Adam for Transformers: Why and Why Not

Yushun Zhang

The Chinese University of Hong Kong, Shenzhen, China

NeurIPS 2024

Presented at Tsinghua, Dec 2024
Thanks Kaifeng for the invitation!

Pipeline to Train LLMs

Your final LLM

Initial model: W^0 → $W^{1\,Trillion}$ → $W^{1\,Trillion+100k}$ → $W^{1\,Trillion+100k+100k}$

Pre-train

Dataset:
>1 Trillion tokens of textbooks and Wikipedia, etc.

Training Objective:
Predict next token

Algorithm:
(1^{st}-order OPT + Adam)

1000+GPU
Takes months

SFT

Dataset:
100k – 1000k tokens of Question & Answer pair

Training Objective:
Predict next token

Algorithm:
(1^{st}-order OPT + Adam)

1-100GPU
Takes days

RLHF

Dataset:
100k – 1000k tokens of human preference data

Training Objective:
maximize reward

Algorithm:
(0^{th}-order OPT+Adam)

1-100GPU
Takes days

Remark: PPO? Not precise!

Pre-training is EXPENSIVE

	Model size	Data size	# A800	Train Time	Rent price (RMB)	Purchase price (RMB)	Remark
预训练 (pre-training)	13B	1T tokens	80 张	约 45 days	691K	**16M**	Entry-level model
预训练 (pre-training)	130B	10T tokens	8000 张	约 140 days	21B	**16B**	Average-level model
领域增量预训练 (continue-training)	13B	200B tokens	80 张	约 9 days	140K	**16M**	Average-level domain-specific model

Remark:

1) 租赁价格, 按单卡 8 元/hr计算. 购买价格按1 台 A800 8 卡 160 万计算.

2) 如果是千卡租赁，可以谈到 3.3 元/hr 的价格，130B 模型可以降到 8600 万左右, 仍然很高。

Message:

a) 从头开始预训练13B 模型，80 张 A100, 一次训练需要一个半月

b) 增量预训练 13B的模型，只用 1/5 的 tokens (200B tokens)，一次训练的时间也很长。

Background

- **Adam** becomes the most popular algorithms in deep learning (DL). ~~(>170,000 citations, by May 2024~~, >198k citations, by Oct 2024)

- **Default in** LLM (large language models)

```
optimizer = optim.Adam(net.parameters(), lr=args.lr, betas=(args.beta1, args.beta2), eps=1e-08,
                       weight_decay=args.weightdecay, amsgrad=False)
```

为了提升大模型训练， 需要理解 Adam

Overview of this talk

Part I:

- Why LLM training requires Adam, not SGD?
- We explain it from Hessian spectrum perspective

Part II:

- **Adam-mini:** A “mini” version of Adam
- Saves memory by **45%-50%;** same loss curve as Adam
- Can achieve **49.6%** higher throughput on Llama2-7B pre-training (saves **30%** training time)

Contents

Let us start with SGD…

- Consider $\min_{x} f(x) \coloneqq \sum_{i=1}^{n} f_i(x)$.

  n: number of samples (or mini-batches of samples)

  x: trainable parameters

- In the k-th iteration: Randomly sample τ_k from $\{1,2,\ldots,n\}$

SGD (Stochastic gradient descent): $x_{k+1} = x_k - \eta_k \nabla f_{\tau_k}(x_k)$

SGD with momentum (SGDM):

$$m_k = (1-\beta_1)\nabla f_{\tau_k}(x_k) + \beta_1 m_{k-1}$$

$$x_{k+1} = x_k - \eta_k m_k$$

1st order momentum

Iterate update

Adam

- $\min_x f(x) := \sum_{i=1}^{n} f_i(x)$. In the k-th iteration: Randomly sample τ_k from $\{1,2,\dots,n\}$

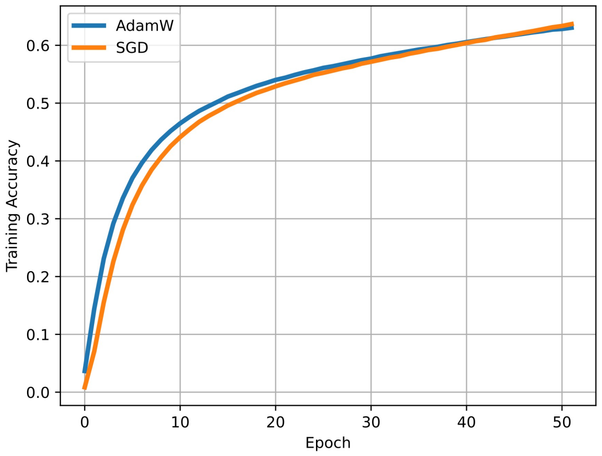
- Adam (Kingma and Ba'15):
- $m_k = (1-\beta_1)\nabla \mathrm{f}_{\tau_k}(\mathrm{x}_k) + \beta_1 m_{k-1}$ ← 1st order momentum
- $v_k = (1-\beta_2)\nabla \mathrm{f}_{\tau_k}(\mathrm{x}_k) \circ \nabla \mathrm{f}_{\tau_k}(x_k) + \beta_2 v_{k-1}$ ← 2nd order momentum
- $x_{k+1} = x_k - \eta_k \frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \frac{m_k}{\sqrt{v_k}}$ ← Iterate update

- β_1: Controls the 1st-order momentum m_k. Default setting: $\beta_1 = 0.9$
- β_2: Controls the 2nd-order momentum v_k. Default setting: $\beta_2 = 0.999$
- One important difference with SGD:

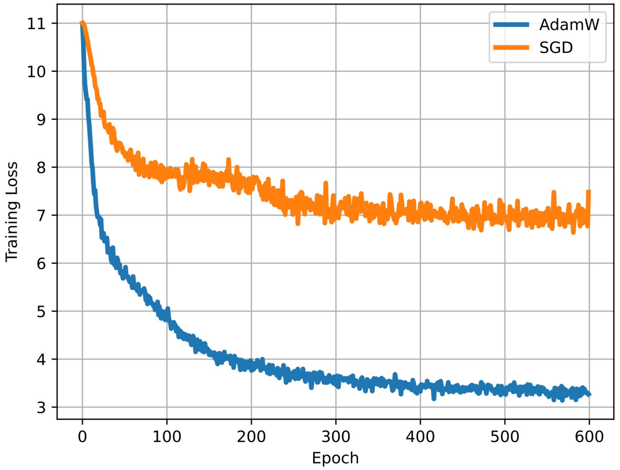
  -- Adam use coordinate-wise lr $\frac{\eta}{v_i}$ ‘

  -- SGD uses single lr η for all

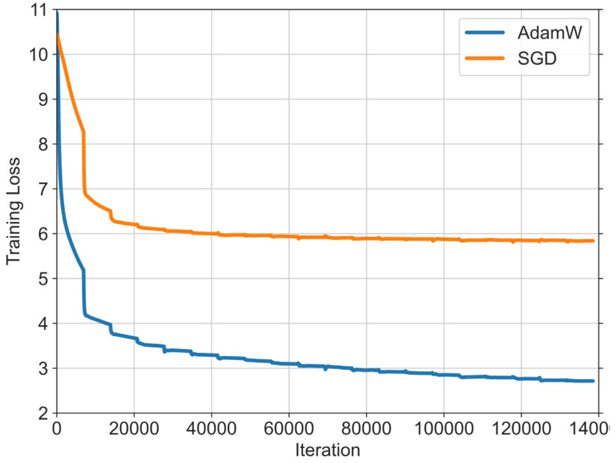
SGD works well on CNN, largely underperforms Adam on Transformers



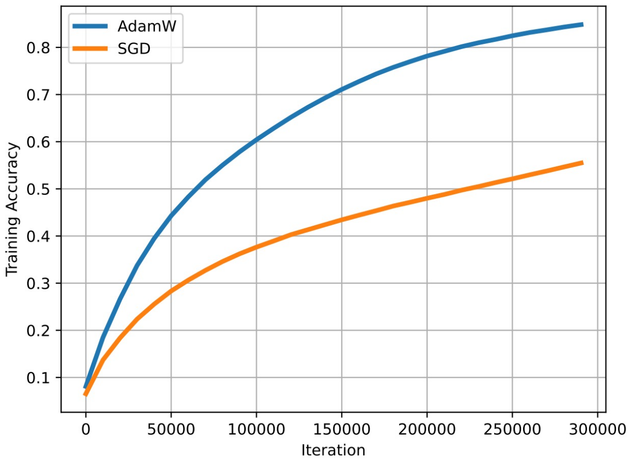
VGG (CNN)



GPT2



BERT



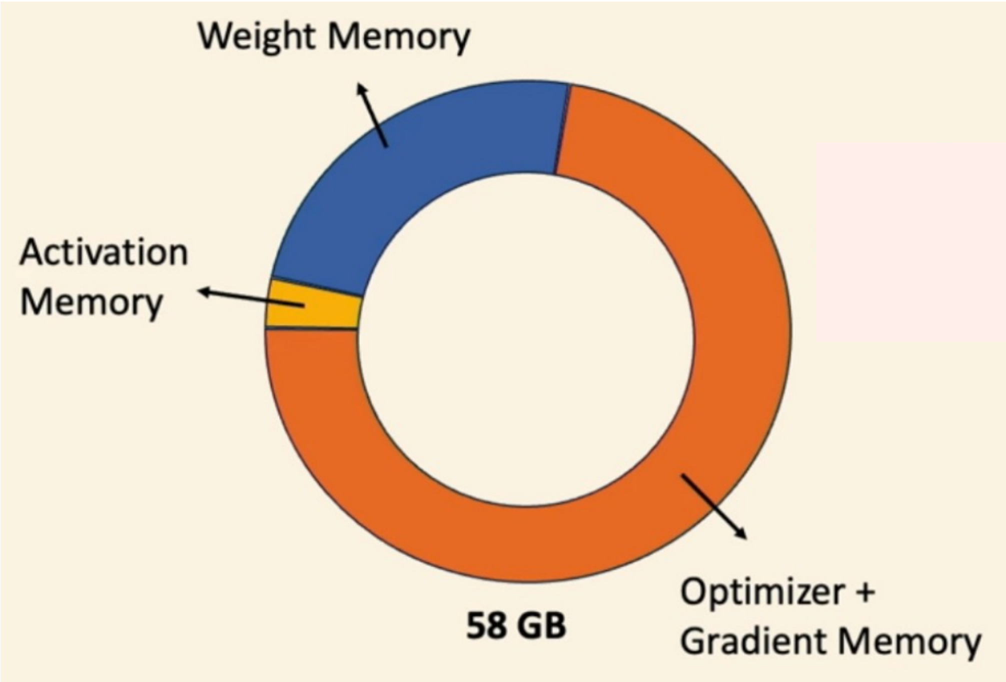
ViT

Transformers Need Adam, but why?

However, Adam is expensive to use…

- Adam needs memory for m and v
  -- In total: 2x model size

  -- becomes a major overhead for LLMs: e.g., for 7B models



[Figure from a recent talk by Meta]

- Palm-540B: Adam alone takes **50x** A100-80GB GPUs …

Literature on Why Adam better than SGD

- One perspective [J. Zhang et al. 19]
  NLP problems exhibit **heavy-tailed noise** in g $\longrightarrow$ SGD fails

- However, [Chen, Kunstner, Schmidt'21] provides negative evidence:
  SGD is worse than Adam on Transformers, even in **full-batch case** (with no stochasticity)

Heavy-tailed noise does not explain the gap between SGD and Adam on Transformers

Jacques Chen, Frederik Kunstner, Mark Schmidt
University of British Columbia

- So there shall be other reasons...

What problem structure might hamper SGD?

- **Hessian eigenvalues** largely decides behavior of gradient methods [Nocedal & Wright'99, Nesterov'13, Goh'17, Sun'2019]

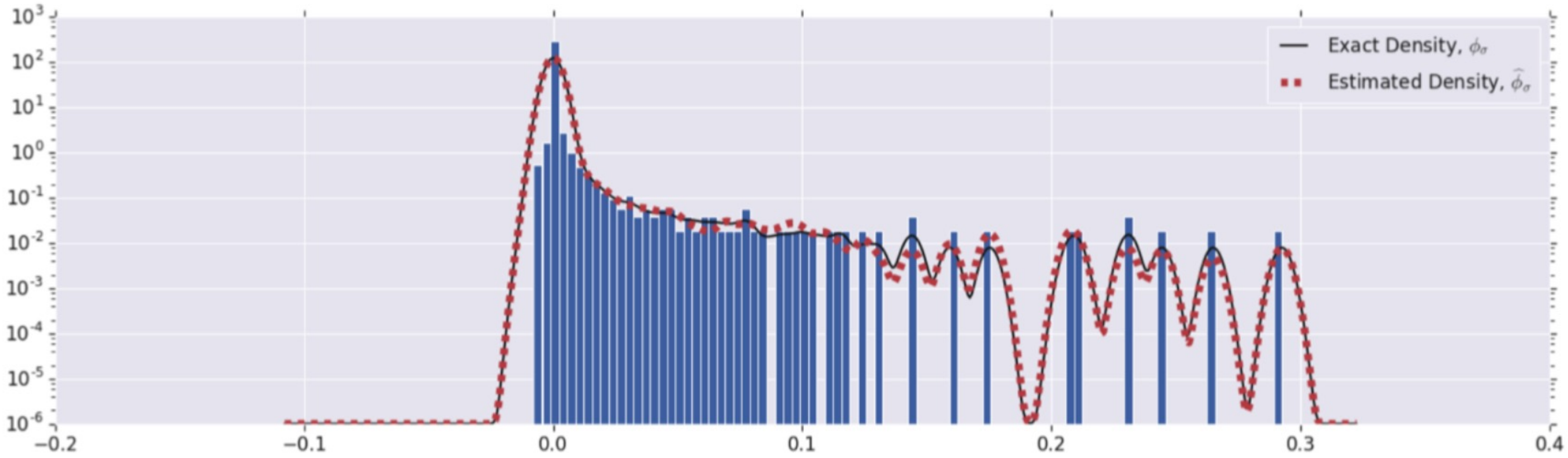
- For instance: ill-conditioning slow down GD

- Can Hessian spectrum explain the gap of SGD and Adam?

- Unfortunately, no (not directly)…

Preliminary: Hessian spectrum

y-axis: frequency



x-axis: Eigenvalues.

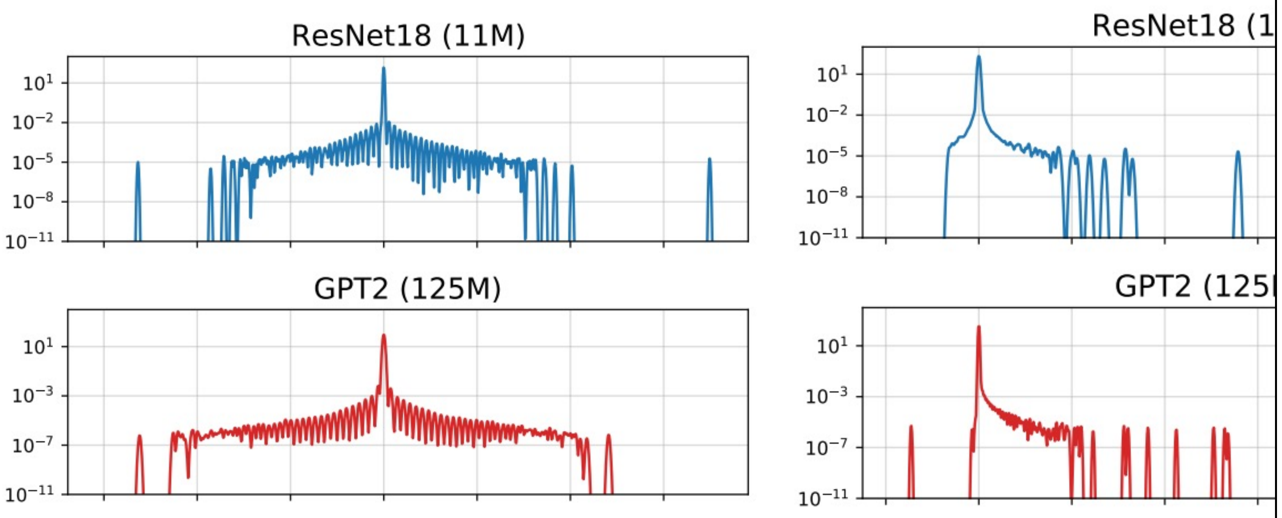
Figure from [Ghorbani et al. 19]

What is spectrum: histogram of eigenvalues

Remark: How to plot Hessian spectrum?
We use Stochastic Lanczos Quadrature (SQL) [Bai, Fahey, and Golub 1996]
(will take >10 pages to explain, omitted today)

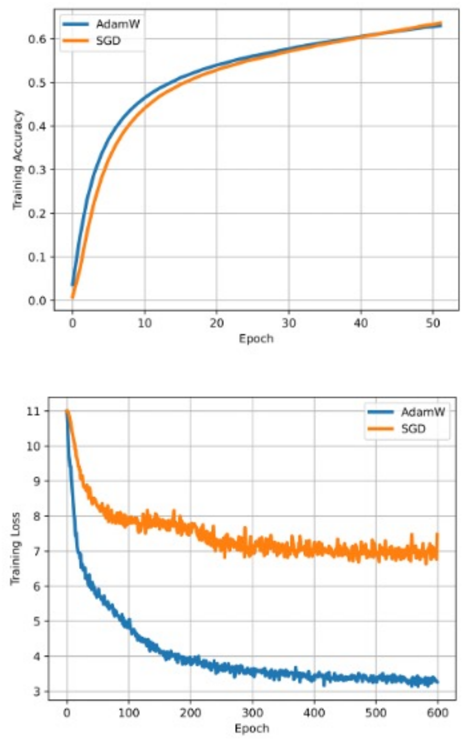
We will compute the Hessian spectrum for a wide range of neural networks

Hessian spectrum cannot explain the gap



(a) Initialization

(c) 50% st



SGD ≈ Adam on CNNs

SGD ≪ Adam on Transformers

CNN and Transformers:
Spectrum looks quite similar!
(see more figures in the paper)

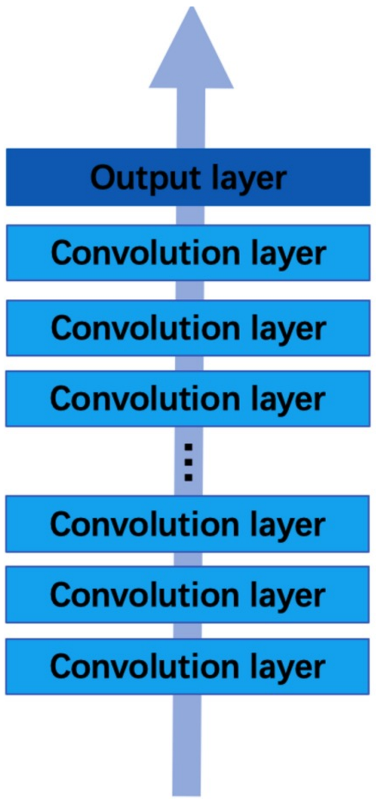
Something must be overlooked…

- Full hessian spectrum does not seem informative enough

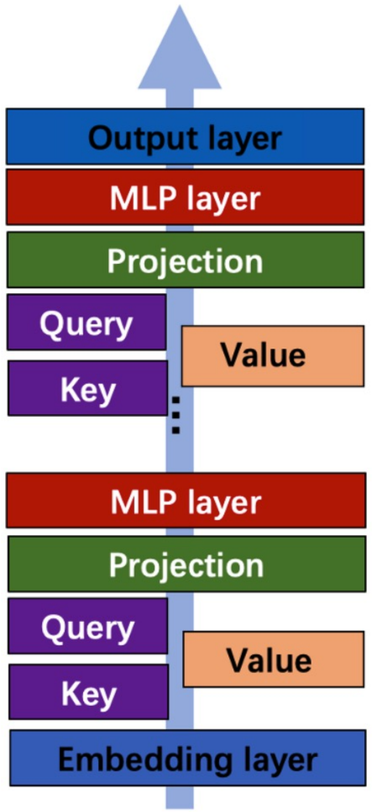
- What else?

- We find one important features that are overlooked:
  The build-up rules of the architecture
  -- Transformers are stacked up by different kinds of layers

Build-up rules of architectures

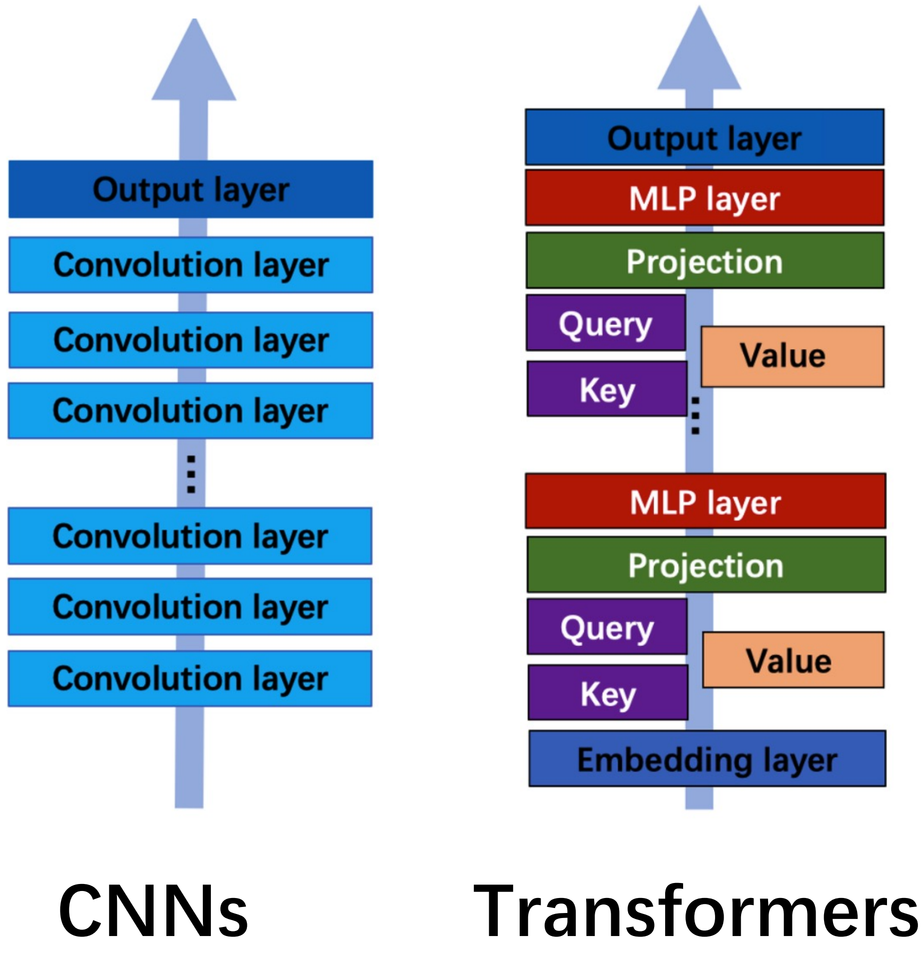


CNNs:
repetitive stack of similar layers



Transformers:
each block follow different design
(e.g., Q, K, V, MLP)

Build-up rules of architectures



We hypothesize:

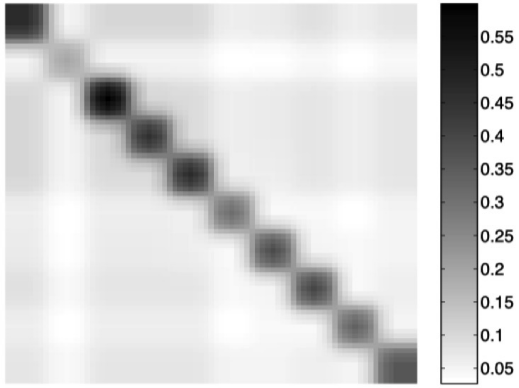
Different designs among parameter blocks

will affect

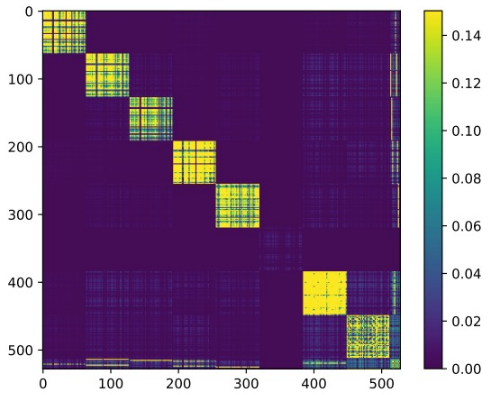
Hessian of these parameter blocks.

This inspires us to investigate the **blockwise Hessian spectra** (i.e., principle diag blocks in Hessian)

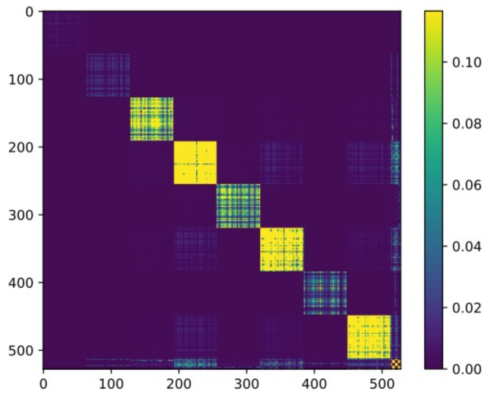
Another reason for studying blockwise Hessian **near Block Diagonal structure**



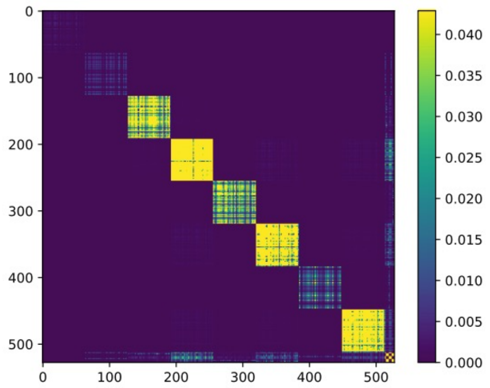
(a) Exact Hessian of a MLP (Collobert, 2004)



(b) Hessian of an MLP at 1% step



(c) Hessian of an MLP at 50% step



(d) Hessian of an MLP at 100% step

Proof (from Collobert 2004): for 1-hidden-layer network $f(\theta, x)$ + **Cross Entropy loss**, we have

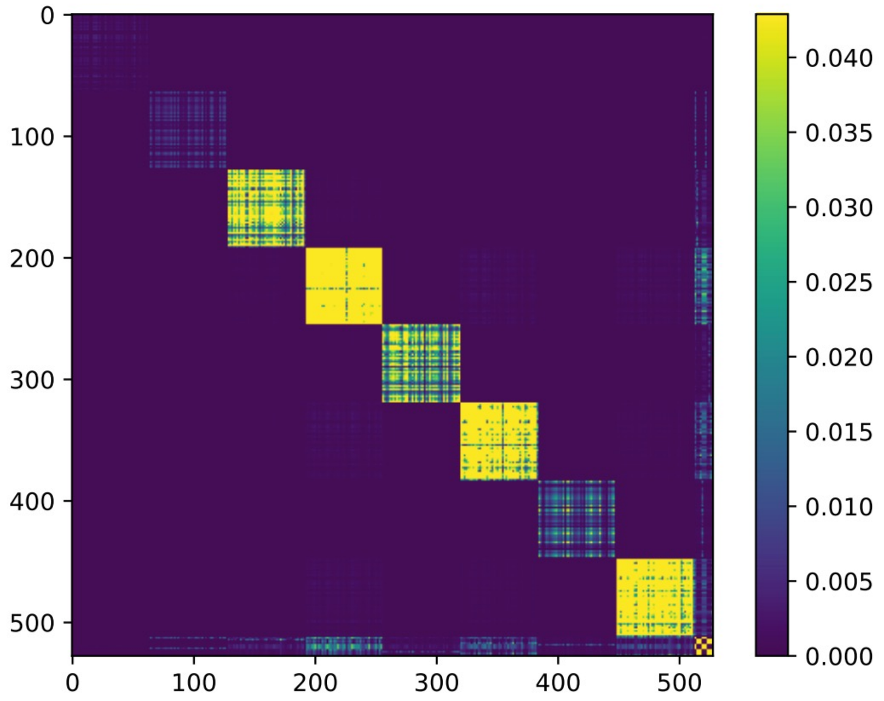
$$\frac{\partial^2 \ell(f(\theta,x),y)}{\partial w_i \partial w_j} = p_\theta(y|x)\,(1-p_\theta(y|x))v_i v_j \phi'\left(w_i^\top x\right)\phi'\left(w_j^\top x\right) x x^\top \quad \text{for } i \neq j,$$

where $w_i \in R^{\,data\ dimension}$: the weight associated with the i-th output neuron

When we maximize $p_\theta(y|x)$, $p_\theta(y|x)\,(1-p_\theta(y|x))$ will quickly shrink to zero

But this structure is largely overlooked for both opt & DL community (sadly…)

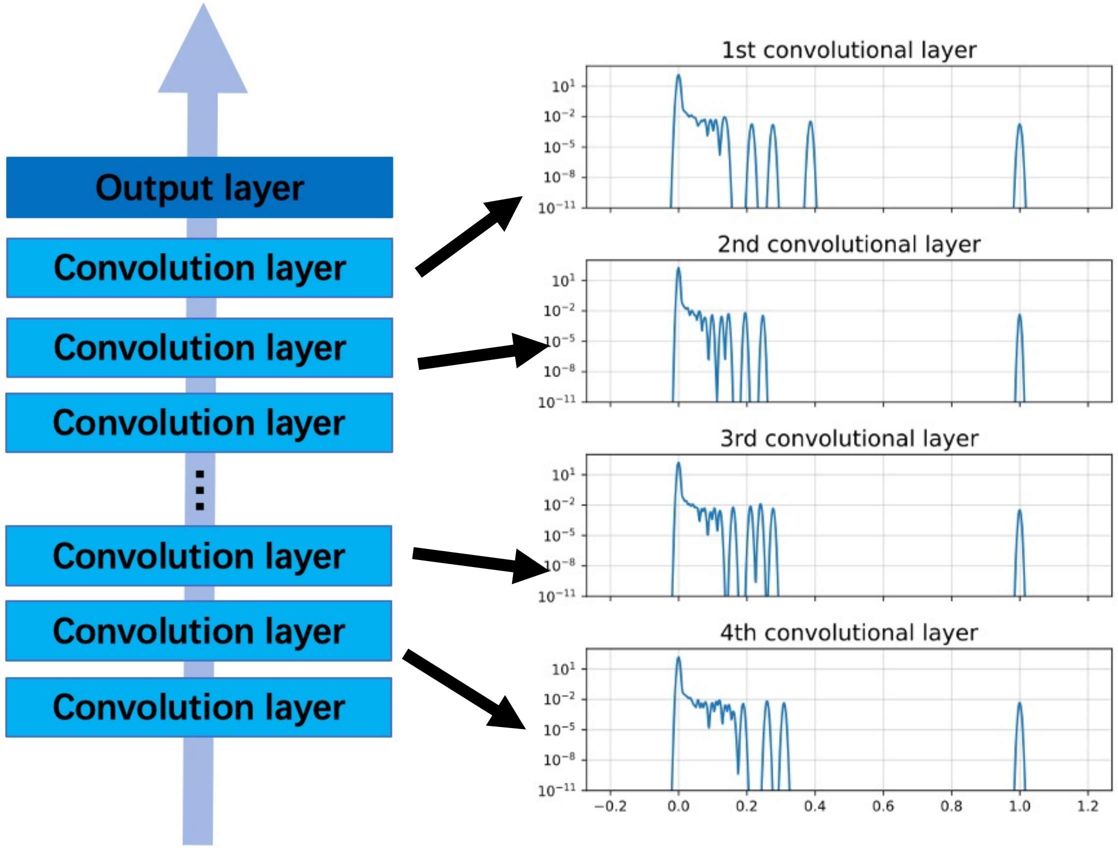
Blockwise Hessian spectrum might matters



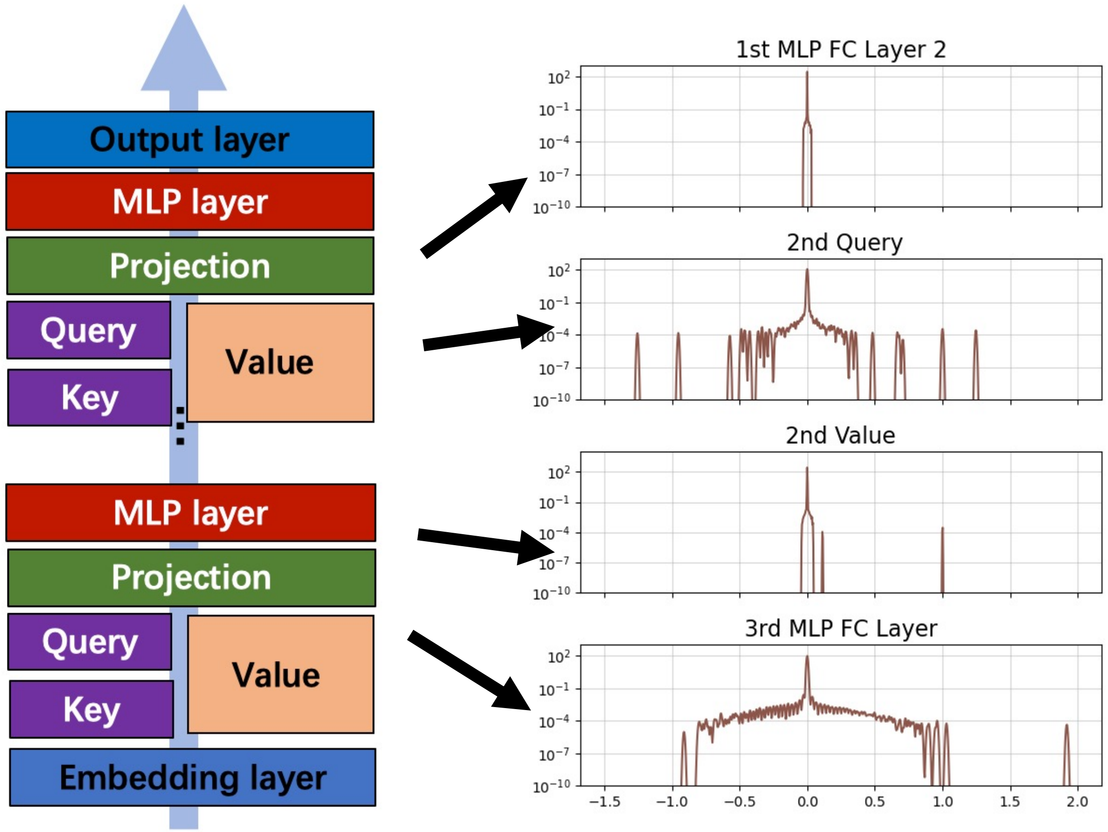
Conjecture: Eigenvalues in **each block (e.g., Q, K, V)** could be important

- What extra info over full spectrum?
- By linear algebra: **location** of eigenvalues

Blockwise Hessian spectrum



ResNet18



BERT

CNNs: blockwise spectrum are quite **similar**
We call it ``**homogeneity**''

Transformers: blockwise spectrum are largely **different**
We call it ``**heterogeneity**''

JS-distance among blocks

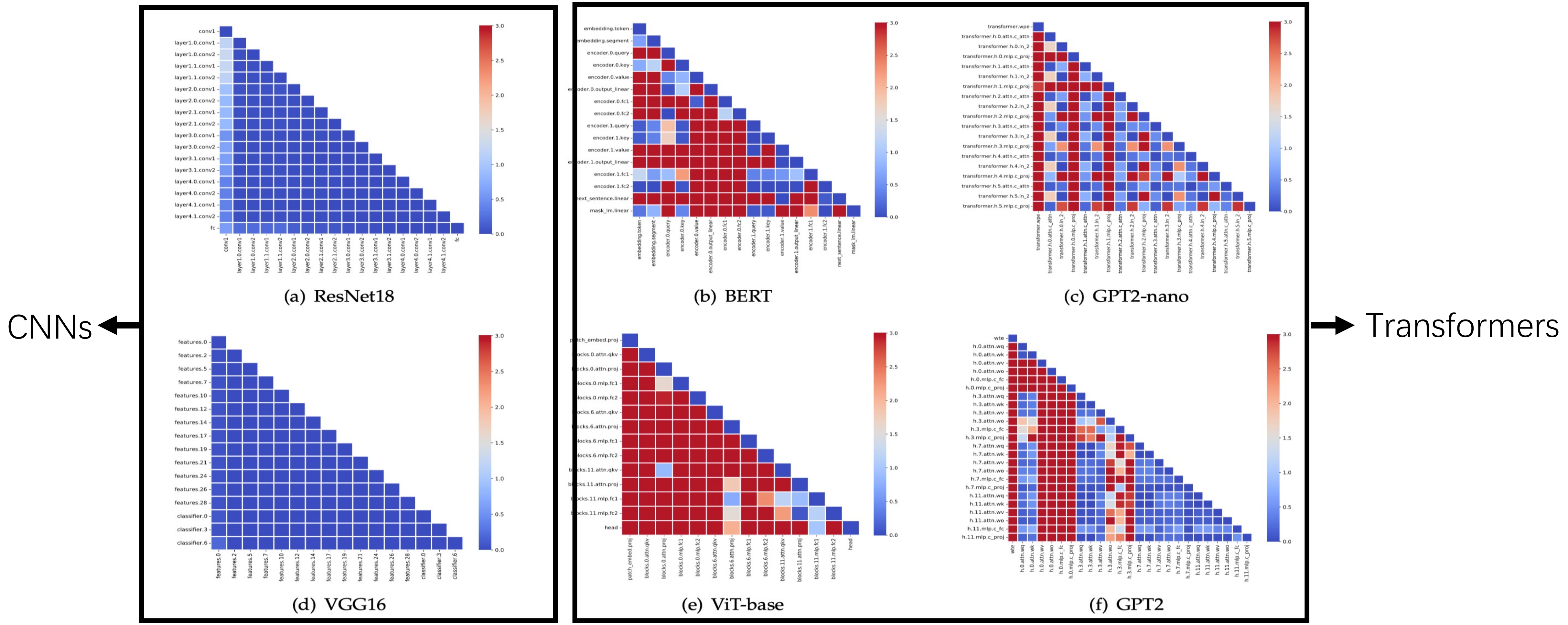


Figure 4: The JS distance among blockwise Hessian spectra for different models at initialization.

Observation 1: Heterogeneity is widely observed in Transformers, but not on CNNs!

Our Explanation: Why Transformer Needs Adam

Observation 1:
Transformer’s block Hessian are heterogeneous

Previous part

Together explains:

Original claim:
Transformer:
SGD worse than Adam.

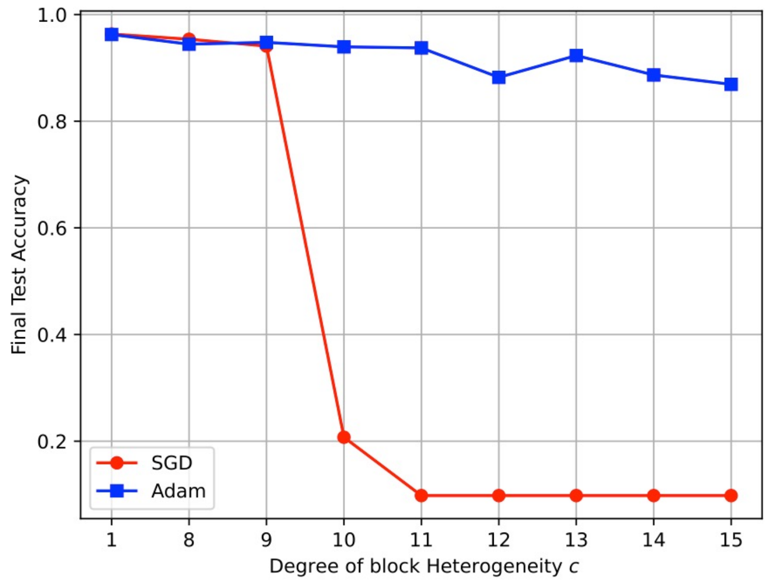
Observation 2:
If block Hessian are heterogeneous,
then SGD worse than Adam.

Next part:

Claim 2: Hetereogeneity causes SGD worse than Adam.

Hetereogeneity makes SGD worse: Example 2 (pure MLP)

- **Q:** Is Transformer the only architecture that is heterogeneous & SGD worse?
- **A:** No! We provide a few more heterogeneous examples that SGD is worse.



x-axis: degree of Heterogeneity
y-axis: final converged accuracy

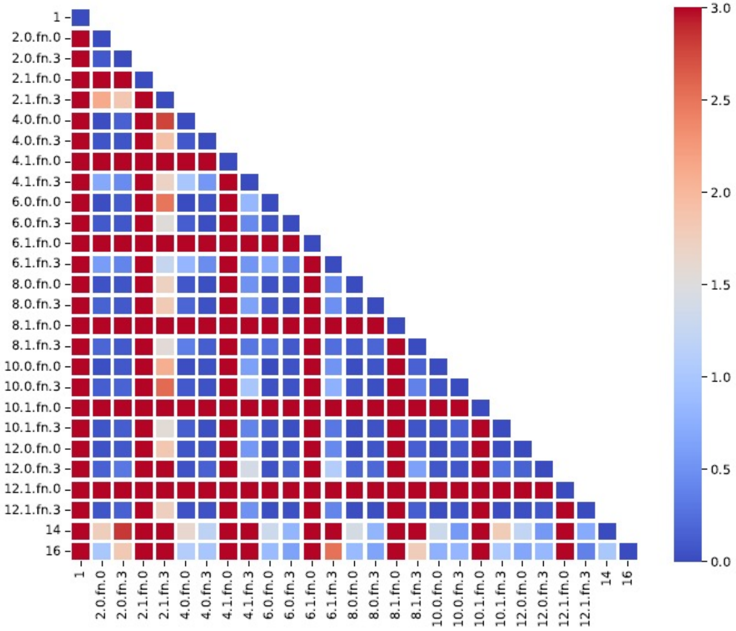
Example 2:
A man-made MLP on MNIST:
We exert heterogeneity by scaling each layer differently

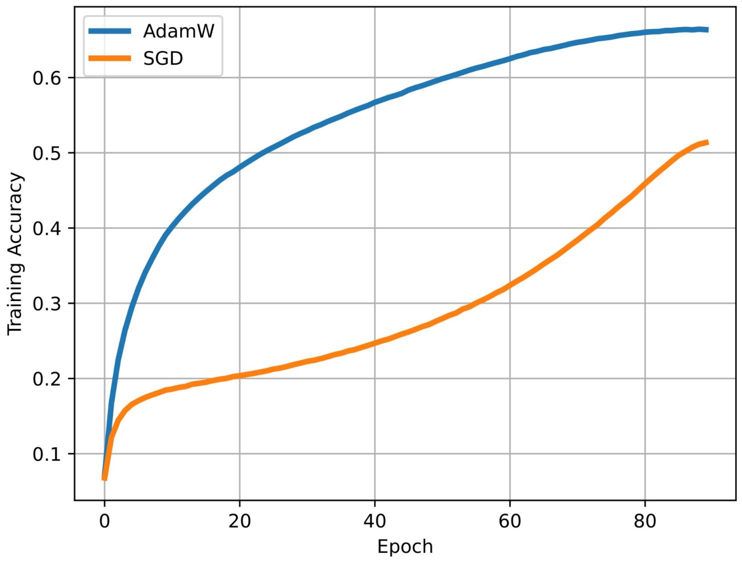
Observation:
SGD fails as heterogeneity grows
while Adam remains unaffected

Hetereogeneity makes SGD worse: Example 3 (MLP-mixer)

Example 3: MLP-mixer [1]

A pure MLP architecture that outperforms CNNs on ImageNet





Observation:

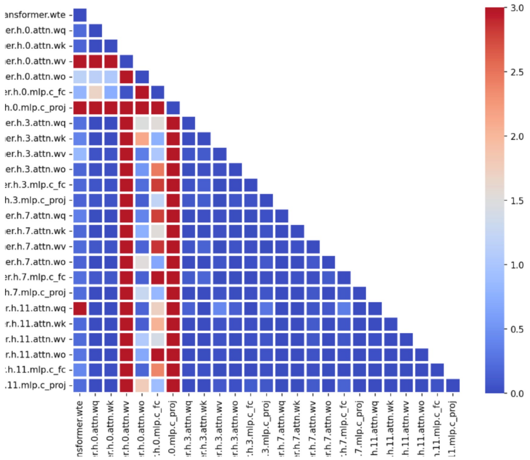
--MLP mixer is heterogeneous

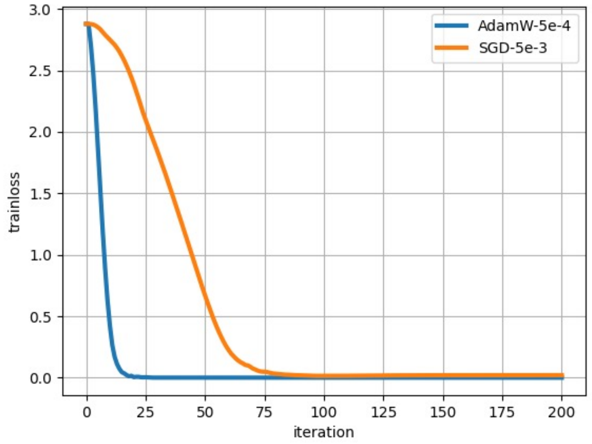
--SGD performs worse than Adam

[1] MLP-mixer: An All-MLP Architecture for Vision. Tolstikhin et al. , NeurIPS 2021

Heterogeneity is redued after training

- We find **pretrained Transformers** suffer less heterogeneity
- May explain why fine-tuning is easier
- SGD could work here: still slower, but can reach similar loss as Adam
- Similar phenomena also holds for ViT-base

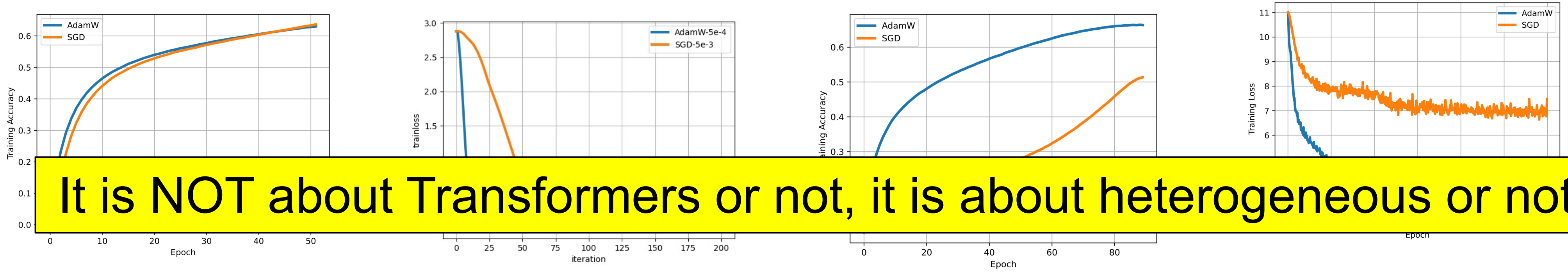




GPT2-125M (pretrained on 25B tokens): finetuning on a subset of Alpaca

Observation 3: Heterogeneity tends to reduce after (pre)-training

SGD performance v.s. Heterogeneity in Hessian



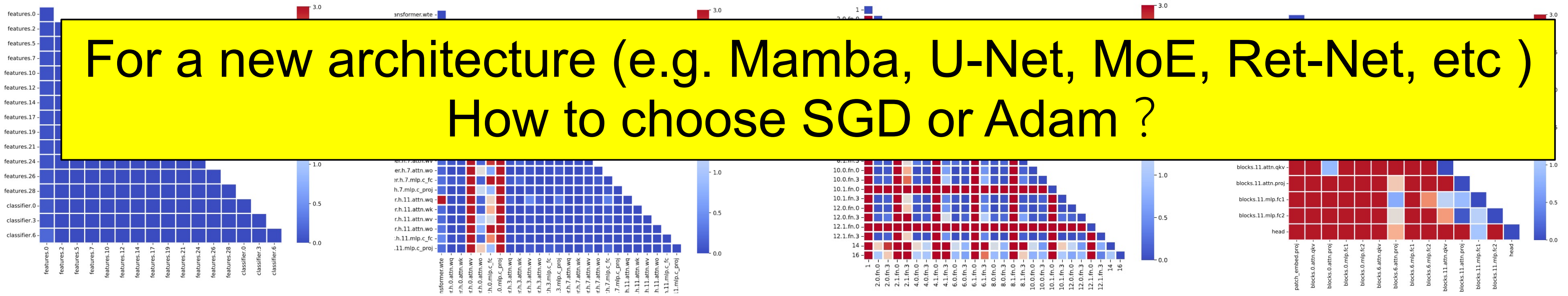
It is NOT about Transformers or not, it is about heterogeneous or not!

Easy for SGD

Difficult for SGD

Homogeneity in Hessian

Heterogeneity in Hessian



For a new architecture (e.g. Mamba, U-Net, MoE, Ret-Net, etc)
How to choose SGD or Adam？

CNN

Transformer (pretrained)

MLP-mixer

Transformer

Empirical guidance: choose SGD or Adam?

- Introduce metric to **predict** the failure of SGD **before launching the training**

- **Our metric:** average JS distance of spectrum among blocks at step = 0, called JS^0
- This metric could be efficiently computed using Stochastic Lanczos Quadrature
  Our **PyTorch implementation**: https://github.com/zyushun/hessian-spectrum

Model	ResNet18	VGG16	GPT2 (pretrained)	MLP-mixer	BERT	GPT2	ViT-base
JS^0	0.10	0.09	18.84	34.90	53.38	83.23	286.41

For CNNs: 100x smaller than Transformers!

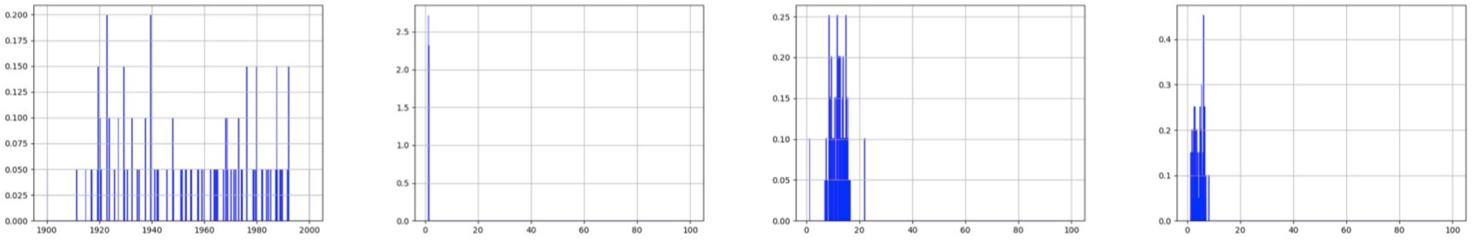
Initial theory

Hetereogeneity makes SGD worse: Quadratic Prob

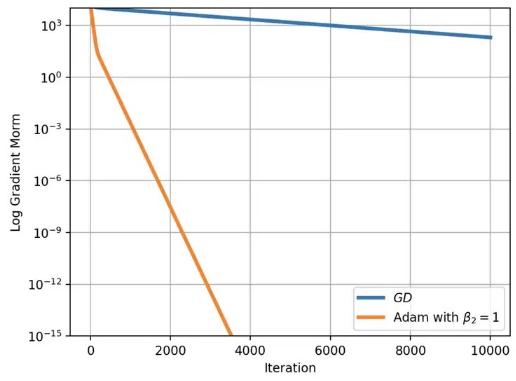
- Case study on quadratic models

$$\min_{w} \frac{1}{2} w^T H w, H = diag(H1, H2, H3, H4), H \text{ is PD}$$

Case 1: H with **Transformer-type spectra:** sampled from GPT2

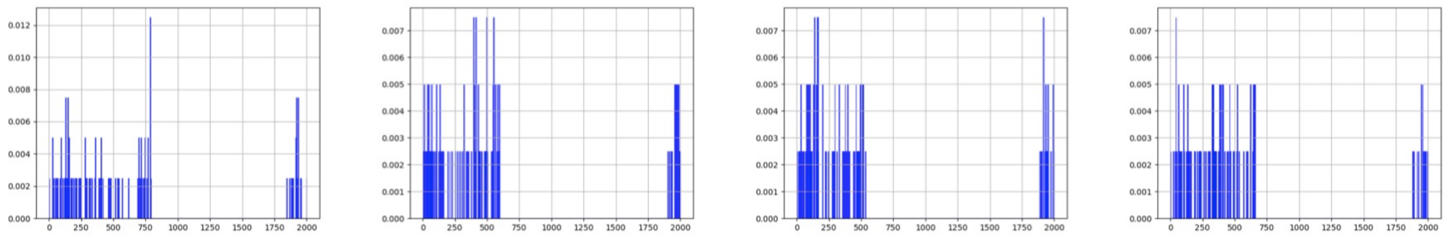


(a) Spectrum of H_1 (b) Spectrum of H_2 (c) Spectrum of H_3 (d) Spectrum of H_4

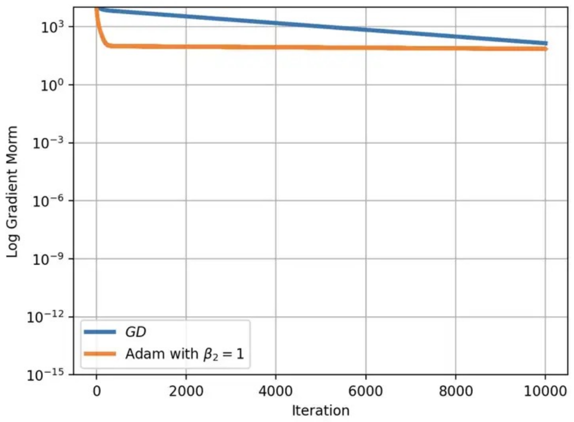


GD is slower than Adam

Case 2: H with **CNN-type spectra:** sampled from ResNet18



(a) Spectrum of H_1 (b) Spectrum of H_2 (c) Spectrum of H_3 (d) Spectrum of H_4



GD is similar as Adam

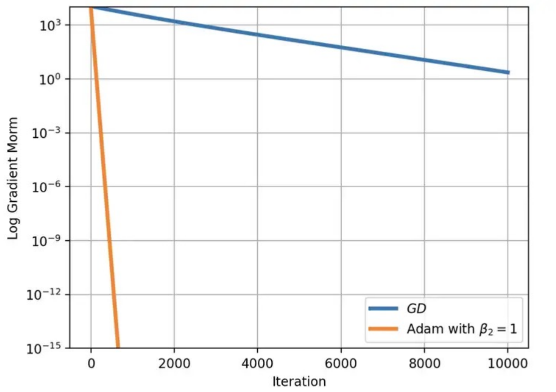
Remark:
Same condition number for case 1 & 2
but the performance is different, due to **homo & heterogeneity**

Hetereogeneity makes SGD worse: Quadratic Prob

$$\min_w \frac{1}{2} w^T H w, H = diag(H1, H2, H3), H \text{ is PD}, \quad H_l \in R^{3\times 3}, l = 1,2,3$$

Case 3: H with simplified **heterogeneous** spectra

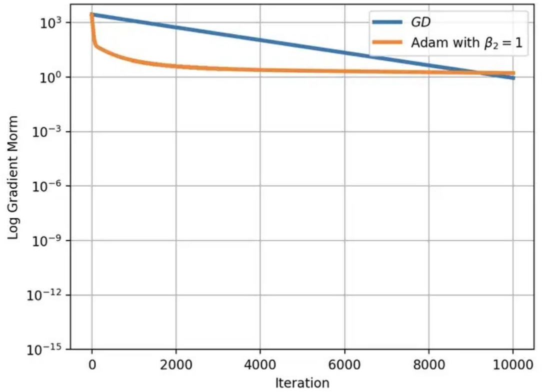
Eigenvalues of H_l :
{ 1, 2, 3 } , { 99, 100, 101 } , { 1998, 1999, 2000 }



GD is slower than Adam

Case 4: H with simplified **homogeneous** spectra

Eigenvalues of H_l :
{ 1, 99, 1998 } , { 2, 100, 1999 } , { 3, 101, 2000 }



GD is similar as Adam

Remark:
All eigenvalues are the same for case 3 & 4
but the performance is different, due to **homo & heterogeneity**

Theoretical results

A well-known result for GD: Consider $\min_w f(w) = \frac{1}{2} w^T H w$, where H is PD, then there exits a H and w^0 :

$$f(w_{GD}^{t+1}) - f^* \geq \left(1 - \frac{2}{\kappa}\right)(f(w_{GD}^{t}) - f^*)$$

where $\boldsymbol{\kappa}$ is the condition number of H

Theorem 1(Adam): Consider $\min_w f(w) = \frac{1}{2} w^T H w$, where H a block-diagonal PD matrix with L blocks, then:

$$f(w_{Adam}^{t+1}) - f^* \leq \max_{l \in [L]} \left(1 - \frac{1}{\kappa_{Adam,l}}\right) (f(w_{Adam}^{t+1}) - f^*)$$

where $\kappa_{Adam,l} = r\, \boldsymbol{\kappa_l}$, $\boldsymbol{\kappa_l}$ is the condition number of H_l; r is a constant related to initialization w^0

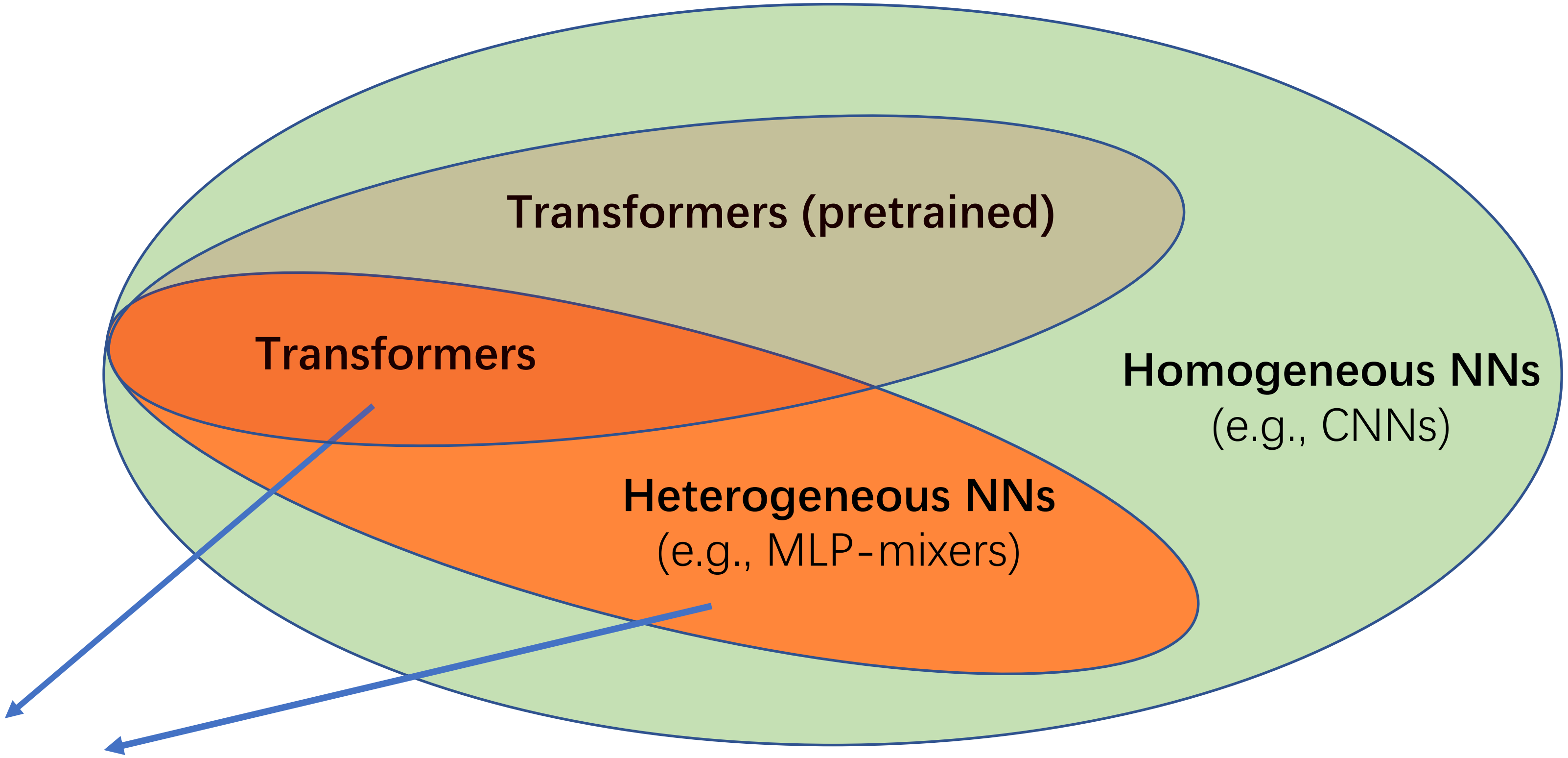
Comparing Two Results

- **Compare GD vs Adam:**
  Adam is faster than GD when $r\, \boldsymbol{max_l\, \kappa_l} \leq \boldsymbol{\kappa}$,

- Happens in the heterogeneous quadratic examples
  Quantity of Adam ~20x smaller, and Adam is also 20x faster

- This provides a partial theoretical explanation why Adam works better than SGD on Heterogeneous case

Summarize in one figure



SGD < Adam here!

Why is SGD slow?
-- SGD assigns **one lr for all** parameter blocks
-- Cannot handle heterogeneity across blocks

Why Adam?
--Each block needs (at least) one customized lr
-- could be provided by v

Contents

Part I Why Transformers need Adam?

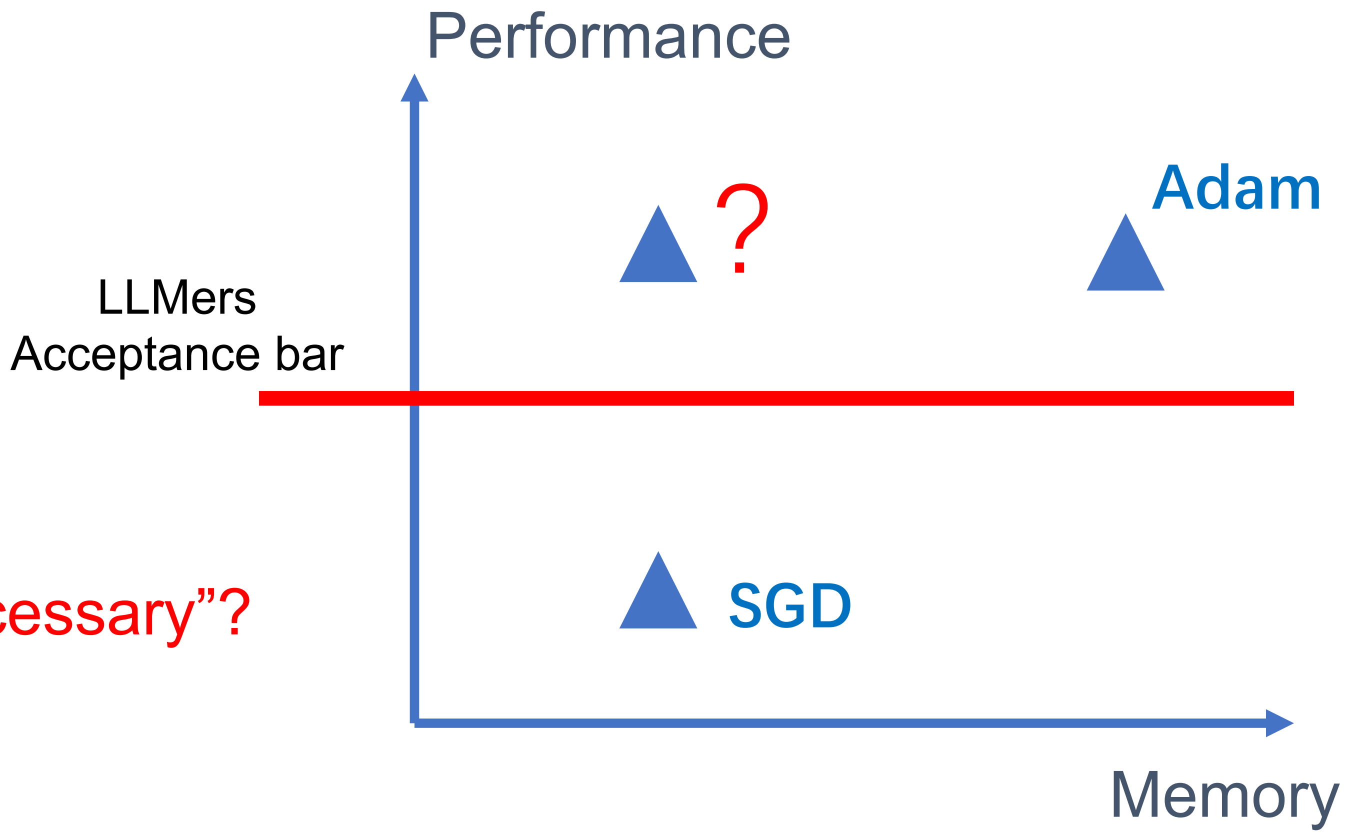
Part II Adam-mini: A lightweight version of Adam

Is Adam really “necessary”?

But… this mainly shows:

SGD is not sufficient,
Adam is “sufficient”

Question: Is Adam really “necessary”?



How to slim down Adam?

Major difference of Adam with SGD: its diagonal preconditioner

$$\boxed{w \leftarrow w - \eta \; D \circ m}$$

-- SGD: $D_{SGD} = I$

-- Adam: $D_{Adam} = Diag(\frac{1}{\sqrt{v_1}}, \frac{1}{\sqrt{v_2}}, \ldots, \frac{1}{\sqrt{v_2}})$

- Why D_{Adam} helps: D_{Adam} assigns different lrs for each parameters
- In optimization theory: converges faster when $\kappa(D_{Adam}H) \leq \kappa(H)$
- **Caveat**:
  D_{Adam} is not always effective!
  Related to an old topic in linear algebra

A bit of history: diagonal preconditioner

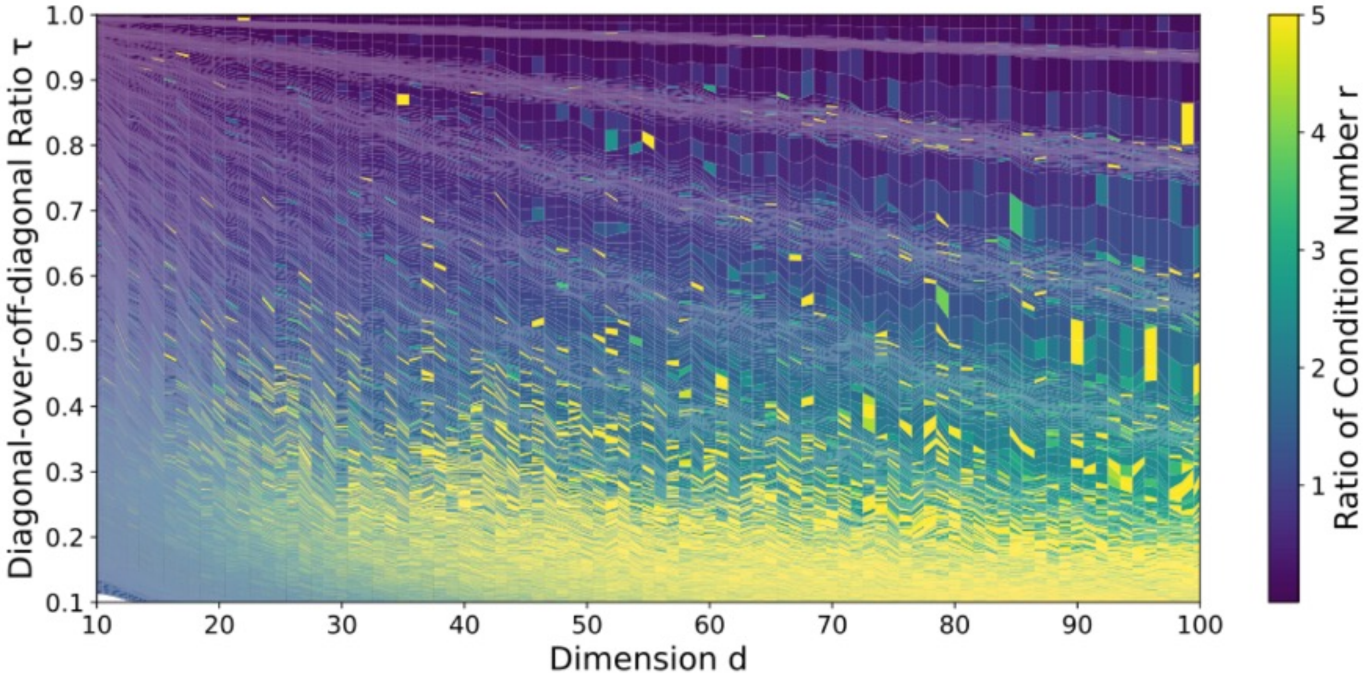
Q1: For what kind of Hessian H do we have $\kappa(D_{Adam}H) \leq \kappa(H)$?

- This is a pure linear algebra problem, but NOT easy to answer…
- In **[1]:** A similar question for $D_{Jacobi} = Diag(\frac{1}{h_{11}}, \dots \frac{1}{h_{nn}})$, not well answered so far
- We still lack theoretical understanding of D_{Adam}

Q1 (numerical-version): what Hessian structure does Neural Nets have?
Is D_{Adam} effective on this class of H?

[1] Worst-case Complexity of Cyclic Coordinate Descent, Sun and Ye, 2017, Mathematical Programming

Numerical results on random H and $\boldsymbol{D_{Adam}}$



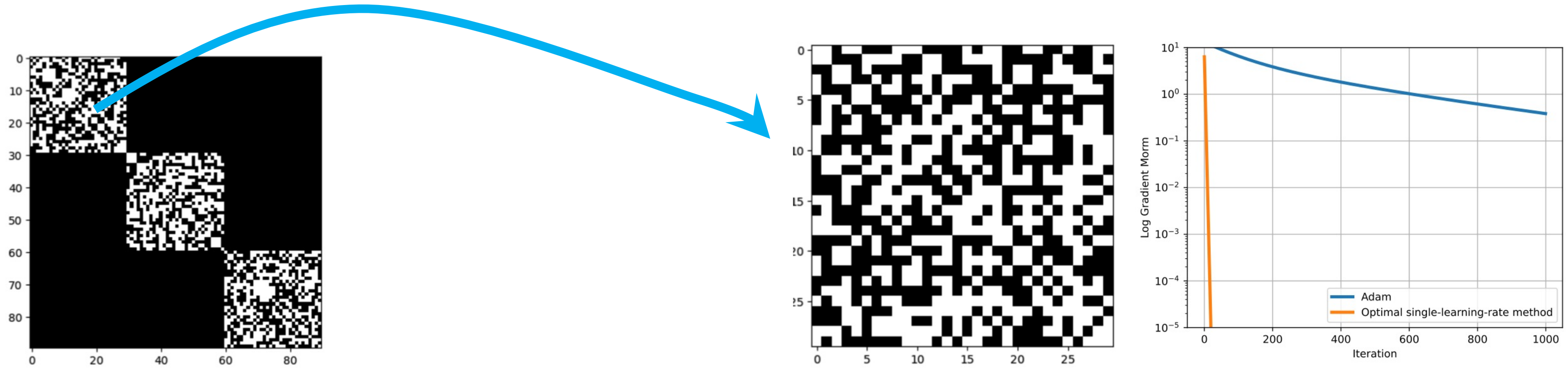
$$\tau = \frac{\sum_i |H_{b,i,i}|}{\sum_{i,j} |H_{b,i,j}|}, \quad r = \frac{\kappa(D_{\text{Adam}} H_b)}{\kappa(H_b)},$$

$\tau \to 1$: $H \to$ pure diagonal

Observation: D_{Adam} **is NOT effective when** H **is dense**

How effective is D_{Adam} on dense block?

$$\min_{x} \frac{1}{2} x^T H x,$$ where $H = Diag\,(H_1, H_2, H_3), H_i$ are random PD (dense) matrices



(a) Hessian matrix (c) Dense sub-block (d) Sub-problem loss

- **Another Case: Dense block.**
  We take the 1st sub-block H_1 in (a) and use it for a new problem (c)

-- Figure (c): **GD with (a different) optimal lr** outperforms **Adam**,
even though Adam uses more lr!
-- This means: D_{Adam} is NOT so effective on H_1!

Our Explanation: Why Transformer Needs Adam

Observation 1:
Adam is good for block-Hessian.

Previous Part 1

Observation 2:
Adam is bad for dense Hessian

Previous slide

Together reveals:

Implication:
Adam mainly handles "cross-block" challenge,
NOT "within-block" challenge.

Further Implication:
We may replace Adam's coordinate-wise lr by block-wise lr

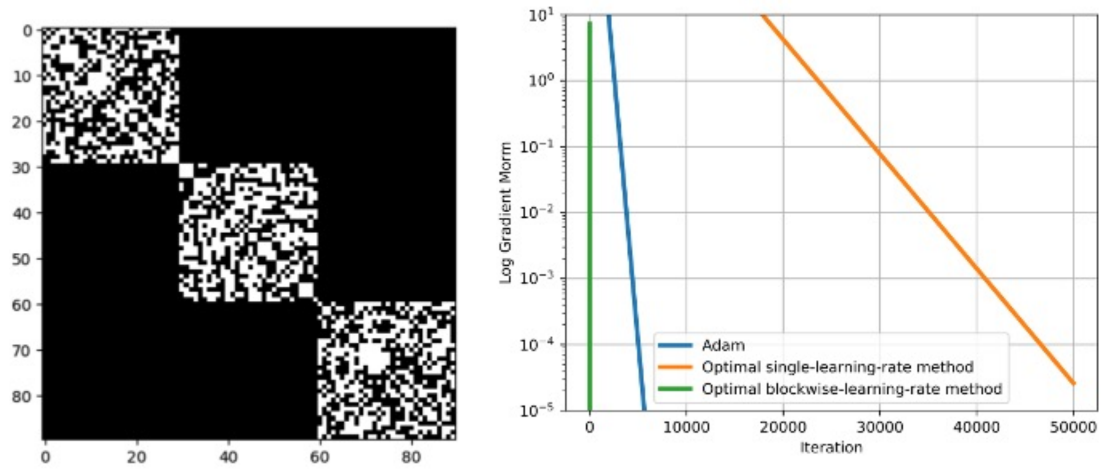
How effective is D_{Adam} on this block-diagonal Hessian?

- We explore it numerically on quadratic functions:

$$\min_x \frac{1}{2} x^T H x, \text{ where } H = Diag\,(H_1, H_2, H_3), H_i \text{ are random}$$

Algorithm (block-wise optimal lr method):
-- Collect optimal lr's for each block
-- Apply them to **"blockwise" version of GD,**

Observation: Figure (b): **blockwise GD** outperforms Adam with only 3 learning rates!



(a) Hessian matrix (b) Total loss

Using Fewer Learning Rates?

- More lrs do not neccessarily bring extra gain
  (This reveals a drawback of design in $Adam$)

- For each block, a single **but good** lr can outperform $Adam$

- But how to find these good lrs without grid-search?

Adam-mini (Framework)

Algorithm 1 Adam-mini (General form)

Input weight-decay coefficient λ and current step t
Partition params into `param_blocks` by **Principle 1** in Section 2.3
for `param` in `param_blocks` **do**
  $\mathtt{g} = \mathtt{param.grad}$
  $\mathtt{param} = \mathtt{param} - \eta_t * \lambda * \mathtt{param}$
  $\mathtt{m} = (1-\beta_1) * \mathtt{g} + \beta_1 * \mathtt{m}$
  $\hat{\mathtt{m}} = \frac{\mathtt{m}}{1-\beta_1^t}$
  $\mathtt{v} = (1-\beta_2) * \mathtt{mean}(\mathtt{g} \odot \mathtt{g}) + \beta_2 * \mathtt{v}$
  $\hat{\mathtt{v}} = \frac{\mathtt{v}}{1-\beta_2^t}$
  $\mathtt{param} = \mathtt{param} - \eta_t * \frac{\hat{\mathtt{m}}}{\sqrt{\hat{\mathtt{v}}}+\epsilon}$
end for

- \- **Step 1**: partition the gradient g into B sub-vectors according to the dense Hessian sub-block $g_b, b = [B]$

  \- **Step 2:** for each g_b, calculate:

$$\boxed{v_b = (1-\beta_2) * \mathbf{mean}(g_b \circ g_b) + \beta_2 * v_b, \quad b = 1, \cdots B}$$

  \- **Step 3:** then use $\frac{\eta}{\sqrt{v_b}}$ as the lr for the parameters associated with g_b

- **Step 1 is important, and will be discussed later.**

Adam-mini: an illustration

- Illustration: For a problem with 5 parameters, $w \leftarrow w - \eta\, u \circ m$

  Adam: u = $\left(\frac{1}{\sqrt{v_1}}, \frac{1}{\sqrt{v_2}}, \frac{1}{\sqrt{v_3}}, \frac{1}{\sqrt{v_4}}, \frac{1}{\sqrt{v_5}}\right)$

  Adam-mini: if block partition is (1,2,3); (4,5), then

  $$u = \left(\frac{1}{\sqrt{(v_1+v_2+v_3)/3}}, \frac{1}{\sqrt{(v_1+v_2+v_3)/3}}, \frac{1}{\sqrt{(v_1+v_2+v_3)/3}}, \frac{1}{\sqrt{(v_4+v_5)/2}}, \frac{1}{\sqrt{(v_4+v_5)/2}}\right)$$

- Benefits: reduce # learning rates: from # parameters to # blocks

- For LLMs, we will show that this would free ≥ 99% elements in v

- **Remark:** Cheap way to find “good lrs” , but might not be optimal

- **Remaining question:** How to partition parameters for a given problem, e.g., Transformers?

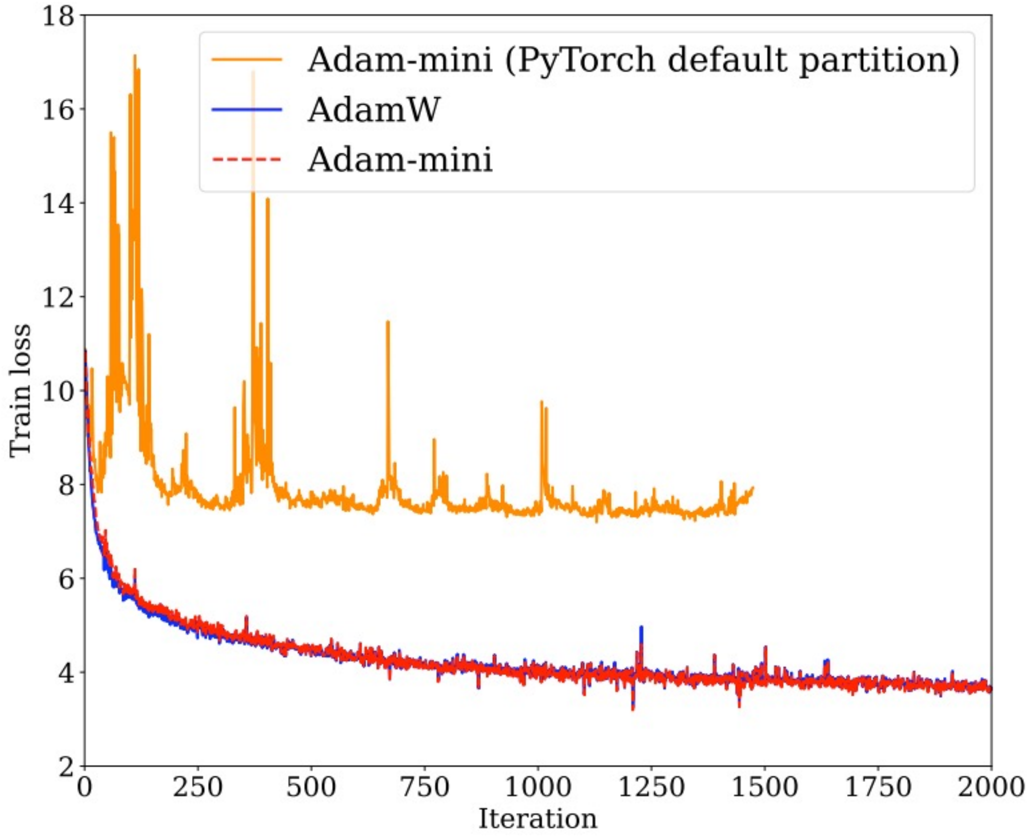
Parameter Partition: Failed Attempt

Failed Attempt:

- Default PyTorch partition strategy **(layer-by-layer)** is a naive candidate

- Unfortunately, this default strategy over-sim

- Observe **training instability** on 1B models

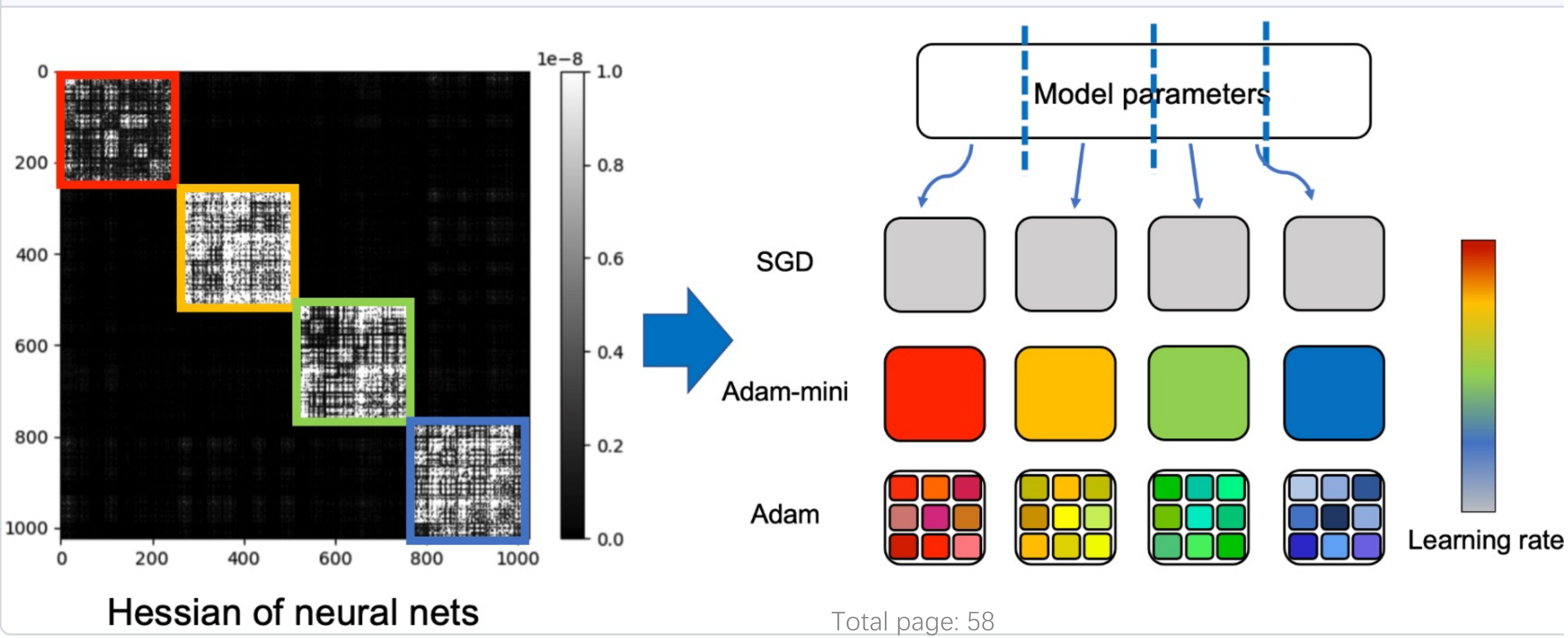


Why?

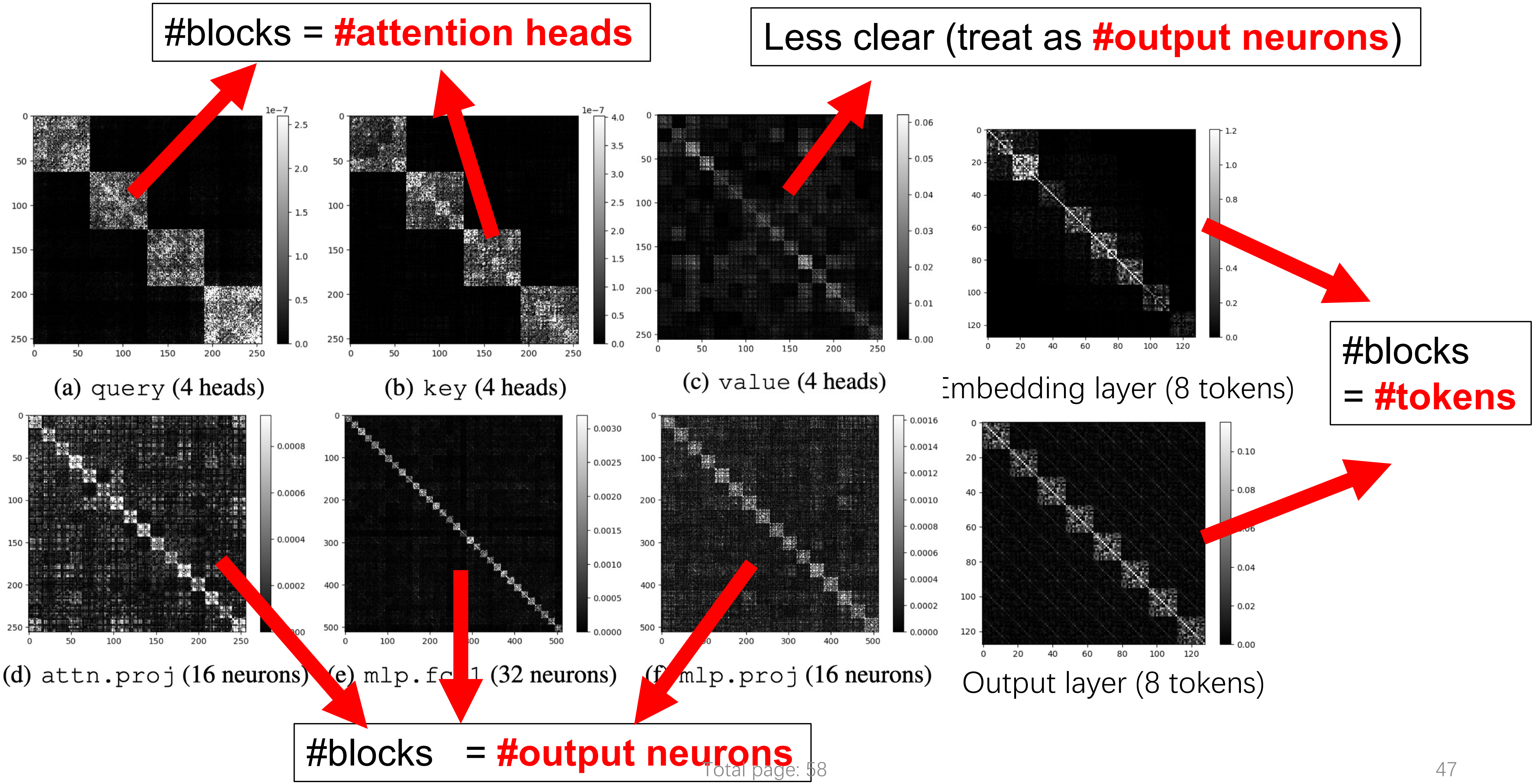
We suspect the default PyTorch partition did not fully capture the **Hessian structure**

Partition Principle

Partition Principle:
Partition parameters into blocks s.t. each block is associated with a dense sub-block in Hessian.



Applying Principle to Transformer **(non-equal sized blocks)**



(a) `query` (4 heads) (b) `key` (4 heads) (c) `value` (4 heads) Embedding layer (8 tokens)

(d) `attn.proj` (16 neurons) (e) `mlp.fc1` (32 neurons) (f) `mlp.proj` (16 neurons) Output layer (8 tokens)

Side story: ”Bug”

Remark: 和初版 (in May, 2024) 不同!

对 embedding & output layer:

初版

每个坐标分配一个 lr

和理想划分策略不一致!

NOT Principled enough!

本版

根据 Hessian blocks划分

和理想划分策略完全一致!

Due to “bug”:

In Pytorch, embedding layer is W^T, extra “transpose”!

Complete form of Adam-mini

Partition for Transformer

Adam-mini given the partition rule

Algorithm 1 Adam-mini in Pytorch style

```
Input weight-decay coefficient λ and current step t
Choose param_blocks from Algorithm 2 or 3
for param in param_blocks do
    g = param.grad
    param = param - η_t * λ * param
    m = (1 − β_1) * g + β_1 * m
    m̂ = m / (1 − β_1^t)
    v = (1 − β_2) * mean(g ⊙ g) + β_2 * v
    v̂ = v / (1 − β_2^t)
    param = param - η_t * m̂ / (√v̂ + ε)
end for
```

Partition for simple models (CNNs, GNNs, Diffusions)

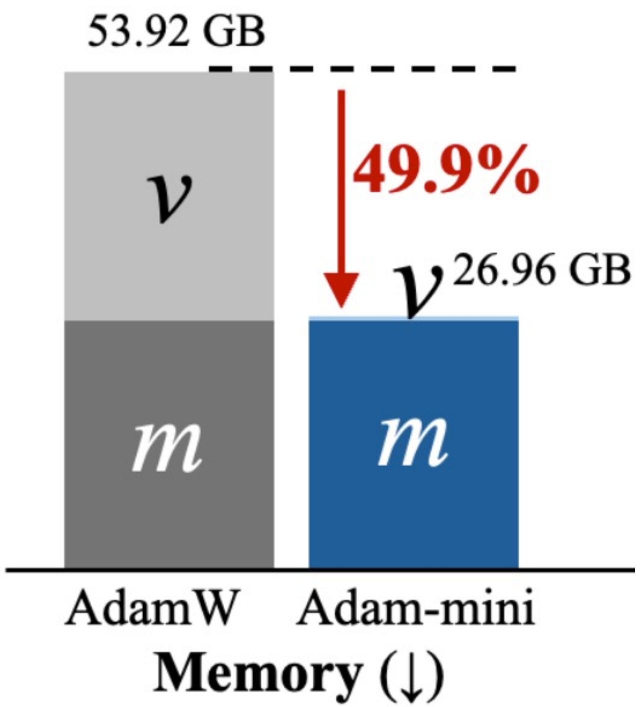
orithm 2 Partition for non-Transformers

```
param_blocks = {}
for name, param in parameters do
    param_blocks[name]=param
end for
return param_blocks
```

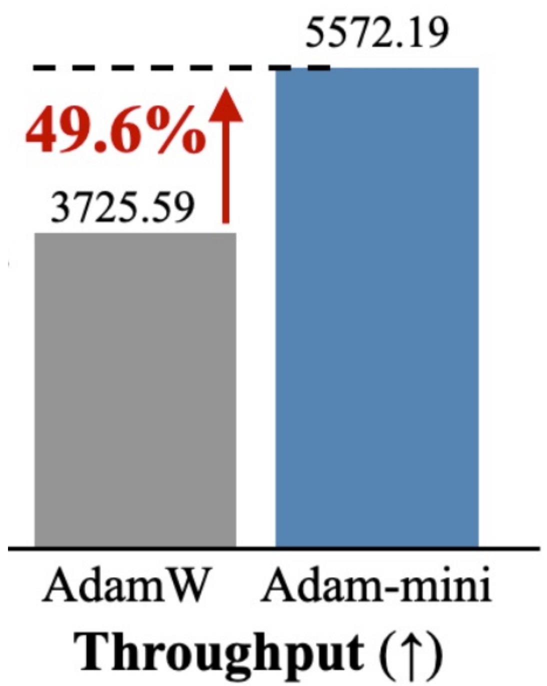
Algorithm 3 Partition for Transformers

```
param_blocks = {}
for name, param in parameters do
    if 'embed' or 'output' in name then
        Partition param by tokens
        for i = 0...tokens-1 do
            param_blocks[name+i]=param[i]
        end for
    else if 'query' or 'key' in name then
        Partition param by heads
        for i = 0...heads-1 do
            param_blocks[name+i]=param[i]
        end for
    else if 'value', 'attn.proj', or 'mlp' in name then
        Partition param by output neurons
        for i = 0...output_neurons-1 do
            param_blocks[name+i]=param[i]
        end for
    else
        param_blocks[name]=param
    end if
end for
return param_blocks
```

Memory cut down & Throughput enhancement



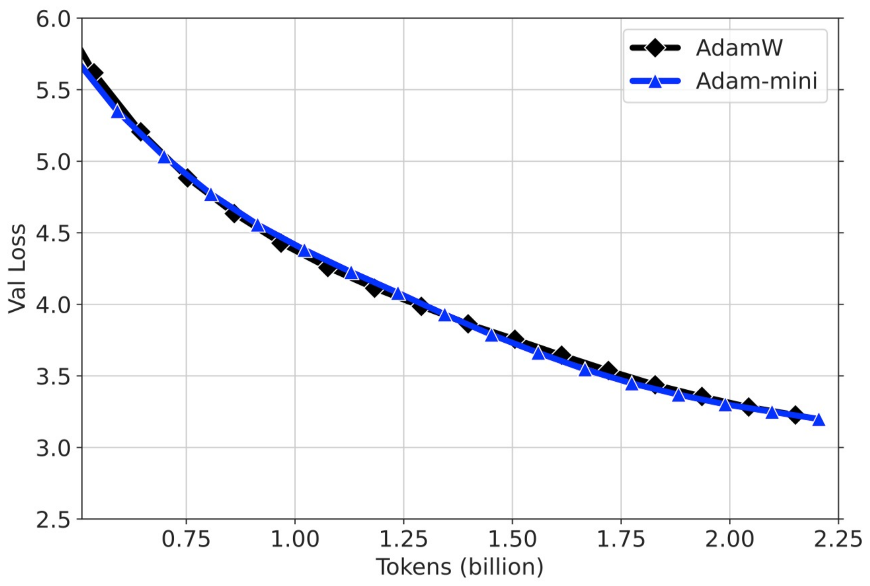
Saves 50% memory of Adam



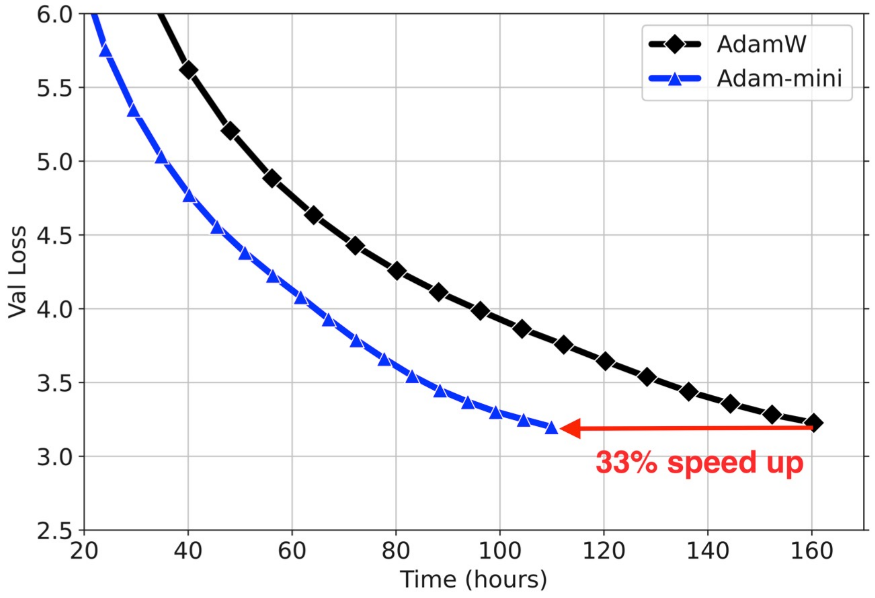
Can increase about 50% throughput of Adam
(# processed data per second)

Why? Reduce communication + larger batch size per GPU

Llama2-7B: pre-training



(b) Loss v.s. iteration



(c) Loss v.s. time

- ➢ Same loss curve as Adam
- ➢ **33% less time** to process the same # tokens (tested on 2x A800-80GB GPUs)

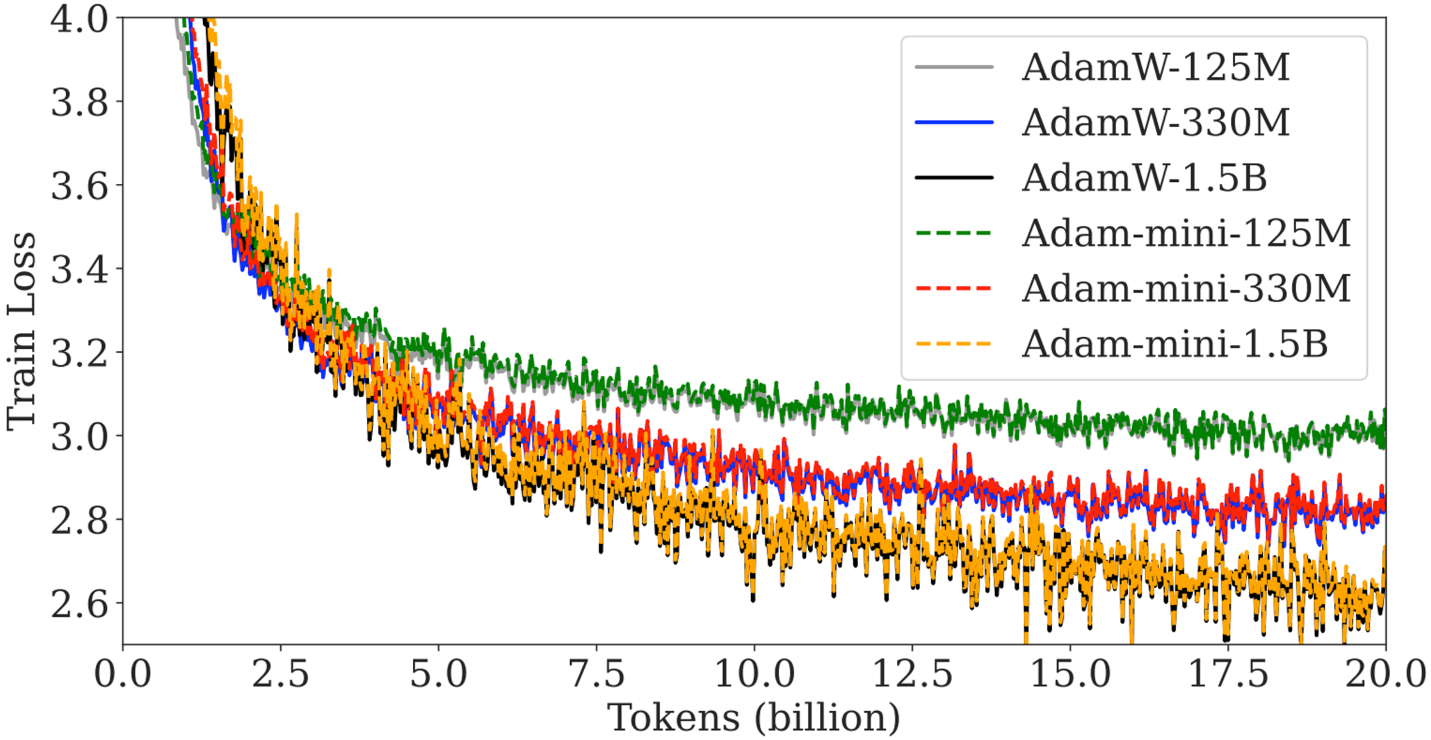
Llama2-7B: pre-training

Optimizer	# Tokens (B)	GPU hours (h) (↓)
AdamW	1	74.56
Adam-mini	1	49.85 (↓ **33.1%**)
AdamW	70	5219.16
Adam-mini	70	3489.55 (↓ **33.1%**)
AdamW	140	10438.32
Adam-mini	140	6979.10 (↓ **33.1%**)

33% less time to process the same # tokens

(tested on 2x A800-80GB GPUs)

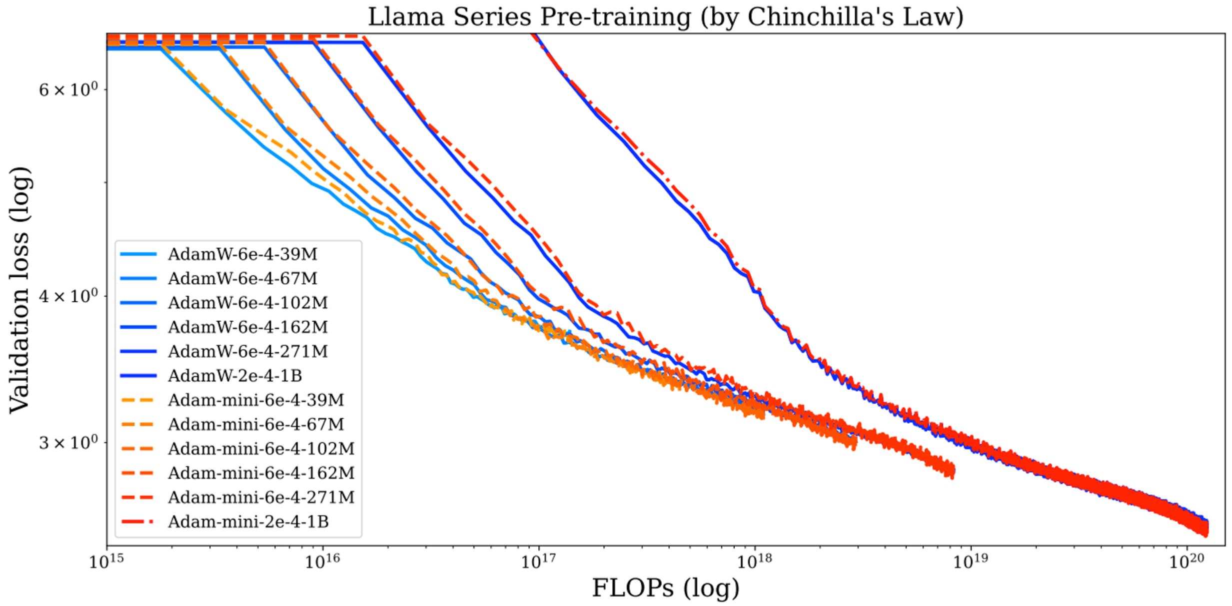
GPT-2 pre-train (125M, 330M, 1.5B)



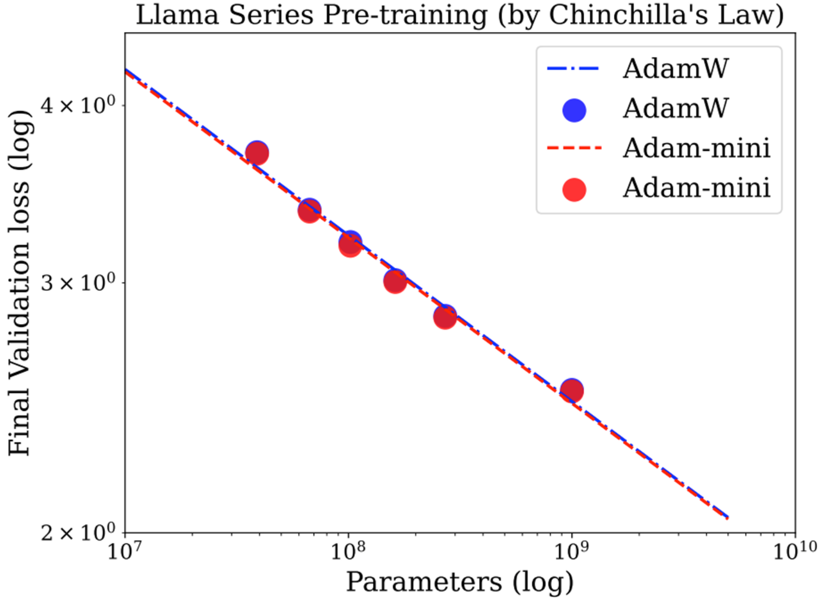
(a) GPT-2 series

The curves of Adam-mini closely resemble the curves of AdamW

Scaling laws of Adam-mini from 39M to 1B



(a) Scaling laws in terms of compute

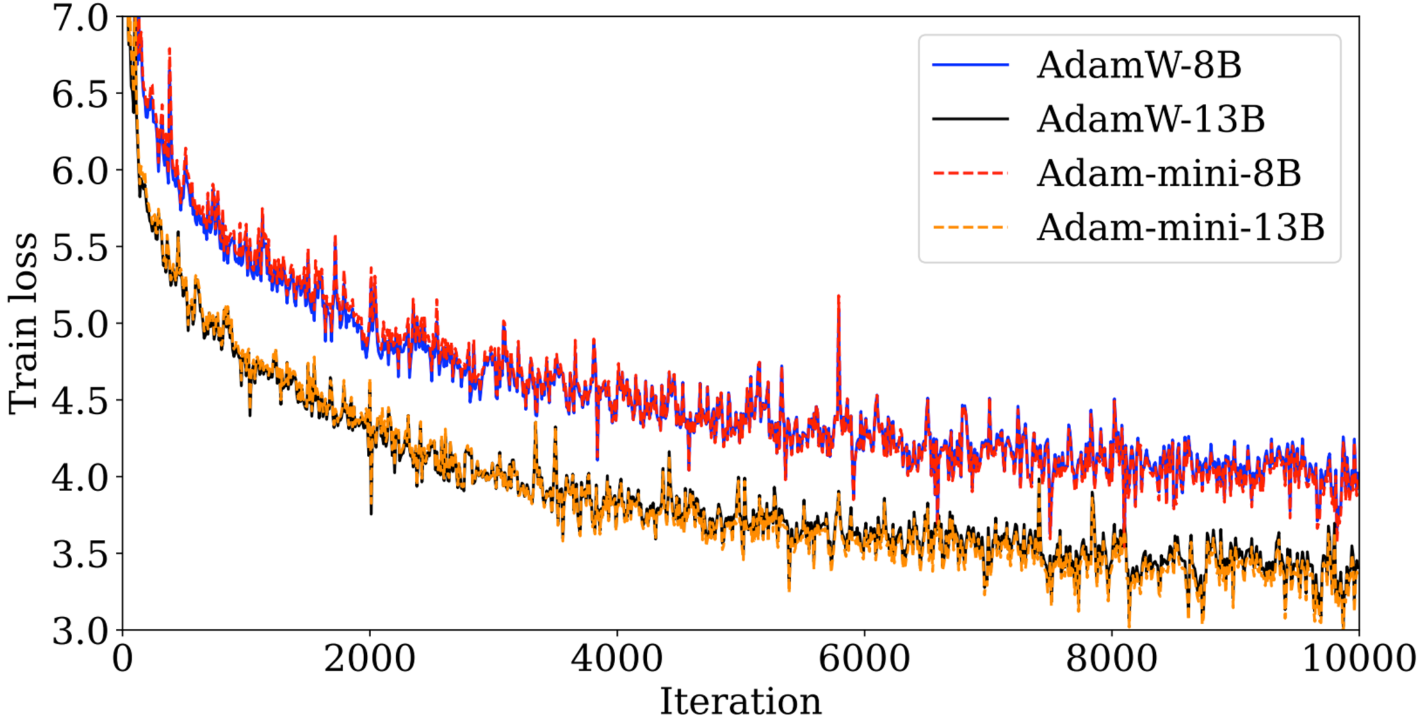


(b) Scaling laws in terms of parameters

- We train Llama series (from 39M to 1B) for complete pre-training runs (“complete” under the definition of Chinchila’s law: # data= 20 * # parameters)
- For all models, Adam-mini performs similarly to AdamW

This serves as an evidence that Adam-mini can work on larger models, e.g., 30B, 100B (if the scaling law holds)

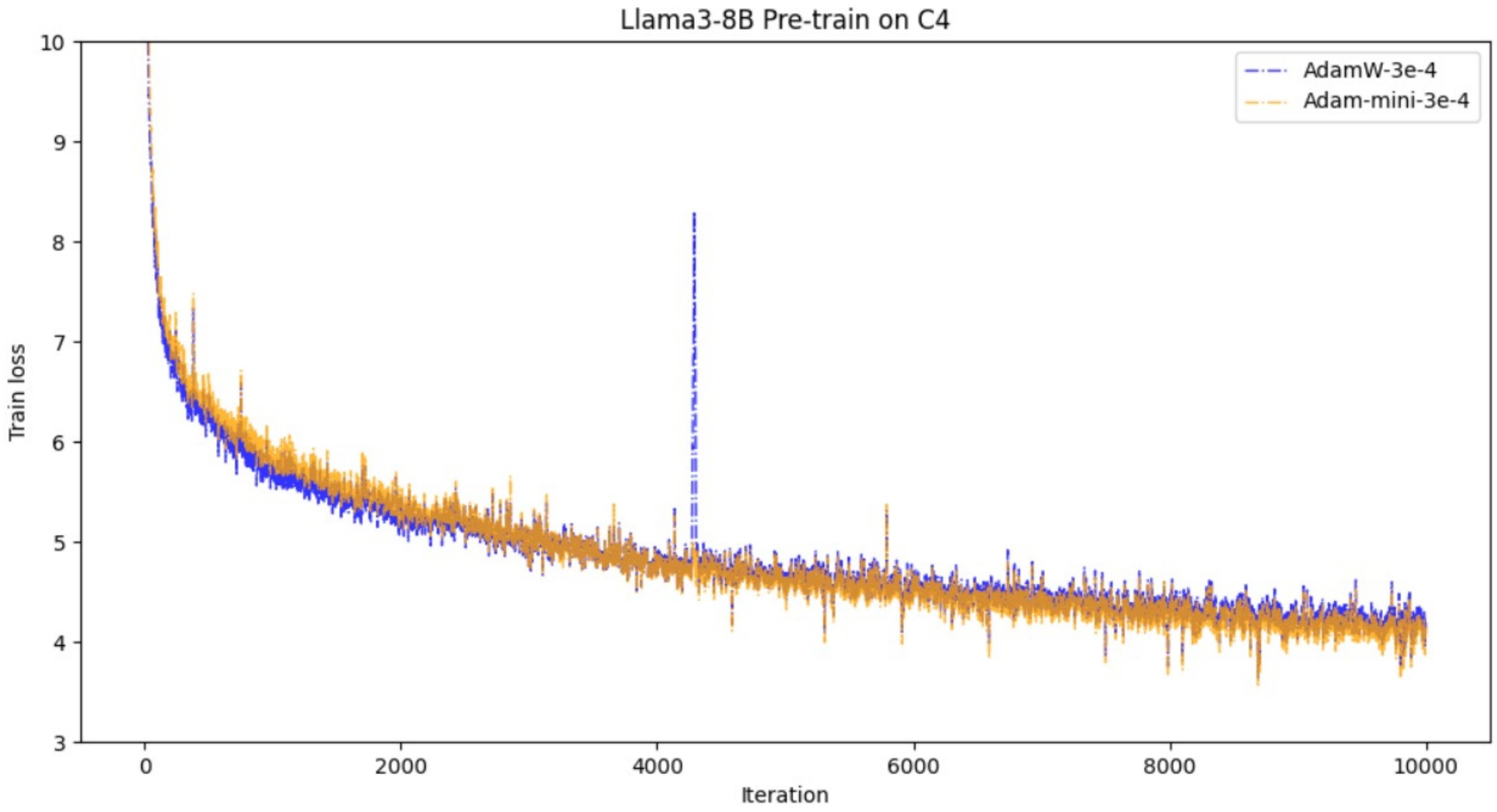
Llama 3-8B and Llama 2-13B



(b) Llama 3-8B and Llama 2-13B

The curves of Adam-mini closely resemble the curves of AdamW

Independent verifier from PyTorch team (Llama3-8B)



lessw2020 commented 5 days ago • edited Author

Hi @zyushun - congrats! With a slight bump in lr (3e-4 mini vs 1e-4 adamw) and mini shows very similar curves but with overall outperformance! This is imo a very big accomplishment as most optimizers can't do this (meet / exceed adamw) at 8B scale and esp not while reducing memory so significantly.

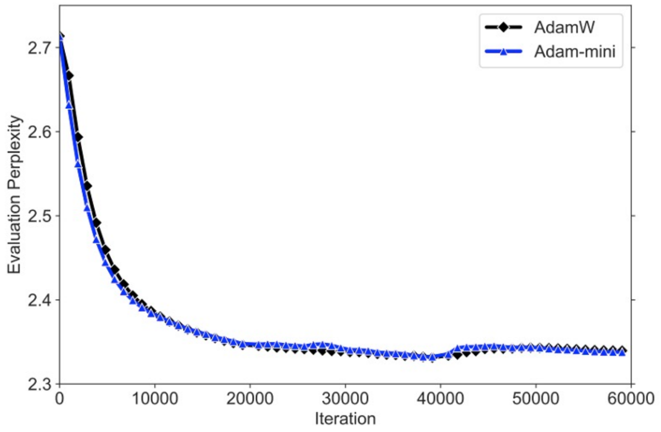
Highlight:

"This is imo a very big accomplishment as most optimizers can't do this (meet / exceed adamw) at 8B

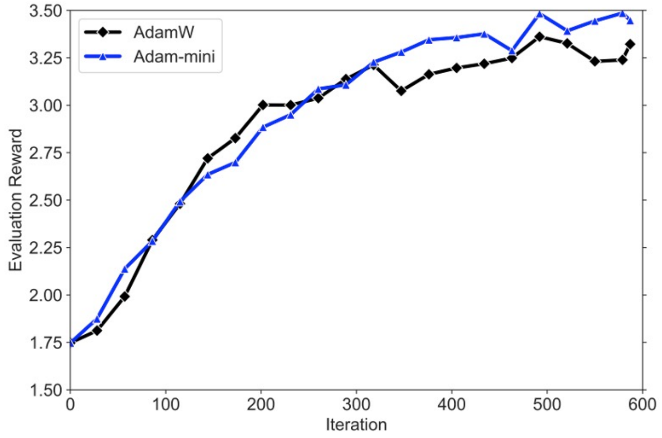
… and especially not while reducing memory so significantly"

Llama2-7B: SFT and RLHF

Finetuning tasks for Llama2-7B pre-trained model (released by Meta).



(b) SFT



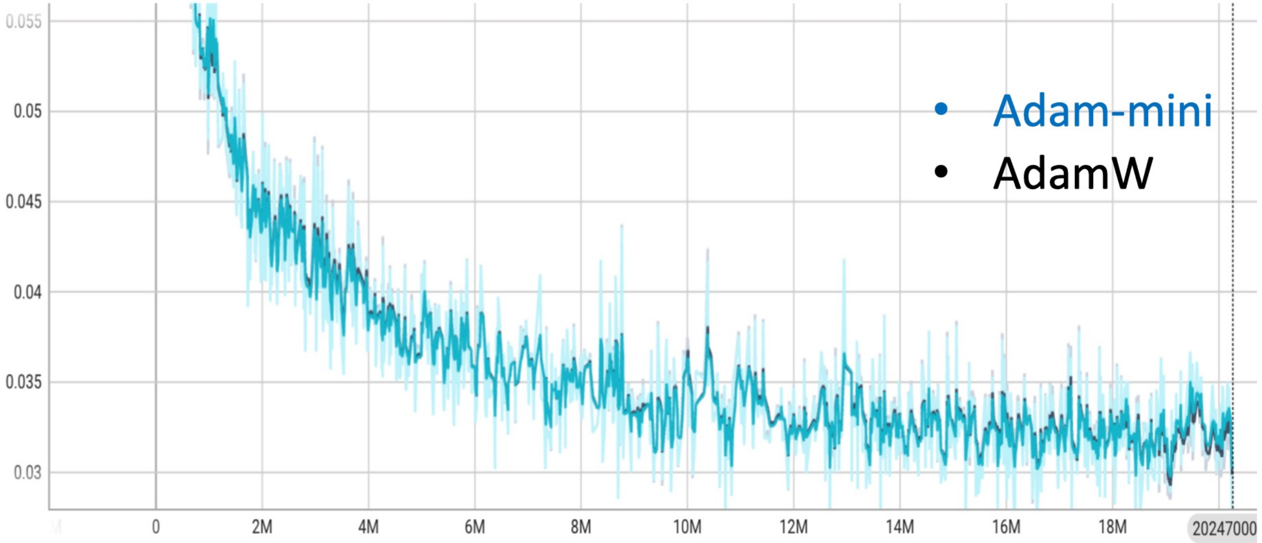
(c) RLHF

Table 3: Averaged GPT-4 evaluation score (↑) of SFT and RLHF on the MT-Bench.

	SFT (LoRA)		SFT		RLHF	
	AdamW	Adam-mini	AdamW	Adam-mini	AdamW	Adam-mini
MT-Bench	4.23	**4.41**	5.37	**5.40**	5.54	**5.68**

Adam-mini performs slightly better than **AdamW**, with **50%** less memory

Diffusion models

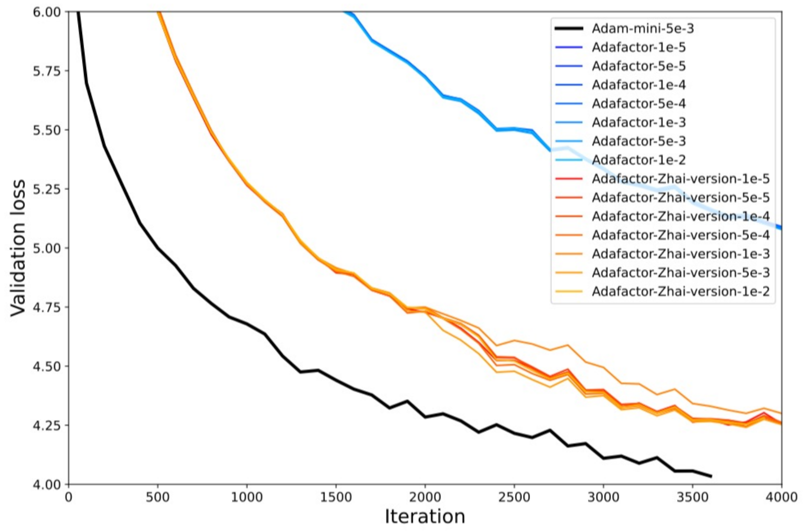


Diffusion model training:
Loss vs. iteration

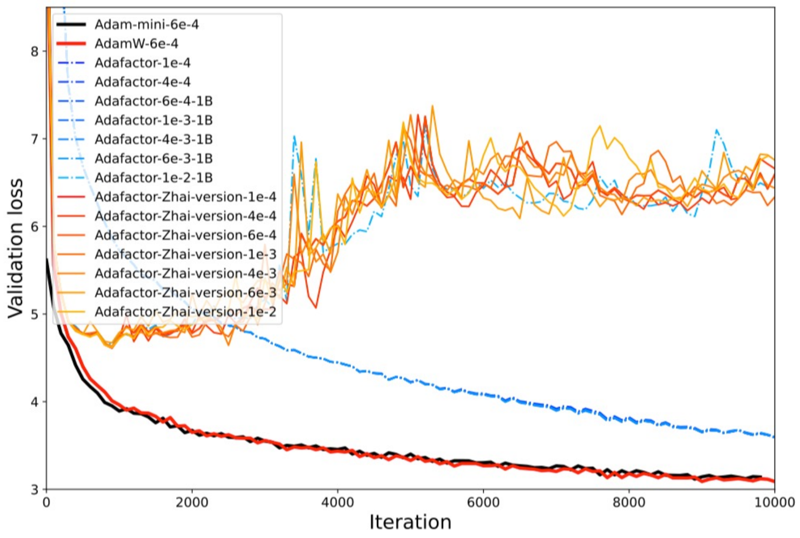
Independent verifier on twitter:
SDXL: Stable Diffusion XL **(sized 2.6B)**
One of the SOTA Diffusion model

Adam-mini performs slightly better than **AdamW**, with **50%** less memory

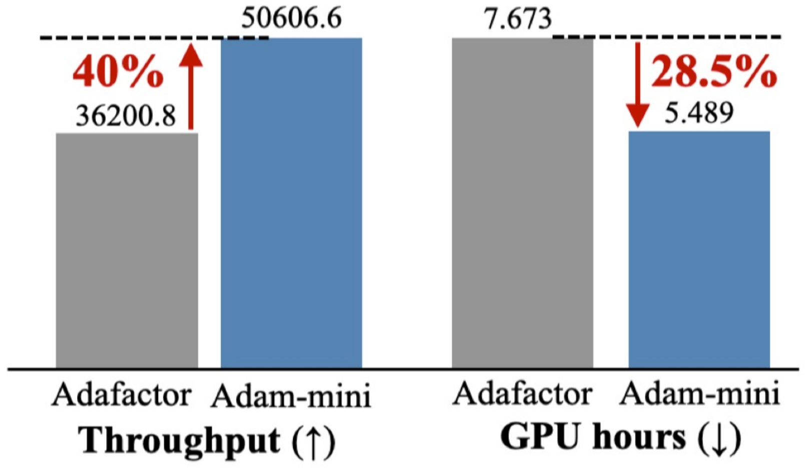
Comparison to Adafactor (variants)



(a) Llama 2-20M



(b) Llama 2-1B



(c) Throughput comparison

We did NOT find the good hyperparams for Adafactor to work
(**9 hyperparams!** Hard to find the correct combination)
Further, Adafactor **has higher latency** (due to more matrix produts)

Summary

- **Part I:** We provide an explanation why Transformer needs Adam, not SGD
  - Hetereogeneity of block-Hessian-spectra is one reason

- **Part II:** We propose a 50%-memory-saving variant of Adam: Adam-mini
  - **Two features**: Principled block partition; block-mean of v

How to Use? Just 1-line code change

```
pip install adam-mini
```

```
from adam_mini import Adam_mini
```

```
optimizer = Adam_mini(
            named_parameters = model.named_parameters(),
            lr = lr,
            betas = (beta1,beta2),
            eps = eps,
            weight_decay = weight_decay,
            model_sharding = True,
            dim = model_config.dim,
            n_heads = model_config.n_heads,
            n_kv_heads = model_config.n_kv_heads,
            )
```

Same values as AdamW!

Your model config

We support: DDP, FSDP, Deepspeed, Torchtitan, HF trainer 🤗

👆 Code for Adam-mini
Currently:
-- 300+ stars

If you like Adam, Adam-mini is a no-brainer switch!

Mainly based on:

- **Zhang**, Chen, Ding, Li, Sun, & Luo, Why Transformers Need Adam: A Hessian Perspective, NeurIPS 24
- **Zhang***, Chen*, Li, Ding, Chen, Kingma, Ye, Luo & Sun, Adam-mini: Use Fewer Learning Rate To Gain More, preprint.

Diederik P. Kingma

Yinyu Ye

Ruoyu Sun

Zhi-Quan Luo

Thanks to all the collaborators!