Preconditioning Layer for Improving LLM Training

孙若愚　Ruoyu Sun

The Chinese University of Hong Kong, Shenzhen

IASM 5-day workshop

Sep 2026

深圳河套學院 Shenzhen Loop Area Institute

香港中文大學（深圳） The Chinese University of Hong Kong, Shenzhen

深圳市大数据研究院 Shenzhen Research Institute of Big Data

Background – Normalization in LLM Training

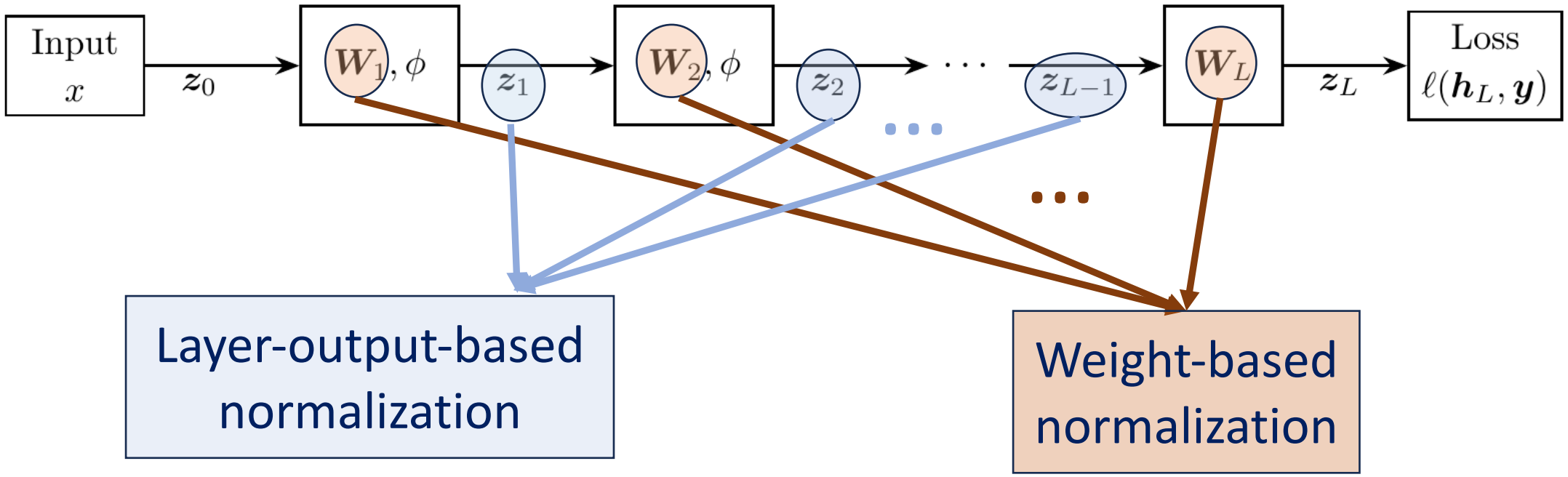
- Training large-scale Transformers often requires a range of techniques to stabilize training & improve performance.

- **Normalization layers**, which control the **scale and/or mean** of intermediate signals, are widely used in LLM training:
  - LayerNorm and its variant RMSNorm
  - QK-Norm
  - …

Remark: They are built-in component of neural-nets & appear in backprop, so they are "normalization layers"

Two Types of Normalization Layers

- Traditionally, there are two types of normalization layers



Layer-output-based normalization

Layer normalization (transformers)
Batch normalization (image classification)
Group normalization
…

Weight-based normalization

Weight normalization
Spectral normalization (popular in GAN)
…

Background – Weight-Based Normalization

- **Weight control**: less explored in large-scale LLM training.

- **Very recently,** several studies on **weight control** for LLM training, e.g.:
  - **Thinking Machines Lab – "Modular Manifolds"**:
    constrain weight matrices to submanifolds (e.g. **Stiefel manifold**).
  - **Deepseek's mHC**: control weights to be **doubly stochastic matrix**.

- However, a clear theoretical principle for design is missing.

Gap Between Theory & Practice

Optimization theory for NNs:

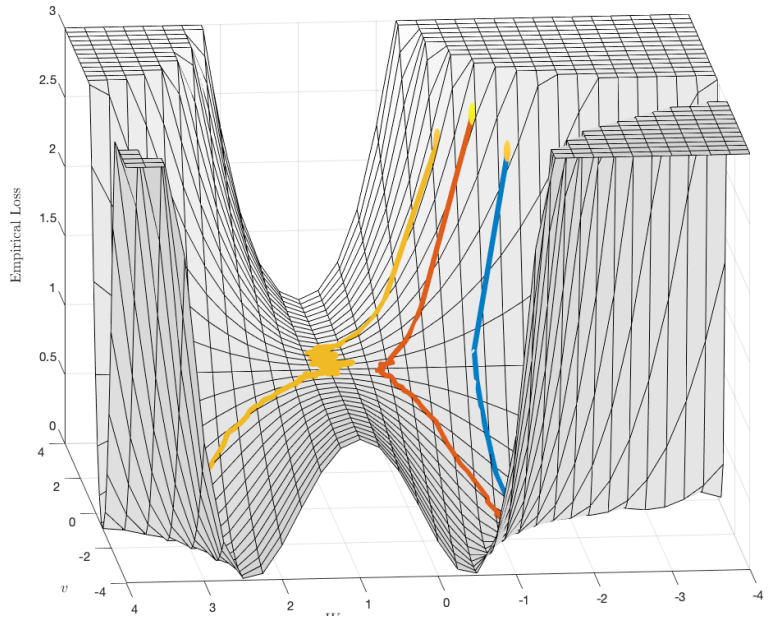
"explain" why GD converges to global optima

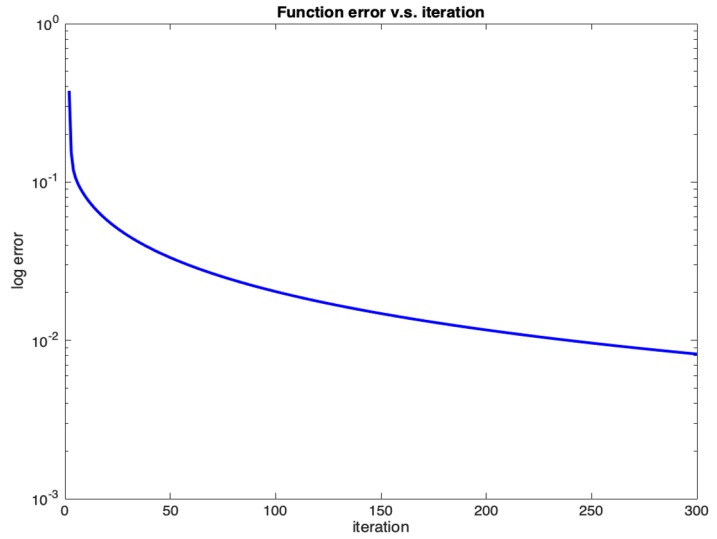
e.g. (Jacot et al. 2018;
Du et al. 2018; Allen-Zhu et al. 2019, etc.)

Algorithm Design:

"Design" algorithm for faster convergence to local optima

e.g. Adam, muon, etc.





Our Contributions

Combining theory & practice：

"design" algorithm， that relates to global optima convergence

Theory

When all weight matrices have bounded condition numbers,
GD has geometric convergence to the **global minimizer** in deep linear networks.

Method

Propose a built-in-layer method: **Preconditioning layer** (**PC layer**).

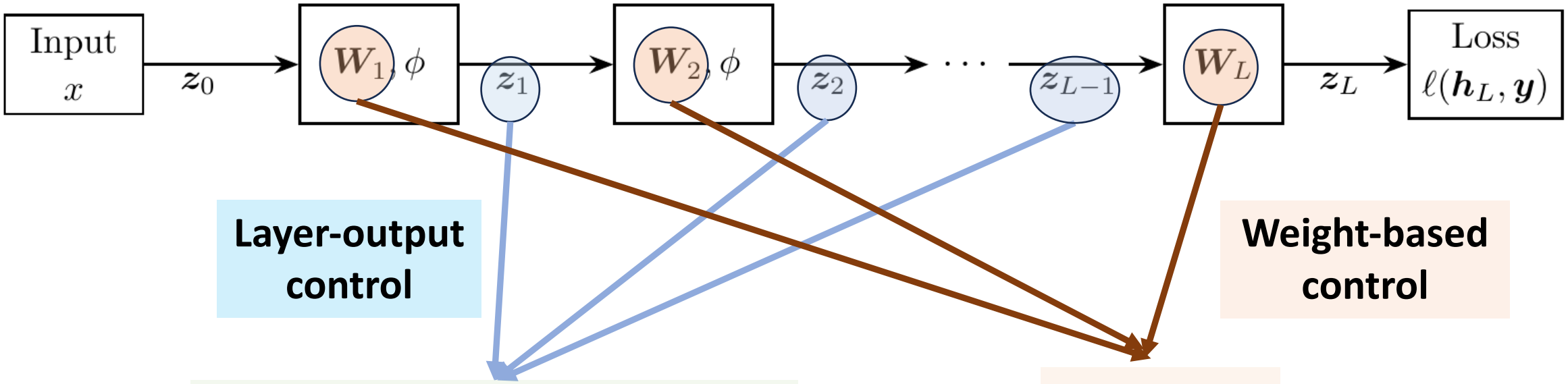
-- uses matrix polynomials to control weight spectrum throughout training.

Performance

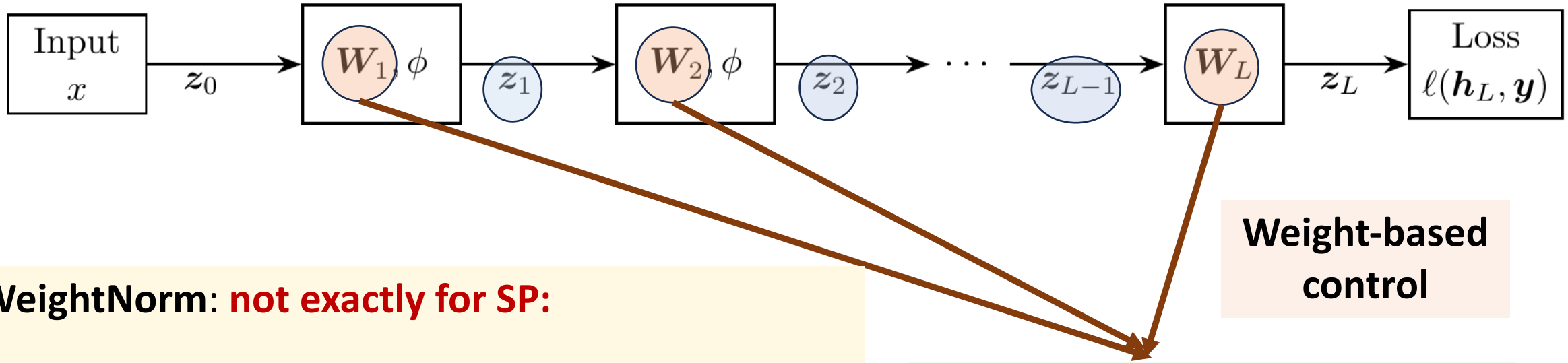
On 1B-Transformer pre-training with AdamW, PC layer achieves 2x more token-efficient and improves downstream performance.

Contents

Motivation: Controlling Weights for What?



One Perspective: Signal Propagation



Weight-based control

WeightNorm: **not exactly for SP:**

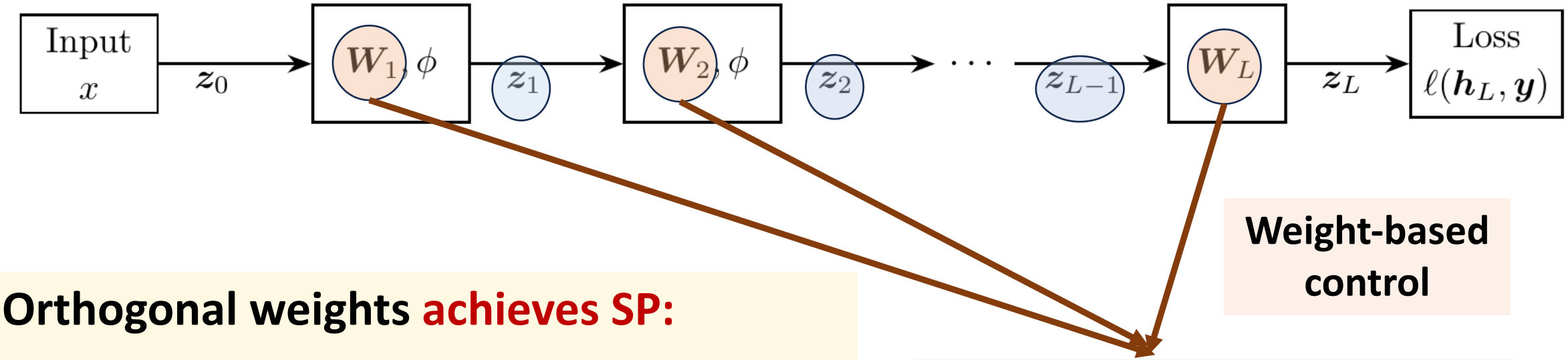
$$y = Wx$$

If ||x|| ≈ 1 and all rows/columns of W have norm 1, ||y|| is **not guaranteed** to be ≈ 1.

"Signal Propagation" (SP)

Hope $\| z_j \| \approx 1$

One Perspective: Signal Propagation



Weight-based control

Orthogonal weights achieves SP:

$$y = Wx$$

If ‖x‖ ≈ 1 and **W is an orthogonal matrix**,
Then **‖y‖ ≈ 1**.

"Signal Propagation" (SP)
Hope $\| z_j \| \approx 1$

Limitation of Orthogonal Weights

Limitation: Representation power of NN with orthogonal weights is weak.

Claim: Suppose $f_\theta(x) = W_L \dots W_1 x$ where W_l are orthogonal, then it cannot represent $f^*(x) = 2x$, i.e.,

$$2x \notin \{W_L \dots W_1 x \mid W_l \in R^{d\times d}, W_l^T W_l = I, l = 1, \dots, L\}$$

Adding scalars?

$$\{Wx \mid W \in \mathbb{R}^{d\times d}\} \neq \{\alpha W_L \dots W_1 x \mid W_l \in \mathbb{R}^{d\times d}, W_l^T W_l = I, l = 1, \dots, L\}$$

e.g. $$\begin{bmatrix} 2 & \\ & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \notin \{\alpha W_L \dots W_1 x \mid W_l \in \mathbb{R}^{2\times 2}, W_l^\top W_l = I, l = 1, \dots, L\}$$

Reducing Condition Number: SP Perspective

$$y = Wx$$

$$\| y \| = \| Wx \| \in (\sigma_{min} \| x \|, \ \sigma_{max} \| x \|)$$

Design principle 1 (signal propagation perspective)
Reduce condition number $\kappa = \sigma_{max}/\sigma_{min}$.

Design principle 2 (representation perspective)
Let $\kappa = \sigma_{max}/\sigma_{min}$ be O(1), but not exactly 1.

Contents

Preconditioning: Brief History

Timeline of Classical Preconditioners

Year	Method	Formulation (M or M^{-1})	Characteristics
1845	**Jacobi**	$M = \text{diag}(A)$	**Diagonal.** Trivial inverse ($1/a_{ii}$). Fast but weak.
1950s	**Polynomial**	$M^{-1} \approx \sum_{k=0}^{p} (I - A)^k$	**MVPs.** No inversion needed. GPU friendly.
1970s	**ILU**	$M = \tilde{L}\tilde{U} \approx A$	**Triangular.** Implicit inverse via back-sub. Serial.
1990s	**SPAI**	$M^{-1} = \arg\min \|AP - I\|$	**Optimization.** Explicit sparse inverse. Fully parallel.

Requirement in NN: Need to be differentiable!

Polynomial Preconditioner

Polynomial preconditioner

A polynomial p such that $p(Q)Q$ has a small condition number,

if Q is a squared matrix.

See, e.g.,

Olin G Johnson, Charles A Micchelli, and George Paul. Polynomial preconditioners for conjugate gradient calculations. *SIAM Journal on Numerical Analysis*, 20(2):362–376, 1983.

Polynomial Precondition **on Rectangular Matrix**

Implementation on neural network: **handle rectangle matrix**

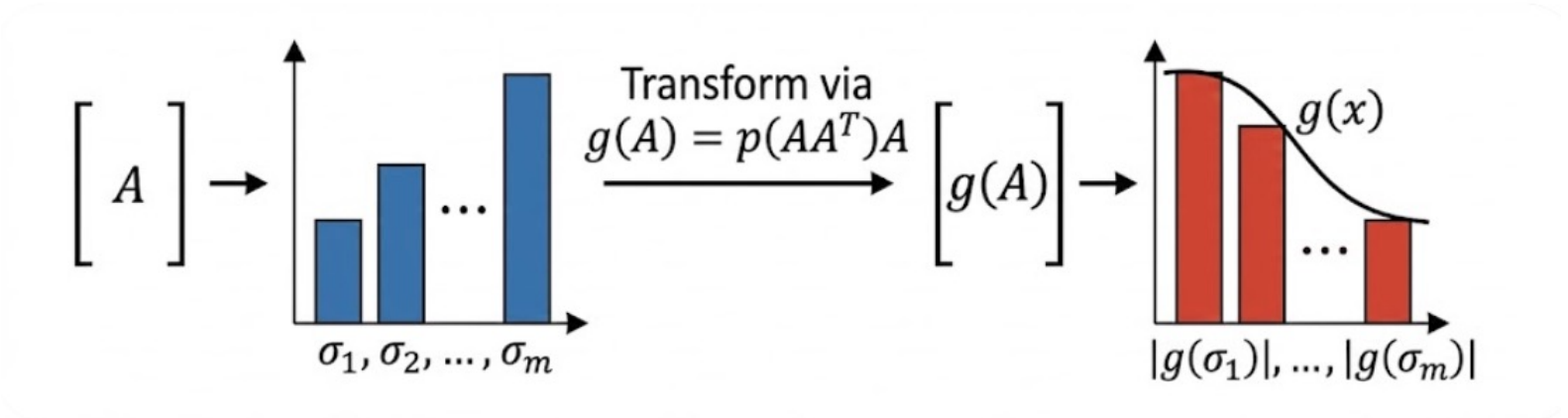
- Use $p(AA^T)A$ as the preconditioned matrix, if A is a rectangle matrix

Basic fact (Linear Algebra fact)

Suppose $A \in \mathbb{R}^{n\times m}$ has singular values $\sigma_1 \leq \cdots \leq \sigma_m$.

Suppose $g(x) = p(x^2)x$ where p is a polynomial.

Then the singular values of $g(A) = p(AA^T)A$ are $|g(\sigma_1)|, \ldots, |g(\sigma_m)|$.



Target of PC

Goal 1 find polynomial p **s.t.** $g(A) = p(AA^T)A$

i) well-conditioned;

ii) κ not exactly 1

Goal 2 find polynomial p **s.t.** $g(x) = p(x^2)x$ maps $[\sigma_1, \sigma_m]$ to $[\epsilon, 1]$ for some ϵ.

Remark: need to rescale A s.t. $\sigma_{max}(A) \sim 1$.

Target of PC

Goal 2 find polynomial p **s.t** $g(x) = p(x^2)x$ maps $[\sigma_1, \sigma_m]$ to $[\epsilon, 1]$ for some ϵ.

$g(x)$ is spectrum mapping, should

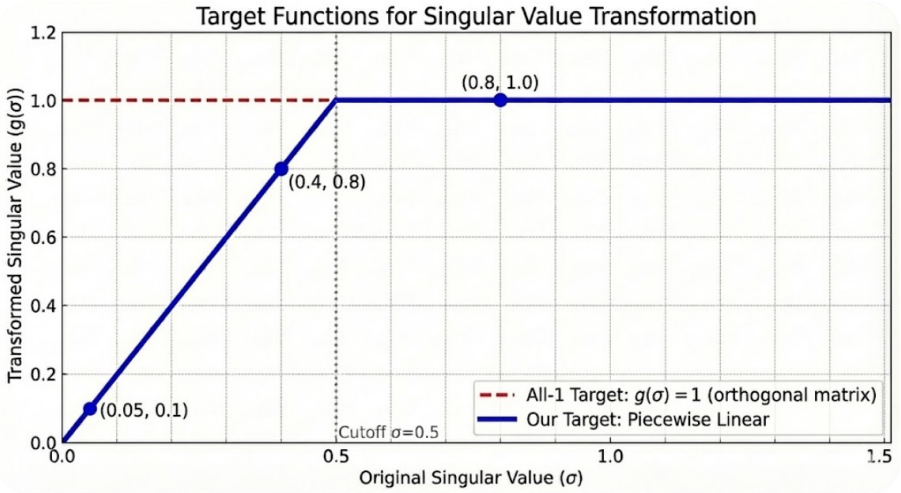
"ill-conditioned" spectrum

well conditioned spectrum

0.02, 0.1, 0.8, 0.9, 1

1, 1, 1, 1, 1

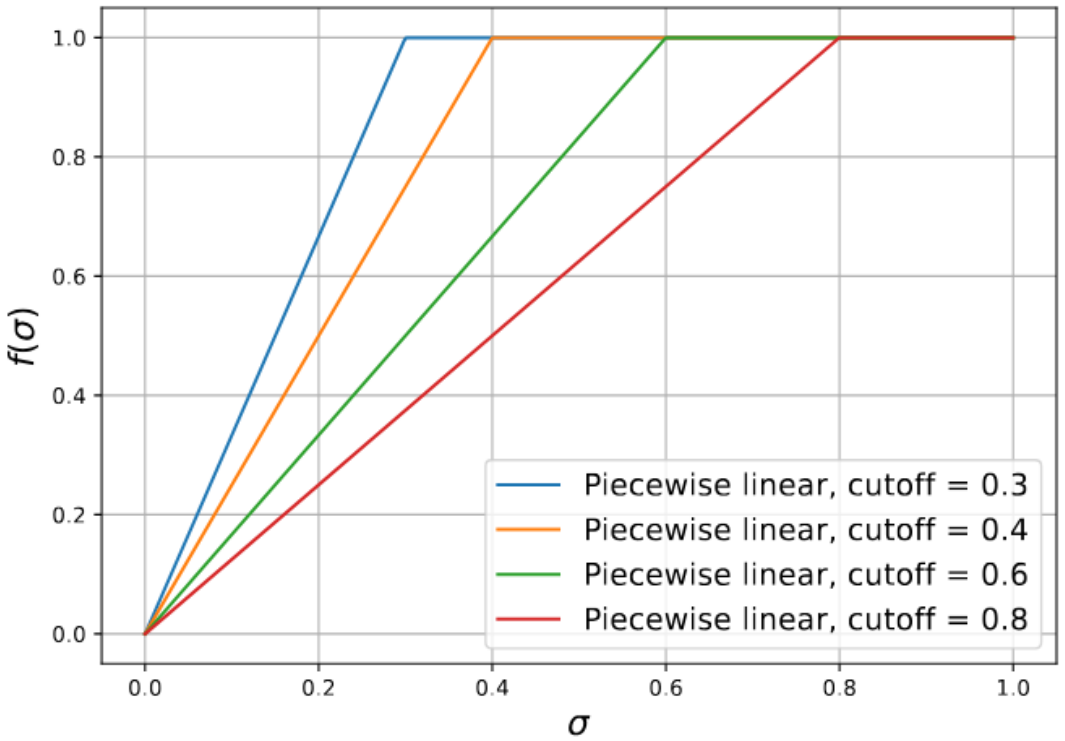
0.04, 0.2, 0.9, 0.95, 1



Target for Spectrum Mapping

Target: balance between **expressivity and optimization**:

- A family of **piece-wise linear functions** as the target functions
- Constraint: $g(0) = 0$



$$\mathrm{PL}_b(\sigma) = \begin{cases} \sigma/b, & \sigma < b, \\ 1, & \sigma \geq b, \end{cases}$$

For instance, b = 0.3, 0.4, 0.6, 0.8

Smaller b → steeper slope (1/b) → stronger conditioning on the spectrum.

Advantages:

1. Obey the natural constraint: $g(0) = 0$
2. Map singular values in [b,1] to 1.

Polynomial Fitting Problem

How to find a spectrum mapping close to target mapping?

Target function f:

$PL_{0.3}, PL_{0.4}, PL_{0.6}, PL_{0.8}$

approximate

Model class:

$G_k = \{g(x) = p(x^2)x \mid \deg(p) <= k\}$.

e.g. $G_5 = \{ax^5 + bx^3 + cx \mid a,b,c \in \mathbb{R}\}$

extra constraint: g(1) = 1 requires a+b+c = 1.

Remark:

Muon also solves a similar problem,

But the target is 1.

Polynomial Fitting Problem: Optimization

How to find a spectrum mapping close to target mapping?

Solve an optimization problem!

Function approximation problem 1:
Given a target function f, find the function in the polynomial family G_k that best approximates f.

(P1) $$\min_{g \in G_k} \int_0^1 |g(\sigma) - f(\sigma)|^2 w(\sigma) d\sigma$$

Function approximation problem 2:
Given a target function f and n random samples σ_i, find the function in G_k that best fits f on these n points.

(P2) $$\min_{a=(a_0,a_1,\ldots,a_k)\in\mathbb{R}^{k+1}} \sum_{i=1}^{n} \left| \sigma_i \sum_{t=0}^{k} a_t \sigma_i^{2t} - f(\sigma_i) \right|^2 w(\sigma_i)$$

What Type of Optimization Problem is P2?

$$\min_{a=(a_0,a_1,\ldots,a_k)\in\mathbb{R}^{k+1}} \sum_{i=1}^{n} \left| \sigma_i \sum_{t=0}^{k} a_t \sigma_i^{2t} - f(\sigma_i) \right|^2 w(\sigma_i)$$

Answer: It is a **Weighted Least Squares** problem.

Math: The polynomial space and $\mathbb{R}^n$ are **isomorphic** (as vector spaces).

Let P_3 be the vector space of polynomials of degree at most 3

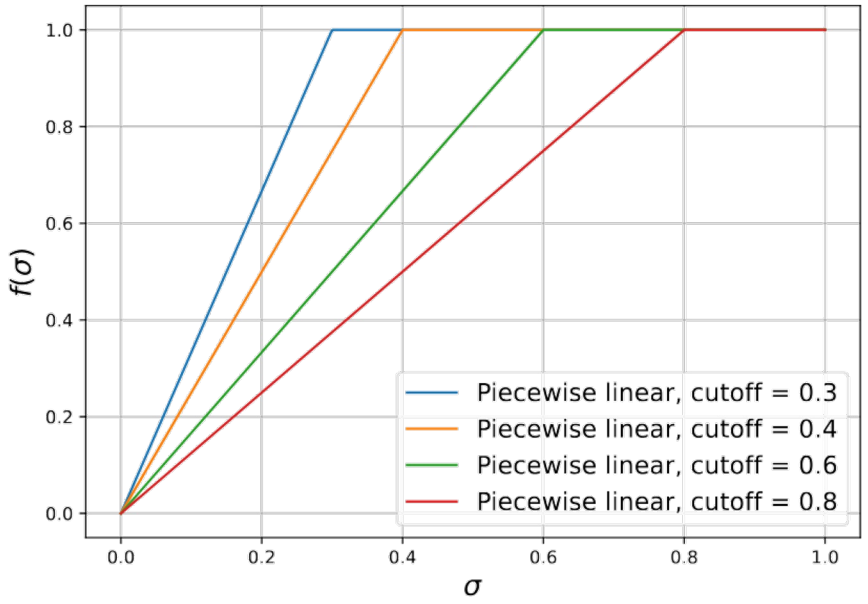
$\mathcal{B} = \{1, x, x^2, x^3\}$.

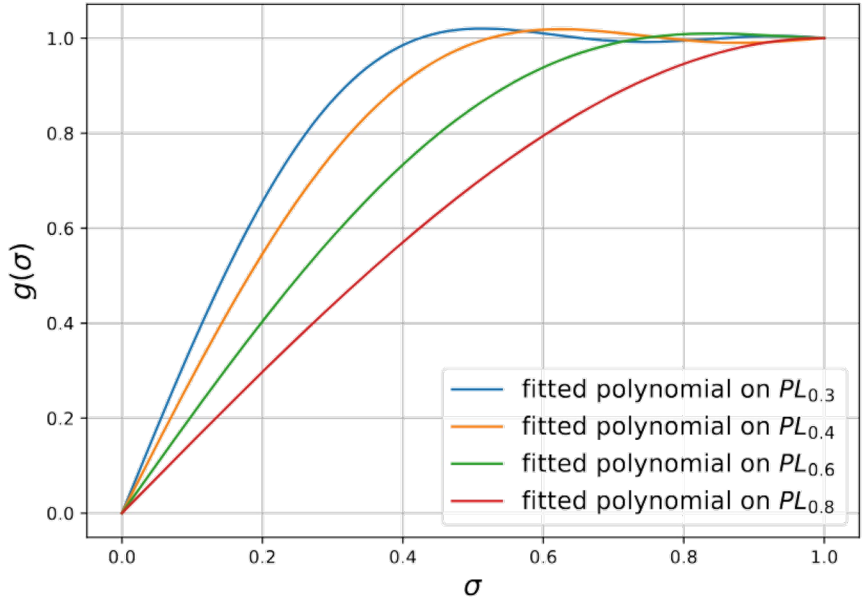
$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$[f(x)]_{\mathcal{B}} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Model Class: Degree-k Polynomials

Target function





Polynomial approximation

- Fit steeper slope with higher degree poly
- Only **two tunable hyperparameters**:
  - The **cutoff b (b = 0.8, 0.6, 0.4, 0.3)**.
  - The **polynomial degree k (k = 3, 5, 7, 9)**.

$$g_1(\sigma) = 1.507\sigma - 0.507\sigma^3,$$
$$g_2(\sigma) = 2.083\sigma - 1.643\sigma^3 + 0.560\sigma^5,$$
$$g_3(\sigma) = 2.909\sigma - 4.649\sigma^3 + 4.023\sigma^5 - 1.283\sigma^7,$$
$$g_4(\sigma) = 3.625\sigma - 9.261\sigma^3 + 14.097\sigma^5 - 10.351\sigma^7 + 2.890\sigma^9.$$

PC Layer

Algorithm 1 PC Layer for Transformers

Input: Transformer architecture $\mathcal{A}$; subset of weight blocks `PC_blocks` $\subseteq$ `param_blocks`; preconditioning polynomial g; a specific matrix norm $\|\cdot\|$; learnable scalar γ (initialized to 1).
for each weight matrix W in `PC_blocks` **do**
  $\widetilde{W} = W/\|W\|$ # *weight normalization (typically ensures* $\sigma_{\max}(\widetilde{W}) \leq 1$*)*
  $g(\widetilde{W}) = p(\widetilde{W}\widetilde{W}^\top)\widetilde{W}$ # *polynomial preconditioning on singular values*
  $\mathrm{PC}(W) \leftarrow \gamma \cdot \|W\| \cdot g(\widetilde{W})$ # *rescale with learnable* γ *and weight norm*
end for
Return: Transformer architecture $\mathcal{A}_{\mathrm{PC}} = \mathcal{A}[W \to \mathrm{PC}(W)]_{W \in \texttt{PC\_blocks}}$.

No inference-time overhead: After training $\mathcal{A}_{\mathrm{PC}}$, we simply store $\mathrm{PC}(W)$ as the new weight and use it as the parameter of the original architecture $\mathcal{A}$ during inference.

Setup of PC Layer for Transformers

Algorithm 1 PC Layer for Transformers

Input: Transformer architecture $\mathcal{A}$; subset of weight blocks `PC_blocks` $\subseteq$ `param_blocks`; preconditioning polynomial g; a specific matrix norm $\|\cdot\|$; learnable scalar γ (initialized to 1).
for each weight matrix W in `PC_blocks` **do**
  $\widetilde{W} = W/\|W\|$ *# weight normalization (typically ensures $\sigma_{\max}(\widetilde{W}) \leq 1$)*
  $g(\widetilde{W}) = p(\widetilde{W}\widetilde{W}^\top)\widetilde{W}$ *# polynomial preconditioning on singular values*
  $\mathrm{PC}(W) \leftarrow \gamma \cdot \|W\| \cdot g(\widetilde{W})$ *# rescale with learnable γ and weight norm*
end for
Return: Transformer architecture $\mathcal{A}_{\mathrm{PC}} = \mathcal{A}[W \to \mathrm{PC}(W)]_{W \in \texttt{PC\_blocks}}$.

PC setups:

- PC_blocks = {attn.o, ffn}
- Choice of $||W||$: Spectral norm $||W||_2$
- Degree-9 PC poly: approximate $PL_{0.3}$

Approximate $||W||_2$ via power iteration (T steps, e.g. 10):

$$||W||_2 = \sigma_{max}(W) \approx u^T W v$$

for i in range (T): $v \leftarrow \frac{W^T u}{||W^T u||}, u \leftarrow \frac{Wv}{||Wv||}$

No SVD needed; only matrix-vector products.

Contents

Experimental Settings

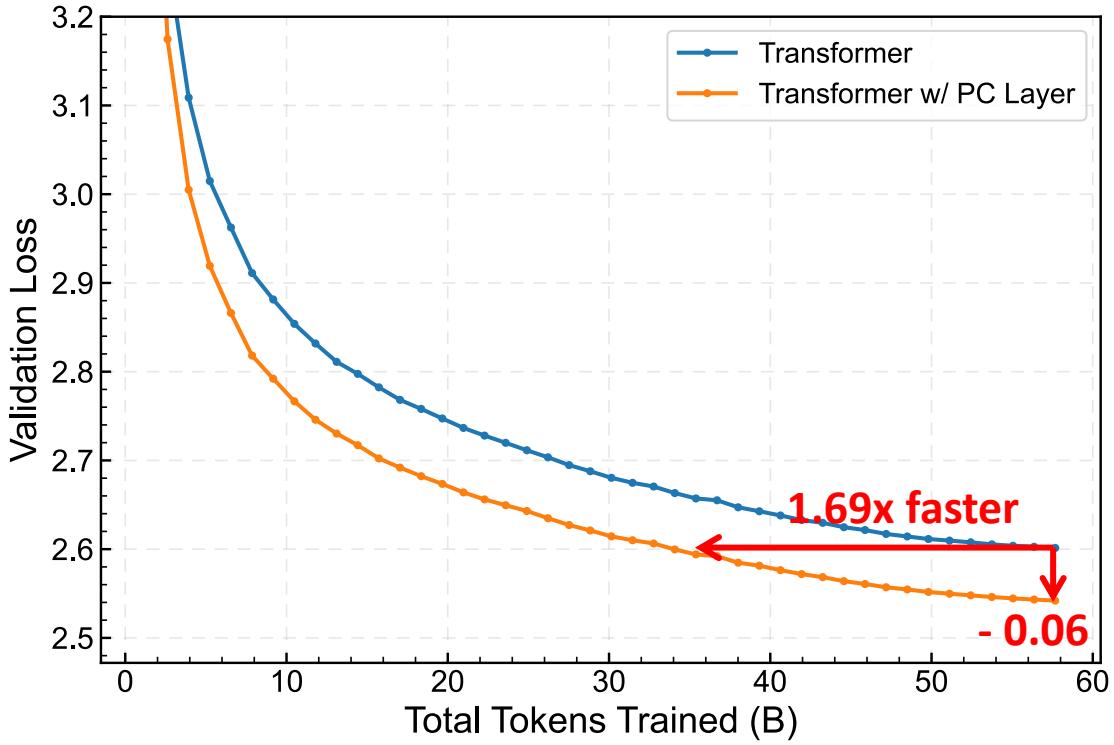
Goal: Compare Transformer with PC Layer and standard Transformer

- **Model:** LLaMA-2 **271M and 1B**

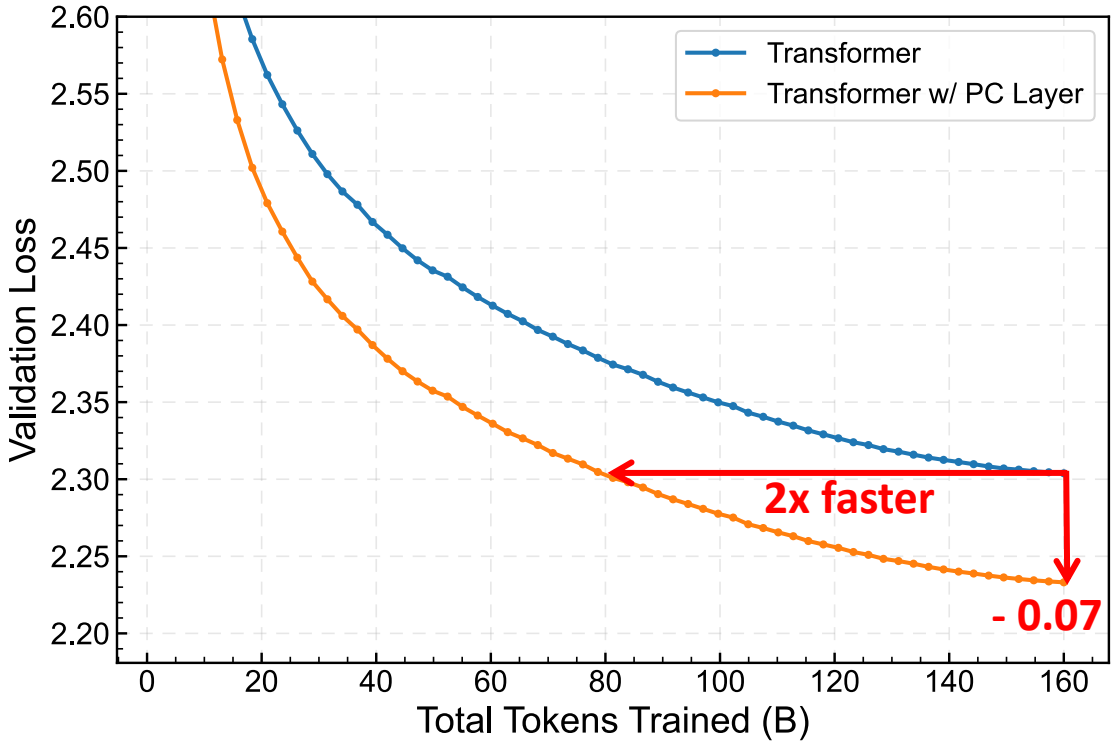
Model size	d_model	n_layers	n_heads	hidden_dim
271M	1024	16	16	2816
1B	2048	18	16	5632

- **Optimizer:** AdamW
- **Data-rich regime** for a more reliable evaluation
  - More than 160x tokens-to-parameter ratio, 8x Chinchilla law
- **Seq_len:** 8192 **Batch size:** 8192 * 320 = 2.6M tokens
- **LR schedule**: Cosine (1% warmup, decaying to 10% of max LR).
- **LR search**: Grid search the optimal LR on standard Transformer.

PC Layer Speeds up LLM Pre-Training



271M model
(57B tokens, LR=8.485e-4)



1B model
(160B tokens, LR=3e-4)

Applying PC layer results in 2x speedup over a standard Transformer (1B model).

Downstream Performance

Model	LMB. (OpenAI) Acc ↑	Hella. Acc ↑	Wino. Acc ↑	PIQA Acc ↑	ARC-E Acc ↑	ARC-C Acc ↑	CSQA Acc ↑	OBQA Acc ↑	COPA Acc ↑	Avg. Acc ↑
Llama-1B	0.5016	0.5129	0.5438	0.7171	0.4655	0.2534	**0.2080**	0.3160	0.6900	0.4676
Llama-1B w/ PC Layer	**0.5341**	**0.5639**	**0.5470**	**0.7427**	**0.5126**	**0.2807**	0.1892	**0.3220**	**0.7300**	**0.4914**

+2.38 pts

Applying PC layer results in better downstream performance.

Tricks for Matrix Polynomial Calculation

Trick 1: Choose the more efficient form:

- $g(A)$ has two equivalent forms: $p(AA^\top)A = Ap(A^\top A)$.
- **Intuition**: use the smaller Gram matrix.
- **Tall** A: choose $Ap(A^\top A)$; **wide** A: choose $p(AA^\top)A$.

Trick 2: Horner's method (秦九韶算法):

- A is an $n \times m$ matrix (WLOG, assume $n \le m$). Denote $B = AA^\top$.
- For example,
  - Naïve method: $p(B) = a_0 + a_1 B + a_2 B^2 + a_3 B^3 + a_4 B^4$. 6 matmuls!
  - Horner's method: $p(B) = a_0 + (a_1 + (a_2 + (a_3 + a_4 B)B)B)B$. 3 matmuls!
- For general degree k of polynomial,
  - Naïve: 1+2+…+(r-1) = r(r-1)/2 matmuls.
  - Horner's method: (r-1) matmuls.

FLOPs analysis

- Count **only matrix multiplications**:
  - Multiplication of $(a \times b)$ matrix with $(b \times c)$ matrix $\Rightarrow$ 2abc FLOPs;
- For $W \in \mathbb{R}^{n \times m}$, choose the smaller Gram matrix. WLOG, assume $n \leq m$.
- r: degree of polynomial p;
- k: degree of polynomial g. $(g(x) = p(x^2)x)$

- **Additional forward-pass FLOPs of PC** (including weight-norm computation):

$$\Delta\text{FLOPs}_{\text{PC,fwd}} \leq 2(r+1)\text{mn}^2$$

- **Additional FLOPs of PC in 1 training step** (forward:backward ≈ 1:2):

$$\Delta\text{FLOPs}_{\text{PC,train}} \approx 3\Delta\text{FLOPs}_{\text{PC,fwd}} \leq 6(r+1)mn^2$$

FLOPs analysis (cont'd)

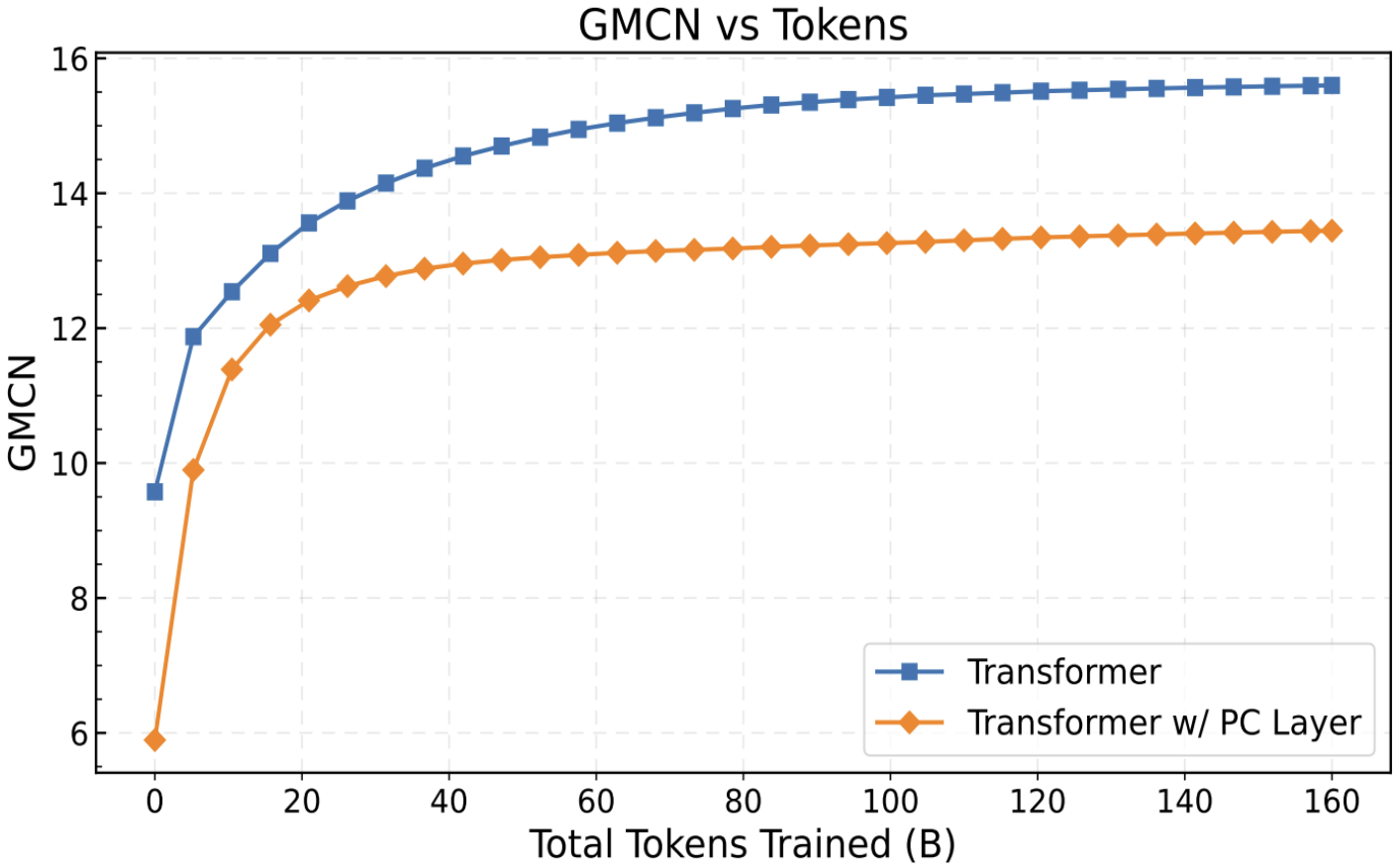
- **Per-step FLOPs of baseline** $\approx 3 \times 2nmB = 6nmB$.
  - Consider linear layer XW, with $X \in \mathbb{R}^{B\times n}$, $W \in \mathbb{R}^{n\times m}$.
  - B is the batch size.

- **Relative FLOPs overhead:**

$$\text{Overhead} = \frac{\Delta\text{FLOPs}_{\text{PC,train}}}{6nmB} \leq \frac{6(r+1)mn^2}{6nmB} = \frac{(r+1)n}{B}.$$

- Example (Llama-1B):
  - Batch size: $B = 2.62\times10^6$ tokens; model width $\text{n} = d = 2048$.
  - When $\text{r} = 4$ $\text{FLOPs Overhead} \leq 0.4\%$

PC Improves the Weight Spectrum



1B model

Modified Condition Number (MCN) of weights:

$$\tilde{\kappa}(W_\ell) = \frac{\bar{\sigma}_{\text{top } 10\%}}{\bar{\sigma}_{\text{bottom } 10\%}}$$

Average of top/bottom 10% singular values.

$\sigma_{\max}/\sigma_{\min}$ is highly susceptible to numerical instability, particularly due to near-zero $\sigma_{\min}$ or outlier $\sigma_{\max}$.

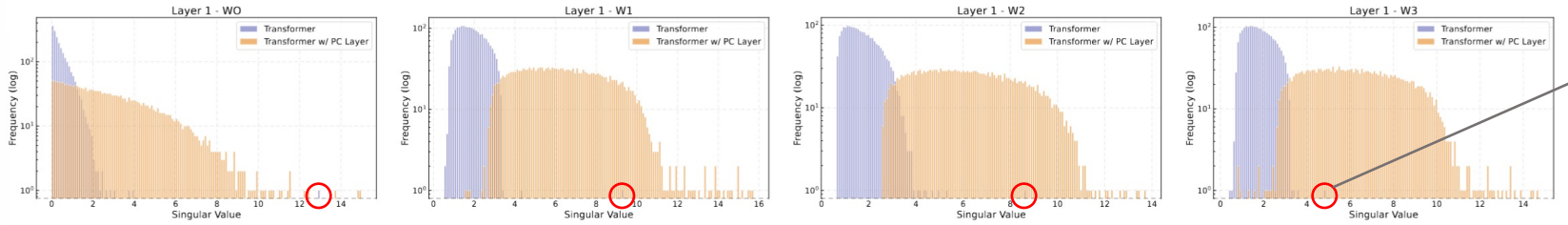
Metric:

Global Modified Condition Number (GMCN)

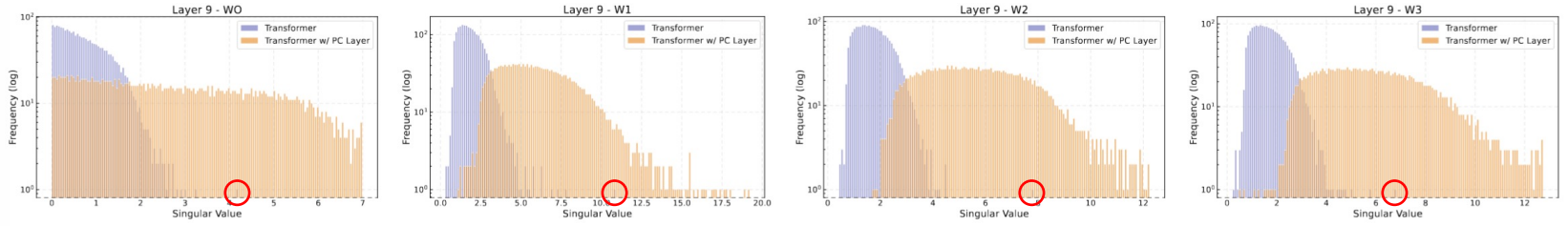
-- Geometric mean of MCNs

PC layer achieves lower GMCN – better weight conditioning.

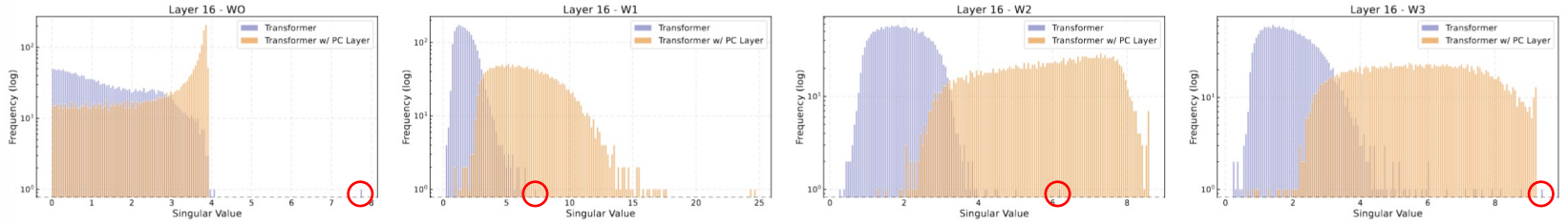
PC Improves the Weight Spectrum



(a) Layer 1: W_O (b) Layer 1: W_1 (c) Layer 1: W_2 (d) Layer 1: W_3



(e) Layer 9: W_O (f) Layer 9: W_1 (g) Layer 9: W_2 (h) Layer 9: W_3



(i) Layer 16: W_O (j) Layer 16: W_1 (k) Layer 16: W_2 (l) Layer 16: W_3

Baseline (Blue): Exhibits outlier singular values (red circles indicate the top singular value)

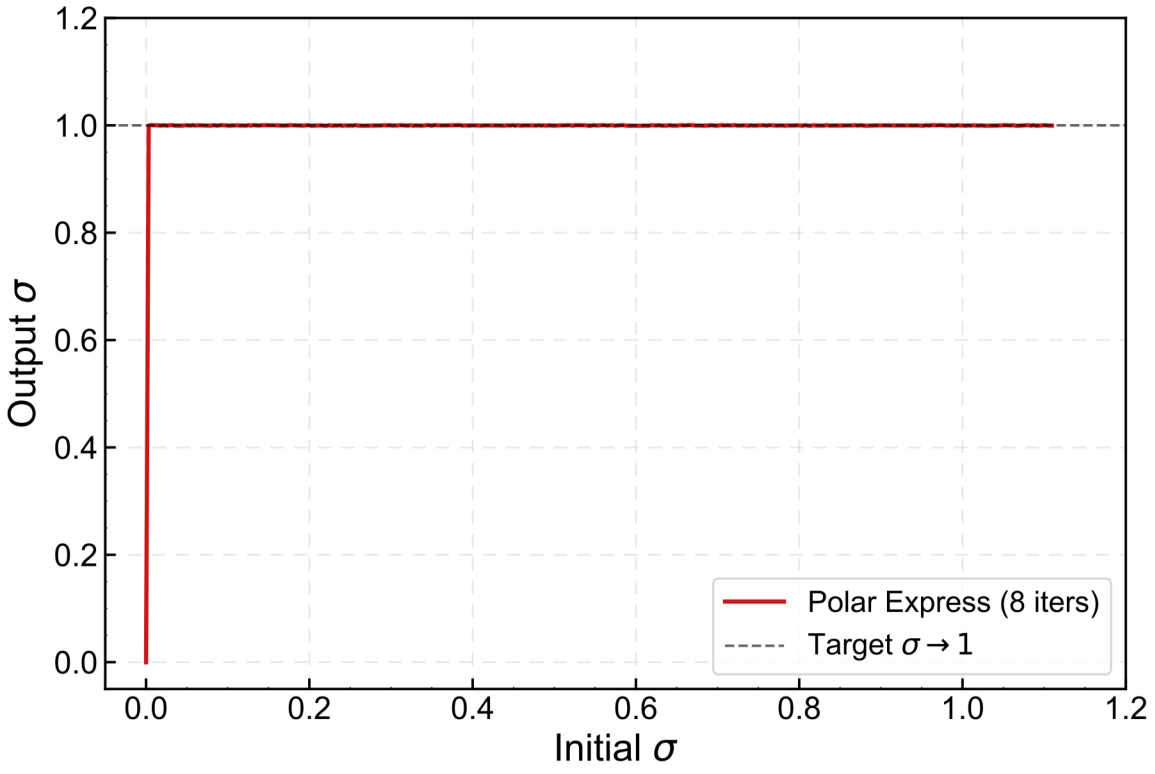
Blue: Spectrum of W in Transformer Baseline

Orange: Spectrum of $\mathrm{PC}(W)$ in PC-Transformer

Standard Transformer: Pile-up of very small singular values, many near-degenerate directions.

PC **lifts the lower end of the spectrum** and mitigates spectral degeneracy.

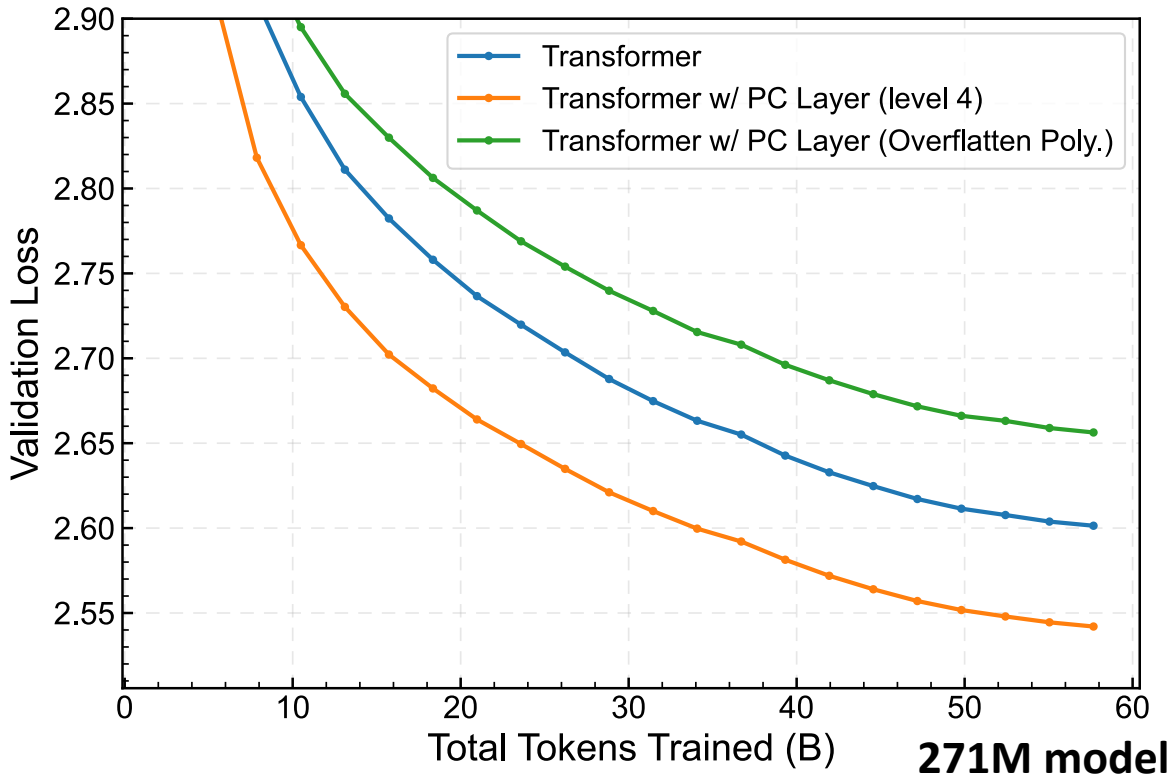
Is a Near Perfect Flat Spectrum better?



Target function: Sign(x) in $[\epsilon, 1]$

Polynomial approx: Polar Express:

Composition of **8 degree-5 polynomials**



271M model

Overflatten achieves lower condition number, but worse final loss than **baseline.**

Recall **Design Principle 2:**

κ should be O(1), but not exactly 1.

Ablation Study: What Makes PC Work?

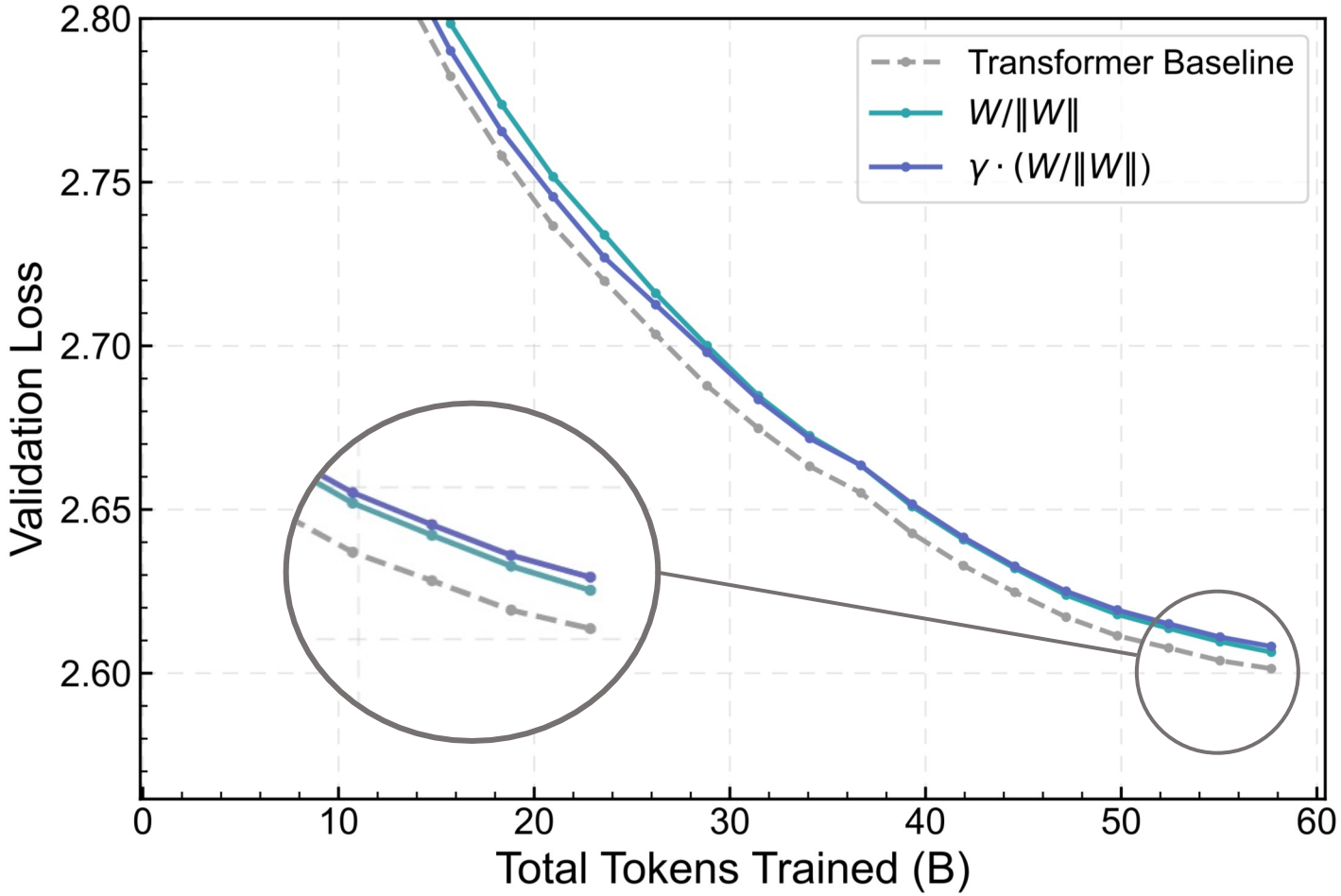
Methods	Weight Formulation
Baseline	W
Learnable Scalar Rescaling	γW
Spectral Normalization (SN)	$W/\|\|W\|\|$
Learnable Scalar + SN	$\gamma \cdot W/\|\|W\|\|$
Polynomial	$g(W/\|\|W\|\|)$
Polynomial + Learnable Scalar	$\gamma \cdot g(W/\|\|W\|\|)$
Polynomial + Multiplying back $\|\|W\|\|$	$\|\|W\|\| \cdot g(W/\|\|W\|\|)$
Polynomial + Learnable Scalar + Multiplying back $\|\|W\|\|$ **(PC)**	$\gamma \cdot \|\|W\|\| \cdot g(W/\|\|W\|\|)$

Is Controlling Only the Top Singular Value Enough?

Setup:
Baseline v.s. Spectral Normalization

Results:
SN (w/ or w/o learnable γ) slightly underperforms baseline.

Finding 1: Top singular value control alone may be insufficient.



271M model

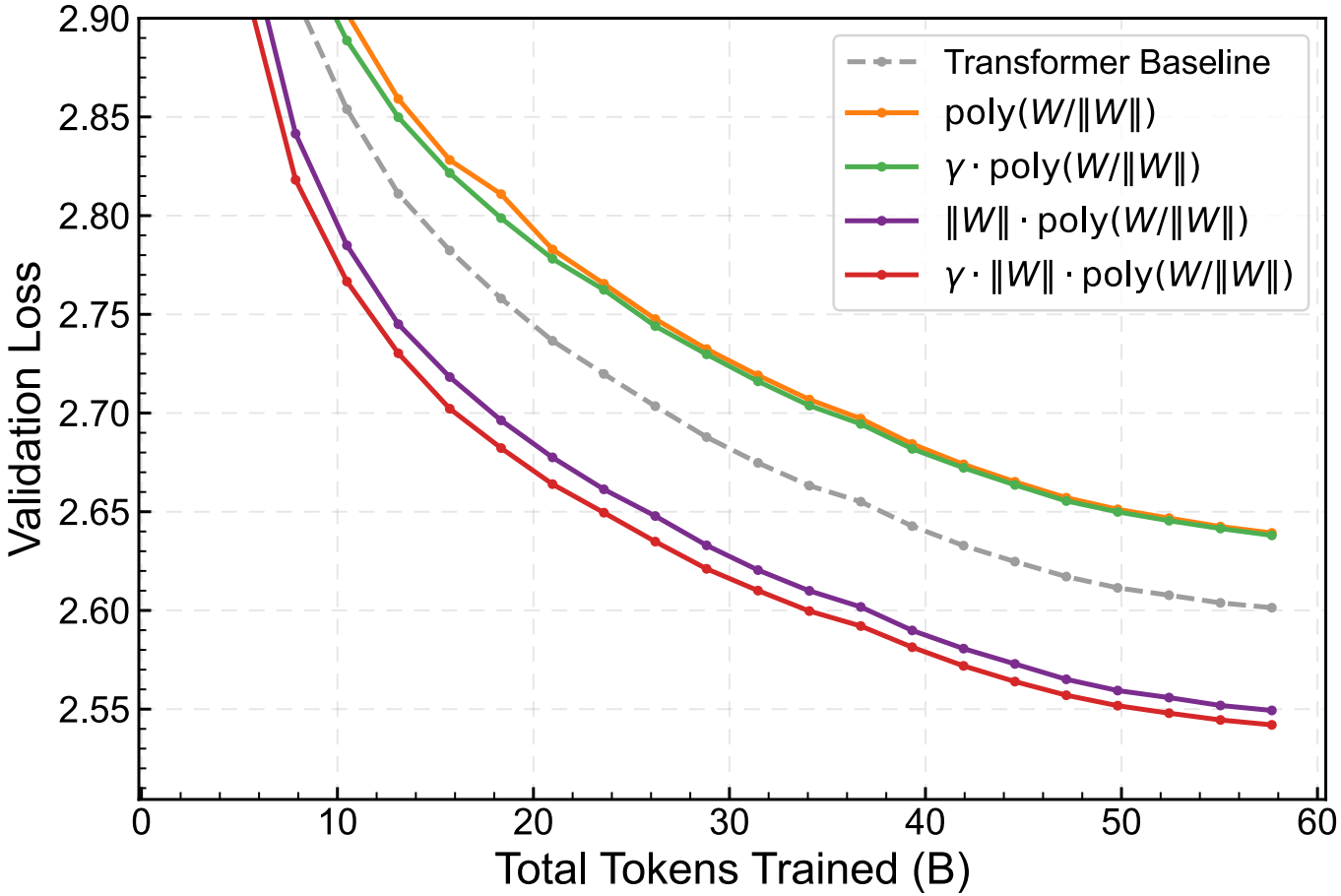
Weight Rescaling after Polynomial Preconditioning

Setup:
Baseline v.s. PC
(different ways of weight rescaling)

Results:

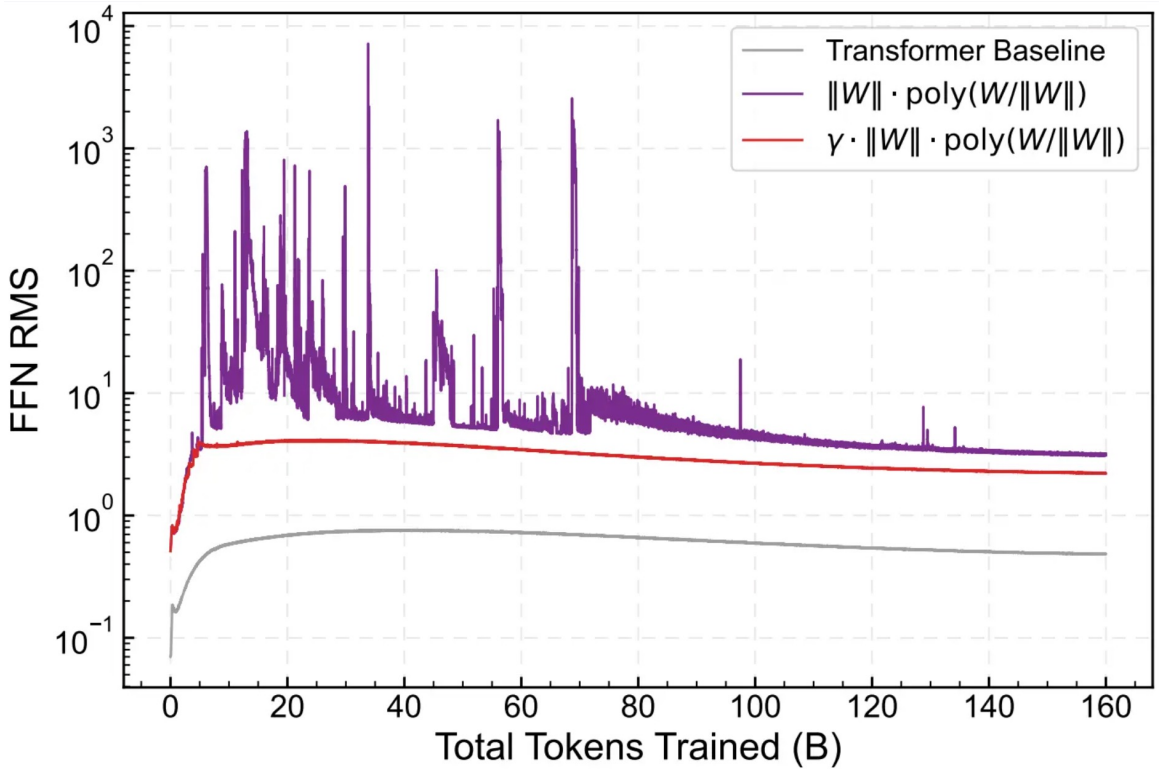
- PC w/o multiplying back $||W||$ underperforms baseline.
- PC w/o multiplying back $||W||$ outperforms baseline.
- Learnable γ matters litter for loss.

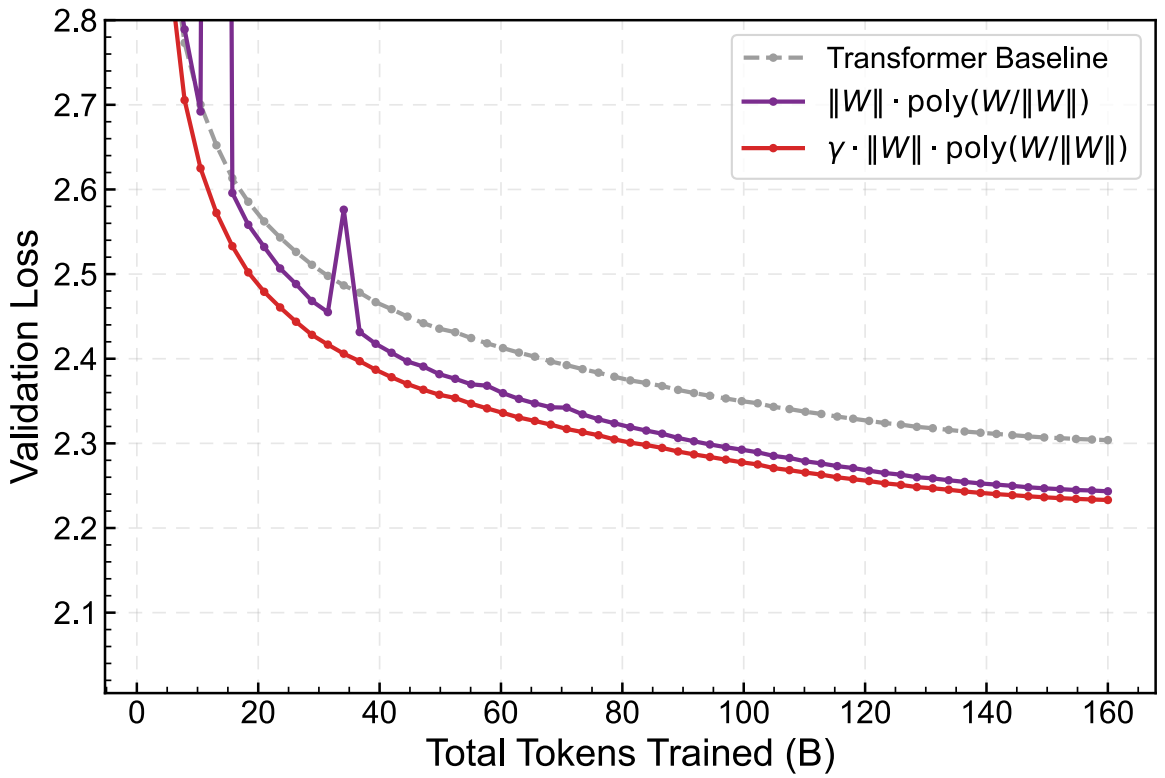
Finding 2: Multiplying back weight norm is essential.



271M model

Learnable γ in PC Layer





Oscillations in activation norms (FFN RMS) **w/o Learnable γ.**

1B model
(Effect not obvious at 271M)

Finding 3: Learnable γ stabilizes SP, thereby stabilizing training.

γ evolves smoothly during training, without oscillatory behavior.

Conclusion for ablation study

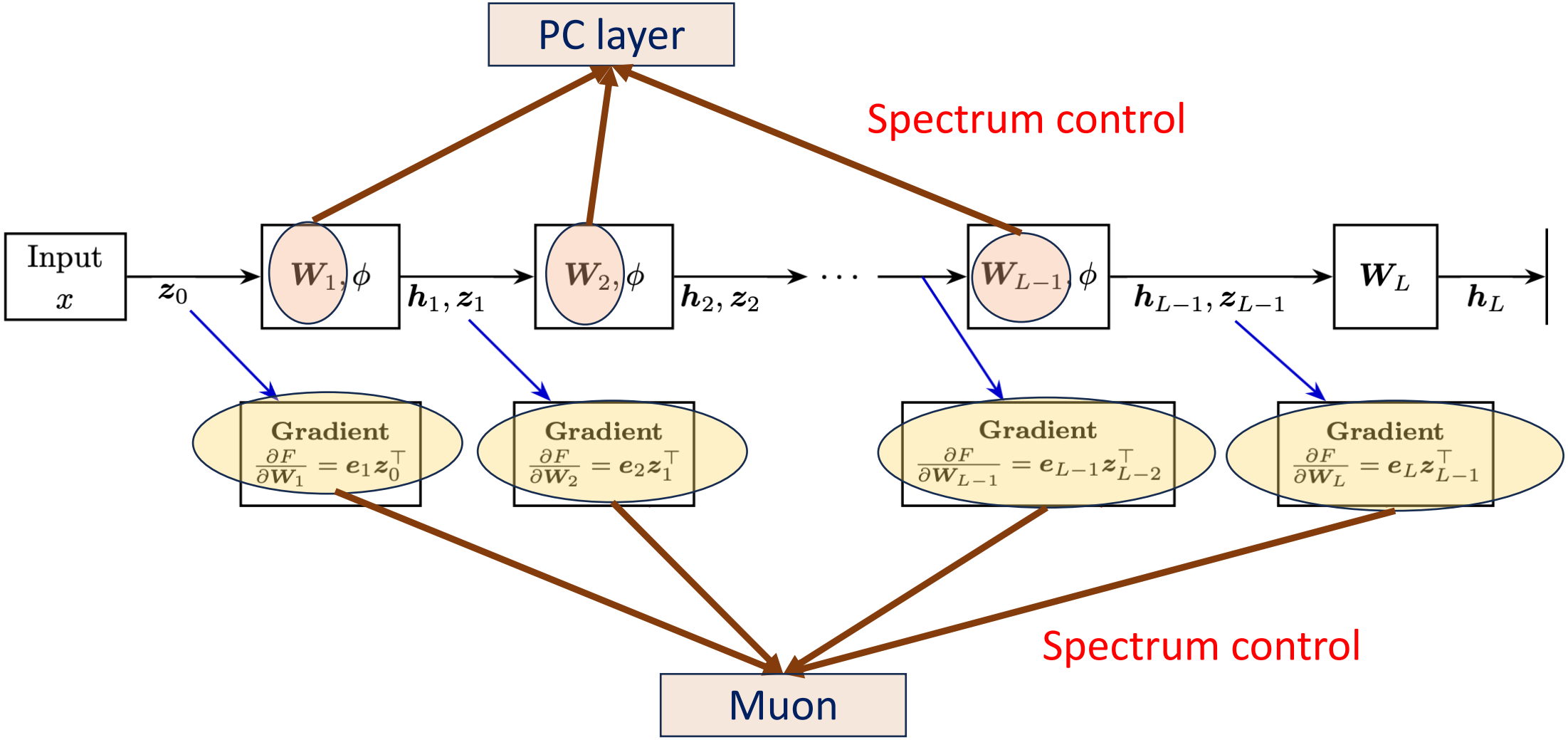
PC Layer: Spectrum Reshaping + Weight Rescaling

- **Spectrum reshaping:** polynomial transformation reshapes the singular value distribution into a better-conditioned range.

- **Weight rescaling:**
  - **Multiplying back $||\boldsymbol{W}||$** is essential for performance gains.
  - **Learnable $\boldsymbol{\gamma}$** stabilizes SP, improving training stability.

Contents

Relationship between PC and Muon



Differences between PC and Muon

on **weights**

SGD under PC network

$$W_{\text{new}} = W - \eta \frac{\partial \mathcal{L}(g(W))}{\partial W}$$

v.s

on **gradients (momentums)**

Muon

$$W_{\text{new}} = W - \eta \cdot g\left(\frac{\partial \mathcal{L}(W)}{\partial W}\right)$$

Target function of polynomials:

Muon: sign function (for orthogonal updates)
PC: piecewise linear function
(for balancing expressivity with optimization)

Remark: Muon uses high-order polynomials,
but purpose is approximating orthogonal gradient.

How about setting up target to be
well-conditioned gradient, instead of orthogonal?
--For what purpose?

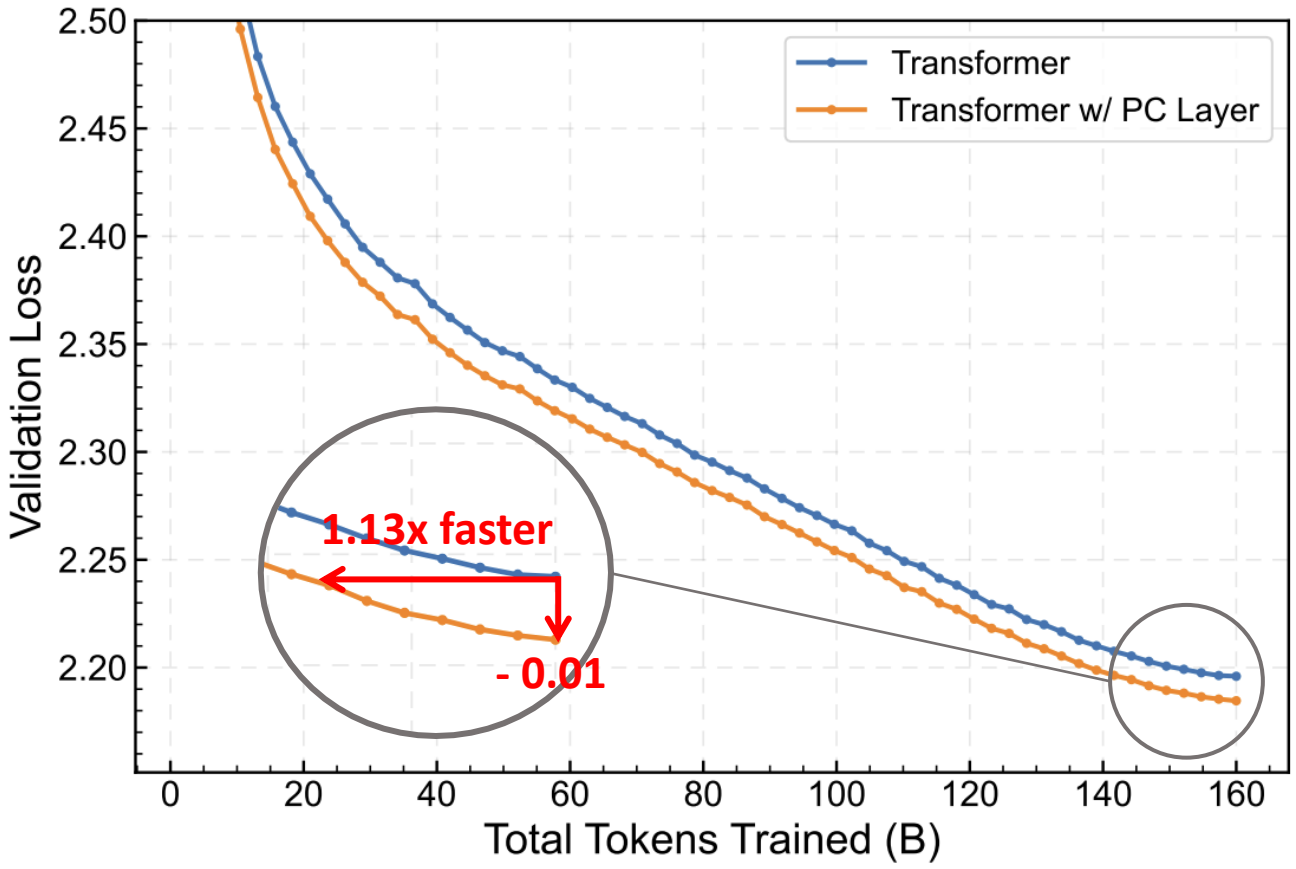
Optimization Theory Design Principle:

PC layer can be justified by global convergence;

Muon: justified by one-step reduction.
(Weijie's talk)
or preconditioned GD (faster local convergence)

PC layer Improves Training under Muon

They are somewhat orthogonal, so PC layer and Muon can combine!



1B model

LR= 1.5e-3

Moonlight version (Kimi) of Muon:
Use RMS matching trick.

Degree-5 PC poly: approximate $PL_{0.6}$

Using Muon as the backbone optimizer, PC achieves 1.1x speedup.

Downstream Performance

Model	LMB. (OpenAI) Acc ↑	Hella. Acc ↑	Wino. Acc ↑	PIQA Acc ↑	ARC-E Acc ↑	ARC-C Acc ↑	CSQA Acc ↑	OBQA Acc ↑	COPA Acc ↑	Avg. Acc ↑
Llama-1B	0.5643	0.6034	0.5699	0.7427	0.5076	**0.3063**	**0.2039**	**0.3500**	0.7400	0.5098
Llama-1B w/ PC Layer	**0.5818**	**0.6071**	**0.5935**	**0.7530**	**0.5404**	0.2935	0.2015	0.3460	**0.7700**	**0.5208**

+1.10 pts

Using Muon as the backbone optimizer, applying PC layer results in better downstream performance.

Contents

Intuition of Global Convergence 1

Suppose $f(\theta; x)$ denotes the output of neural net with input x.

Objective function: $\mathrm{L}(\theta) = \frac{1}{2}\left|\left|\mathrm{Y} - F(\theta; X)\right|\right|^2$, where $\mathrm{F}(\theta; X) = (f(\theta; x_1), \ldots, f(\theta; x_n))$ and $X = (x_1, x_2, \ldots, x_n)$.

Jacobian: $G(\theta) = \left(\frac{\partial f(\theta; x_1)}{\partial \theta}, \ldots, \frac{\partial f(\theta; x_n)}{\partial \theta}\right)$; NTK: $K(\theta) = G(\theta)^\top G(\theta)$

Lemma 1: Any critical point θ with full rank NTK is a global minimum (with objective value 0).

Intuition of Global Convergence 2

- Continuous-time gradient dynamics: $\frac{d\,\theta(t)}{dt} = -\frac{\partial F(\theta(t))}{\partial \theta}$.

- **Assumption 1:** On the interval [0,t], the minimal eigenvalue of the NTK is bounded below by λ_{min}.

- **Lemma 2**: Under Assumption 1, the loss value satisfies

$$L\big(\theta(t)\big) \leq L\big(\theta(0)\big)^{-2t\lambda_{min}}.$$

NTK Spectrum and Global Convergence

- Controlling spectrum of NTK is important for stabilizing NN training (Pennington et al. 2017; Xiao et al. 2018; 2020)
  - If the **NTK stays full rank**, then GD on **wide NNs** converges to global minima. (Jacot et al. 2018; Du et al. 2018; Allen-Zhu et al. 2019; Zou et al. 2018)
  - The initial NTK is full-rank and stays full rank during training can be fulfilled by **orthogonal init.** (Hu et al. 2020)

- Some drawbacks:
  - **Theory**: Assume ultra-wide neural network to **ensure Assumption 1.**
  - **Design**: Does NOT directly provide guidance on algorithmic improvement.

- But at least leaves an intuition:
  - **Better NTK spectrum → convergence**

Notation and Assumption

- Notations
  - A linear neural network $f(\theta; x) = W_L W_{L-1} \dots W_1 x \in \mathbb{R}^{d_y \times 1}$
  - $\theta = (W_1, W_2, \dots, W_L)$: parameters; W_j: matrix of dimension $d_j \times d_{j-1}, j = 1, \dots, L$
  - **Assumption 3**: diamond network : there exists $r \in \{1, .., L\}$, such that $d_y = d_L \leq d_{L-1} \leq \cdots \leq d_r$, and $d_r \geq d_{r-1} \geq \cdots \geq d_0 \geq n$.

  - Residual after the k^{th} update via gradient descent : $e(k) = F(\theta; x) - y$
  - For given $\tau_l \geq 1, \mu_l \geq 0, l = 1, \dots, L$, define
    - $R \triangleq \{\theta = (W_1, \dots, W_L) | \tau_l \geq \sigma_{max}(W_l) \geq \sigma_{min}(W_l) \geq \mu_l, \forall l\}$
    - $\beta \triangleq L \left|\left|X\right|\right|_2 (\tau_L \tau_{L-1} .. \tau_1)^2 \left(\left|\left|e(0)\right|\right| + \left|\left|x\right|\right|_F\right)$
    - $\mu \triangleq (\mu_1 \dots \mu_L)^2 \sigma_{min}(x)^2$
- We use GD with stepsize η

Weight Conditioning and Convergence

Theorem 1 *Suppose* $\eta = \frac{1}{\beta}$. *Assume* $\theta(k) \in R,\ k = 0, 1, \ldots, K$. *Then we have*

$$\|e(k+1)\|^2 \leq (1 - \frac{\mu}{\beta})\|e(k)\|^2, \quad k = 0, 1, \ldots, K.$$

- In words, if the weight condition number is upper bounded for K iterations, then the loss decreases at a linear rate for these K iterations.
- If $\theta(k)$ stays in the nice region R for all iterations, then we obtain global convergence to zero.

We observe that

$$\frac{\beta}{\mu} = C\frac{(\tau_L \ldots \tau_1)^2}{(\mu_L \ldots \mu_1)^2},$$

implying that smaller weight condition numbers lead to a faster convergence rate.

Note: if the weights move out of the "nice regime", then the theory says nothing.

Proof Sketch

Theorem 1 *Suppose* $\eta = \frac{1}{\beta}$. *Assume* $\theta(k) \in R,\ k = 0, 1, \ldots, K$. *Then we have*

$$\|e(k+1)\|^2 \leq (1 - \frac{\mu}{\beta})\|e(k)\|^2, \quad k = 0, 1, \ldots, K.$$

Proof sketch.

1. Well-conditioned $\theta(k) \in \mathcal{R}$ implies well-conditioned NTK: $\mu I \preceq K(\theta(k)) \preceq \beta I$.
2. Lower bound μ: PL inequality; upper bound β: smoothness of the loss function.
3. Standard convergence proof for GD under PL inequality and smoothness.

It supports our intuition:

Better weight conditioning → Better Jacobian spectrum → Better convergence

Theory → Design

- **This theorem has a direct implication on the algorithm design.**

- **Interpretation**:
- If we can improve the condition numbers of weights during training,
- Then the updated parameter trajectory may stay in R for a longer time,
- and thus lead to smaller loss values.

- **Disclaimer**: of course, the theory is just for linear networks; but still, we believe such a result is still informative.

  Extension to non-linear networks is also possible.

Extension to Transformers

- We did not intend to extend the proof to transformers.
- But one insight from the theory:

  no need to perform PC on Q,K.
- **Reason**: $\kappa(Q), \kappa(K)$ have little impact on the whole $\kappa(L)$.

Summary

Motivation: Weight Spectrum Control for better training

Algorithm Design: PC layer – Polynomial Preconditioning

Experiments: Speedup compared with standard 1B Transformer.

Relationship with Muon: PC and Muon are orthogonal, and can combine.

Theory: Convergence to global minima under controllable conditions,
relating weight spectrum to global convergence.

Reference: PC Layer: Polynomial Weight Preconditioning for Improving LLM Pre-Training. Senmiao Wang*, Tiantian Fang*, Haoran Zhang*, Yushun Zhang, Kunxiang Zhao, Alex Schwing, and Ruoyu Sun. To release soon.

Thank You!

Email 邮箱：sunruoyu@cuhk.edu.cn
Website 个人网站：ruoyus.github.io

Interested in LLMs, machine learning, optimization, communications, etc.
研究兴趣：大模型、机器学习、优化、通信