nGPT: Normalized Transformer

nGPT: Normalized Transformer with Representation Learning on the Hypersphere

Yiming Wang[1] and Andreas Plesner[2]

5/20/2025

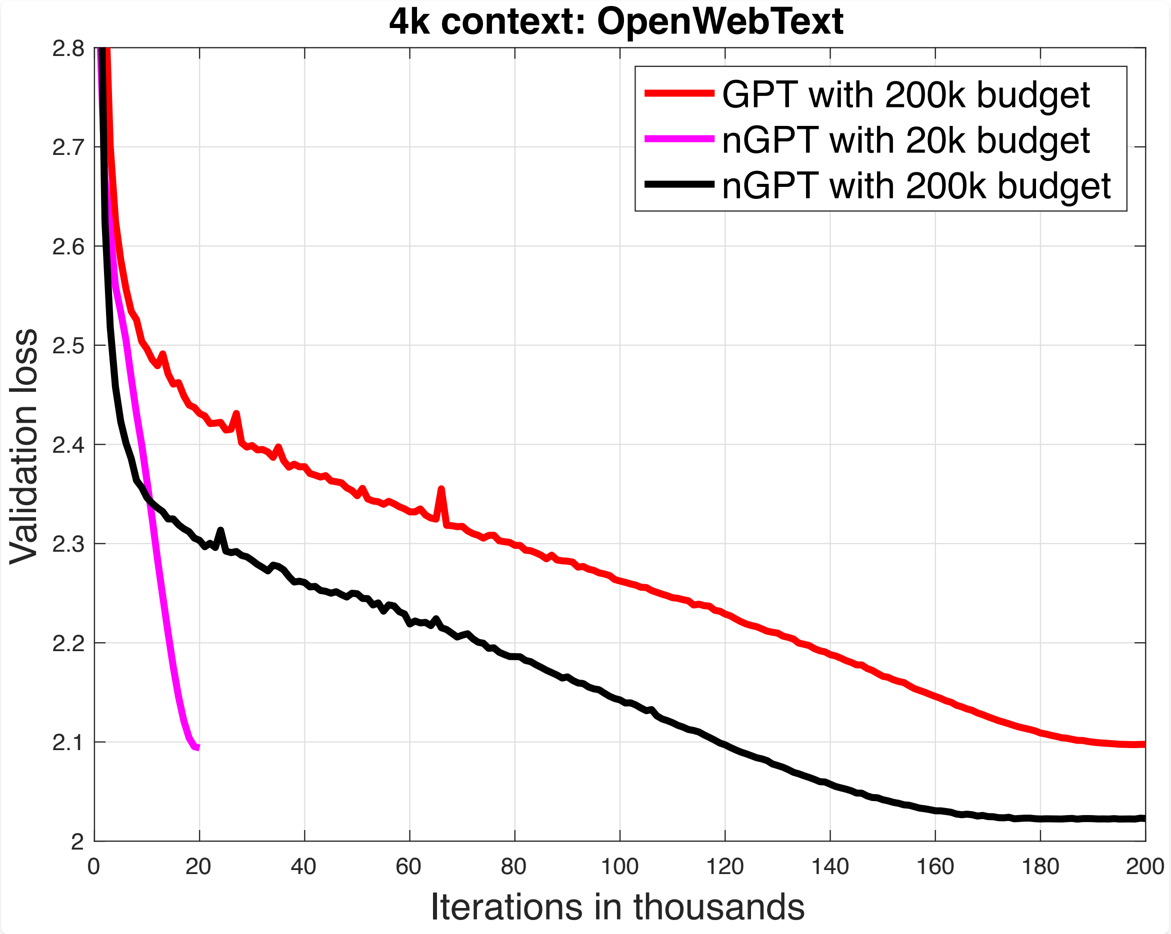
[1] Presenter

[2] Advisor

ETH zürich

I. Prologue

A. Faster and Better

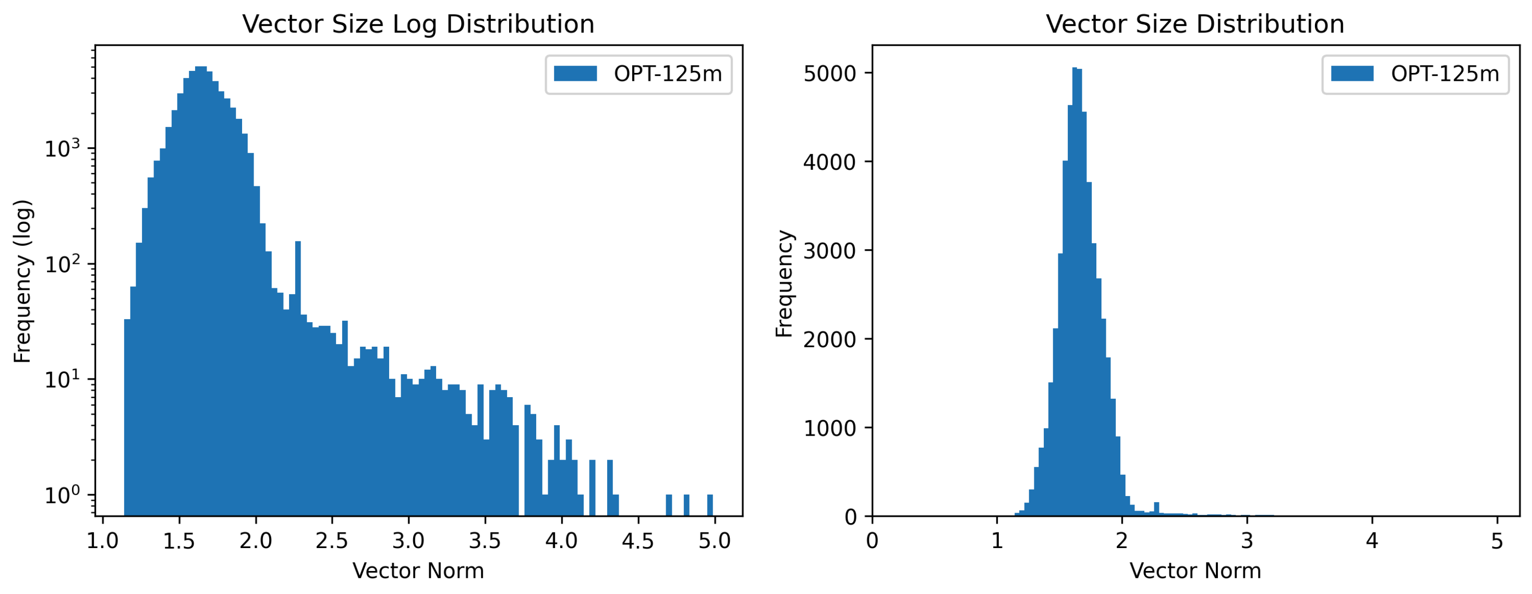


How possible?

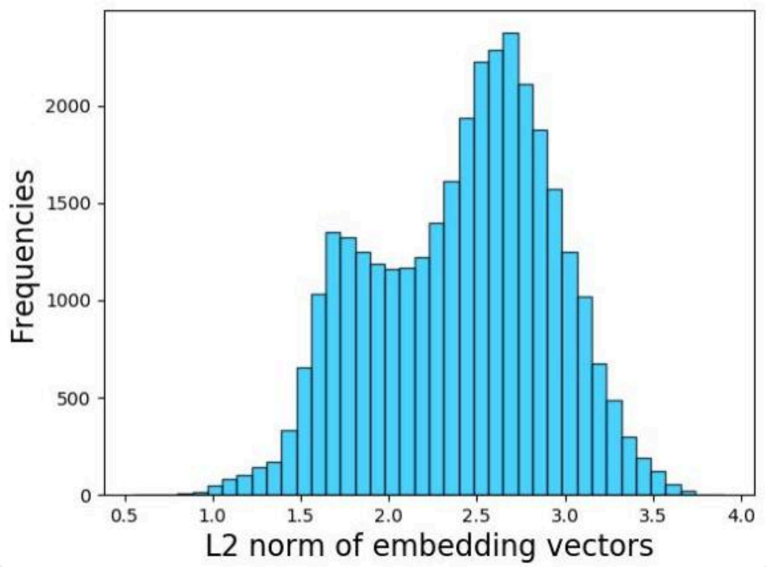
How does it work?

I.B. The Unconstraint

From blog-NickyP we see that:

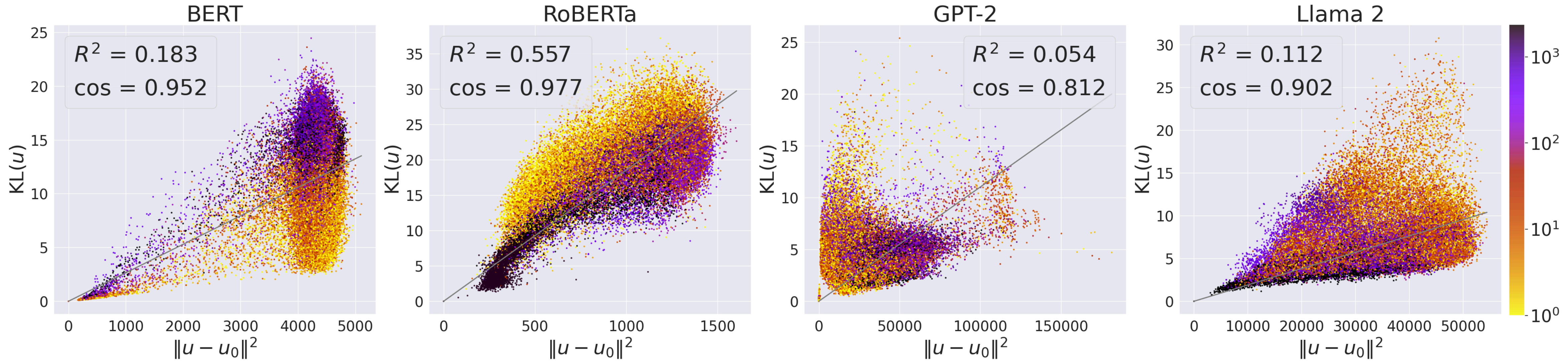


Similar results also shown in Liu2020,



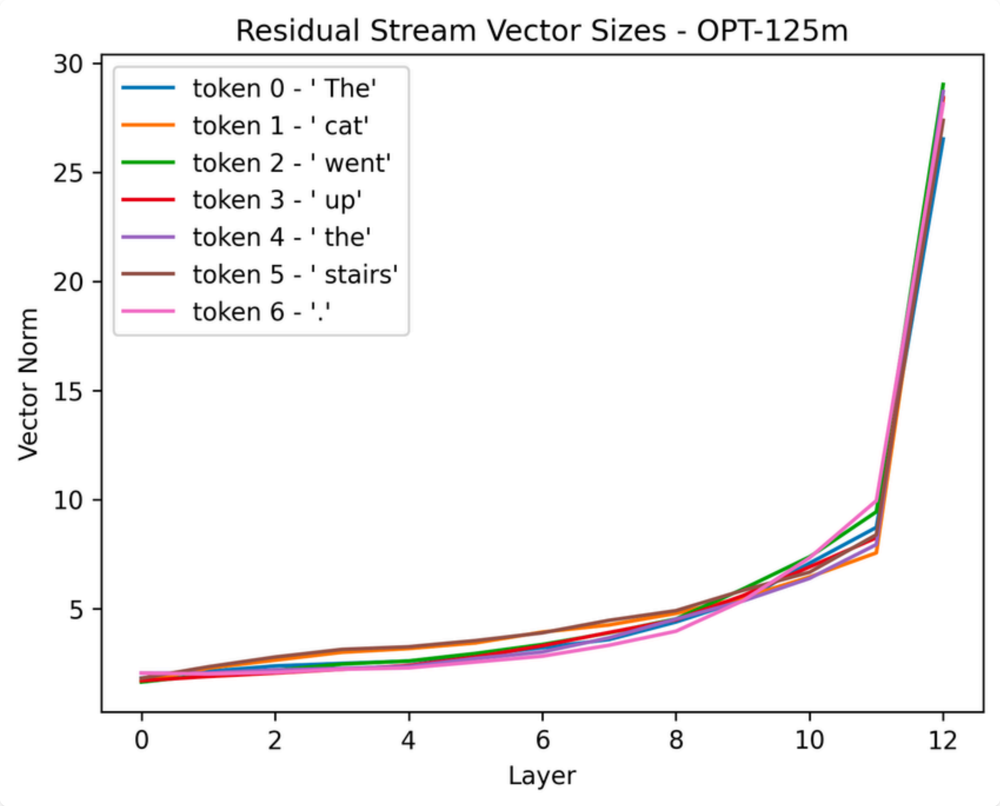
I.B. The Unconstraint

Althought the norms of word embedding do carry *information*, as Oyama2022 shown:



I.B. The Unconstraint

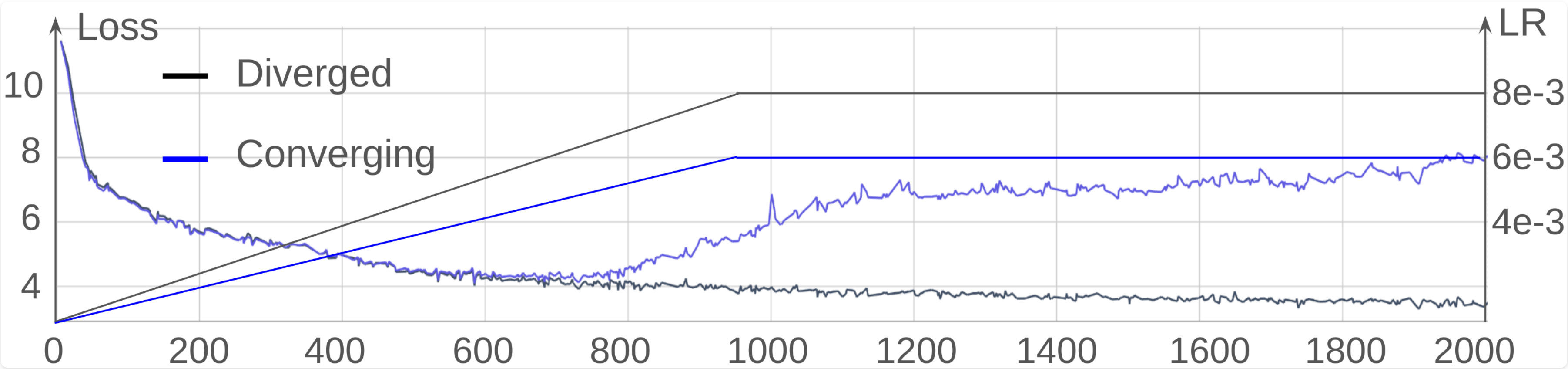
However, the norms of these embeddings keep *unconstraintly* **increasing** during training, as Oyama2022,shown:



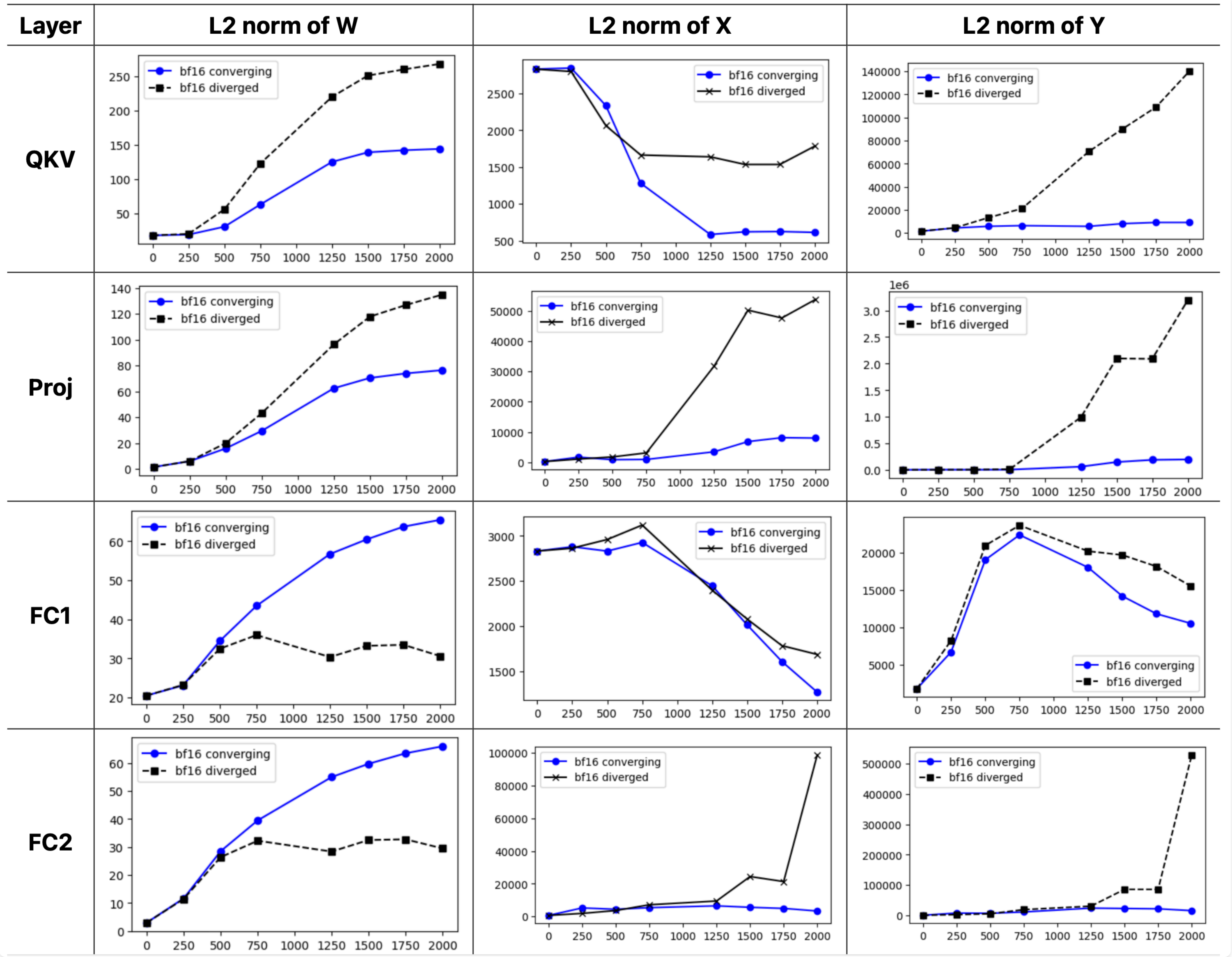
So what?

I.B. The Unconstraint

Rybakov2024 compared two models (diverged vs converging)



I.B. The Unconstraint

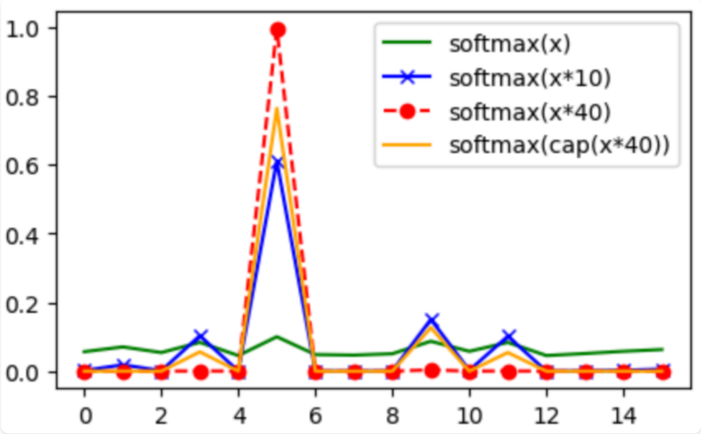


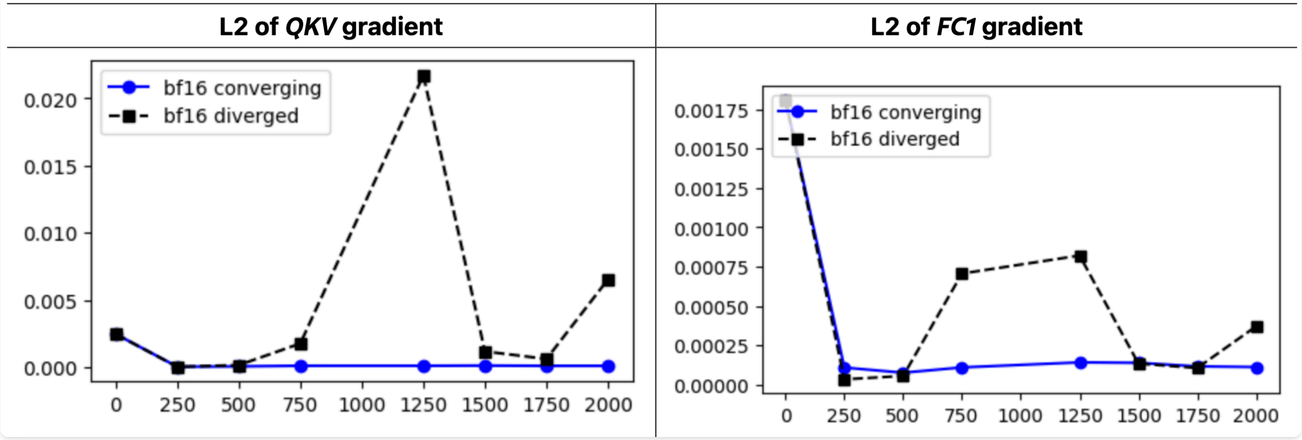
, and Rybakov2024 showed that:

- **unbounded** output L2 norm growth
- **exploding** input gradients
- disrupting training **stability**

I.B. The Unconstraint

The influence of the unbonded l_2-norm of the linear layer on logits and gradients Rybakov2024 shown:





II. From GPT to nGPT

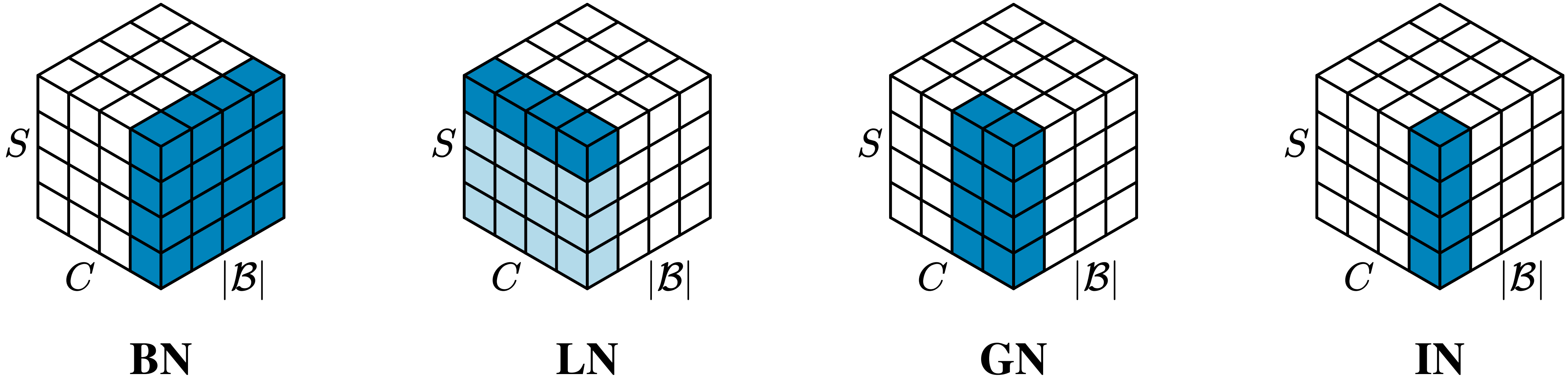
Architecture Modification

Transformer	Normalized Transformer
$h_A \leftarrow \mathrm{ATTN}(\mathrm{RMSNorm}(h))$	$h_A \leftarrow \mathrm{Norm}(\mathrm{ATTN}(h))$
$h \leftarrow h + h_A$	$h \leftarrow \mathrm{Norm}(h + \alpha_A(h_A - h))$
$h_M \leftarrow \mathrm{MLP}(\mathrm{RMSNorm}(h))$	$h_M \leftarrow \mathrm{Norm}(\mathrm{MLP}(h))$
$h \leftarrow h + h_M$	$h \leftarrow \mathrm{Norm}(h + \alpha_M(h_M - h))$
Final: $h \leftarrow \mathrm{RMSNorm}(h)$	

Transformer vs. Normalized Transformer.

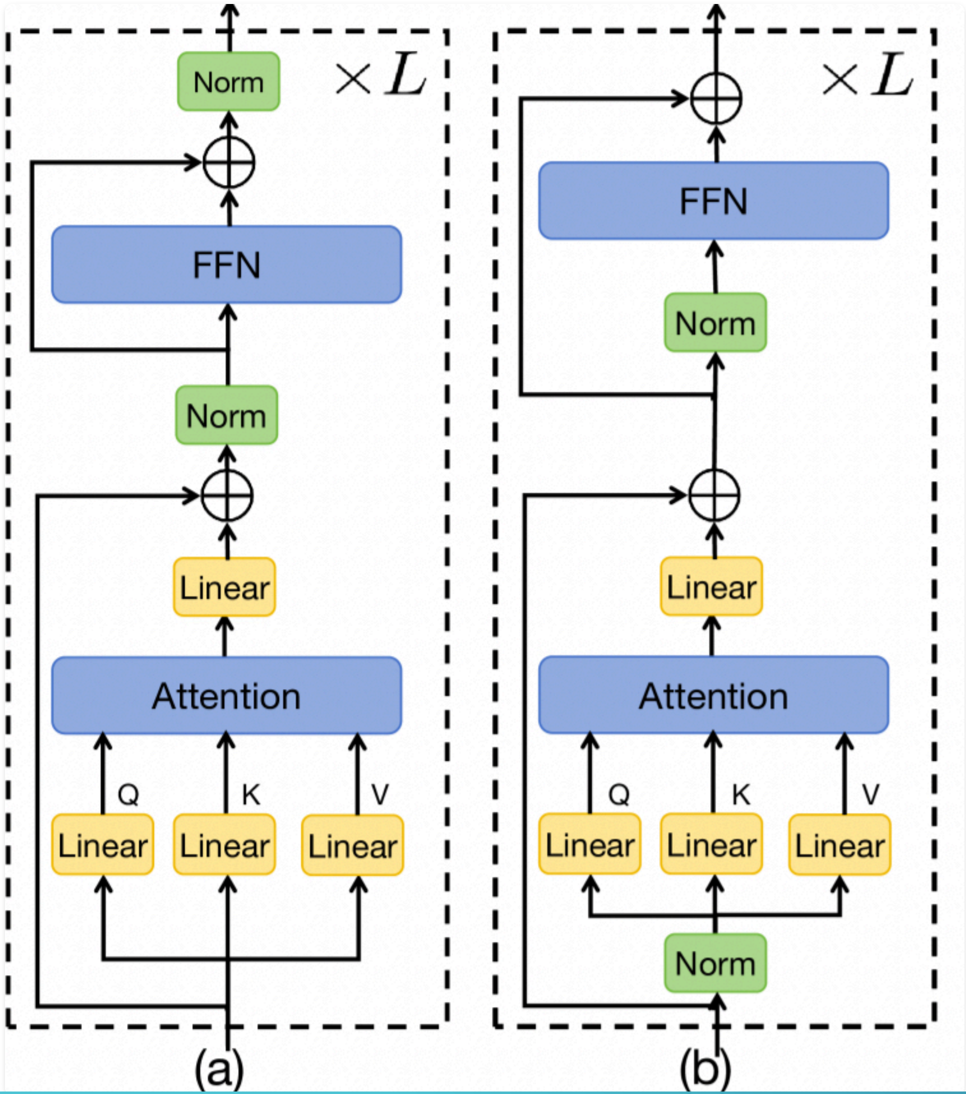
II.A. Which Normalization?

Right now we have **BatchNorm** (Ioffy2015), **LayerNorm** (Ba2016), **post-LayerNorm** (Xiong2020), **MixNorm** (Hu2021), **qk-norm** (Henry2020), **DeepNorm** (Wang2022), **HybridNorm** (Zhuo2025) , even **NormFormer** (Shleifer2021).



II.A. Which Normalization?

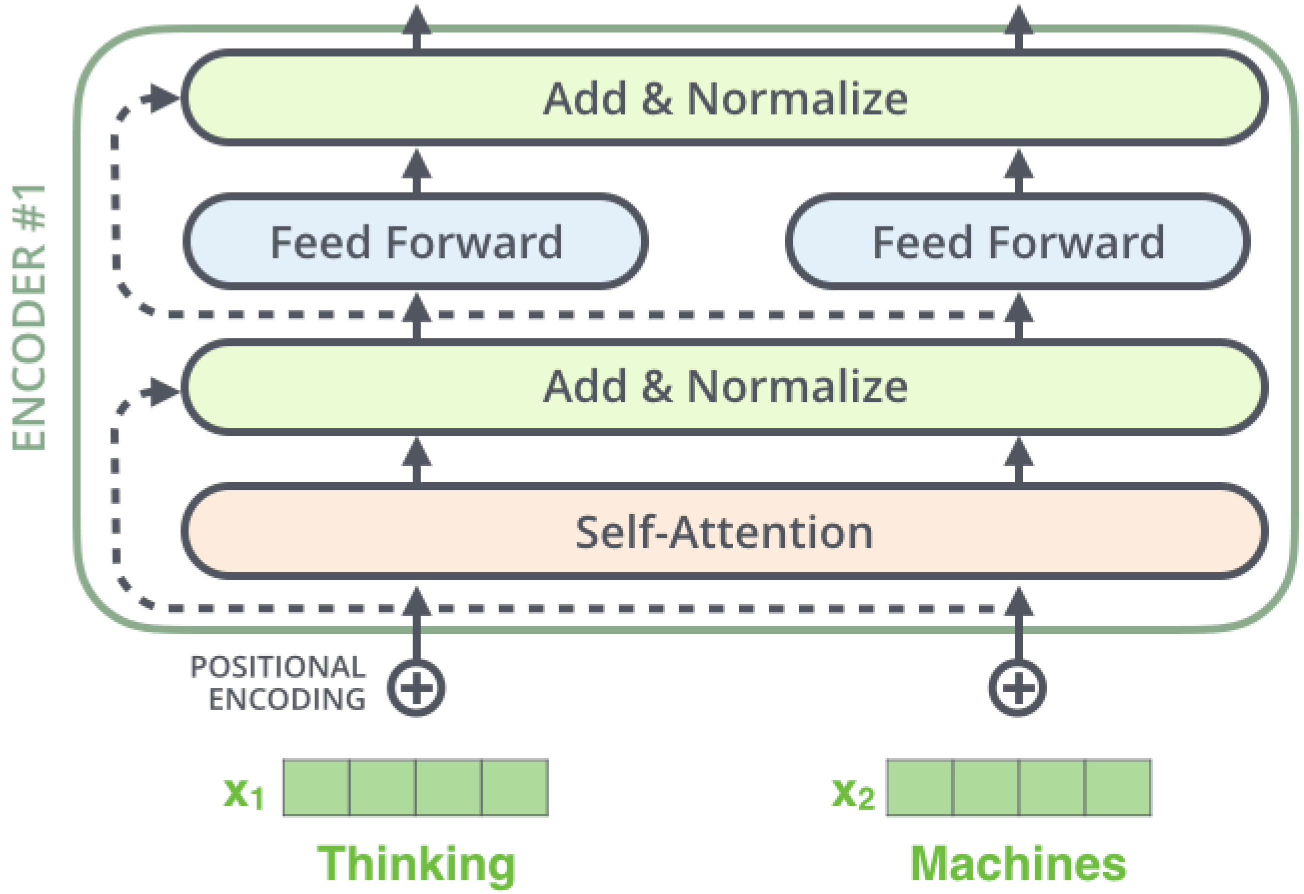
It seems that we need to normalzize the embeddings, but **which** kind of normalizaiton are you talking about?



II.A. Which Normalization?

post-LN is deploy in original Transformer Vaswani2017

- a residual connection followed up with a post-LN for each sublayer in each encoder.

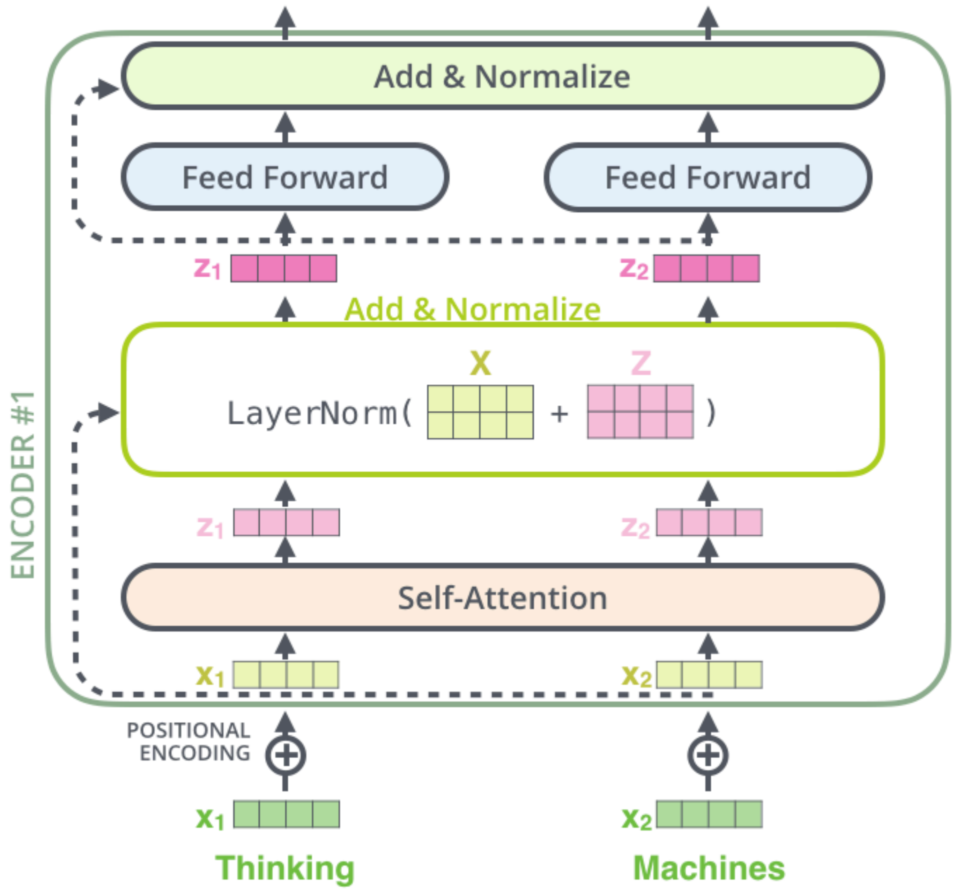


`

II.A. Which Normalization?

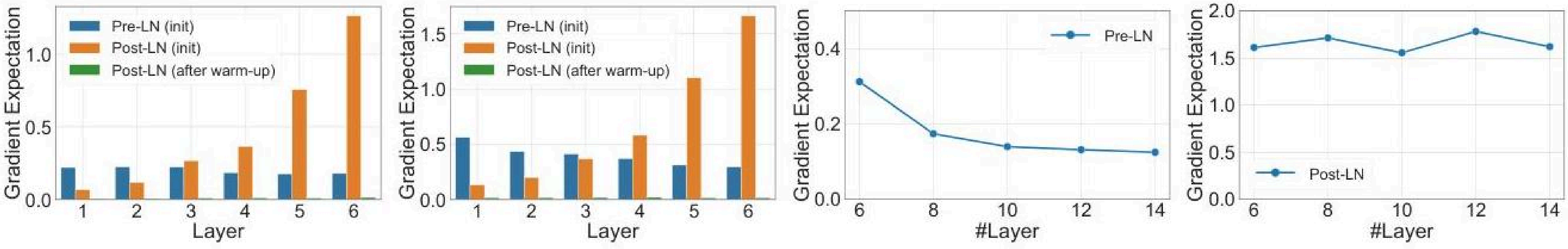
post-LN is deploy in original Transformer Vaswani2017:

- visualizing the vectors within post-LN operation and self-attention



II.A. Which Normalization?

However, Xiong2020 shown that post-LN must learning rate `warmup`



(a) W^1 in the FFN sub-layers (b) W^2 in the FFN sub-layers (c) Pre-LN Transformer (d) Post-LN Transformer

- to **pre-LN**? or to **post-LN**?

II.B. Totally Normlization!

Transformer	Normalized Transformer
$h_A \leftarrow \mathrm{ATTN}(\mathrm{RMSNorm}(h))$	$h_A \leftarrow \mathrm{Norm}(\mathrm{ATTN}(h))$
$h \leftarrow h + h_A$	$h \leftarrow \mathrm{Norm}(h + \alpha_A(h_A - h))$
$h_M \leftarrow \mathrm{MLP}(\mathrm{RMSNorm}(h))$	$h_M \leftarrow \mathrm{Norm}(\mathrm{MLP}(h))$
$h \leftarrow h + h_M$	$h \leftarrow \mathrm{Norm}(h + \alpha_M(h_M - h))$
Final: $h \leftarrow \mathrm{RMSNorm}(h)$	

Transformer vs. Normalized Transformer.

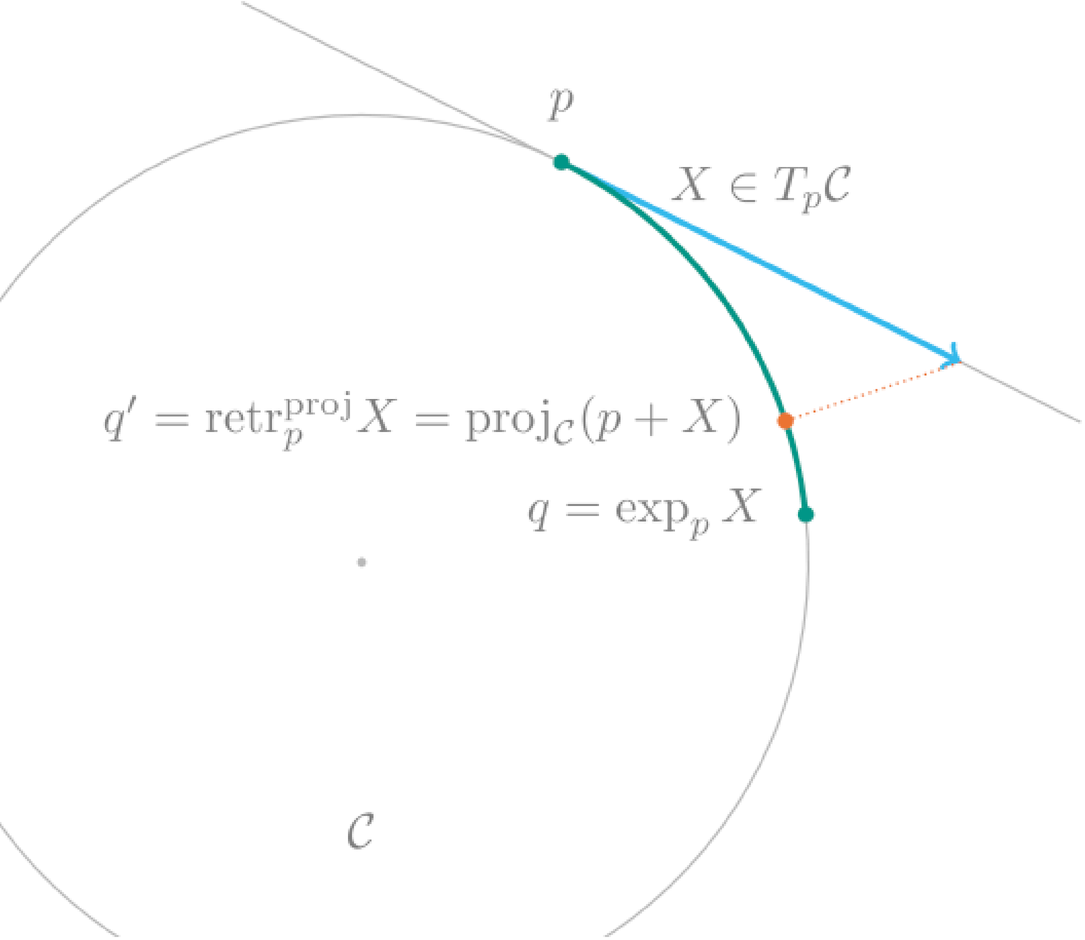
II.B. Totally Normlization!

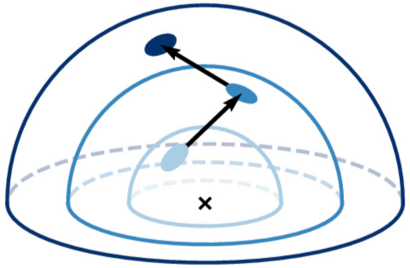
- After normalization, the outputs from the attention and MLP blocks ($\mathbf{h}_A$ and $\mathbf{h}_M$) can be seen as **target points** on a hypersphere. The vectors $\mathbf{h}_A - \mathbf{h}$ and $\mathbf{h}_M - \mathbf{h}$ act as **direction vectors** pointing toward these goal points.

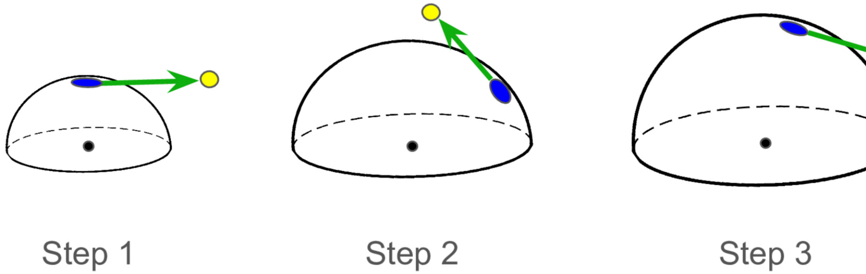
$$h_A \leftarrow \text{Norm}(\text{ATTN}(h))$$
$$h \leftarrow \text{Norm}(h + \alpha_A(h_A - h))$$
$$h_M \leftarrow \text{Norm}(\text{MLP}(h))$$
$$h \leftarrow \text{Norm}(h + \alpha_M(h_M - h))$$

II.B. Totally Normlization!

- After normalization, the outputs from the attention and MLP blocks ($\mathbf{h}_A$ and $\mathbf{h}_M$) can be seen as **target points** on a hypersphere. The vectors $\mathbf{h}_A - \mathbf{h}$ and $\mathbf{h}_M - \mathbf{h}$ act as **direction vectors** pointing toward these goal points.







II.B. Totally Normlization!

- After applying `Norm` for "inner" hidden updates, which retracts the updated state back onto the hypersphere (a retraction step), nGPT effectively performs updates along the **geodesic** on the manifold.

$$\mathrm{LERP}(\mathbf{a}, \mathbf{b}; \alpha) = (1 - \alpha)\mathbf{a} + \alpha\mathbf{b}$$

$$\mathrm{SLERP}(\mathbf{a}, \mathbf{b}; \alpha) = \frac{\sin(1 - \alpha)\theta}{\sin(\theta)}\mathbf{a} + \frac{\sin(\alpha\theta)}{\sin(\theta)}\mathbf{b}$$

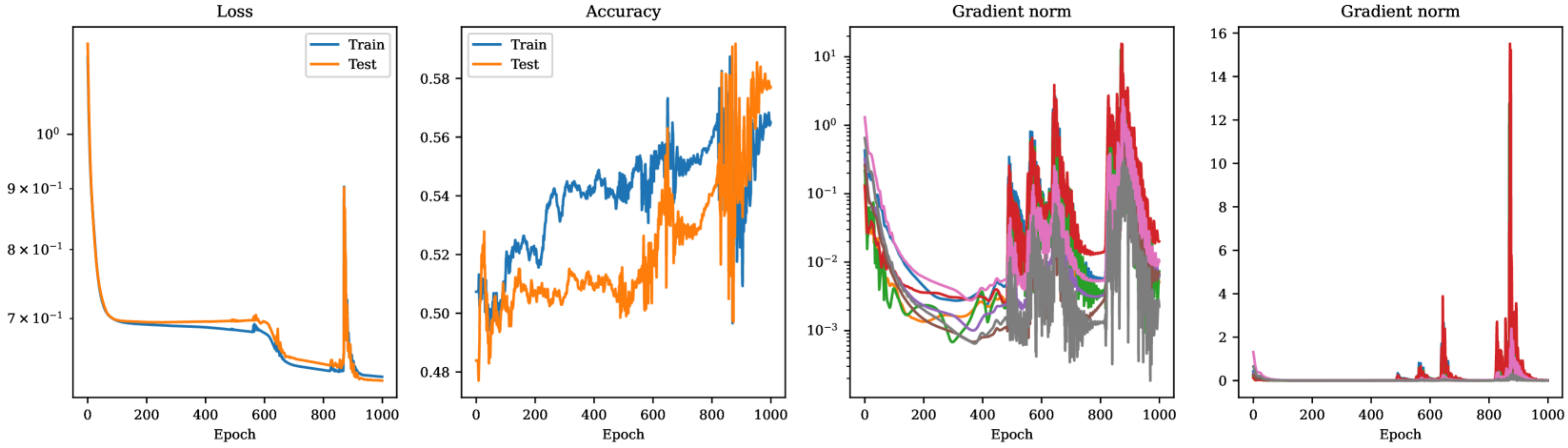
II.C. What's changed?

1. The shape of Loss Landscape

Loss landscape: the **surface** of global loss function $\mathcal{L}$ on parameter space Θ

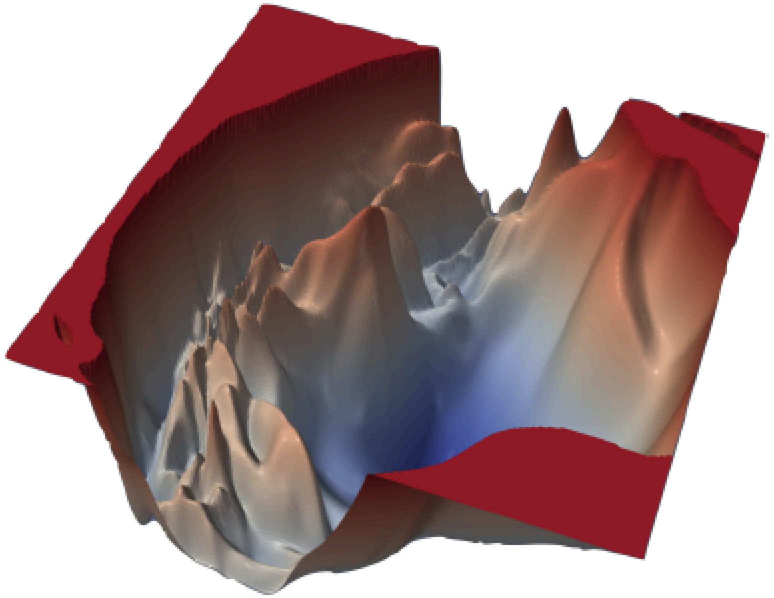
As Odonnat2024b, Odonnat2024b shown:

- loss spikes usually co-occur with **high gradient norm** updates
- loss spikes are linked to the **high curvature** of internal network functions.

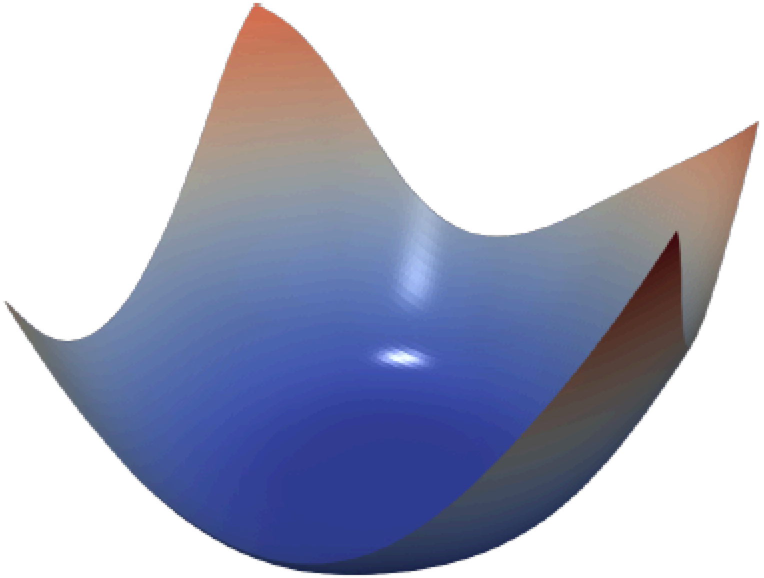


2. A better loss landscape for optimizaiton

Researchers (e.g. Li2017) shown that: specific architecture design can make neural network's optimization more **smooth**. e.g. skip-connection.



(a) ResNet-110, no skip connections

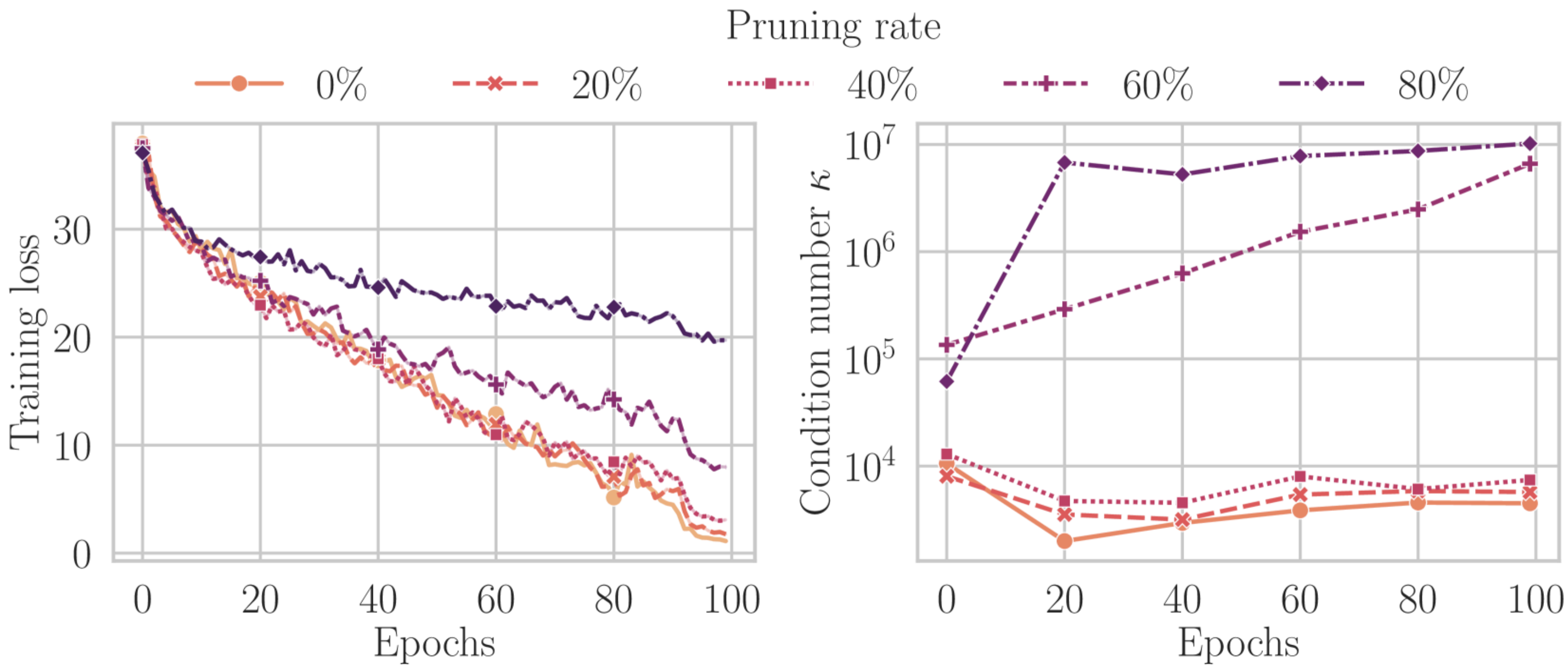


(b) DenseNet, 121 layers

2. A better loss landscape for optimizaiton

From the perspective of **condition number** of matrix, Zhao2024:

- the matrix's conditional number increase as the network grow deeper ($O(L^2)$)
- large conditional number indicate that the loss landscape has directions with significant differences in **curvature**

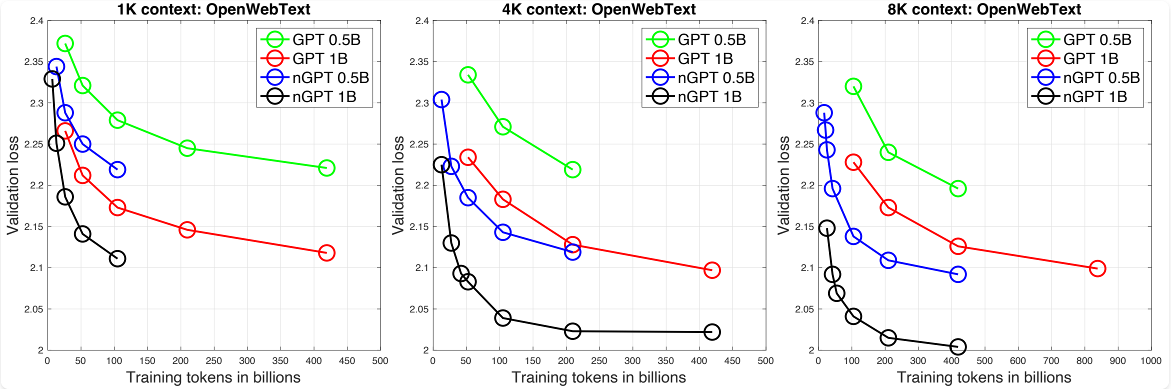


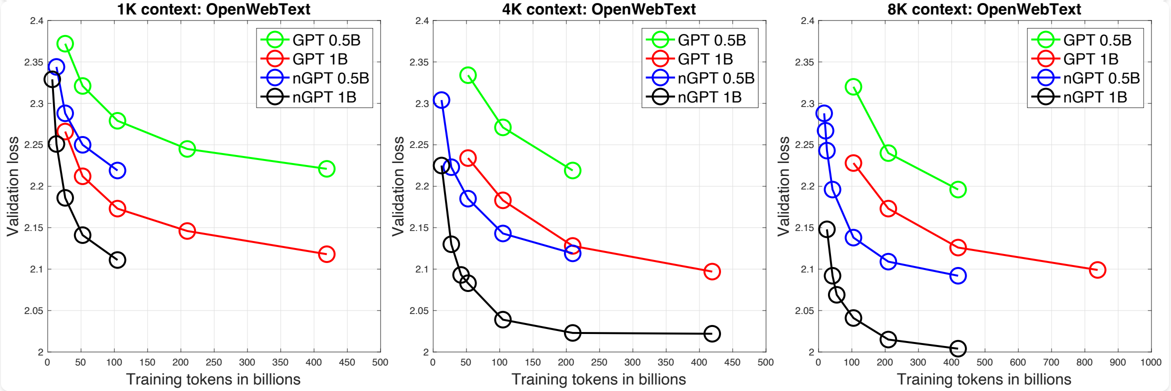
III. Epilogue

III.A. Results

1. Faster

nGPT's results are imprising: The training of 0.5B and 1B nGPT models is about **4x, 10x and 20x faster** (in terms of tokens) on 1k, 4k and 8k context lengths, respectively.

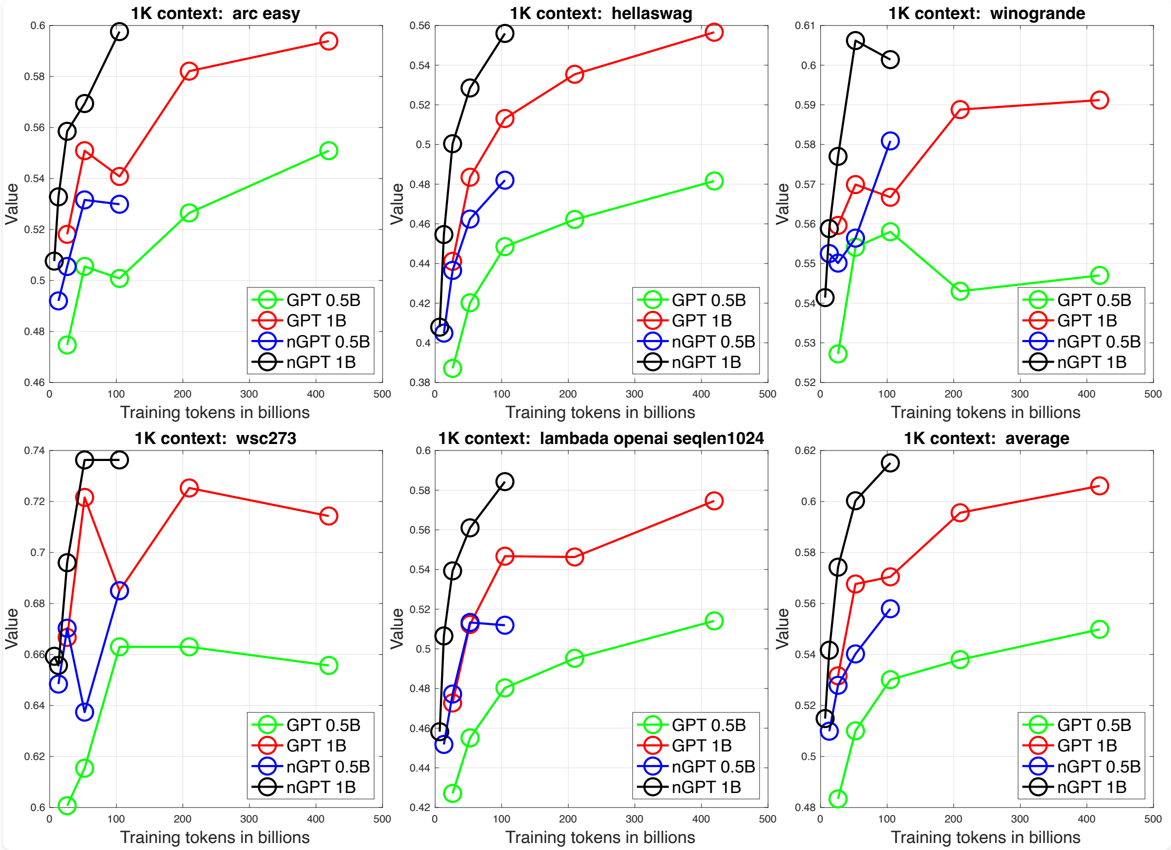




III.A. Results

2. Better

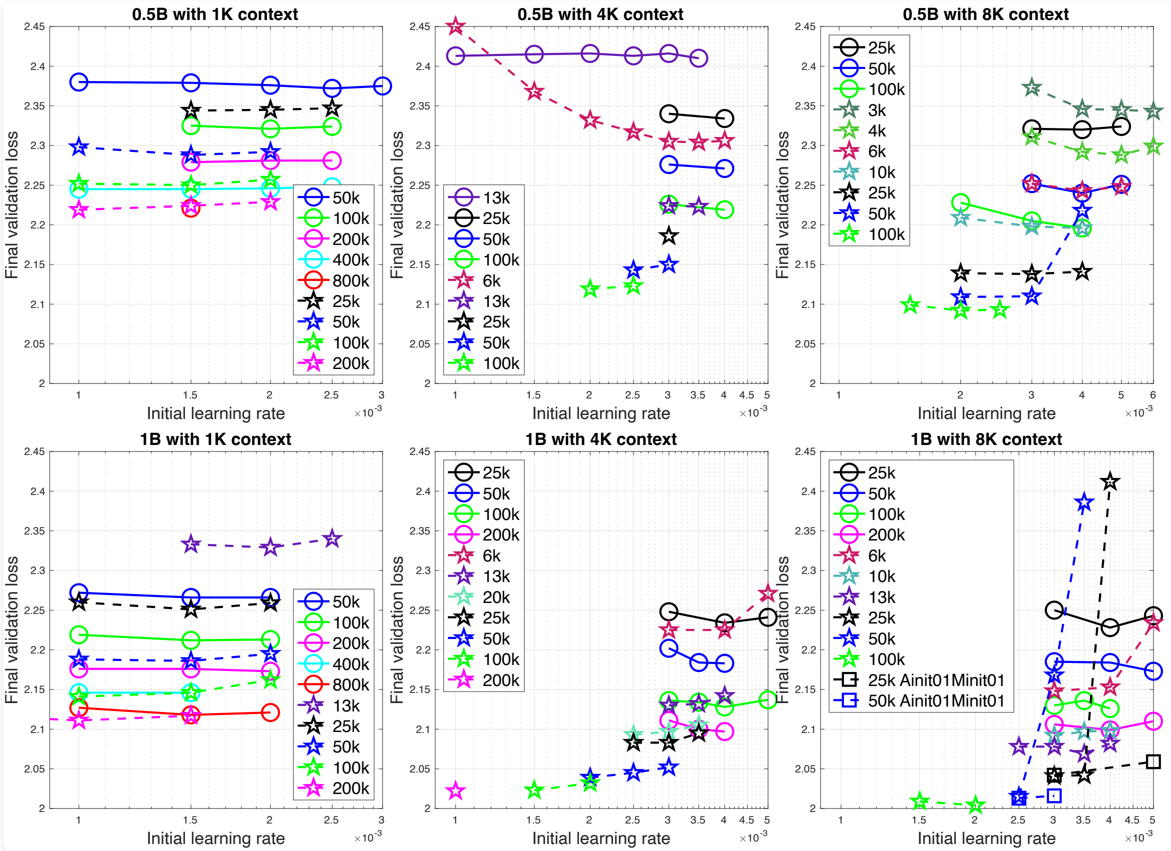
nGPT's results are imprising:



III.A. Results

2. Better

nGPT's results are imprising:



III.A. Results

3. Why Faster&Better?

- According to Section 2.3, figure 6 and Apendix A.2, in fact, astonishingly, nGPT learns to apply only **modest** eigen learning rates to update, and shares **similar** optimal (initial) learning rate with baseline GPT2. But much much faster, why?

III.A. Results

3. Why Faster&Better?

According to Section 2.3, figure 6 and Apendix A.2, in fact, astonishingly, nGPT learns to apply only **modest** eigen learning rates to update, and shares **similar** optimal (initial) learning rate with baseline GPT2. But much much faster, why?

- **Firmer update steps**
- **Reshapedthe loss landscape**

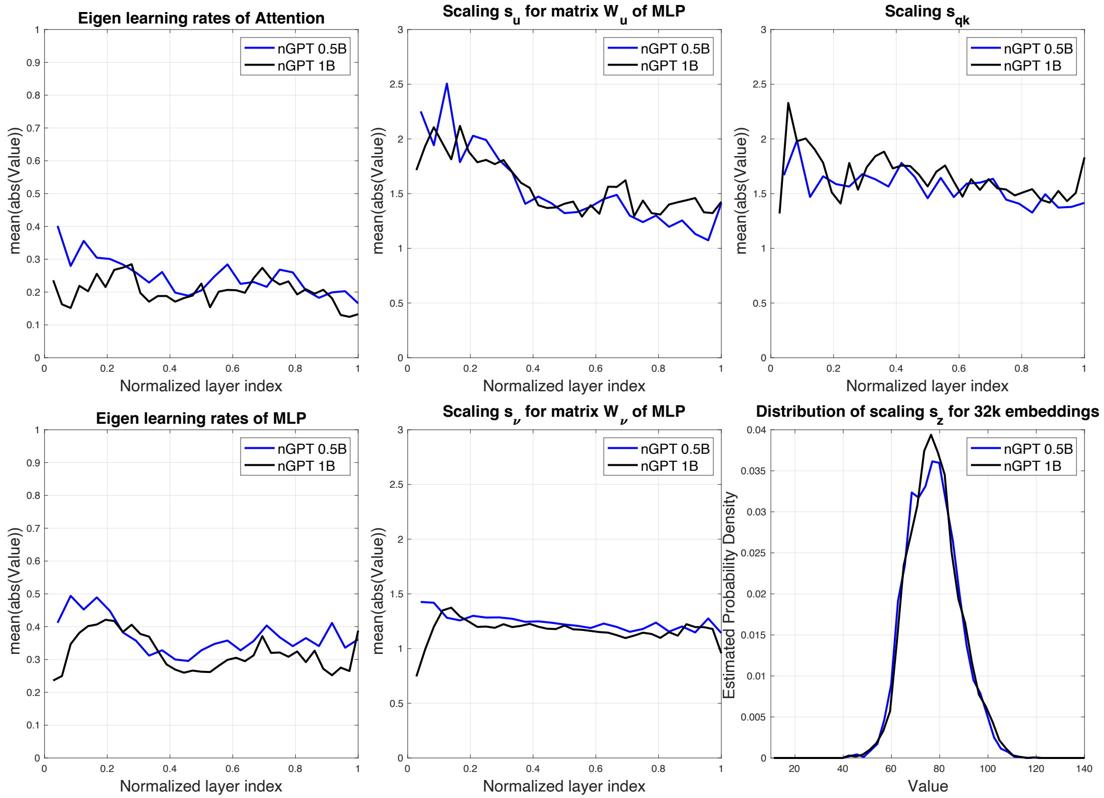
III.A. Results

3. Why Faster&Better?

- **Firmer update steps**

  - The update steps of model's inner states, $\alpha_X(\mathbf{h_X} - \mathbf{h})$, recall figure 6, the inner updates that controlled by α_A, α_M are surely modest, ranging from 20%~40%. We can see these "modest" as **firmer**, more **efficient** information updates.

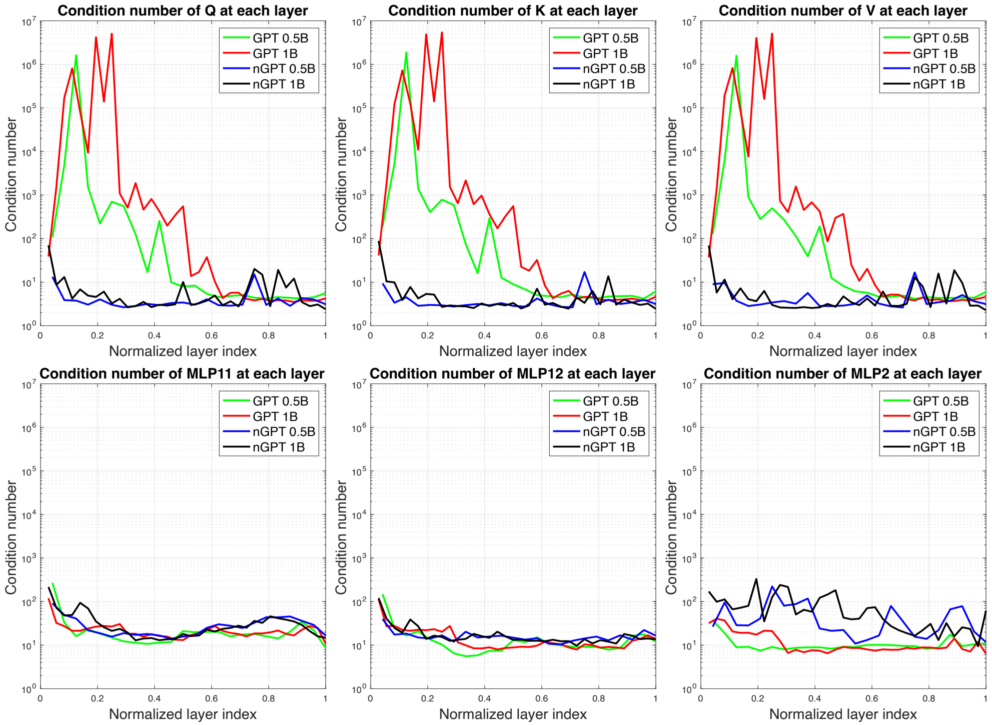
  - Since the inner update of nGPT is firmer, the quality of gradient signals that Adam received is better, leading to **efficient** parameter updates.



III.A. Results

3. Why Faster&Better?

- **Reshapedthe loss landscape**
  - frequently normalization and hypersphere constraint, by constraining the parameter search space (the zone that optimizer "walk" on), such that it is more friendly, more **navigable** for AdamW to iterate.
  - Figure 5 shows that the conditional number of nGPT’s weight matrices are lower, hence uttering more **stable** , uniform, mathematically healthy gradient updates,



III.A. Results

4. Summary

faster convergence means: When walking towards a more generalizable solution,

- each step is a **high quality** optimizing update,
- within a **smoother** loss landscape for optimizer to wander,
- more **effectively**, **directly**.
- Hence, promising a better result, under similar training budgets.

III.B. Takeaways

1. the l_2-norms of embedding vectors, weight matrices columns vectors, hidden vectors… are **unconstrained** in GPT2

III.B. Takeaways

1. the l_2-norms of embedding vectors, weight matrices columns vectors, hidden vectors… are **unconstrained** in GPT2
2. normalized them on the unit **hypersphere**
3. - all inner-products are cosine similarity, which is bounded between $[-1, 1]$

III.B. Takeaways

1. the l_2-norms of embedding vectors, weight matrices columns vectors, hidden vectors… are **unconstrained** in GPT2

2. normalized them on the unit **hypersphere**

3. **Riemannian** gradient descent: updating gradients on hypersphere, with eigen-learning-rate

III.B. Takeaways

1. the l_2-norms of embedding vectors, weight matrices columns vectors, hidden vectors… are **unconstrained** in GPT2
2. normalized them on the unit **hypersphere**
3. **Riemannian** gradient descent: updating gradients on hypersphere, with eigen-learning-rate
4. **FASTER** training, **BETTER** results

III.C. Open Quesitons

1. Really? "baseline" Transformer?

> What is still taken for granted in Transformer? All that is solid melts into air...

- **w/o Attention Heads**
  - Removing **attention heads** in transformer (Voita2019, Michel2019) can maintain performance.

1. Really? "baseline" Transformer?

What is still taken for granted in Transformer? All that is solid melts into air...

- **w/o Normalization**

  - Zhu2025 replaces Transformer normalization layers (like LayerNorm or RMSNorm) with a very simple element-wise operation DyT(x) = $\gamma \times \tanh(\alpha x) + \beta$.

$$\tanh(\alpha x) = \alpha x - \frac{\alpha^3 x^3}{3} + O(x^5)$$

2. Really? Hypersphere?

- **Intrinsic Dimension:** While neural activations (vectors) can be very high-dimensional (e.g., thousands of dimensions), their actual arrangement might be simpler locally. It might be possible to describe their local structure with fewer 'directions' (dimensions) after suitable transformations. This 'fewer number of directions' is the estimated 'intrinsic dimension.'
- For **tree-like** or hierarchical data, significant research (e.g. Ganea2018, Tifrea2018, Chami2019, Bachmann2019, Skopek2019, Shimizu2020, Mettes2022, Yang2024 on HyperbolicNN, Ermolov2022, Bdeir2023 on hyperbolic ViT)) suggest that non-Euclidean geometry is a promising direction, from embedding space design to model architecture.

Thank You & Further Discussion

Questions?

IV. Appendix

A. Faster and Better

Data level

- high-quality, large-scale datasets
- data-preprocessing, pipeline optimization
- ...

Architecture level

- efficient attention mechanisms, e.g. Flash Attention Dao2022
- proper optimizing techniques, e.g. optimizers, warmup, weight decay, dropout
- ...

Engineering tricks

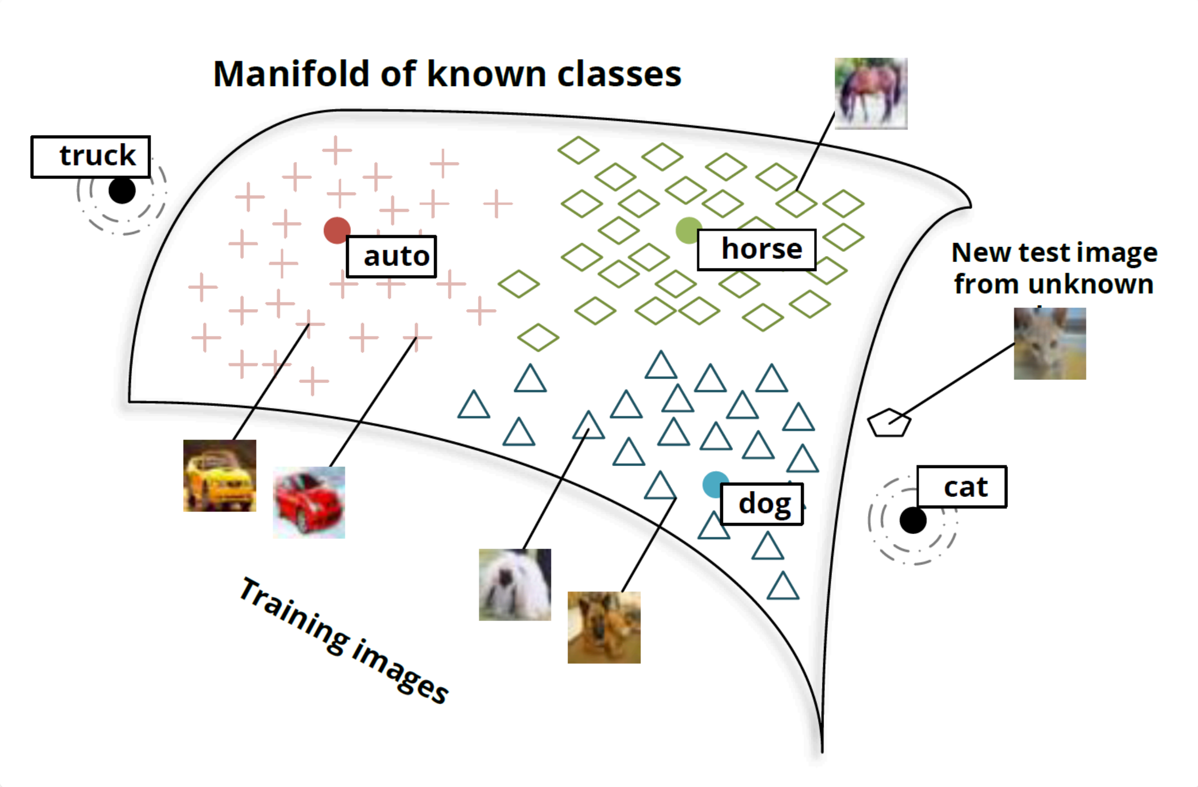
- mixed precision training
- parallelism strategies
- ...

IV.B. Manifold Hypothesis

Where do high-dimensional data reside on?

Instead of filling the entire $R^{d_{\text{model}}}$ space, meaningful **word embeddings** (or hidden states) reside on one or more **low-dimensional representation manifolds**[1] (some locally Euclidean spaces) possessing specific geometric structure.

For instance, (semantically) similar items are closer in the embedding space; this proximity might be the geodesic distance on a curved manifold.

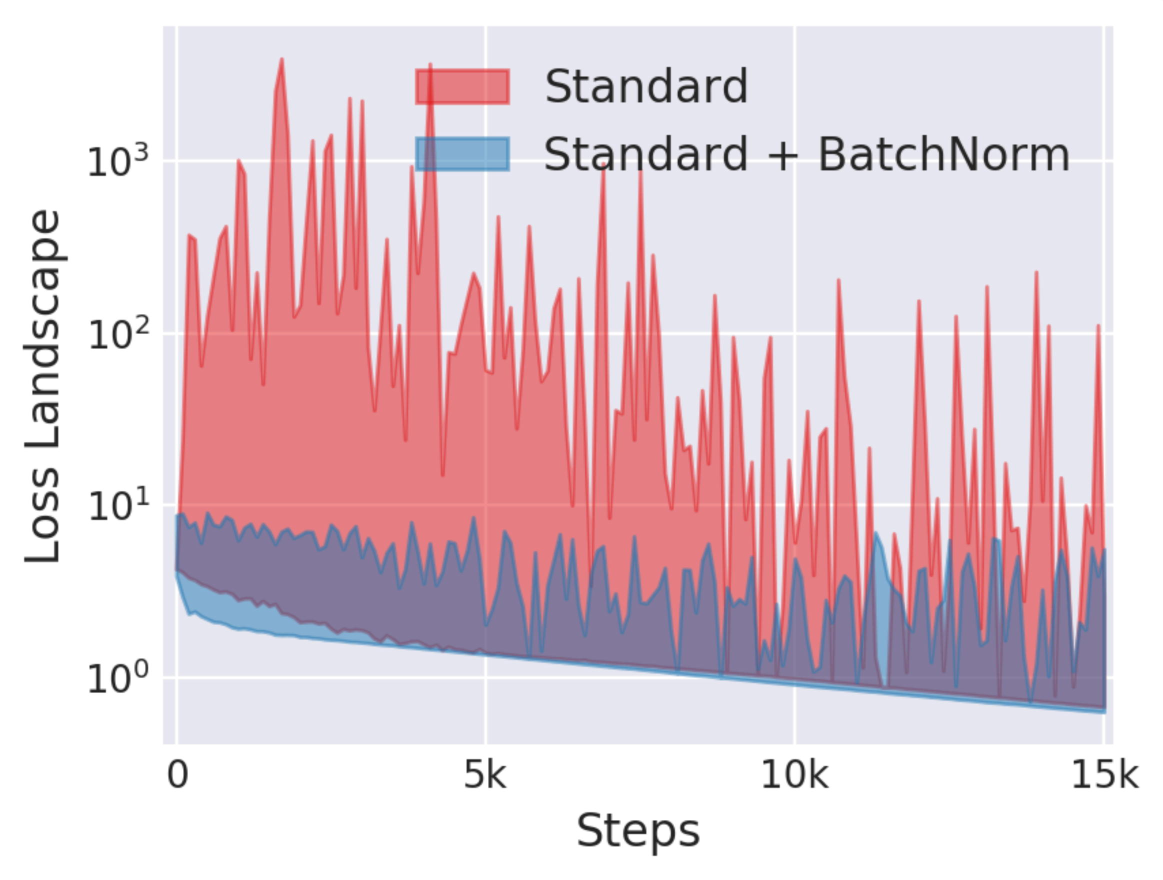


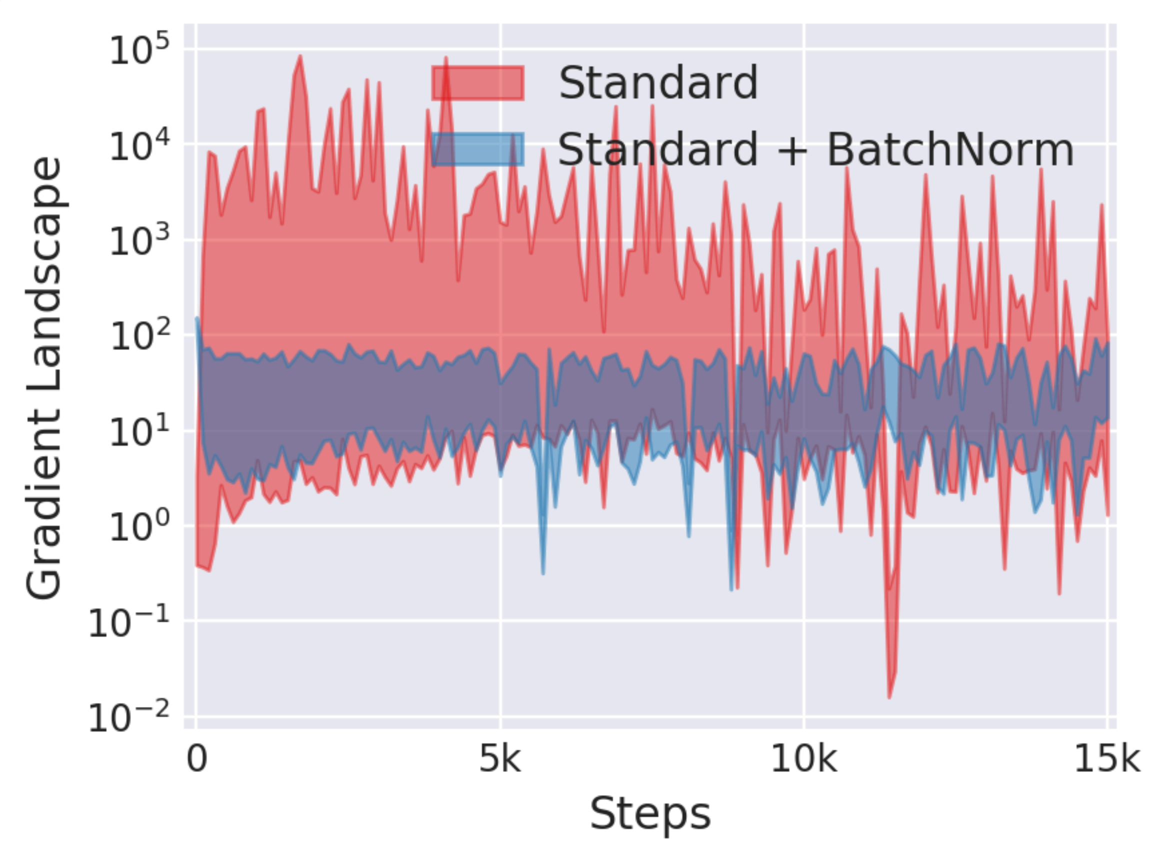
1. one can refer to blog-Colah-a, blog-Colah-b, Robinson2025 for fun. ↩

IV.C. Loss Landscape

1. Towards a better landscape

Researchers (e.g.Santurkar2018,Balestriero2022) shown that: normalizaiton optimizes **loss landscape**, not only smooothing it, but also let model to stand at a better optimizing starting point.



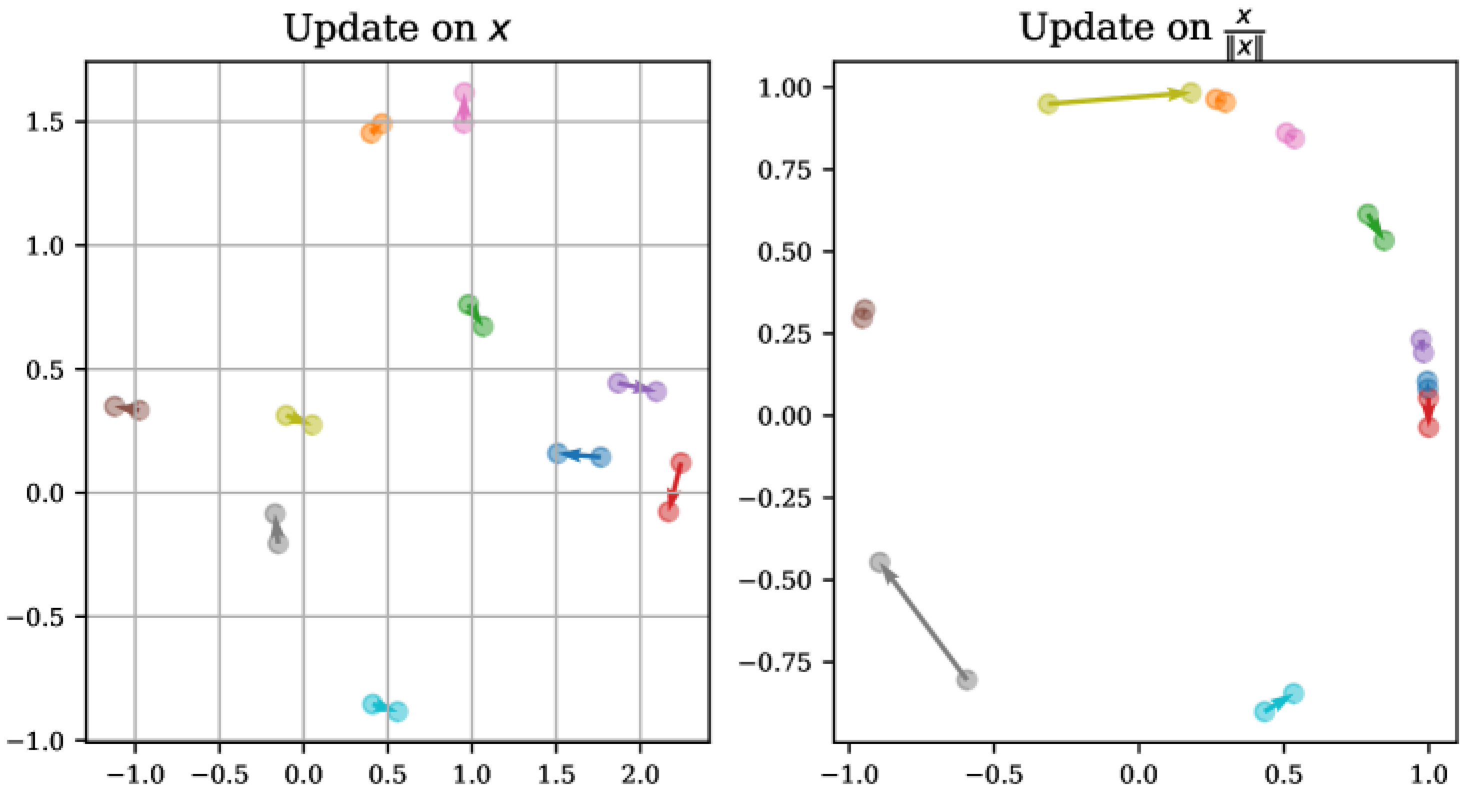


IV.C. Loss Landscape

2. Towards the mininum…

> **Loss spikes** are linked to the **high curvature** of internal network functions. A small update to an element $\mathbf{x}$ can result in a substantial change to its normalized version $\frac{\mathbf{x}}{||\mathbf{x}||_2}$, significantly altering the network's subsequent behavior. – Odonnat2024b

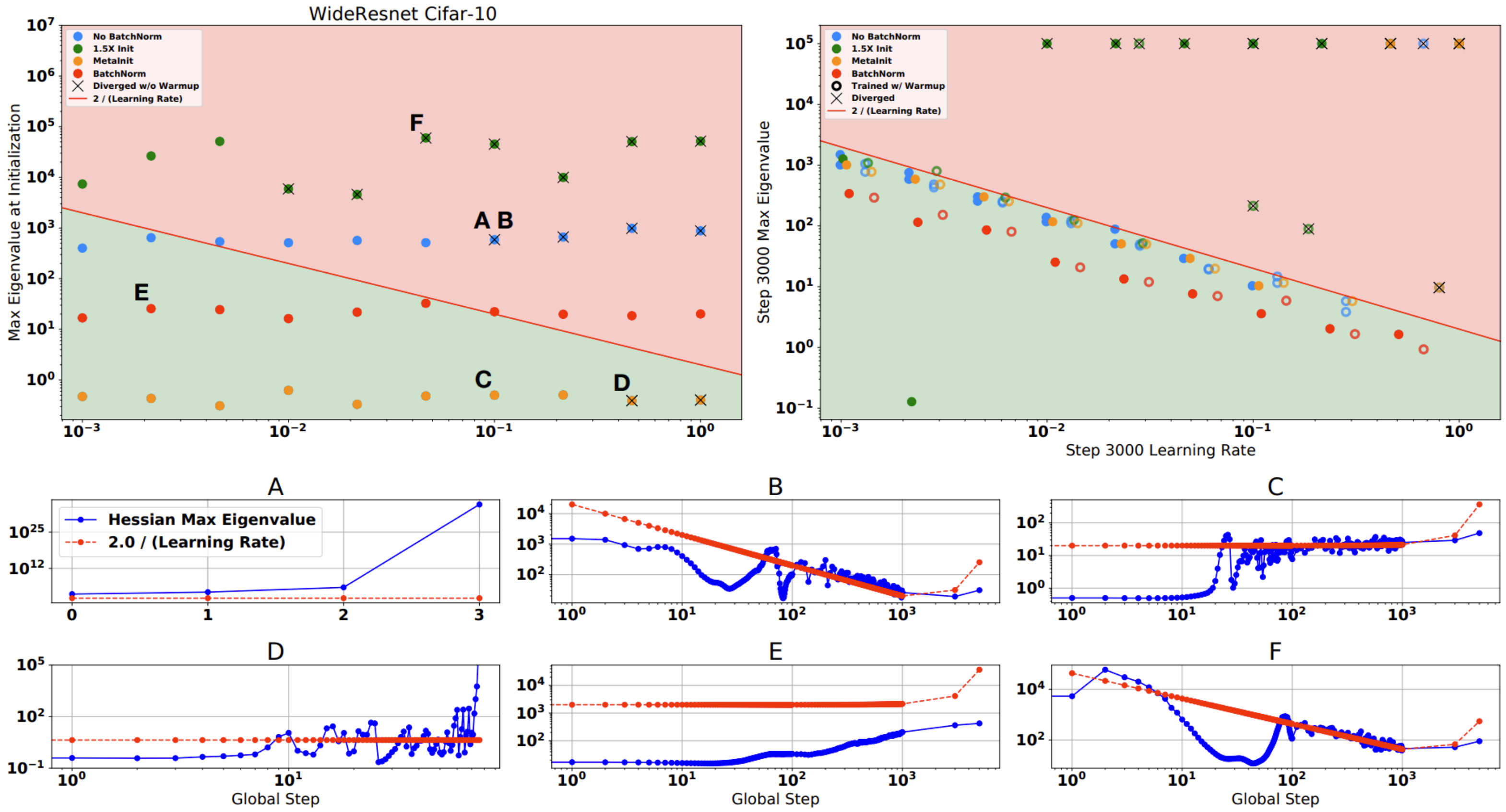
such as $f(x) = \frac{\mathbf{x}}{||\mathbf{x}||_2}$ near the origin:



IV.C. Loss Landscape

2. Towards the mininum…

From the perspective of **optimizing progress**, Gilmer2020 shown that: successful model and hyperparameter choices allow the **early optimization trajectory** to either avoid – or navigate out of – regions of **high curvature** and into **flatter** regions that tolerate a higher learning rate.

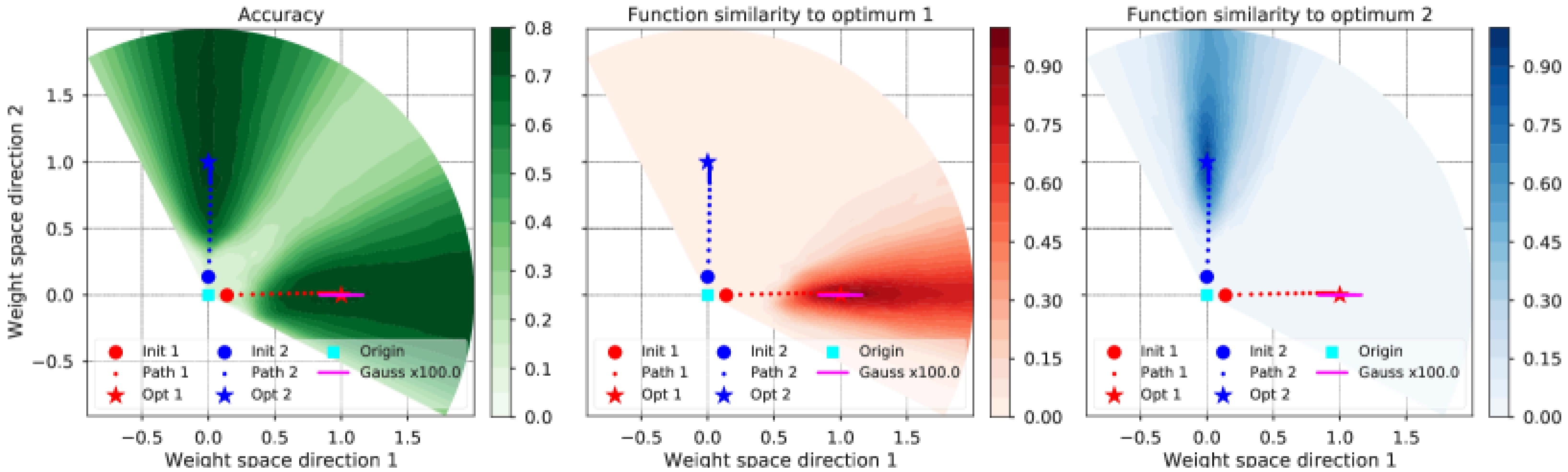


IV.C. Loss Landscape

2. Towards the mininum...

From the perspective of initializaiton, researchers(e.g. Fort2018, Fort2019a, Fort2019b) shown that,

- Random initializations explore entirely **different** modes,
- Faster training implys exploring the relatively **flat** zones ,by all means

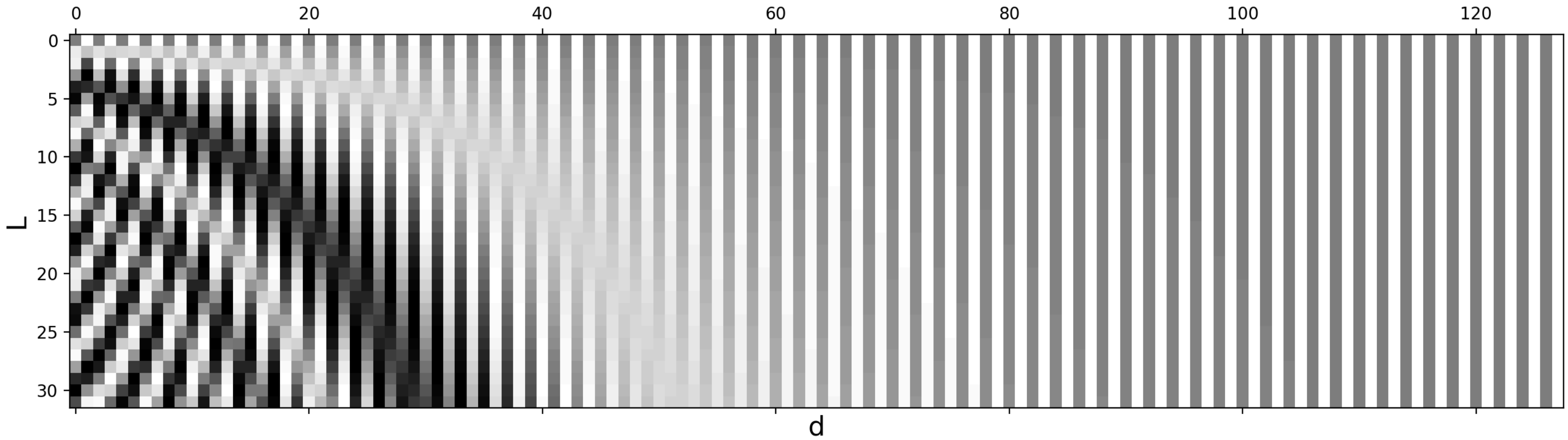


IV.D. Which PE?

0. Positional Encoding vs Positional Embedding

Denote the size of vocabulary as V, token to predict as t_i, token input sequence as $\mathbf{x} = \{t_1, t_2, \ldots, t_{i-1}\}$, learnable, unconstrained embedding matrices $\mathbf{E}_{\text{input}}, \mathbf{E}_{\text{output}} \in \mathbb{R}^{V \times d_{\text{model}}}$

- self-attention operation is *permutation invariant*, it is important to use proper positional encoding to provide *order information* to the model. – blog-weng



IV.D. Which PE?

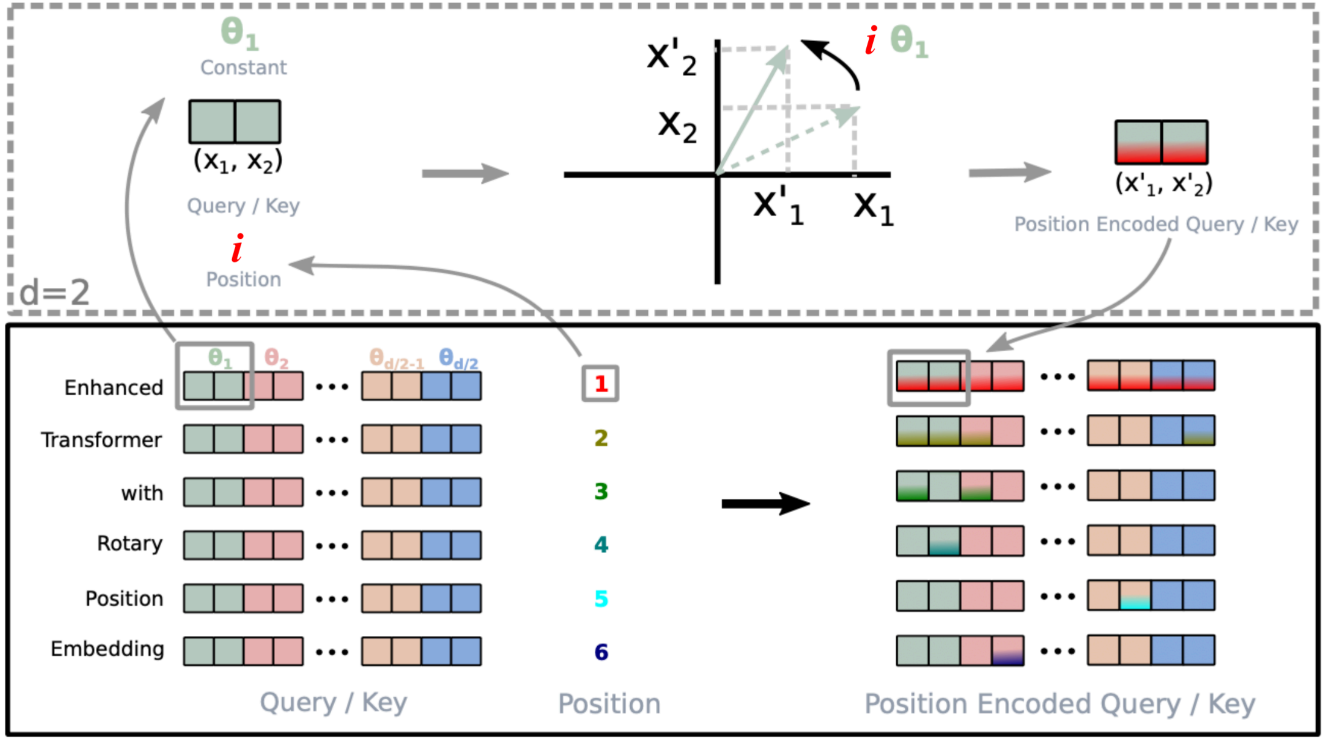
1. Positional Encoding vs Positional Embedding

Can we give the position information to input tokens with specific embedding, thus **differentiating** tokens based on their locations?

IV.D. Which PE?

2. RoPE: Su2021

- By using a rotation matrix, Rotary Positional Embedding (RoPE) (Su2021) unifies advantages of both ***absolute*** [1] and ***relative*** [2] positional embedding schemes.
- images from blog-weng

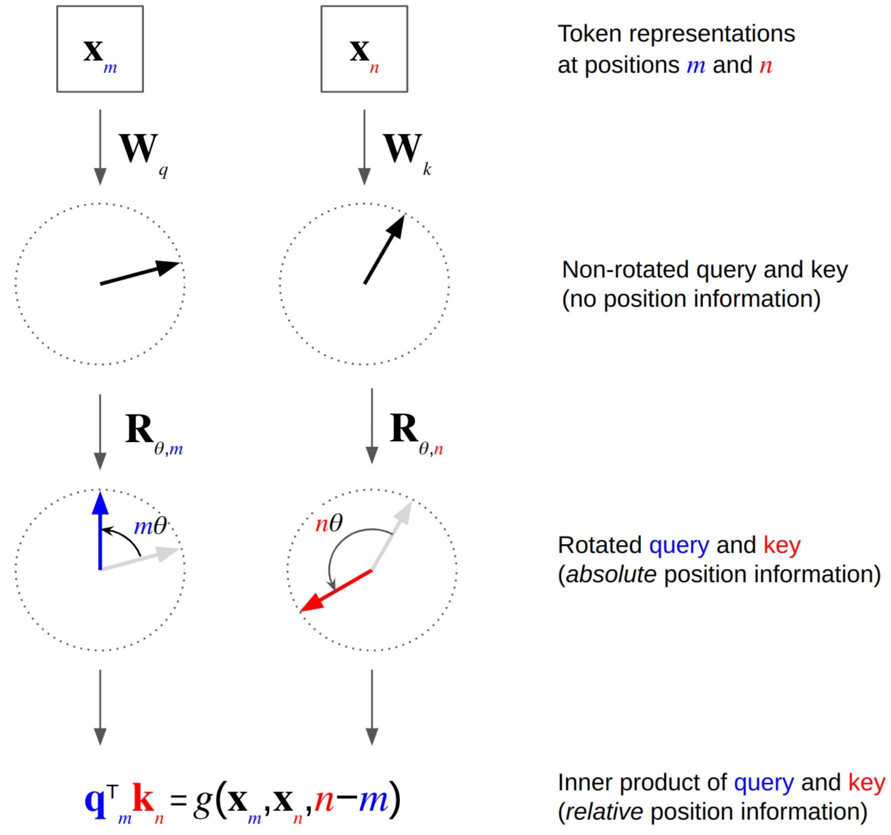


1. Su2024: through sinusoidal or learnable embedding ↩
2. Su2024: by adding relative biases ↩

IV.D. Which PE?

2. RoPE: Su2021

RoPE first multiplies queries and keys with a rotation matrix i.e. it rotates $\mathbf{W_q x_m}$ and $\mathbf{W_k x_n}$ before taking their inner product. The rotation matrix is a function of ***absolute*** position. Calculating the inner products of rotated queries and keys results in an attention matrix that is a function of ***relative*** position information only. -blog-krasserm



IV.D Output Logits

Denote the "transformed" output vector as $\mathbf{h}_i \in \mathbb{R}^{d_{\text{model}}}$, i as the index of position for predicted sequence, and $\mathbf{z}_i \in \mathbb{R}^V$ as the unconstrained probabilities for each token (aka `logits`),

$$\mathbf{z}_i = \mathbf{E}_{\text{output}} \mathbf{h}_i$$

An overview of the evolution and types of activation functions:

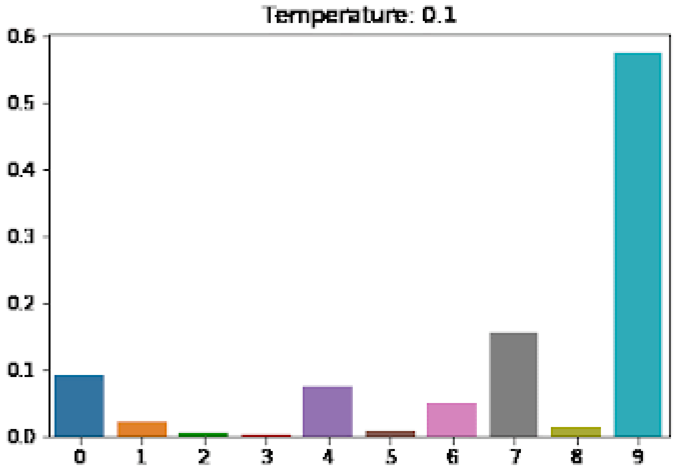
IV.D Output Logits

As usual, our logits $\mathbf{z}_i$ come to the cradle of `softmax` to convert them into probabilities. Say the chosen token (as prediction token) as τ_j, then denote z_{i,τ_j} as the corresponding logit, then:

$$\mathbb{P}(\tau_j|\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}) = \frac{\exp(z_{i,\tau_j})}{\sum_{k=1}^{V} \exp(z_{i,\tau_k})}$$

IV.D Output Logits

Feeling like missing something right? Yes, your intuition is sharp, we add a temperature term $\mathbf{s}_z \in \mathbb{R}^V$:



![](https://charlielehman.github.io/post/visualizing-tempscaling/tempscale_10.gi