TnT - The Normalized Transformer: Eigen-Learning Rates Meet Stepwise Normalization

Anonymous ACL submission

Abstract

Traditional Transformer architectures process input sequences through self-attention mechanisms and feed-forward networks operating in Euclidean space. However, this representation may not fully capture the complex, hierarchical relationships inherent in structured data. To address this limitation, we introduce N Transformer, an enhanced encoder-decoder architecture that applies normalization to all embeddings, attention mechanisms, and hidden states to maintain unit norm. N-Transformer ensures that token representations traverse the surface of a hypersphere with each layer contributing a displacement towards the target output predictions.These displacements are defined by the MLP and attention blocks, whose vector components also reside on the same hypersphere. Our experiments demonstrate that N-transformer achieves superior performance compared to standard transformer architectures within a comparable parameter range. Through our experiments we show that N-transformer has superior performance compared to standard transformer architectures on the same parameter scale. Furthermore, N-transformer achieves an average 11.31% improvement on GLUE tasks and great improvements on the Super-GLUE tasks, 11.5% on WiC and 3.79% on BoolQ. Additionally, pre-training experiments on the OpenWebText dataset reveal a 7% lower validation loss than the baseline transformer model after the same number of training iterations.

1 Introduction

The transformer (Vaswani et al., 2017) architecture has become the foundation of modern natural language processing, with numerous variants proposed to enhance its capabilities across different tasks. While much research has focused on architectural modifications like attention mechanisms and positional encodings such as RoPE(Su et al., 2024), the role of normalization in these models remains an active area of investigation. It has been observed that applying various normalization techniques is beneficial (Salimans and Kingma, 2016), leading to experiments incorporating normalization layers such as LayerNorm (Ba, 2016) and RMSNorm (Zhang and Sennrich, 2019) at nearly every possible position within the network (Xiong et al., 2020).

Another approach to the model normalization is through controlling the norm of weights using weight decay (Loshchilov, 2017).Recent studies (D'Angelo et al., 2023) suggest reevaluating the role of weight decay and taking a closer look at rotations rather than focusing solely on vector norms (Kodryan et al., 2022).

The recently released nGPT paper by NVIDIA (Loshchilov et al., 2024) serves as a major source of inspiration for our work. We extend the functionality of the methodology outlined in their technical report by implementing a Normalized encoder-decoder architecture with representation learning on the hypersphere.

We define the Normalized Transformer (N-transformer), which incorporates normalization throughout the architecture to ensure all computations occur on the unit hypersphere. Previous approaches apply normalization primarily for training stability, N-transformer uses normalization as a fundamental design principle that shapes how the model processes and transforms sequences. By constraining embeddings, attention matrices, and hidden states to unit norm, it creates a more structured optimization landscape where each layer's transformations are naturally bounded and interpretable as movements along the hypersphere. The key technical innovation lies in its reformulation of the standard Transformer blocks. Rather than applying unrestricted linear transformations followed by non-linearities, the attention and feed-forward layers compute normalized displacements that preserve the hyperspherical constraint.

Standard Transformer	N-transformer
$h_A \leftarrow \text{ATTN}(\text{RMSNorm}(h))$	$h_A \leftarrow \text{ATTN}(h)$ with normalized Q, K scaled by s_{qk}
$h \leftarrow h + h_A$	$h \leftarrow \mathcal{N}(\mathcal{N}(h) + \alpha_A(\mathcal{N}(h_A) - \mathcal{N}(h)))$
$h_C \leftarrow \text{CROSS}(\text{RMSNorm}(h))$	$h_C \leftarrow \text{CROSS}(h)$ with normalized Q, K scaled by s_{qk}
$h \leftarrow h + h_C$	$h \leftarrow \mathcal{N}(\mathcal{N}(h) + \alpha_C(\mathcal{N}(h_C) - \mathcal{N}(h)))$
$h_M \leftarrow \text{MLP}(\text{RMSNorm}(h))$	$h_M \leftarrow \text{MLP}(h)$ with scaled intermediate states s_{uv}
$h \leftarrow h + h_M$	$h \leftarrow \mathcal{N}(\mathcal{N}(h) + \alpha_M(\mathcal{N}(h_M) - \mathcal{N}(h)))$
Final: $h \leftarrow \text{RMSNorm}(h)$	No final normalization needed
Parameters unconstrained	All matrices and embeddings normalized along embedding dimension after each batch

Table 1: Transformer vs. N-transformer Architecture Comparison where $\mathcal{N}(\cdot)$ represents L2 normalization. α_A, α_C, α_M are learnable update rates for attention, cross-attention, and MLP respectively. s_{qk} is the learned scaling for query-key interactions. s_{uv} is the learned scaling for MLP intermediate states.

Our contributions are as follows:

- **Unified Normalization in Encoder-Decoder Architecture:** We normalize all vectors that form the embedding dimensions of network matrices to lie on a unit norm hypersphere. This operation converts matrix vector products into cosine similarities, limiting the range to [-1, 1]. The normalization renders weight decay unnecessary. Our approach maintains normalized representations throughout the information flow, including the cross-attention mechanism. This unified approach reduces the parameter count by 2.5%.

- **Adaptive Scaling Mechanism for Component Integration:** We introduce learnable eigen learning rates for both encoder and decoder components. These scaling factors automatically balance the contributions of self-attention, cross-attention, and feed-forward networks, allowing the model to dynamically adjust the importance of each component during training.

- **Faster Convergence:** The N-transformer model demonstrates faster convergence, attaining a lower validation loss compared to the standard transformer model for the same number of training iterations, indicating more efficient optimization.

2 Design and Architecture Details

This section outlines the baseline Transformer and the modifications necessary to derive its normalized version.

2.1 Self Attention

The self-attention mechanism in N-transformer builds upon traditional transformer architectures while introducing normalization-based optimizations for both encoder and decoder components.

2.1.1 Standard Transformer Self-Attention

In the standard Transformer, self-attention operates on an input sequence $X \in \mathbb{R}^{B\times T\times D}$, where B is the batch size, T is the sequence length, and D is the embedding dimension. The attention mechanism is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

where Q, K, and V are linear projections of the input:

$$Q = XW_Q\ K \quad = XW_K\ V = XW_V \quad (2)$$

2.1.2 N-transformer Self-Attention

The N-transformer modifies this mechanism by introducing normalized query and key representations:

$$\text{normalized}(x) = \frac{x}{|x|_2} \quad (3)$$

The modified attention computation becomes:

$$Q_{\text{norm}} = s_{qk} \cdot \text{normalized}(Q) \quad (4)$$

$$K_{\text{norm}} = s_{qk} \cdot \text{normalized}(K) \quad (5)$$

where s_{qk} is a learned scaling parameter initialized as:

$$s_{qk} = \frac{1}{\sqrt{D}} \cdot s_{\text{init}} \tag{6}$$

The attention scores are computed by taking the dot product of the query and key vectors, scaling by $\frac{1}{\sqrt{d_k}}$, and applying a softmax function to obtain attention weights. A masking matrix M prevents attending to future tokens, ensuring causal attention:

$$\text{Attention}(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \text{softmax}\left(\frac{\boldsymbol{q}\boldsymbol{k}^\top}{\sqrt{d_k}} + M\right)\boldsymbol{v}. \tag{7}$$

Multi-head attention is implemented using separate projections for each attention head, with results concatenated and projected back to the model's dimension:

$$\boldsymbol{h}_A = \text{Concat}(\text{head}_1, ..., \text{head}_{n_{\text{heads}}})W_O. \tag{8}$$

2.2 Cross-Attention Mechanism

Cross-attention is employed within the encoder-decoder framework to enable interactions between the encoder's contextual representations and the decoder's token embeddings. Unlike self-attention, which operates within a single sequence, cross-attention allows the decoder to attend to the encoder's output, making it essential for tasks such as sequence-to-sequence learning. Given the encoder's output hidden states $E \in \mathbb{R}^{B\times S\times D}$ and the decoder's current hidden states $H_d \in \mathbb{R}^{B\times T\times D}$, cross-attention enables the decoder to selectively focus on relevant parts of the input sequence during generation.

2.2.1 Standard Transformer Cross-Attention

In the standard Transformer, cross-attention computes queries from the decoder's hidden states while deriving keys and values from the encoder's output:

$$\text{CrossAttention}(Q_d, K_e, V_e) = \text{softmax}\left(\frac{Q_d K_e^T}{\sqrt{d_k}}\right)V_e \tag{9}$$

This mechanism allows each decoder position to attend to all encoder positions, creating a direct information pathway between the input and output sequences.

2.2.2 N-transformer Cross-Attention

The N-transformer enhances cross-attention by applying the same normalization principles used in self-attention. This modification ensures consistent scale management throughout the network and stable gradient flow between encoder and decoder. The normalized queries and keys are computed as:

$$Q_{d_norm} = s_{qk} \cdot \text{normalized}(Q_d) \tag{10}$$

$$K_{e_norm} = s_{qk} \cdot \text{normalized}(K_e) \tag{11}$$

where the scaling factor s_{qk} maintains the same initialization as in self-attention. The final cross-attention computation becomes:

$$\text{CrossAttention}(Q_d, K_e, V_e) = \text{softmax}\left(\frac{Q_{d_{\text{norm}}} K_{e_{\text{norm}}}^T}{\sqrt{d_k}}\right)V_e \tag{12}$$

This normalized cross-attention ensures that the interaction between encoder and decoder representations occurs in a well-conditioned space, potentially leading to more stable training and better convergence.

2.3 MLP Updates

The Multi-Layer Perceptron (MLP) component serves as a position-wise feed-forward network that processes each position independently, introducing non-linearity and increasing the model's representational capacity. This component is crucial for capturing complex patterns and transformations that cannot be modeled by attention mechanisms alone.

2.3.1 Standard Transformer MLP

The MLP block in this transformer architecture processes the input hidden state x through a sequence of operations. First, the input is normalized using RMSNorm:

$$\hat{x} = \text{RMSNorm}(x) \tag{13}$$

The normalized input is then projected through a single expansion matrix to a higher-dimensional space:

$$z = \hat{x}W_1 \tag{14}$$

where $W_1 \in \mathbb{R}^{d\times 4d}$ expands the representation to four times the model dimension. This expanded representation is processed through the SiLU (Elfwing et al., 2018) activation function:

$$h = \text{SiLU}(z) = z \cdot \sigma(z) \tag{15}$$

where $\sigma(z)$ is the sigmoid function. Finally, the activated representation is projected back to the original dimension through a second linear transformation:

$$y = hW_2 \tag{16}$$

where $W_2 \in \mathbb{R}^{4d \times d}$. Unlike more complex variants, this implementation uses a straightforward structure without gating mechanisms, focusing on dimensional expansion and contraction through the two linear transformations with the SiLU activation providing non-linearity. The complete transformation can be summarized as:

$$\text{MLP}(x) = \text{SiLU}(xW_1)W_2 \tag{17}$$

This MLP operates within a residual framework in the transformer block, where its output is added to the input:

$$x = x + \text{MLP}(\text{RMSNorm}(x)) \tag{18}$$

2.3.2 N-transformer MLP

The N-transformer introduces crucial modifications to the MLP block by incorporating scaled updates and normalized computations. Unlike the standard transformer, this variant operates directly on the input without an initial RMSNorm layer when in N-transformer mode. The computation begins with a scaled intermediate representation:

$$\text{UV} = s_{uv} \cdot W_1 x \tag{19}$$

The scaling factor s_{uv} is carefully initialized and learned during training:

$$s_{uv} = s_{\text{init}} \cdot \frac{1}{s_{\text{scale}}} \cdot \sqrt{D} \tag{20}$$

where s_{init} is set to 1.0 and s_{scale} is the initialization scaling factor. This scaling ensures that the intermediate activations remain in a controlled range, particularly important given the expansion factor of the feed-forward layer. The expanded representation is then split into two components:

$$u, v = \text{split}(\text{UV}) \tag{21}$$

where both u and v are in $\mathbb{R}^{4d}$. These components interact through a SiLU activation:

$$x_{\text{mlp}} = u \odot \text{SiLU}(v) \tag{22}$$

The final projection is computed through:

$$h_{\text{mlp}} = W_2 x_{\text{mlp}} \tag{23}$$

A key distinction of the N-transformer lies in its update mechanism. Instead of a simple residual connection, it employs a normalized update:

$$\alpha = |\alpha_{\text{mlp}}| \cdot \frac{\alpha_{\text{init}}}{\alpha_{\text{scale}}} \tag{24}$$

where α_{mlp} is a learned parameter, α_{init} is 0.05, and α_{scale} is the base scaling factor. The final output is computed through a normalized interpolation. Let's define the normalization operator $\mathcal{N}(x)$ that performs L2 normalization:

$$\mathcal{N}(x) = \frac{x}{|x|_2} \tag{25}$$

Then the final update equation becomes:

$$x_{\text{out}} = \mathcal{N}(\mathcal{N}(x) + \alpha(\mathcal{N}(h_{\text{mlp}}) - \mathcal{N}(x))) \tag{26}$$

More concisely, if we let $\hat{x} = \mathcal{N}(x)$ and $\hat{h}\text{mlp} = \mathcal{N}(h\text{mlp})$:

$$x_{\text{out}} = \mathcal{N}(\hat{x} + \alpha(\hat{h}_{\text{mlp}} - \hat{x})) \tag{27}$$

This normalization-based update ensures that the network maintains stable gradients and controlled feature magnitudes throughout training, while the learned scaling factors allow the model to adaptively adjust the influence of the MLP transformation at each layer.

2.4 Residual Updates

The N-transformer introduces a novel approach to residual connections that fundamentally differs from the standard additive skip connections. Let $x \in \mathbb{R}^{B \times T \times D}$ be the input to a layer and $h \in \mathbb{R}^{B \times T \times D}$ be the transformed representation (either from self-attention, cross-attention, or MLP). The residual update is computed as follows: First, we define the L2 normalization function for any vector v:

$$\mathcal{N}(v) = \frac{v}{|v|_2} \tag{28}$$

The residual update in the N-transformer then becomes:

$$x_{\text{out}} = \mathcal{N}\Big(\mathcal{N}(x) + \alpha \cdot \big(\mathcal{N}(h) - \mathcal{N}(x)\big)\Big) \tag{29}$$

where:

α is a learnable parameter initialized to a small value $\mathcal{N}(x)$ and $\mathcal{N}(h)$ are the L2-normalized input and transformed representations The difference term $\big(\mathcal{N}(h) - \mathcal{N}(x)\big)$ represents the update direction in normalized space. The outer normalization ensures the final output maintains unit norm.

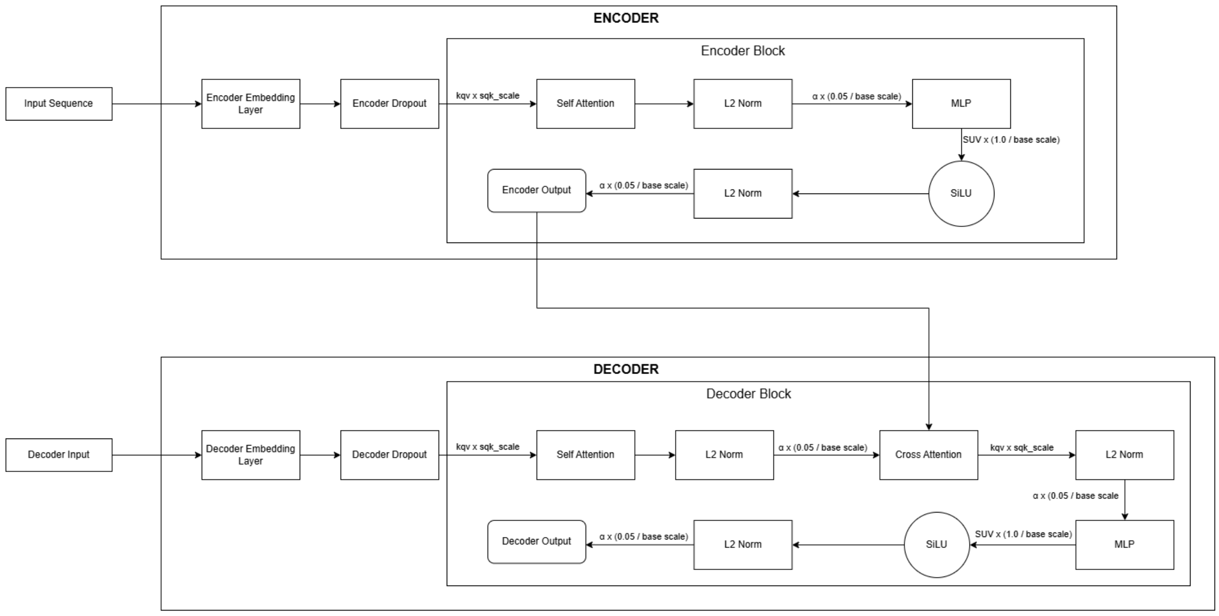


Figure 1: Architecture diagram of the N-transformer model.

This formulation can be interpreted as a form of spherical linear interpolation between normalized representations, with α controlling the magnitude of the update. The key advantages of this approach are:

- Scale Invariance: All operations occur in normalized space, making the updates independent of the magnitude of the input representations.
- Controlled Updates: The learned α parameter allows the model to adaptively determine the optimal interpolation between input and transformed representations.
- Stability: The double normalization ensures that the residual path maintains consistent scales throughout the network.
- Gradient Flow: The normalized difference provides a well-conditioned gradient path, potentially improving optimization.

α is typically initialized to a small value and is allowed to adapt during training, enabling the model to learn the optimal balance between preserving input information and incorporating transformations.

2.5 Comparison: Transformer vs. Normalized Transformer

In the normalized Transformer, all matrices and embeddings are normalized along their embedding dimension after each batch pass. The hidden state updates are controlled by learnable vectors of eigen learning rates α_A and α_M.

Model	Block Size	Validation Loss
Baseline transformer	1024	4.54303
Baseline transformer(GELU)	1024	4.31210
Baseline transformer	512	4.50281
N-transformer	1024	4.28749
N-transformer(GELU)	1024	4.33878
N-transformer	512	4.38358

Table 2: Validation loss at 500 iterations for different models and block sizes.

3 Experiments

We begin by pre-training both the baseline Transformer and the normalized Transformer on the OpenWebText (Gokaslan et al.)dataset using the sequence-to-sequence objective and evaluate them on a set of standard downstream tasks. We experiment with medium-sized models with the N-transformer having 325.24 million parameters and the baseline transformer having 333.93 million parameters, including embeddings. A detailed description of the setup and hyperparameters is in Appendix A.1.

We evaluate our models on the GLUE benchmark (Wang, 2018), following the approach of (Raffel et al., 2020). Since the original test sets are not publicly available, we adopt the data split strategy proposed by (Zhang et al., 2020). For smaller datasets (RTE, MRPC) with fewer than 10K samples, we divide the original validation set equally into validation and test sets. For larger datasets, we create a validation set by reserving 1K samples from the training data and use the original validation set for testing. The same approach is applied to the WiC (Pilehvar and Camacho-Collados, 2018) and BoolQ (Clark et al., 2019) datasets from the SuperGLUE (Wang et al., 2019) benchmark. For the MNLI task, we report results on the MNLI matched dataset. We fine-tune both N-transformer and the baseline transformer models on downstream tasks for 100 iterations each, using a batch size of 64. Our results indicate a significant performance improvement of the N-transformer over the baseline transformer, with an average performance gain of 21%. All experiments were run on a single NVIDIA A100 40GB GPU.

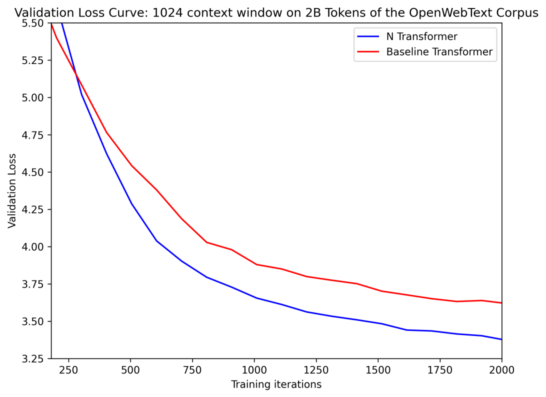
4 Results

4.1 Analysis

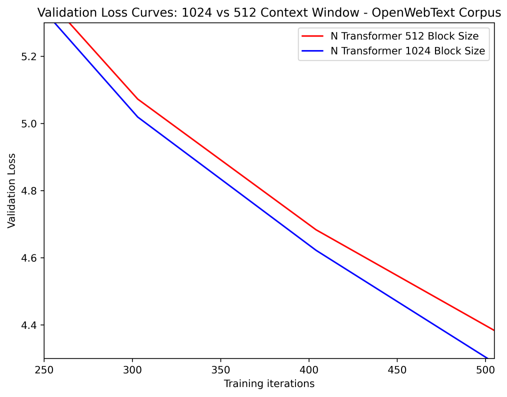
The eigen learning rate distributions in Figure 3 reveal fundamental differences in learning dynamics between encoder and decoder components. The decoder components demonstrate significantly higher learning rates, with attention rates ranging from 0.125 to 0.27 and MLP rates from 0.18 to 0.237, compared to the encoder's more conservative rates of 0.021-0.039 for attention and 0.016-0.032 for MLP layers. This substantial disparity (approximately 5-10x) in learning rates between encoder and decoder components suggests a more aggressive learning strategy in the decoder, aligning with its role in generating diverse outputs. The encoder's attention mechanism shows a gradual increase in learning rates across layers (0.0213 to 0.039), indicating progressively finer adaptation in deeper layers, while maintaining overall conservative updates. In contrast, the decoder's attention exhibits higher initial learning rates that decrease in later layers (0.27 to 0.125), suggesting more aggressive learning in early layers followed by refinement. Notably, MLP learning rates in both components display more stable patterns than their attention counterparts, with decoder MLP rates maintaining consistently higher values (around 0.22) compared to encoder MLP rates (around 0.02). This stability in MLP rates, coupled with the more variable attention rates, indicates a well-structured learning process where feature transformation remains consistent while attention mechanisms adapt more dynamically based on their position and role in the network. Figure 2a shows the comparison of the validation loss between the N-transformer and the baseline transformer with a context window of 1024 and impact with block size. The N-transformer model achieves a lower validation loss (3.37) than the baseline transformer model (3.62). Figure 2b depicts that using a larger context window of 1024 shows a lower validation loss at 500 iterations compared to a context window of 512 for the N-transformer. Figure 4 provides an analysis of the scaling factors after pre-training the N-transformer model on the OpenWebText dataset. The scaling factors reveal distinct operational patterns across the encoder-decoder architecture. The decoder's attention scaling (sqk) demonstrates a notable monotonic decrease from 0.28 to 0.142 across layers, indicating a progressive refinement of attention focus as information flows deeper into the network. This contrasts sharply with the encoder's attention scaling, which maintains consistently low values (0.026-0.039), suggesting more focused attention processing throughout. The feed-forward network scaling factors (su, sv) exhibit remarkable stability in both components, with decoder values slightly above unity (1.02-1.05) indicating minor feature amplification, while encoder values remain marginally below unity (0.98-0.99), suggesting subtle feature dampening. This dichotomy in scaling behavior, particularly in the attention mechanisms, aligns with the theoretical understanding of encoder-decoder architectures, where encoders focus on creating precise contextual representations while decoders progressively refine their attention from broad context consideration to specific token generation. The stability of feed-forward scaling factors across all layers indicates robust gradient flow and effective normalization, while the systematic difference between encoder and decoder scaling suggests specialized roles in sequence processing.

Parameter	N-transformer	Baseline Transformer
Number of Encoder Layers (n_{layers})	6	6
Number of Decoder Layers (n_{layers})	6	6
Model Dimension (d_{model})	1024	1152
Number of Attention Heads (n_{heads})	8	8
Key Dimension (d_k)	$d_{\text{model}}/n_{\text{heads}}$	$d_{\text{model}}/n_{\text{heads}}$
MLP Dimension (d_{MLP})	$4d_{\text{model}}$	$4d_{\text{model}}$
Total Parameters	325.24M	333.94M

Table 3: Architectural configurations of N-transformer and Baseline Transformer models.



(a) Validation Loss Curve for 1024 Context Window



(b) Validation Loss Curve Comparison between context windows

Figure 2: Figure 2a: Validation loss curve comparing N-transformer and Baseline Transformer models with a 1024 context window. Figure 2b: Validation loss curves comparing N-transformer models with 512 and 1024 context window

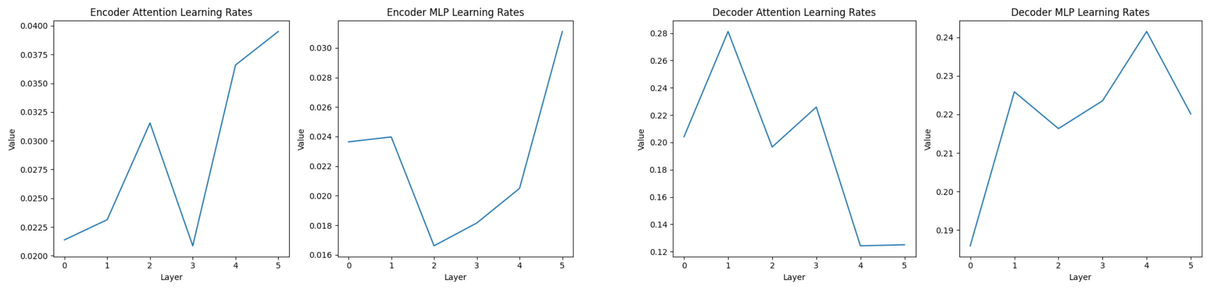


Figure 3: Layer-wise distribution of eigen learning rates across encoder and decoder components, illustrating distinct learning dynamics between attention and MLP layers, with decoder components exhibiting substantially higher learning rates compared to their encoder counterparts.

4.2 Downstream Tasks Results

We evaluate N-transformer and the Baseline transformer on the GLUE Benchmark and WiC and BoolQ from the SuperGLUE benchmark. The N-transformer model exhibits an average increase in performance by **11.96%** after training for only about 100 iterations. This result shows the increase in convergence speed of the N-transformer in various language modeling tasks.

5 Conclusion

This work builds on important artifacts and findings made by NVIDIA in their report 'nGPT' and presents a novel encoder-decoder architecture that leverages hyperspherical normalization and adaptive scaling factors. Through careful analysis of the scaling factors and eigen learning rates, we observe distinct behavioral patterns between encoder and decoder components that provide insights into the model's learning dynamics. The decoder's attention scaling factors show a progressive focusing behavior, while maintaining stable feed-forward scaling slightly above unity, indicating effective feature amplification. In contrast, the encoder maintains consistently low attention scaling with feed-forward values slightly below unity, suggesting precise contextual representation learning. These patterns are complemented by the eigen learning rate distributions, where decoder components exhibit significantly higher rates compared to encoder components, enabling more aggressive learning in the decoding phase while maintaining stable feature transformations. The effectiveness of this architecture is demonstrated by its substantial performance improvement of 11.96% over the baseline transformer on GLUE and SuperGLUE benchmarks.

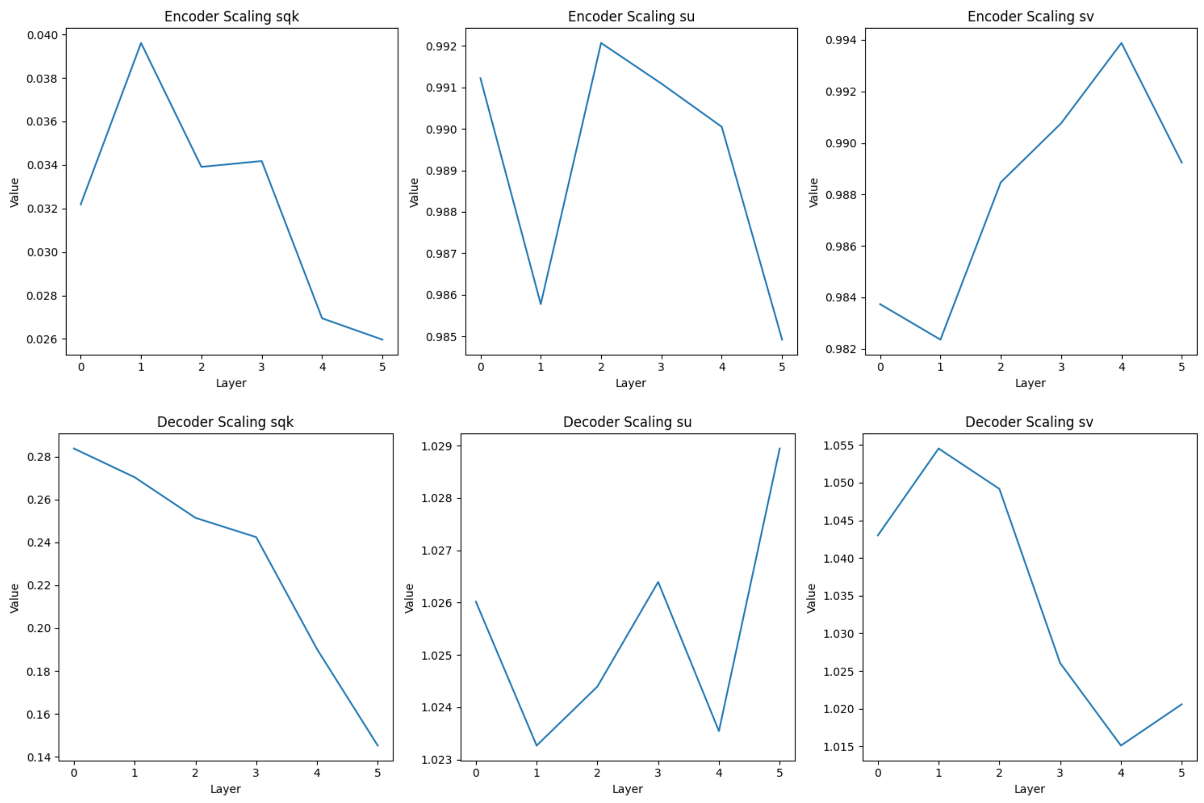


Figure 4: Evolution of scaling factors across encoder and decoder layers (0-5) of the N-transformer, showing distinct patterns between attention (sqk) and feed-forward (su, sv) components.

Model	MRPC (Acc / F1)	QNLI	QQP (Acc / F1)	RTE	MNLI	SST-2	BoolQ	WiC	Overall Avg.
Transformer	**61.54 / 71.43**	56.12	72.45 / 52.94	49.31	33.38	77.84	63.66	51.72	59.04
N-Transformer	**61.54** / 69.47	**62.72**	**79.31 / 69.92**	**56.94**	**57.31**	**80.00**	**66.07**	**57.66**	**66.10**

Table 4: Performance comparison between Transformer and N-transformer across multiple datasets. Scores for MRPC, QQP, and MNLI are reported as (Accuracy / F1), while other datasets display accuracy. Overall Avg. represents the average performance across all datasets. The N-transformer model exhibits an average increase in performance by 11.96% after training for 100 iterations with a batch size of 64

6 Limitations

While the N transformer demonstrates promising results in terms of early learning capabilities and performance improvements, several limitations must be acknowledged. The most significant drawback is the computational overhead introduced by the double normalization process, which increases each training step's duration by approximately 80%. This substantial overhead presents challenges for scaling the architecture to larger models or longer training durations. Additionally, the current implementation should be viewed primarily as a proof of concept for a novel training paradigm rather than a direct competitor to established architectures on downstream tasks, as it has only undergone limited pre-training iterations.

Future work could address these limitations in several directions. First, optimizing the normalization computations through techniques like fused operations or adaptive normalization schedules could potentially reduce the computational overhead. Second, scaling the N transformer to larger network sizes and evaluating its performance on real-world datasets would provide valuable insights into its practical applicability. The architecture could be extended by exploring adaptive scaling mechanisms that dynamically adjust based on layer depth and attention patterns, or by investigating hybrid normalization schemes that balance computational efficiency with representational power. An extension of the architecture towards hybrid architectures (Dao and Gu, 2024) is straightforward. Furthermore, investigating the integration of sparse attention mechanisms or conditional computation paths could enhance the model's efficiency while maintaining the benefits of hyperspherical normalization. The promising early learning behavior also suggests potential applications in few-shot learning scenarios, where rapid adaptation is crucial.

References

Jimmy Lei Ba. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Francesco D'Angelo, Maksym Andriushchenko, Aditya Varre, and Nicolas Flammarion. 2023. Why do we need weight decay in modern deep learning? *arXiv preprint arXiv:2310.04415*.

Tri Dao and Albert Gu. 2024. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11.

Aaron Gokaslan, Vanya Cohen, E Pavlick, and S Tellex. Openwebtext corpus. 2019. *URL http://Skylion007. github. io/OpenWebTextCorpus*, page 9.

Maxim Kodryan, Ekaterina Lobacheva, Maksim Nakhodnov, and Dmitry P Vetrov. 2022. Training scale-invariant neural networks on the sphere can happen in three regimes. *Advances in Neural Information Processing Systems*, 35:14058–14070.

I Loshchilov. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Ilya Loshchilov, Cheng-Ping Hsieh, Simeng Sun, and Boris Ginsburg. 2024. ngpt: Normalized transformer with representation learning on the hypersphere. *arXiv preprint arXiv:2410.01131*.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2018. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. *arXiv preprint arXiv:1808.09121*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Tim Salimans and Durk P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

Alex Wang. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. 2020. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2020. Revisiting few-sample bert fine-tuning. *arXiv preprint arXiv:2006.05987*.

A Appendix

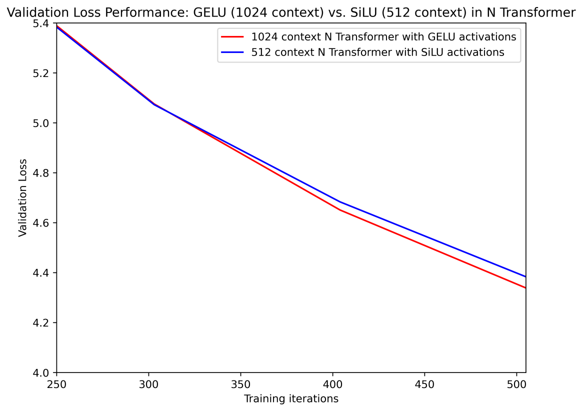


Figure 5: Validation loss curves comparing the N transformer with SiLU activations having a lower block size and an N transformer model using GELU activations having a higher block size on the OpenWebText dataset

A.1 Effect of SiLU in N transformer

Figure 5 demonstrates an intriguing relationship between activation functions and context length in N-transformer architectures. The validation loss curves show that an N-transformer using SiLU activations with a 512-token context length achieves comparable performance to one using GELU activations with a 1024-token context length, with only a marginal difference in validation loss (approximately 0.05 higher for SiLU across iterations). This finding is particularly significant given the model architecture's use of normalized attention mechanisms and residual pathways. The SiLU activation $\sigma(x) \cdot x$ in the feed-forward network, combined with the model's normalization scheme, appears to compensate for the reduced context length, suggesting that the choice of activation function can have architectural implications beyond simple non-linearity. The convergence patterns indicate that while both variants show consistent improvement over training iterations (250-500), the GELU variant with longer context maintains a slight edge in final performance (validation loss of 4.33 vs 4.38). However, the computational efficiency gained from halving the context length while maintaining comparable performance could present a valuable trade-off, especially in resource-constrained environments.

A.2 Pre-training hyperparameter settings

Parameter	**N-transformer**	**Transformer**
Batch Size	16	16
Dropout	0.1	0.1
Training Iterations	2000	2000
β_1	0.9	0.9
β_2	0.95	0.95
Learning Rate	15×10^{-4}	15×10^{-4}
Weight Decay	0.0	0.1
Warmup Iterations	0	100
Base Scale	$1.0/\sqrt{n_{\text{embd}}}$	0.02
Encoder Layers	6	6
Decoder Layers	6	6
Model Dimension	1024	1152
Attention Heads	8	8
Block Size (Context Window)	1024	1024
Total Parameters	325.24M	333.94M

Table 5: Comparison of hyperparameters between N-transformer and the Baseline Transformer.

Table 6: Comparison of architectural parameters between N-transformer and Baseline Transformer.

Table 6 describes the hyperparameter settings used for pre-training both the N transformer and the baseline transformer on the OpenWebText dataset with the sequence-to-sequence objective.

A.3 Downstream tasks results analysis

From Table 4, we can observe that the N transformer model outperforms the baseline transformer by wider margins on tasks with samples having longer sequences such as RTE, MNLI and QNLI and the performance gap narrows as the sequence lengths of the samples reduce as seen in MRPC, SST-2 and QQP tasks in the GLUE Benchmark. The performance patterns observed across different sequence lengths may be attributed to several architectural features of the N transformer. The model's normalized attention mechanism, implemented through L2 norm and carefully calibrated scaling factors, appears to facilitate more stable gradient flow in longer sequences. The architecture's approach to normalization may be particularly beneficial in maintaining consistent gradient magnitudes across varying sequence lengths, potentially enabling better capture of long-range dependencies. Interestingly, the narrowing performance gap in tasks with shorter sequences, such as MRPC, SST-2, and QQP, suggests that the benefits

of the normalization strategy might become more pronounced as sequence length increases. This behavior could indicate that while the normalized architecture provides advantages across all sequence lengths, its impact is most notable in scenarios requiring the model to process and maintain relationships over longer distances.