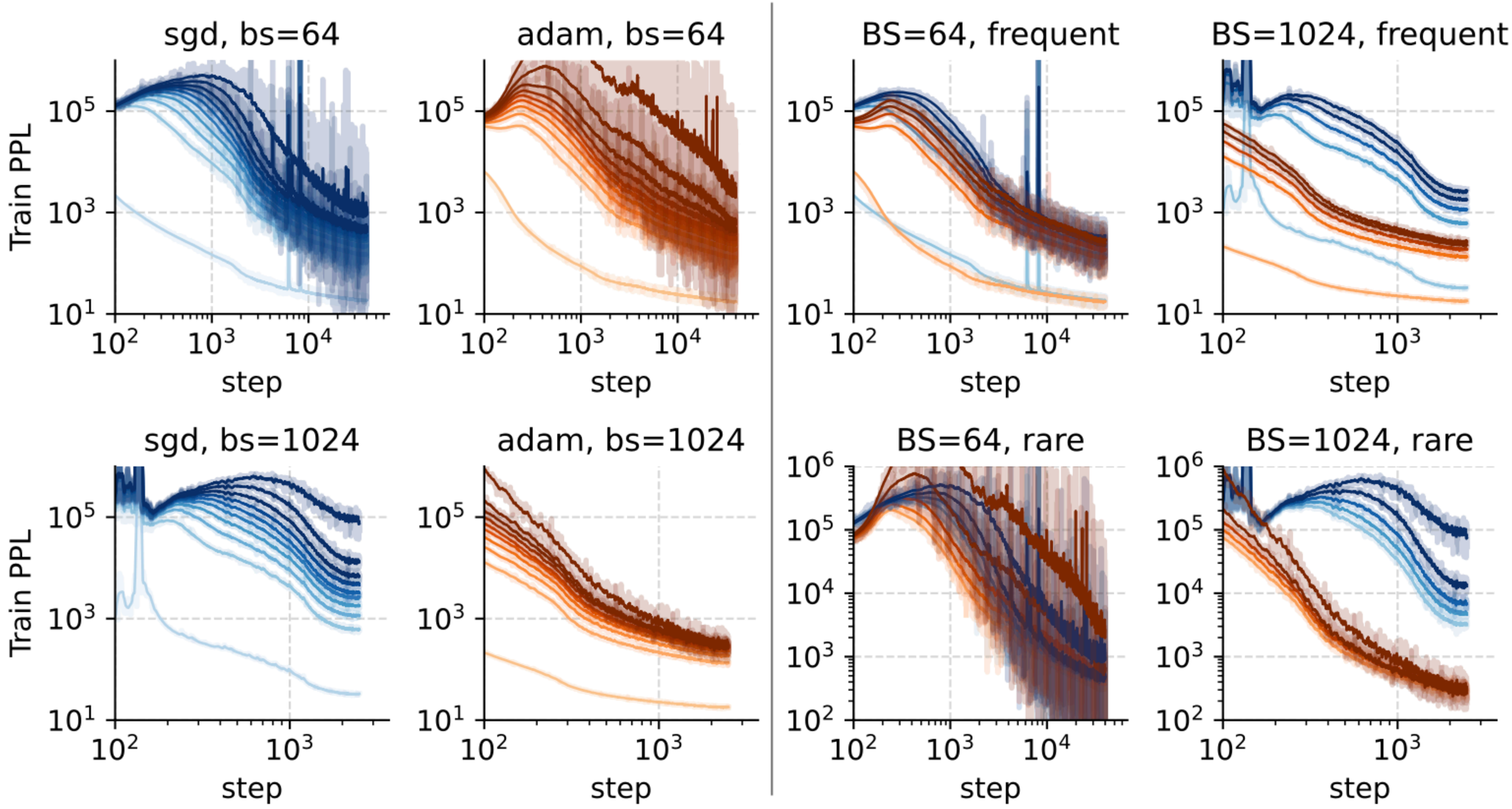
Fineweb dataset
sequence length 2048
12 layers Transformer.



SlimPajama dataset
sequence length 2048
24 layers Transformer



Can the gap be explained by class imbalance?
Looks like **rare tokens are hard to learn for SGD, but only at big batch..**



AdamW

For boosting generalization, a standard technique is doing L2 regularization, helping driving not-needed parameters to be close to zero. A soft constraint.

$$\min_w L(w) \quad \longrightarrow \min_w \tilde{L}(w) := L(w) + \frac{\lambda}{2}\|w\|^2$$

People were initially feeding $\nabla \tilde{L}(w)$ into the Adam moving averages:

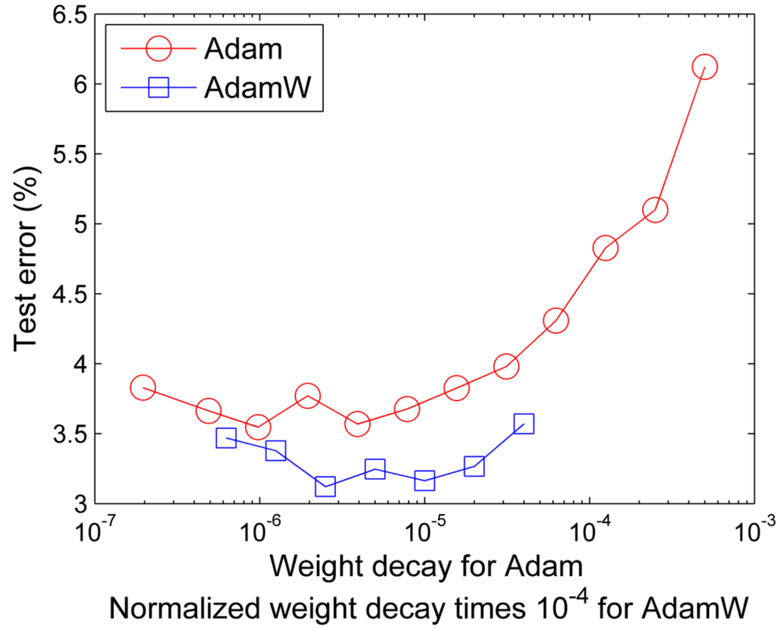
Algorithm 2 Adam with L_2 regularization

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) \right.$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Algorithm 2 Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Basically, Adam on the loss, SGD on the regularizer!



Today, basically every LLM is trained with AdamW.

Note however that in LLMs this phenomenon is not really about test loss, SGD (with or without regularization cannot optimize —even the train loss!)