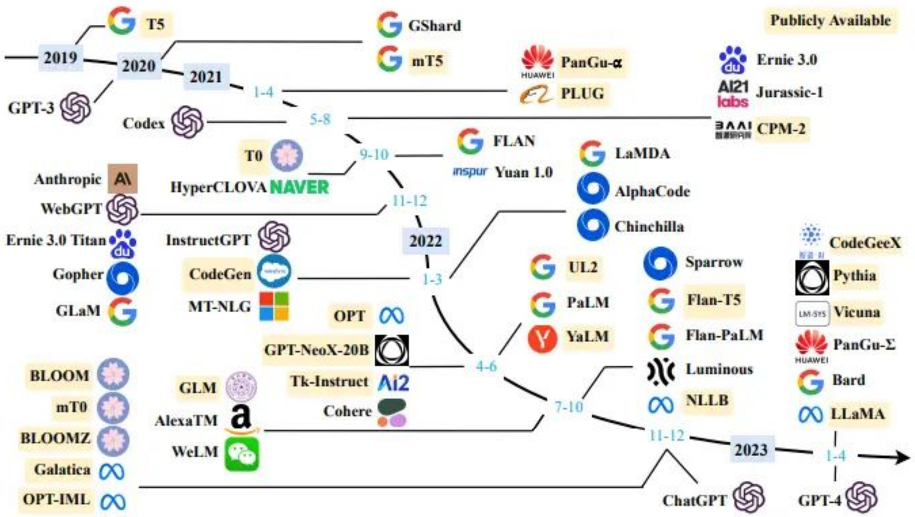
Evaluating Optimizers for Language Model Training: From SGD to Adam (and Beyond?)

Rosie Zhao

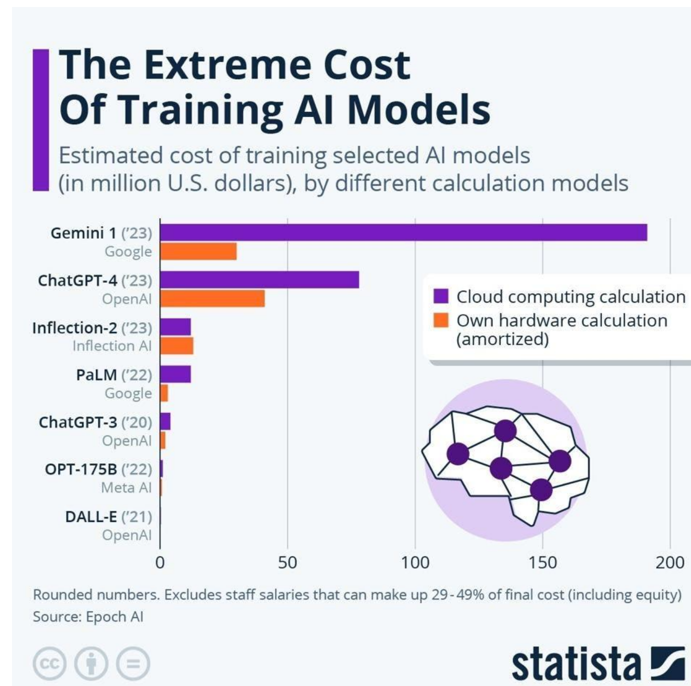
UIUC ML Seminar - September 27, 2024

Harvard John A. Paulson
School of Engineering
and Applied Sciences

Models are getting bigger and pretraining is expensive!



Llama 3.1 405B FLOPs = 10^{21} x AlexNet FLOPs

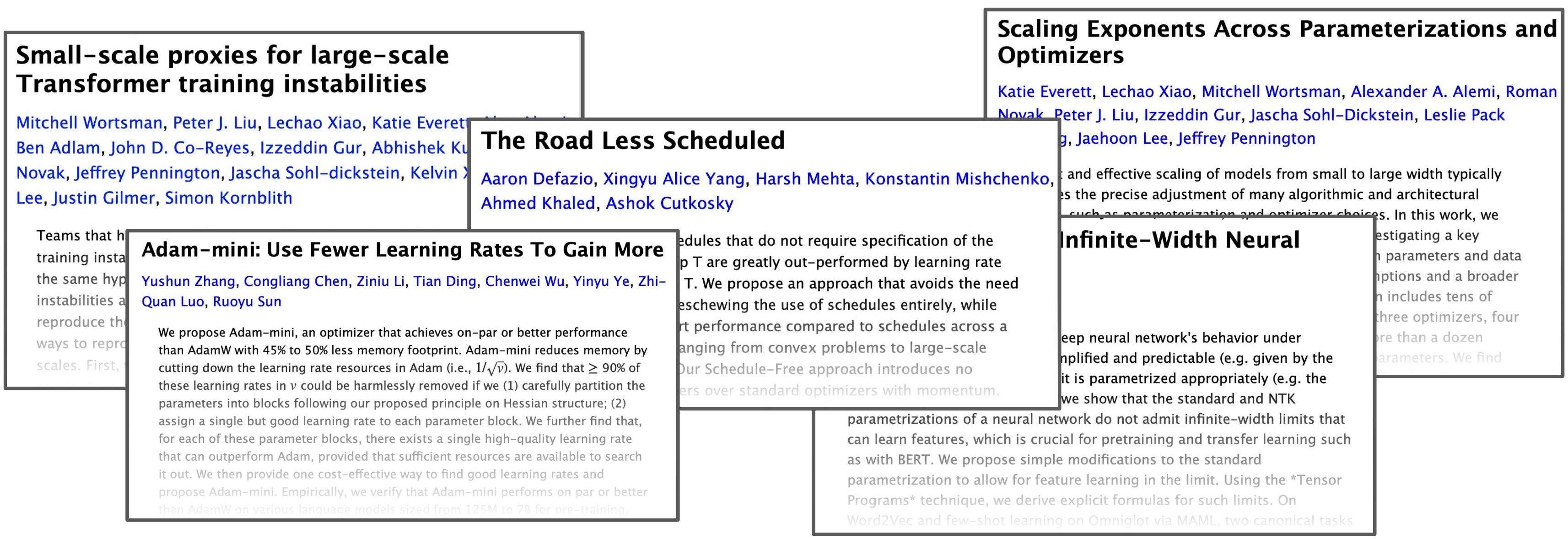


Harvard John A. Paulson School of Engineering and Applied Sciences

Image credit: [1], [2], [3]

What research matters?

- Even industry labs can really only commit to **one training run**
  - **Hyperparameter transfer** across scales
  - **Mitigating training instabilities** at scale
- There's a growing need for **more efficient algorithms**
  - **Distributed training**
  - **More efficient optimizers**



Small-scale proxies for large-scale Transformer training instabilities

Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett
Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Ku
Novak, Jeffrey Pennington, Jascha Sohl-dickstein, Kelvin X
Lee, Justin Gilmer, Simon Kornblith

Teams that h
training insta
the same hyp
instabilities a
reproduce th
ways to repro
scales. First,

The Road Less Scheduled

Aaron Defazio, Xingyu Alice Yang, Harsh Mehta, Konstantin Mishchenko,
Ahmed Khaled, Ashok Cutkosky

edules that do not require specification of the
p T are greatly out-performed by learning rate
T. We propose an approach that avoids the need
eschewing the use of schedules entirely, while
t performance compared to schedules across a
anging from convex problems to large-scale
Our Schedule-Free approach introduces no
ers over standard optimizers with momentum.

Adam-mini: Use Fewer Learning Rates To Gain More

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, Ruoyu Sun

We propose Adam-mini, an optimizer that achieves on-par or better performance than AdamW with 45% to 50% less memory footprint. Adam-mini reduces memory by cutting down the learning rate resources in Adam (i.e., $1/\sqrt{v}$). We find that $\geq$ 90% of these learning rates in v could be harmlessly removed if we (1) carefully partition the parameters into blocks following our proposed principle on Hessian structure; (2) assign a single but good learning rate to each parameter block. We further find that, for each of these parameter blocks, there exists a single high-quality learning rate that can outperform Adam, provided that sufficient resources are available to search it out. We then provide one cost-effective way to find good learning rates and propose Adam-mini. Empirically, we verify that Adam-mini performs on par or better than AdamW on various language models sized from 125M to 7B for pre-training

Scaling Exponents Across Parameterizations and Optimizers

Katie Everett, Lechao Xiao, Mitchell Wortsman, Alexander A. Alemi, Roman Novak, Peter J. Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack
g, Jaehoon Lee, Jeffrey Pennington

and effective scaling of models from small to large width typically
es the precise adjustment of many algorithmic and architectural
such as parameterization and optimizer choices. In this work, we
estigating a key
parameters and data
nptions and a broader
n includes tens of
three optimizers, four
re than a dozen
arameters. We find

Infinite-Width Neural

eep neural network's behavior under
nplified and predictable (e.g. given by the
it is parametrized appropriately (e.g. the
we show that the standard and NTK
parametrizations of a neural network do not admit infinite-width limits that can learn features, which is crucial for pretraining and transfer learning such as with BERT. We propose simple modifications to the standard parametrization to allow for feature learning in the limit. Using the *Tensor Programs* technique, we derive explicit formulas for such limits. On Word2Vec and few-shot learning on Omniglot via MAML, two canonical tasks

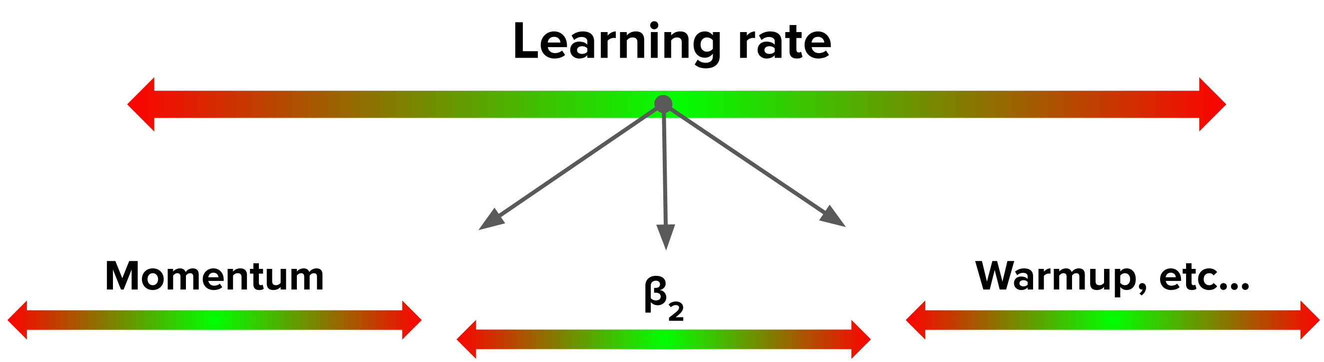
Deconstructing What Makes a Good Optimizer for Language Models

Zhao*, Morwani*, Brandfonbrener*, Vyas*, Kakade.

Harvard John A. Paulson School of Engineering and Applied Sciences

Which optimizers are best?

- We perform a comprehensive sweep for training **autoregressive language models** across **different optimizers, hyperparameters, architectures, and scale**
- Both **optimal performance** and **learning rate stability** are important
- We focus on optimizers with **diagonal preconditioning:** Adam, Adafactor*, Lion, SignSGD with momentum
- We perform **one-dimensional sweeps,** which doesn't account for 2D interactions



Harvard John A. Paulson School of Engineering and Applied Sciences

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

Optimizers Review

- At time t, given weight matrix $W_t \in \mathbb{R}^{m \times n}$, gradient $G_t \in \mathbb{R}^{m \times n}$ with vectorized form $g_t = \mathrm{vec}(G_t) \in \mathbb{R}^{mn}$

Adagrad: second order method maintaining preconditioner H

$$H_t = H_{t-1} + g_t g_t^\top; \quad w_t = w_{t-1} - \eta H_t^{-1/2} g_t$$

Adam: maintains EMA of gradients and elementwise gradients squared

$$W_t \leftarrow W_{t-1} - \eta \frac{M_t}{\sqrt{V_t}}$$

Adafactor*: maintains rank-1 approximation of elementwise gradients squared

$$W_t \leftarrow W_{t-1} - \eta \frac{M_t}{\sqrt{V_t'}}$$

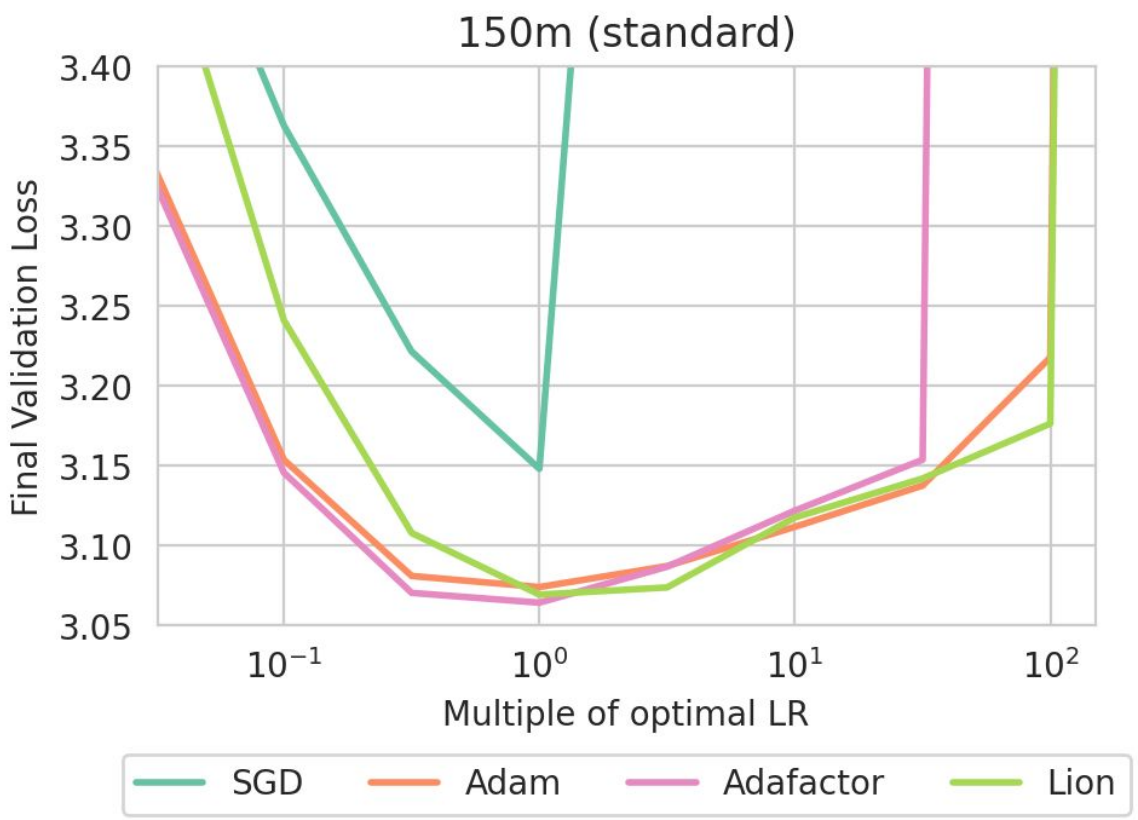
Others of interest: signSGD with momentum (**Signum**), **Lion**, etc…

Other Training Details

- We study two architectures:
  - **With QK-LayerNorm and z-loss** (“standard”) and without
- We train decoder-only language models on **C4 tokenized with the T5 tokenizer**, at multiple scales (**150m, 300m, 600m, 1.2b**)
- Other standard training choices: batch size of 256, sequence length of 512, training with “chinchilla optimal" number of tokens (~20x)...

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

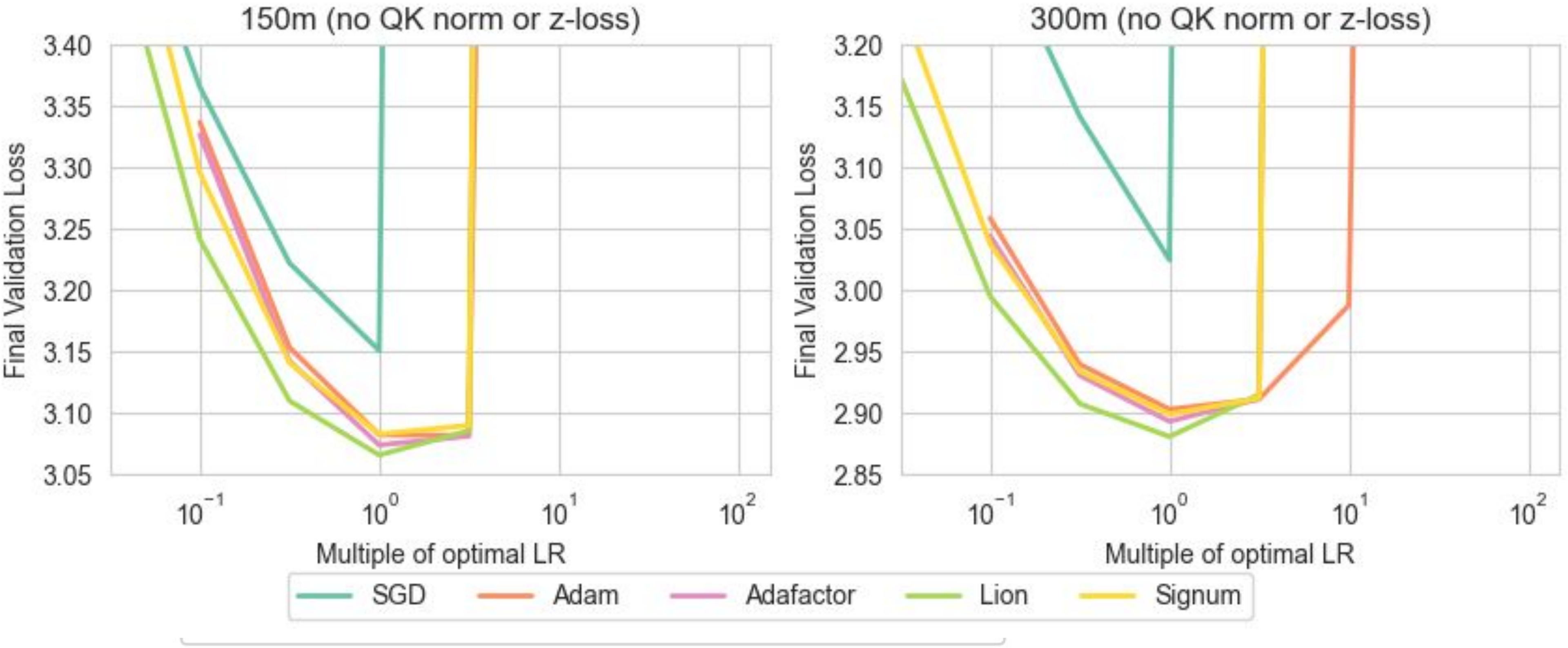
Initial Sweep Results



***SGD is shown here with 0.98 momentum**

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

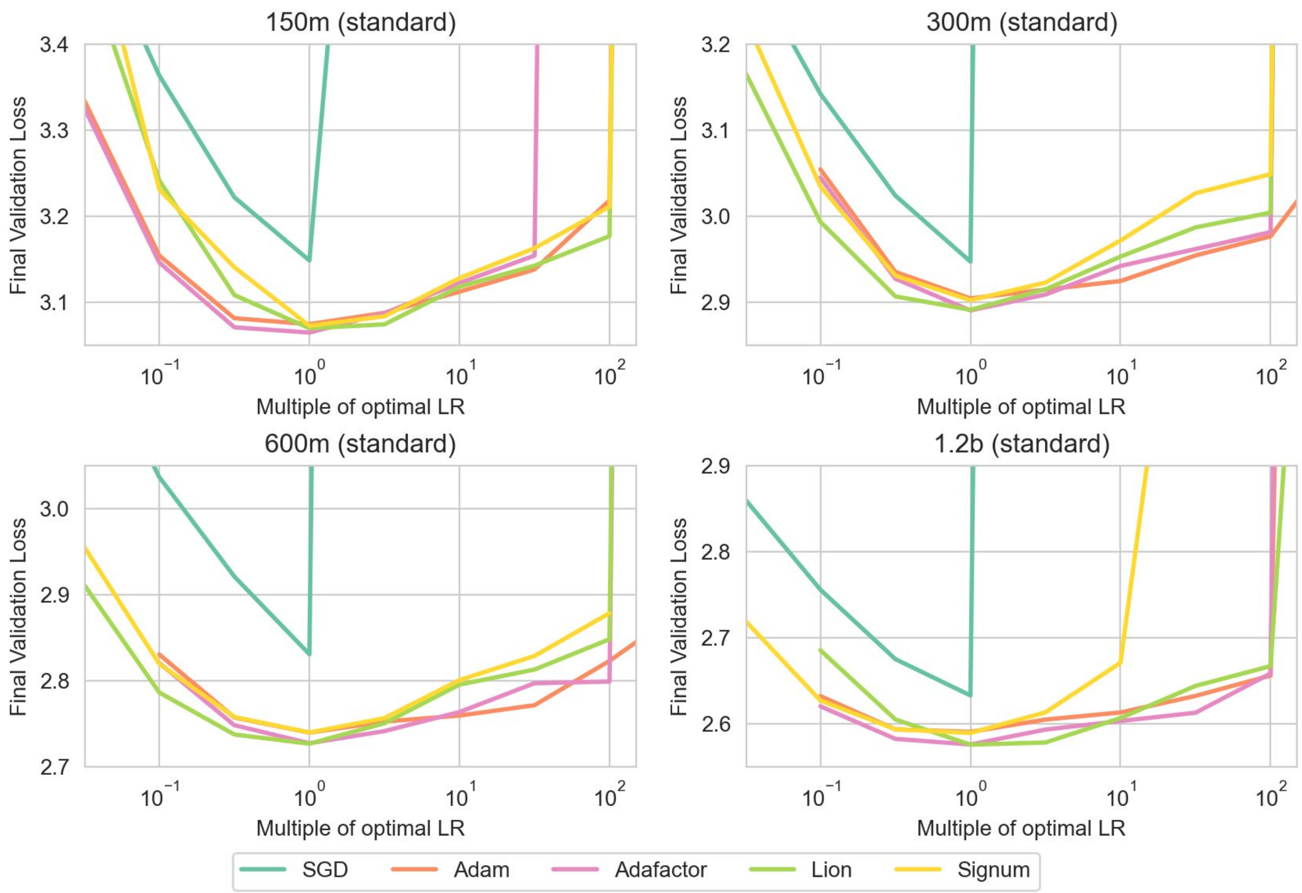
Initial Sweep Results



Harvard John A. Paulson School of Engineering and Applied Sciences

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

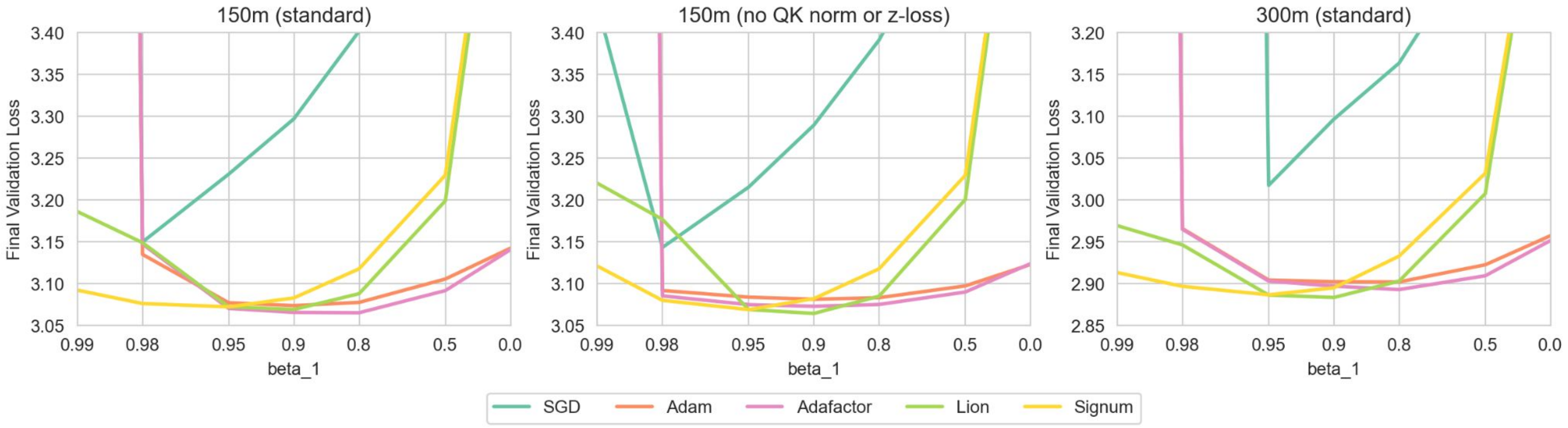
This holds across multiple scales...



Takeaway: besides SGD, performance and stability to learning rate are comparable!

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

Other Hyperparameter Sweeps - Momentum

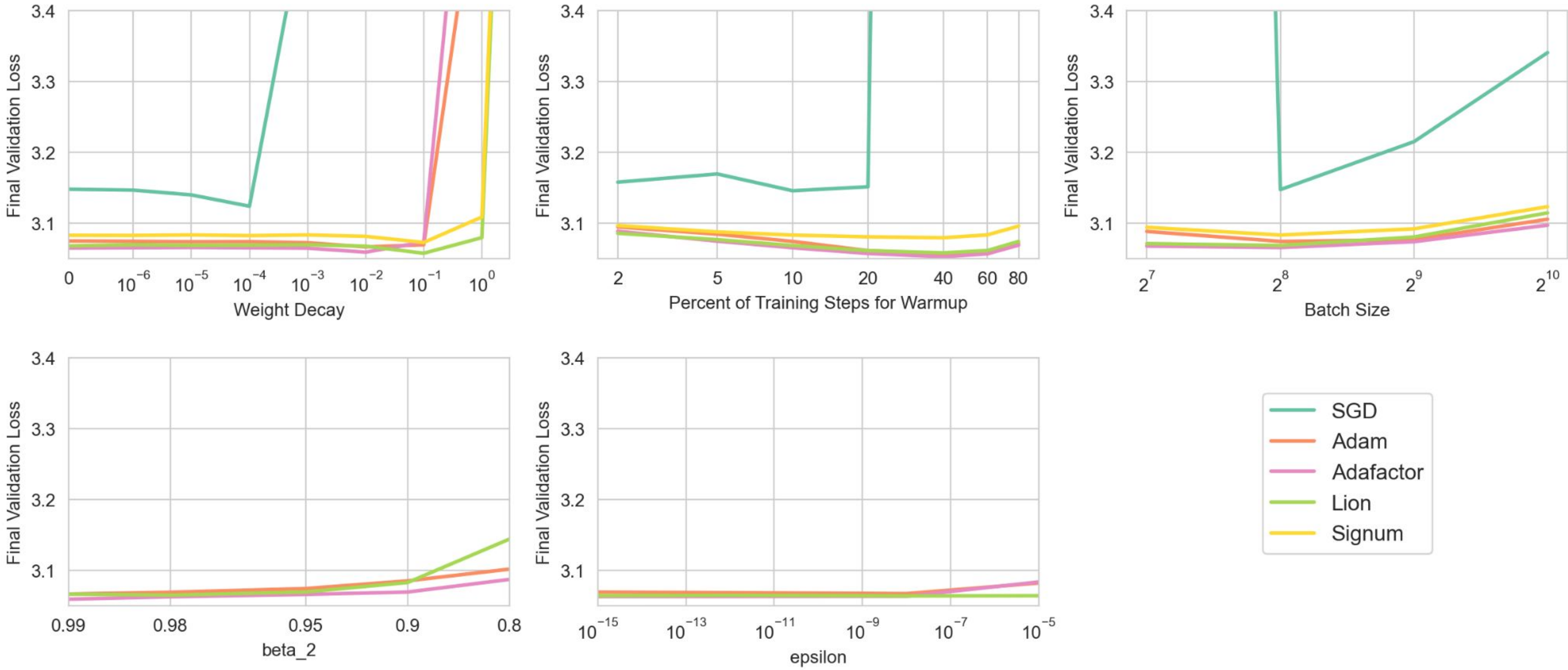


- Most sensitive hyperparameter other than learning rate
- SGD very sensitive, Adam and Adafactor are surprisingly robust, and Lion/Signum get worse at low momentum values

Takeaway: besides SGD, performance and stability to learning rate are comparable at standard momentum values.

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

Other Hyperparameter Sweeps

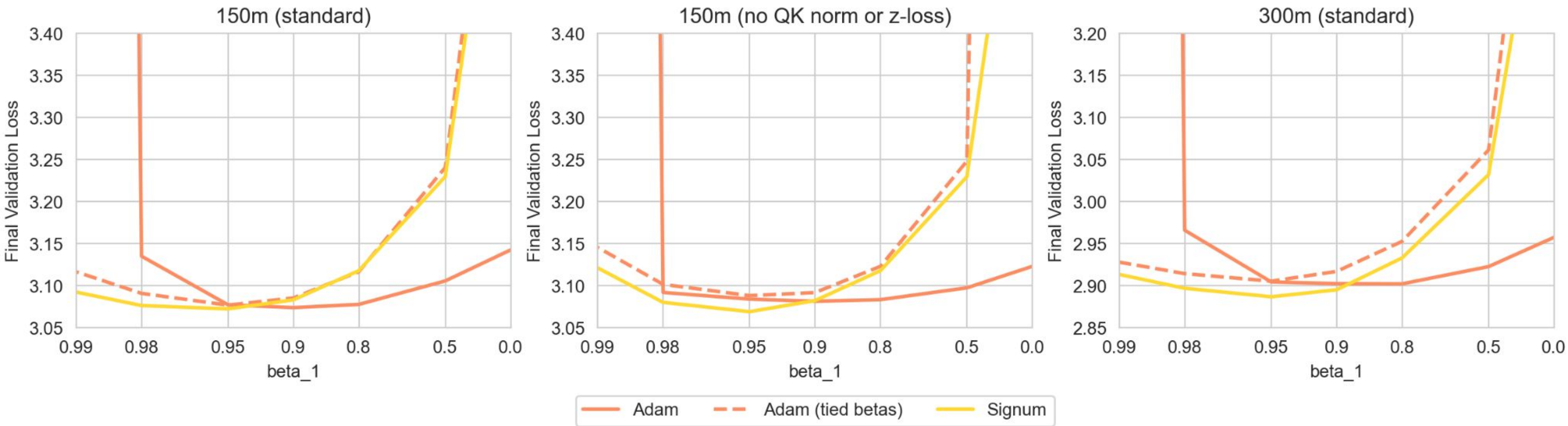


Takeaway: besides SGD, very little performance gain with respect to other parameters. **Prioritize tuning learning rate and momentum.**

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

signSGD

- Adam performs similarly to Signum, even at scale!
- Result from Balles and Hennig (2018) shows that Adam performs variance-adjusted signSGD - if $\beta_1 = \beta_2$, they should match more



Takeaway: Adam behaves similarly to Signum for $\beta_1 = \beta_2$, with standard settings being similar to this ($\beta_1 = 0.9$, $\beta_2 = 0.95$)

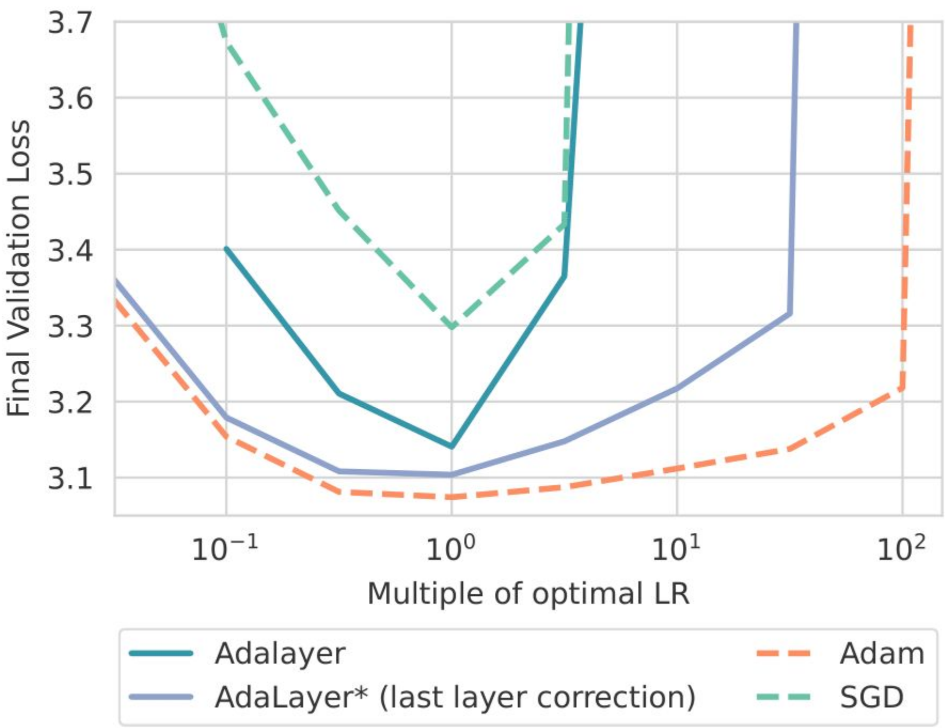
Digging Deeper - Use anything but SGD?

- All diagonal preconditioning optimizers are similar! But why?
- We want to understand the **role of preconditioning** for performance and stability
- **To what extent is this adaptivity needed for different parameters of the network? Can SGD achieve similar benefits with minimal modifications?**

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

Adalayer

- "Layer-wise" version of Adam for ease of study
- Stores a **single scalar** which is the average of the second moment matrix for a given "block" (eg. a layer)

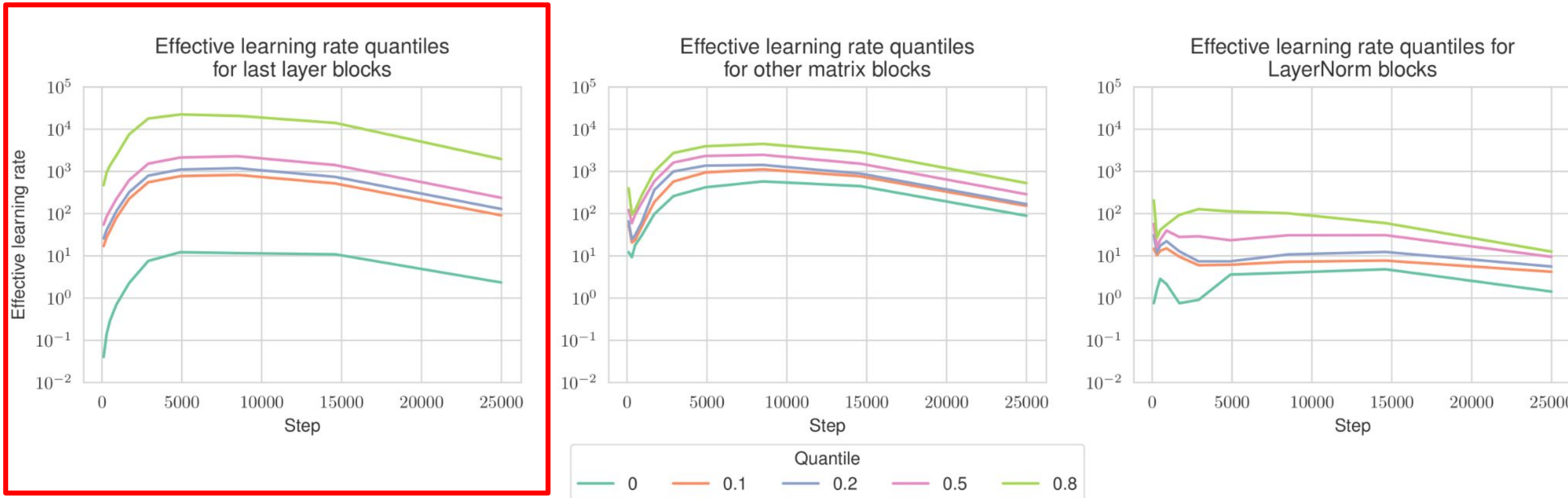


Need to perform a **correction to last layer**: each set of weights feeding into a logit is **its own block**

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

Adalayer Effective Learning Rate Quantiles

- Given layer l, we report effective learning rates $\frac{\eta_t}{\sqrt{v_t^l} + \epsilon}$ over training

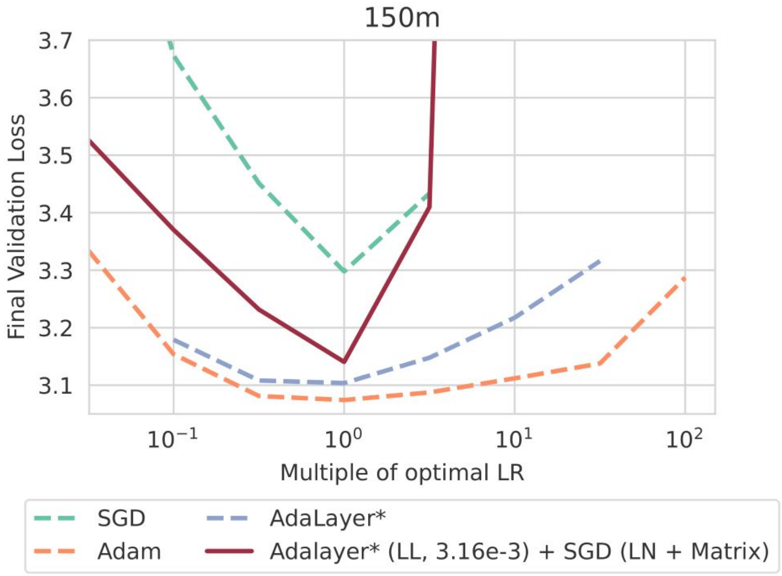


- Learning rates across **logits** vary across **multiple orders of magnitude**

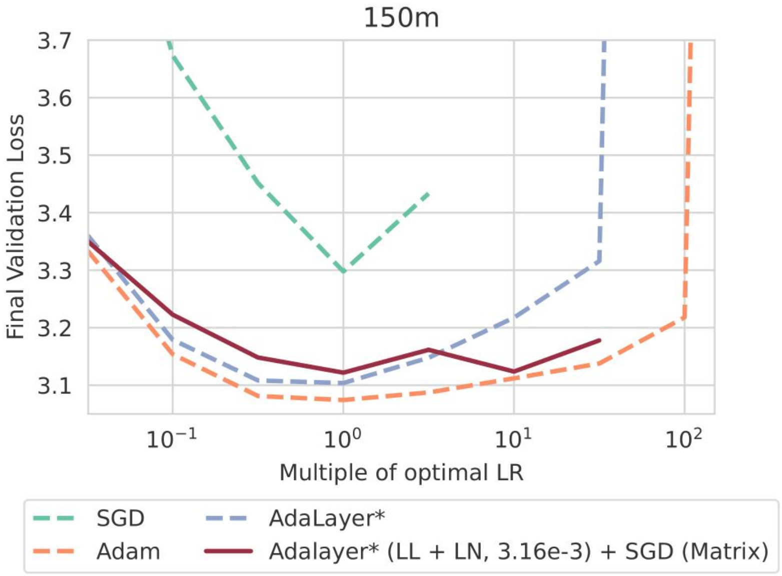
Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

SGD + Adalayer

- Quantiles suggest that all layers but the last layer needs an iteration-dependent scalar correction to their learning rate - **can they actually be trained with SGD?**



Adalayer on just the last layer is not sufficient…

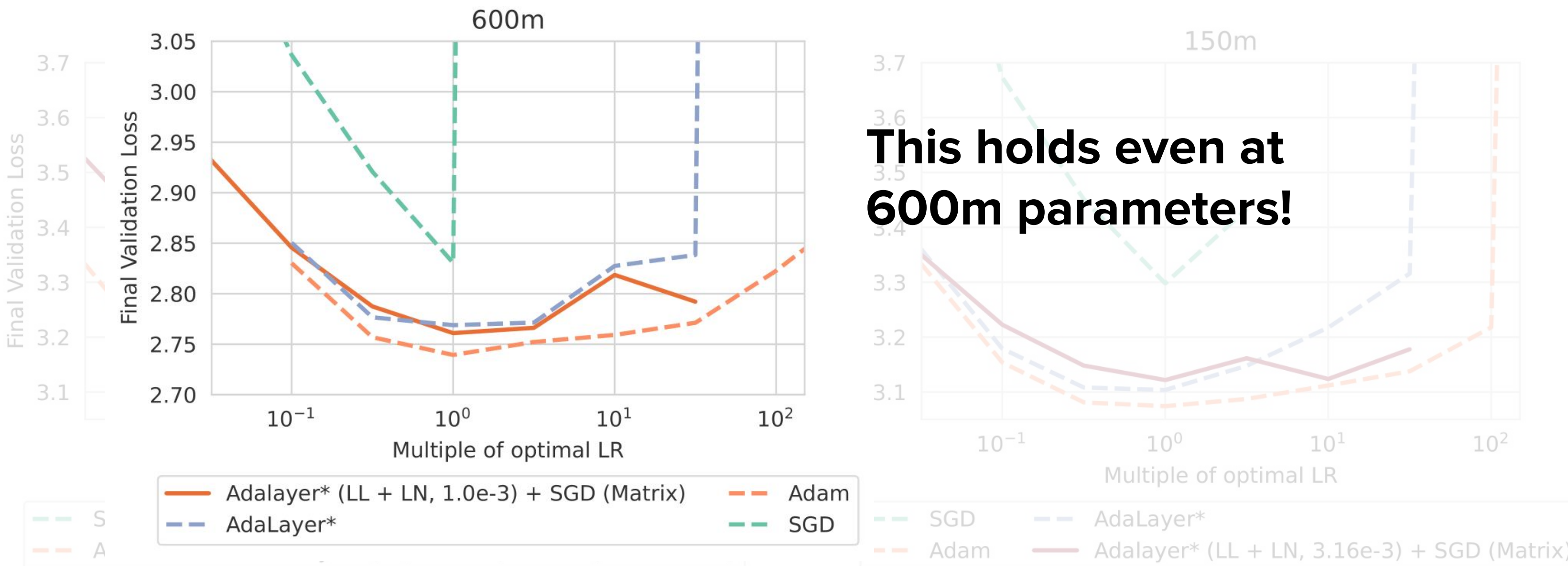


… but Adalayer on just the last layer and LayerNorm parameters is!

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

SGD + Adalayer

- Quantiles suggest that all layers but the last layer needs an iteration-dependent scalar correction to their learning rate - **can they actually be trained with SGD?**



This holds even at 600m parameters!

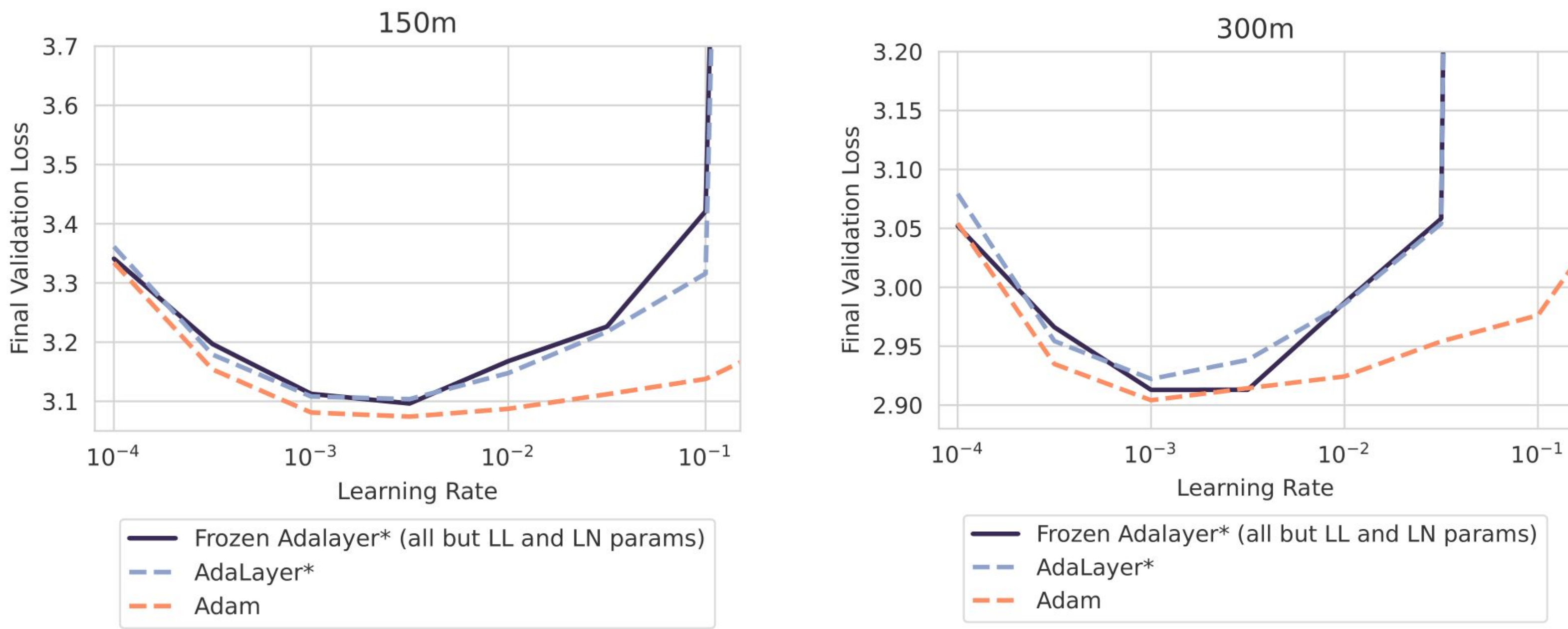
Adalayer on just the last layer is not sufficient…

… but Adalayer on just the last layer and LayerNorm parameters is!

Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

Frozen Adalayer

- We also **fix Adalayer learning rate ratios from initialization**, with the exception of last layer and LayerNorm parameters



Deconstructing what makes a good optimizer for language models. **Zhao*,** Morwani*, Brandfonbrener*, Vyas*, Kakade. Submitted.

Summary and Takeaways

- **Optimizers with diagonal preconditioners are roughly equivalent** both in terms of optimal performance and hyperparameter stability
- It seems that most of the benefits of adaptive optimizers arise from their treatment of the **last layer and LayerNorm parameters**
  - Why? Further investigations into LayerNorm?
- For practitioners: **tune learning rate and momentum**, other hyperparameters are stable around these optimal values
- Optimizer choice might not be the optimal point of intervention for increasing efficiency? At least for diagonal preconditioning optimizers…

SOAP: Improving and Stabilizing Shampoo with Adam

Vyas, Morwani, **Zhao,** Shapira, Brandfonbrener, Janson, Kakade.

What’s next after diagonal preconditioners?

- As we just saw, most diagonal preconditioner optimizers perform similarly to AdamW – **need to explore non-diagonal preconditioning methods**
- **Second-order optimization methods:** Adagrad, Newton’s method require storing and inverting matrices of size |P| x |P| (P = # parameters)
- Hessian-free and Hessian estimation methods (eg. KFAC [Martens & Grosse, 2015], Shampoo [Gupta et al., 2018] and follow up enhancements)

Shampoo

- For a given weight matrix $W \in \mathbb{R}^{m \times n}$, maintain two preconditioners $L_t \in \mathbb{R}^{m \times m} \quad R_t \in \mathbb{R}^{n \times n}$
- Update rule with learning rate η as follows:

$$L_t \leftarrow L_{t-1} + G_t G_t^T; \quad R_t \leftarrow R_{t-1} + G_t^T G_t; \quad W_t \leftarrow W_{t-1} - \eta L_t^{-1/4} G_t R_t^{-1/4}$$

- Previous work (collaborators): **Shampoo^2** (i.e. exponent -½ instead of -¼) is better than Shampoo in practice, and is provably close to the **optimal Kronecker product approximation of the Adagrad preconditioner**.

Distributed Shampoo implementation won Algoperf benchmark!

An equivalence between Shampoo^2 and Adafactor

Algorithm 1 Single step of idealized Shampoo with power 1/2.

Sample batch B_t.
$G_t \in \mathbb{R}^{m\times n} \leftarrow -\nabla_W \phi_{B_t}(W_t)$
$L \leftarrow \mathbb{E}_B[G_B G_B^T]$ {Where the expectation is over a random batch B.}
$R \leftarrow \mathbb{E}_B[G_B^T G_B]$
$\hat{H} \leftarrow L \otimes R/\mathrm{Trace}(L)$
$W_t \leftarrow W_{t-1} - \eta \hat{H}^{-1/2} G_t = W_{t-1} - \eta L^{-1/2} G_t R^{-1/2}/\mathrm{Trace}(L)^{-1/2}$

"Idealized Shampoo": highlighted changes in red

Algorithm 2 Single step of idealized Adafactor in Shampoo's eigenspace.

Sample batch B_t.
$G_t \in \mathbb{R}^{m\times n} \leftarrow -\nabla_W \phi_{B_t}(W_t)$
$L \leftarrow \mathbb{E}_B[G_B G_B^T]$
$R \leftarrow \mathbb{E}_B[G_B^T G_B]$
$Q_L \leftarrow \texttt{Eigenvectors}(L)$
$Q_R \leftarrow \texttt{Eigenvectors}(R)$
$G'_t \leftarrow Q_L^T G_t Q_R$
{Idealized version of code for Adafactor taking G'_t to be the gradient}
$G'_{B_t} \leftarrow Q_L^T G_{B_t} Q_R$
$A = \mathbb{E}_B[G'_B \odot G'_B]1_m$ where $G'_B = Q_L^T G_B Q_R$
$C = 1_n^T \mathbb{E}_B[G'_B \odot G'_B]$
$\hat{V}_t = \frac{AC^T}{1_n^T A}$ {Elementwise division}
$G''_t \leftarrow \frac{G'_t}{\sqrt{\hat{V}_t + \epsilon}}$ {Elementwise division and square root}
$G'''_t \leftarrow Q_L^T G''_t Q_R$ {Projecting back to original space}
$W_t \leftarrow W_{t-1} - \eta G'''_t$

"Idealized Adafactor": Get rank-1 estimates in rotated space given by Q matrices and rotate them back to update weights

Theorem: These two algorithms are equivalent!

Insights from the idealized algorithms

- In practice, Shampoo and Adafactor in Shampoo's eigenbasis are NOT equivalent and differ:
  - When using dataset averages vs running averages
  - When the eigenvector decomposition of L and R is not computed at every step
- Key insight: eigenvector decomposition is expensive, but updating the second moment estimates in the rotated space is inexpensive!
- **Why not opt for Adam instead of Adafactor? (or any other diagonal preconditioner?)**

SOAP!

- **S**hampo**O** with **A**dam in the **P**reconditioner's eigenbasis
- Part of a broader space of second order algorithms where **first order methods** are run in the space provided by **a second order method's preconditioning**
- Much fewer hyperparameters compared to Shampoo, and adds one additional hyperparameter to Adam - **preconditioning frequency**

SOAP: Improving and Stabilizing Shampoo Using Adam. Vyas, Morwani, **Zhao**, Shapira, Brandfonbrener, Janson, Kakade. Submitted.

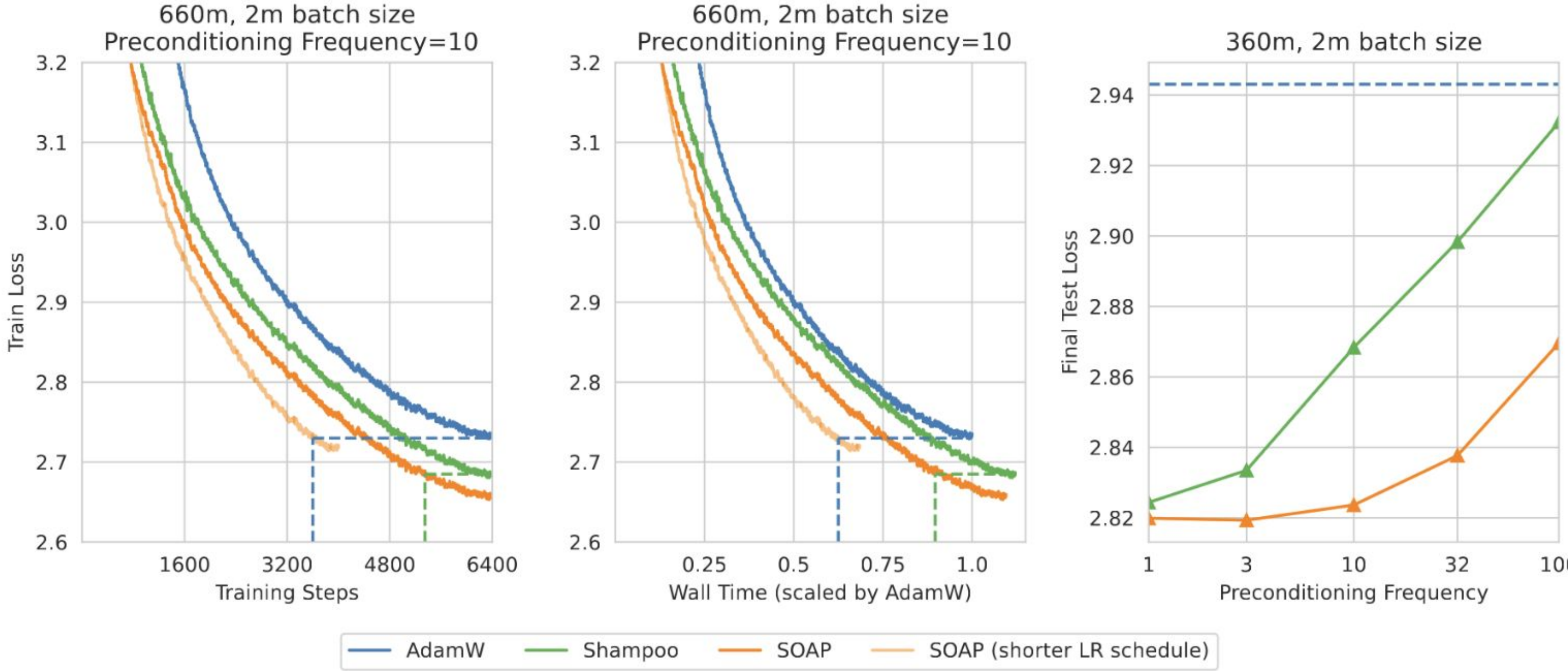
SOAP algorithm

Algorithm 3 Single step of SOAP for a $m \times n$ layer. Per layer, we maintain four matrices: $L \in \mathbb{R}^{m \times m}$, $R \in \mathbb{R}^{n \times n}$ and $V, M \in \mathbb{R}^{m \times n}$. For simplicity we ignore the initialization and other boundary effects such as bias correction. Hyperparameters: Learning rate η, betas $= (\beta_1, \beta_2)$, epsilon ϵ, and preconditioning frequency f.
An implementation of SOAP is available at `https://github.com/nikhilvyas/SOAP`.

Sample batch B_t.
$G \in \mathbb{R}^{m \times n} \leftarrow -\nabla_W \phi_{B_t}(W_t)$
$G' \leftarrow Q_L^T G Q_R$
$M \leftarrow \beta_1 M + (1 - \beta_1) G$
$M' \leftarrow Q_L^T M Q_R$
{Now we "run" Adam on G'}
$V \leftarrow \beta_2 V + (1 - \beta_2)(G' \odot G')$ {Elementwise multiplication}
$N' \leftarrow \frac{M'}{\sqrt{\hat{V}_t + \epsilon}}$ {Elementwise division and square root}
{Now that we have preconditioned by Adam in the rotated space, we go back to the original space.}
$N \leftarrow Q_L N' Q_R^T$
$W \leftarrow W - \eta N$
{End of gradient step, we now update L and R and possibly also Q_L and Q_R. }
$L \leftarrow \beta_2 L + (1 - \beta_2) G G^T$
$R \leftarrow \beta_2 R + (1 - \beta_2) G^T G$
if t % f == 0 **then**
    $Q_L \leftarrow \texttt{Eigenvectors}(L, Q_L)$
    $Q_R \leftarrow \texttt{Eigenvectors}(R, Q_R)$
end if

SOAP: Improving and Stabilizing Shampoo Using Adam. Vyas, Morwani, **Zhao**, Shapira, Brandfonbrener, Janson, Kakade. Submitted.

Experiments



- 40% reduction in iterations and 35% reduction in wall clock time with respect to Adam, and 20% reduction to both with respect to Shampoo
- More robust to higher preconditioning frequency

SOAP: Improving and Stabilizing Shampoo Using Adam. Vyas, Morwani, **Zhao**, Shapira, Brandfonbrener, Janson, Kakade. Submitted.

Summary and Conclusions

- More results in paper: throughput, smaller batch sizes, efficiency improvements
- SOAP outperforms both AdamW and Shampoo on language modeling tasks
- Need to explore further improvements (lower precision, distributed implementation) and using SOAP on other domains (try it!)
- Second order methods potentially have further untapped potential - **diagonal preconditioning optimizers are all similar, and second order methods like SOAP/Shampoo seem to be better!**

Thank you!





Deconstructing Optimizers

SOAP

Theory on Shampoo (collaborators)