Transformers without Normalization

Étienne Perron, Edgar Kappauf, Gaetan Butault
Department of Computer Engineering and Software Engineering, Polytechnique Montréal, Canada
April 26, 2025

Abstract

This report investigates the viability of training Transformers without relying on standard normalization layers such as Layer Norm (LN) or RMSNorm. We also put focus on the Dynamic Tanh (DyT) proposed by [Zhu *et al.*, 2025] and propose our own variant of it. We also compare three different way of placing the normalizations layers in the model.

1 Introduction

Normalization layers have long been recognized as key components in deep neural networks. Starting with Batch Normalization (BN) in 2015 [Ioffe and Szegedy, 2015], which is credited with accelerating and stabilizing training in deep convolutional models, various normalization methods have since emerged. For sequence and Transformer models [Vaswani *et al.*, 2017], Layer Normalization (LN) was introduced to normalize activations on a per-token basis, quickly becoming the standard in Transformers because it stabilizes training without depending on batch size. A variant known as Root Mean Square Normalization (RMSNorm) [Zhang and Sennrich, 2019] offers similar benefits and has gained traction in large-scale models like T5 [Raffel *et al.*, 2019] and LLaMA [Touvron *et al.*, 2023]. Overall, modern architectures typically include a normalization layer to enhance stability, convergence speed, and generalization. Nevertheless, researchers have investigated whether deep networks (particularly Transformers) could be trained without normalization. Early approaches, such as Fixup initialization [Zhang et al., 2019] and SkipInit [De & Smith, 2020], adopted special parameter initializations or learnable scalars in residual connections to reduce reliance on normalization layers. Other methods, like σ-Reparameterization [Zhai et al. 2023], maintain stable activations by controlling spectral norms. Despite these efforts, in practice, thorough tuning is often required, and most models still default to normalization.

[Zhu *et al.*, 2025] propose a simpler alternative: replacing LN or RMSNorm with Dynamic Tanh (DyT). Observing that LN effectively scales mid-range activations and squashes extremes in a tanh-like manner, DyT applies $\tanh(\alpha x)$ with a learnable α, plus optional per-channel scaling and bias. This direct approach bypasses computing means or variances, treating each activation independently and removing the need for batch-level or token-level statistics. Notably, Transformers using DyT generally match or exceed the performance of their normalized counterparts on diverse tasks, without significant hyperparameter adjustments, and also see modest improvements in training and inference speed.

By eliminating normalization layers, DyT challenges the assumption that normalization is indispensable for stable, high-performing Transformers. Its results suggest the core benefits of normalization, such as controlled activation scales, can be replicated through a straightforward, element-wise function. This opens new avenues for future work on normalization-free architectures and the broader implications for deep network design.

2 Background

Modern Transformers use *normalization layers* (like LayerNorm or RMSNorm) to make training more stable, keep activations under control, and speed up convergence. The results of [Zhu *et al.*, 2025] show that LayerNorm behaves like a scaled `tanh` function: values near zero change little, while very large or small values get squashed. This suggests that the key benefit of normalization may come from this *nonlinear squashing*, rather than from computing the mean and variance.

2.1 Dynamic Tanh (DyT)

To test this hypothesis, [Zhu *et al.*, 2025] introduce the **Dynamic Tanh** layer, defined for each activation x as:

$$\mathrm{DyT}(x) \;=\; \gamma\,\tanh\big(\alpha x\big) + \beta, \tag{1}$$

where

- α is a learnable scalar that rescales the global variance of activations;
- γ and β are learnable per-channel affine parameters identical to those in `LayerNorm`.

DyT is a simple replacement for normalization layers. It works on each value separately and doesn't need to compute

statistics, while still keeping activations within a safe range thanks to the `tanh` function.

2.2 Theoretical Approach

The theoretical foundation of Dynamic Tanh (DyT) is rooted in the observation that Layer Normalization (LN) in Transformers produces input-output mappings that resemble the S-shaped curve of a scaled `tanh` function [Zhu *et al.*, 2025]. Specifically, LN linearly scales activations near zero while non-linearly squashing extreme values, a behavior that stabilizes training by preventing activation explosion or vanishing gradients. This insight suggests that the primary role of normalization is not the computation of per-token statistics (mean and variance) but the non-linear transformation of activations to maintain them within a controlled range.

The DyT layer, defined in Equation 1, formalizes this hypothesis by applying a `tanh` function scaled by a learnable parameter α, followed by per-channel affine transformations (γ and β). The `tanh` function bounds activations within $[-1, 1]$, mimicking LN's squashing effect, while α adjusts the input range to approximate the inverse of the global standard deviation of activations. Unlike LN, DyT operates element-wise, eliminating the need for statistic aggregation and reducing computational overhead.

DyT achieves training stability through two mechanisms:

1. **Non-linear squashing**: The `tanh` function compresses extreme activations, preventing unstable gradient propagation, similar to LN's handling of outliers.
2. **Learned scaling**: The parameter α adapts during training to normalize the activation range implicitly, without explicit mean or variance computation.

This approach challenges the necessity of normalization layers by replicating their stabilizing effects with a simpler operation. DyT's generality allows it to replace LN or RMSNorm across various Transformer architectures (e.g., vision, language, speech) with minimal hyperparameter tuning, suggesting that non-linear transformations are the core of normalization's benefits. However, its limitations, such as reduced effectiveness in replacing Batch Normalization in ConvNets, highlight the need for further theoretical exploration. Nonetheless, these results question the idea that deep networks always need normalization layers, and suggest that we can build Transformers without them, leading to simpler and more efficient models.

3 Experiences

In this section, we analyze the results obtained from experimenting with different normalization strategies within the architecture. Normalization method like BatchNorm, LayerNorm and RMSNorm were tested in three configurations: applied before the main operation (Pre), after it (Post), or in both positions (Both) as shown in figure 1. The identity method was method was only used to serve as a baseline when no normalization was applied. Furthermore we also train without any normalization layers (identity function). Finally, we conducted a more detailed series of experiments focusing on DyT:

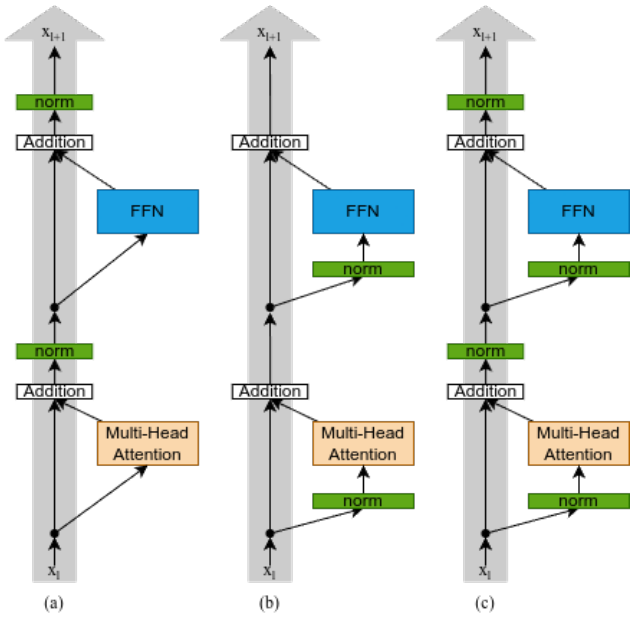

Figure 1: (a) Post-Norm Transformer layer; (b) Pre-Norm Transformer; (c) Both-Norm Transformer layer.

3.1 Comparison of Normalization methods

- **Positioning:** DyT was inserted in Pre, Post, and Both configurations relative to the main operation.
- **Alpha scaling:** We experimented with values of alpha of 0.05, 0.5 (default value) and sometimes 1.0.
- **Activation variations:** We tested with `Hardtanh` and `LeakyHardtanh`. Default activation function is `Tanh`.

We decided to test the different normalization methods on a traditional transformer, as specified by [Vaswani *et al.*, 2017]. The definition of the activation variations that we used for DyT are,

$$\text{Hardtanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases},$$

$$\text{LeakyHardtanh}(x) = 0.99 \cdot \text{Hardtanh}(x) + 0.01 \cdot x$$

and simply `Tanh(x)`.

[Zhu *et al.*, 2025] observe better learning using Tanh than HardTanh, we hypothesize that the cause is the complete lack of gradient outside of the [-1;1] range, LeakyHardtanh is created to test this hypothesis by allowing smaller gradient flow outside that range.

First, we compare the normalization methods BatchNorm, DyT, LayerNorm and RMSNorm in Table 1. As we can see, placing the normalization layer before the main operation always results in better performance. Unlike the authors of the original paper, we were not able to show that DyT yields slightly better performance than the other normalization techniques, it in fact ended up performing worse than identity function. We can also observe very low performance form DyT when placed in Post-norm or Both, we hypothesize this effect to alpha being lower than 1.0 and as such leading to vanishing gradients when applied repetitively to x. We then test this hypothesis in section 3.4. However we where not able to measure any differences between tanh, hardtanh, and leakyhardtanh for DyT.

	Normalization position		
	Post	Pre	Both
BatchNorm	1.74	**1.69**	1.73
DyT (tanh)	3.71	**1.78**	5.50
DyT (hardtanh)	3.26	**1.78**	3.37
DyT (leakyhardtanh)	3.25	**1.78**	3.23
LayerNorm	1.77	**1.70**	1.77
RMSNorm	1.78	**1.70**	1.77
Identity	∅	∅	**1.71**

Table 1: Loss of different normalization methods with different positions.

3.2 DyT at different learning rates

As second experiment, we tried to train the transformer with DyT and different learning rates as [Zhu *et al.*, 2025] indicates instability at higher learning rates. The results are shown in Table 2. As we can see, a substantial change in the hyperparameter value did not result in largely different performance compared to a layernorm baseline, which shows that the DyT normalization is stable with different learning rates. We where not able to reproduces the instabilities shown by [Zhu *et al.*, 2025].

Learning Rate	DyT Eval Loss	Layernorm Eval Loss
0.000025	2.67	2.53
0.000050	2.19	2.08
0.000100	1.34	1.70
0.000200	1.49	1.43
0.000400	1.31	1.29

Table 2: Loss of different normalization methods with DyT and different learning rates.

3.3 DyT at different dimensions

In our third experiment, we trained the model with a doubled hidden dimension size (512) to see how it would affect the performance. The results are available in Table 3 and show that increasing the hidden dimension did improve the performance of Hardtanh and LeakyHardtanh, but hindered Tanh when placed in Pre. When in Post, however, every activation function showed improved performance with a doubled hidden dimension. It is interesting to show that placing the DyT layer before the main operation resulted in substantial improvement in performance, no matter the hidden dimension size. This is consistent with our first observation described earlier.

	Pre		Post	
Hidden Dimension size	256	512	256	512
Hardtanh	1.78	**1.34**	3.26	3.22
LeakyHardtanh	1.78	**1.34**	3.25	3.12
Tanh	**1.31**	1.35	3.71	3.56

Table 3: Loss of different combinations of hidden dimensions and DyT functions for Pre and Post contexts.

3.4 DyT Post and vanishing gradients

We also wanted to learn more about the impact of α on the gradient flow and on the performance of the model with DyT Post to test hypothesis presented in section 3.1. Firstly, we can see on Figure 2 (Epoch 1) that the gradient keep it's norm through the layers with $\alpha = 1.0$ but completely vanishes with $\alpha = 0.5$ and $\alpha = 0.05$. However, Figure 3, which shows Epoch 5, proves that the gradient flow improves after a few iteration of training. This is because the DyT parameters (α, β and γ) are adjusted as the model trains and they are encouraged to learn a value that helps gradient flow. $\alpha = 0.05$ is the only model that couldn't learn to remove the vanishing gradient problem at epoch 5. This is due to α being initialized too low, so the model hasn't had the time to recover after 5 epochs. This prove our hypothesis, and we can confidently hypothesize that similarly an alpha above 1.0 would likely lead to exploding gradients, this however wasn't tested in any of our experiments.

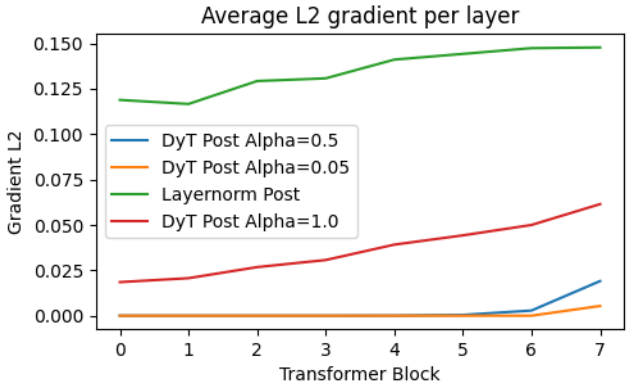

Figure 2: Gradient L2 norm across layers at Epoch 1. Comparing DyT Alpha values to Layernorm Baseline.

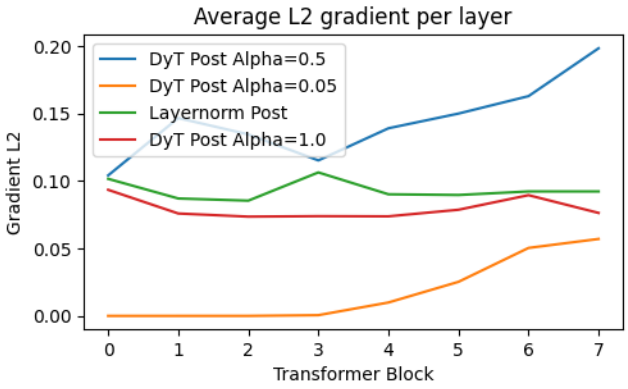

Figure 3: Gradient L2 norm across layers at Epoch 5. Comparing DyT Alpha values to Layernorm Baseline.

In Table 4, we once again evaluate the impact of α, but now on the performance. We compare different positions and different activation function. The results show that the models that are not suffering from vanishing gradients always results in better performance, no matter the activation function.

Activation function	Position	Alpha	Eval Loss
Hardtanh	Post	0.05	6.64
		0.50	3.26
		1.00	**2.18**
LeakyHardtanh	Post	0.05	6.67
		0.50	3.25
		1.00	**2.18**
Tanh	Post	0.05	5.72
		0.50	3.71
		1.00	**2.33**

Table 4: Eval Loss for different DyT functions and Alpha values.

3.5 DyT Pre vs DyT Post

We observe in table 5 that Pre is always the best configuration, once again being consistent with previous observations, but also that the best results are the same for each activation function. This shows that using DyT with $\alpha = 1.0$ will always result with the best performance, no matter which activation function is chosen, however it's important to note that this claim is shown to be wrong by [Zhu *et al.*, 2025] and that optimal alpha for Pre-DyT (Post-DyT was not tested by the authors) is shown to be dependent on embedding dimensions, as this effect is visible for dimensions larger than 4096 we where not able to test this due to hardware limitations.

Activation function	Position	Alpha	Eval Loss
Hardtanh	Post	0.05	6.64
		0.50	3.26
		1.00	**2.18**
	Pre	0.05	1.87
		0.50	1.78
		1.00	**1.78**
LeakyHardtanh	Post	0.05	6.67
		0.50	3.25
		1.00	**2.18**
	Pre	0.05	1.87
		0.50	1.78
		1.00	**1.78**
Tanh	Post	0.05	5.72
		0.50	3.71
		1.00	**2.33**
	Pre	0.05	1.87
		0.50	1.78
		1.00	**1.78**

Table 5: Eval Loss for different DyT functions and Alpha values.

4 Critical Analysis of Our Approach

To further validate our findings, it would be interesting to experiment with more extreme learning rates: both much lower and much higher—to observe how sensitive DyT is to these changes compared to traditional normalization methods. Additionally, running similar experiments on different Transformer architectures (e.g., deeper models or variants used in vision and speech) could help assess the generalization of DyT across tasks and domains. Also, we initially scheduled experiments based on training time, but had to adjust several settings along the way. For a more rigorous analysis, it would be important to control and report training time consistently across all configurations.

References

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[Raffel *et al.*, 2019] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[Touvron *et al.*, 2023] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[Zhang and Sennrich, 2019] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *CoRR*, abs/1910.07467, 2019.

[Zhu *et al.*, 2025] Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization, 2025.