THE BENEFITS OF NORMALIZATION LAYERS IN TRANSFORMERS

Kangqi Yu 121090735
Chinese University of Hong Kong, Shenzhen
Shenzhen, China
kangqiyu

ABSTRACT

A number of studies examining the benefits of normalization layers in transformers have primarily focused on machine translation. To examine the benefit of layer normalizaiton, batch normalizaiton and no normalization to transformers on sentiment analysis tasks, this study conducts tests covering the convergence condition, the initial gradient norm magnitude, the initialization needs, Lipschitz of loss function and smoothnesss of loss function. Finally, layer normalizaiton is the best choice among them and bacth normalizaiton is not favored for the dynamic length of texts data makes it's statistics unstable. No normalizaiton layer is acceptable when model is shallow, while not recommended when training deep models.

1 INTRODUCTION

Transformers have revolutionized Natural Language Processing (NLP) by leveraging self-attention mechanisms to capture long-range dependencies (1). However, as deep networks, Transformers can encounter challenges related to training instability, vanishing or exploding gradients, and poor convergence (2; 3; 4). Normalization layers, particularly Layer Normalization (LayerNorm), have emerged as critical components within the Transformer architecture, addressing these issues by normalizing inputs across features for each training example (4).

LayerNorm has been shown to stabilize the training process in deep networks, especially in recurrent models, as first introduced by Ba et al. (5). Its effectiveness in maintaining consistent variance during training has led to its widespread adoption in Transformers. While studies have demonstrated the efficacy of normalization layers in machine translation tasks (4; 6), the exploration of these techniques in domain-specific tasks, such as financial sentiment analysis, remains limited. This research aims to address this gap by investigating how normalization layers, including LayerNorm and potential alternatives like Batch Normalization (BatchNorm), influence Transformer-based models within the context of financial sentiment analysis.

To this end, this study will utilize the portion of the Financial PhraseBank dataset where all researchers unanimously agree on the labels. a widely recognized benchmark that is annotated by experienced financial researchers (7). I demonstrate the benefits of normalization in enhancing model robustness and improving sentiment classification performance through the convergence condition, the initial gradients magnitude, the initialization needs, Lipschitz of loss function, smoothness of loss function.

The second section provides a concise literature review on sentiment analysis and the benefits of normalization. The third section outlines the experimental settings, followed by the fourth section, which presents the experimental results and discussion. Finally, the last section concludes the study.

2 Related Work

2.1 Sentiment Analysis

Sentiment analysis is a subfield of NLP that focuses on classifying emotions and sentiments expressed in written text. With the exponential growth of social media, the availability of written text is rapidly increasing, providing vast resources for sentiment analysis tasks (8). This surge in data has elevated sentiment analysis to a prominent position among NLP tasks due to its applications in opinion mining, customer feedback analysis, and market prediction.

Early sentiment analysis methods mainly relied on traditional machine learning algorithms such as Naive Bayes and Support Vector Machines (SVM). These models often coupled with manually engineered features like n-grams (9), part-of-speech tags (10), and sentiment lexicons (11). While effective to some extent, these approaches were limited by their dependence on feature engineering and inability to capture contextual nuances in text.

Deep Neural Networks (like Convolutional Neural Networks (CNNs) (12) and Recurrent Neural Networks (RNNs) (13)) revolutionized sentiment analysis by automatically extracting features from data, reducing the reliance on manual feature engineering. Long Short-Term Memory (LSTM) networks (14) and Gated Recurrent Units (GRUs) (15) further improved sentiment analysis by capturing long-range dependencies in text. However, these sequential models struggled with scalability and parallelization (6), which are critical for scaling up.

In recent years, Transformer-based architectures have emerged as the state-of-the-art for sentiment analysis and other NLP tasks. Transformer encoders, such as BERT (Bidirectional Encoder Representations from Transformers) (16), have demonstrated remarkable performance by leveraging self-attention mechanisms to capture global contextual information. BERT's bidirectional nature allows it to understand the context of words more effectively than unidirectional models, making it particularly well-suited for sentiment analysis, where nuanced understanding of text is crucial. Some variant of BERT (like RoBERTa (17)) also attract researchers' attention.

Overall, Transformer-based models have redefined the landscape of sentiment analysis by combining state-of-the-art performance with scalability and robustness. These models have paved the way for new applications, enabling sentiment analysis to handle complex, real-world scenarios with unprecedented precision.

2.2 Normalization Benefits

Normalization layers are essential components in deep learning, designed to address training challenges such as internal covariate shift, slow convergence, and unstable gradients. By normalizing the activations or weights, these layers enhance both the stability and efficiency of deep neural networks, playing a crucial role in their widespread adoption across diverse tasks (18).

Batch Normalization (BatchNorm) (18), proposed by Ioffe and Szegedy, is one of the earliest and most widely used normalization methods. BatchNorm normalizes the inputs of each layer using the mean and variance calculated across a mini-batch, followed by scaling and shifting with learnable parameters. This technique reduces internal covariate shift, enabling higher learning rates and faster convergence (18). BatchNorm has been particularly successful in convolutional neural networks (CNNs) and has contributed to the training of deep architectures like ResNet. However, BatchNorm's reliance on batch statistics makes it sensitive to batch size. When the batch size is small, the estimated statistics may become unstable, leading to weaker performance.

To overcome the limitations of BatchNorm, especially in sequence modeling and tasks with small batch sizes, Layer Normalization (LayerNorm) (19) was introduced. LayerNorm computes the mean and variance across the features of each individual sample rather than across a batch, making it insensitive to batch-size, which is helpful to implement parallelization and federated learning (20). This property has made LayerNorm a cornerstone in sequence-to-sequence models and Transformer architectures, including BERT and GPT. By normalizing at the feature level, LayerNorm ensures stable training dynamics in tasks requiring complex temporal or contextual understanding.

Instance Normalization (InstanceNorm) (21) is another variant that focuses on normalizing each feature map of an individual sample in convolutional networks. Initially proposed for tasks like style

transfer, InstanceNorm excels in applications where preserving spatial structure and style adaptability is critical. Unlike BatchNorm and LayerNorm, InstanceNorm applies normalization independently to each spatial dimension, emphasizing style consistency over content integrity.

Group Normalization (GroupNorm) (22) generalizes the concept of normalization by dividing the features into groups and normalizing within each group. GroupNorm balances the strengths of BatchNorm and LayerNorm, offering robust performance in vision tasks while maintaining stability across varying batch sizes. Its group-based approach makes it an excellent choice for applications where memory constraints limit batch sizes.

Weight Normalization (WeightNorm) (23), another noteworthy method, differs from the previous techniques by normalizing the weights instead of activations. This approach simplifies optimization and improves convergence speed without the need for batch statistics. Although less commonly used than BN or LN, Weight Normalization has demonstrated benefits in specific architectures and tasks.

The evolution of these normalization techniques underscores their central role in modern deep learning. By addressing distinct challenges such as batch-size sensitivity, sequence modeling, and style adaptability, they have expanded the applicability and robustness of neural networks. As research continues, more adaptive and efficient normalization methods are expected to emerge, further solidifying their impact on deep learning advancements. In this study, BatchNorm and LayerNorm are focused for the nature of sentiment analysis.

3 Experiment Setting

3.1 Dataset and Tokenizer

Financial PhraseBank dataset is a well-developed human-labeled financial sentiment analysis dataset (7). To ensure the quality of experimental data, this study selects the data with labels agreed by all researchers. Further, the whole dataset is shuffled and divided into training dataset and validation dataset. Then, this study utilizes the tokernizer of BRET to tokernize text data.

3.2 Model Structure

The model utilized in this study is the original Transformer Encoder (6), with Post-LayerNorm strucuture rather than Pre-LayerNorm structure. Although Pre-LayerNorm indeed has the advantage to alleviate graident exploding and vanishing and the detailed proof is in A.1, this study chooses Post-Norm for the experimental model is not so deep, which means the advantages in controlling gradient exploding and vanishing are unimportant, and the performance of Post-LayerNorm is better than Pre-LayerNorm due to potential degradation proposed by Shleifer et al. (24).

Except the feed-forward networks with relu as activation function take Kaiming Initialization (25), all other layers take Xavier Initialization (26).

3.3 Training

This study did not employ any training technique specifically designed for Transformer, such as the warm-up stage. Instead, all training methods followed standard approaches (like Stochastic Gradient Descent) to reduce the number of hyperparameters.

3.4 Results and Discussions

Convergence Condition The learning rate parameter is tuned to test the convergence of each model. The narrow model refers to the widths of feed-forward networks are 32. The normal-width model refers to the widths of feed-forward networks are 320. The wide model refers to the widths of feed-forward networks are 3200. Through the experiment results in B.1, the Transformer Encoder with LayerNorm is not affected by the changes in model depth and width, making it easy to train; the Transformer Encoder with BatchNorm and without normalization are affected by the depth and width of the model, making them relatively hard to train.

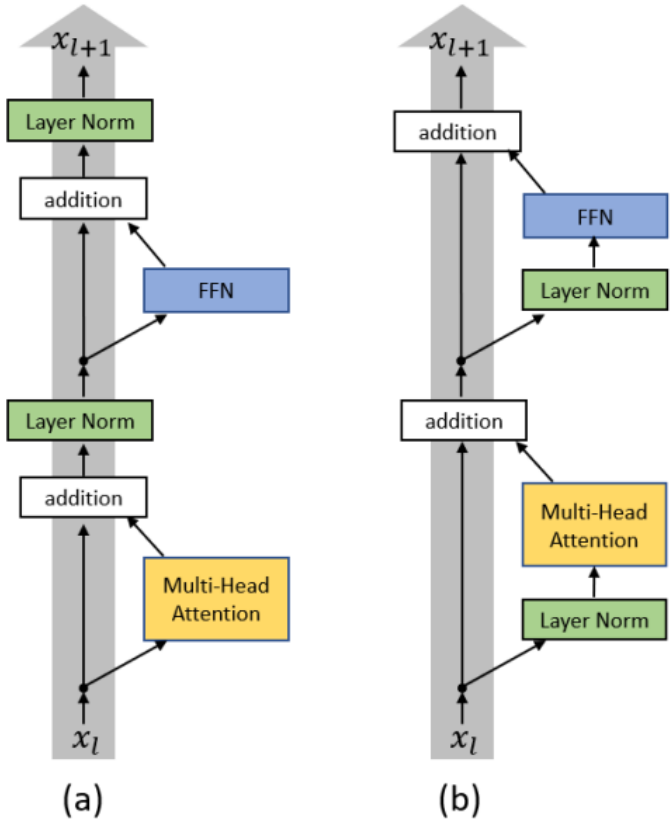

Figure 1: Example of Post-LayerNorm (a) and Pre-LayerNorm (b). Reproduced from (4).

Initial Gradients Magnitude Many researchers checked the expected initial gradients norm magnitude to assees whether the model is easy to train (4; 27). I use 100 different random seeds and record the initial graident norm of these models. Detailed gradient norm magnitudes are shown below:

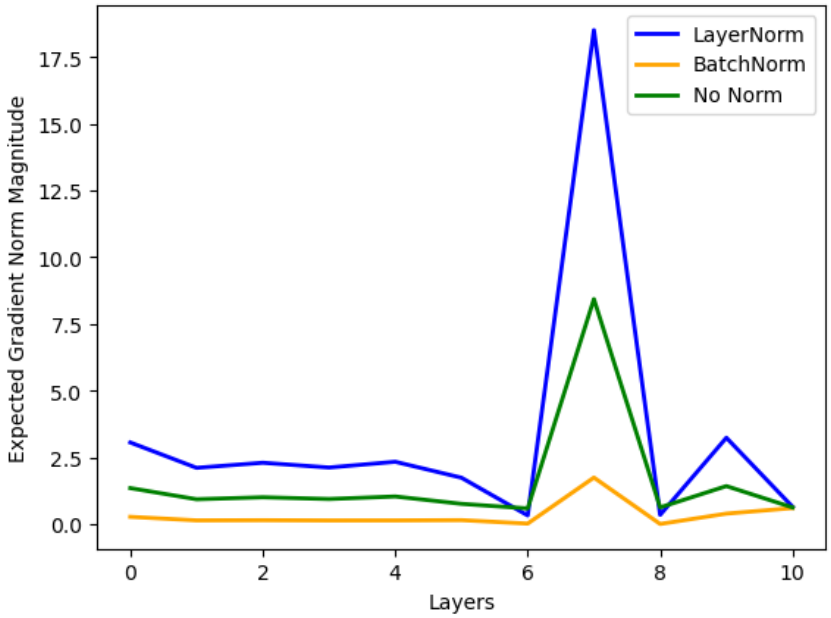

Figure 2: Expected Gradient Norm Magnitude of Layers - Shallow Models

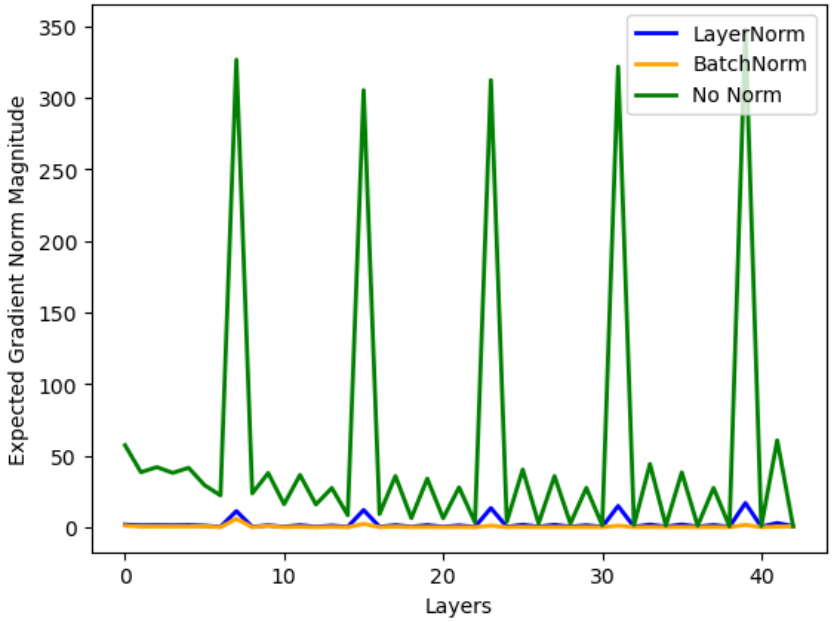

Figure 3: Expected Gradient Norm Magnitude of Layers - Deep Models

The deep models with normalizaiton, no matter LayerNorm or BatchNorm, have a series of more stable expected gradient norm across layers, making it easy to train. However, this benefit is not so obvious in shallow model with LayerNorm.

Initialization Needs As stated in the model section, all layers of model in the previous experiments are well-initialized. However, this study will test the robustness of model with different normalization layers/without normalization layer in this experiment, setting all parameters initialized randomly from -0.1 to 0.1 and -100 to 100, respectively. The results show small scale uniform initialization little affects these models and large scale bad Initialization will make them all diverge.

Lipschitz and Smoothness of Loss Function Lipschitz of loss function is defined as (lr refers to learning rate) (28)

$$\max_{\alpha \in (0, lr)} \|\mathcal{L}(W) - \mathcal{L}(W - \alpha \nabla \mathcal{L})\|$$

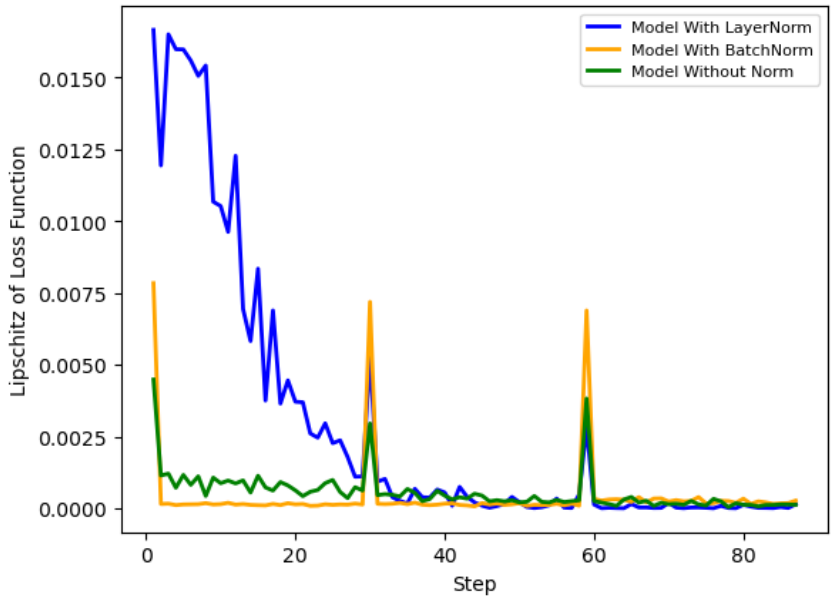

Figure 4: Lipschitz of Loss Function

Smoothness of loss function is defined as (28)

$$\max_{\alpha \in (0, lr)} \|\nabla \mathcal{L}(W) - \nabla \mathcal{L}(W - \alpha \nabla \mathcal{L})\|$$

Both the Lipschitz condition and smoothness of loss function from 4, 5 indicates BatchNorm is more stable in gradients than LayerNorm. However, the loss plot 6 of it indicates BatchNorm is not so suitable to sentiment analysis tasks for the dynamic length of texts, which makes the statistics in BatchNorm unstable and leaves a loss peak in the loss plot.

4 Conclusion

LayerNorm is the most suitable normalization choice for Transformers in sentiment analysis tasks, offering significant advantages in robustness, convergence speed, and the magnitude of initial gradient norms. While BatchNorm performs comparably and occasionally better than LayerNorm in certain experiments, it is generally not favored due to its limitations in handling the dynamic length of text data and its slower convergence speed. Omitting normalization is not recommended for deep Transfomers.

To expand this discussion to other tasks, the further studies will focus on computer vision tasks. In these taskes that input data is regular, the above conclusion may be different for the smooth gradients of BatchNorms and no dynamic issue about their statistics.

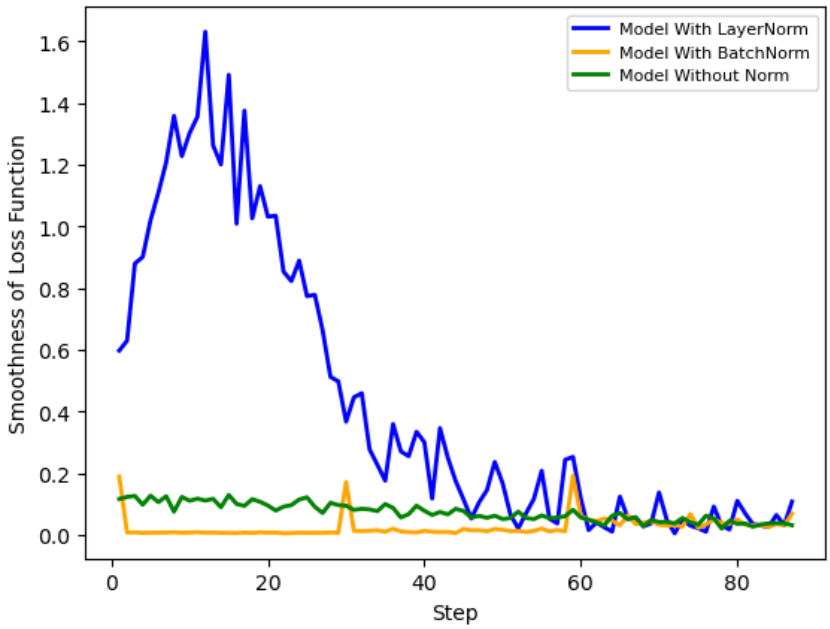

Figure 5: Smoothness of Loss Function

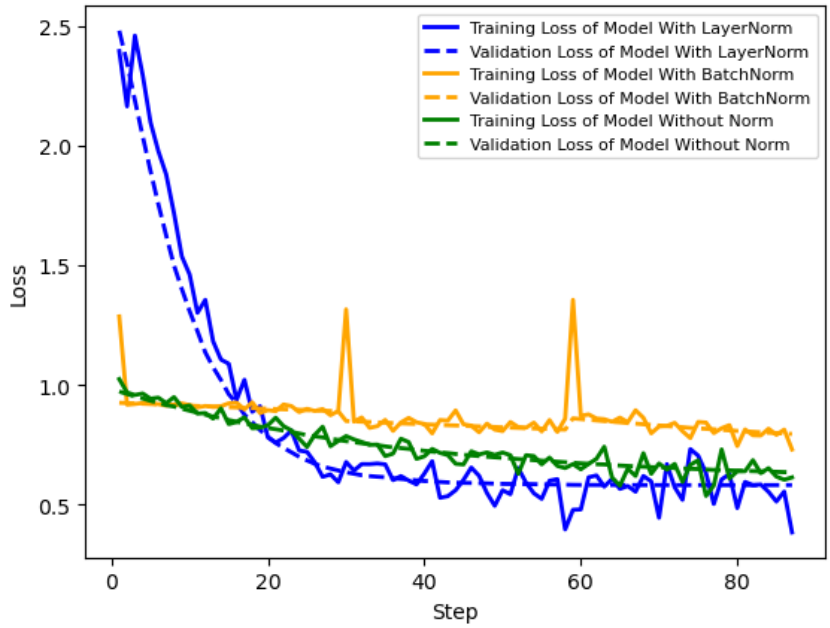

Figure 6: Loss Plot

REFERENCES

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[2] Akhil Kedia, Mohd Abbas Zaidi, Sushil Khyalia, Jungho Jung, Harshith Goka, and Haejun Lee. Transformers get stable: An end-to-end signal propagation theory for language models, 2024.

[3] Alireza Naderi, Thiziri Nait Saada, and Jared Tanner. Mind the gap: a spectral analysis of rank collapse and signal propagation in transformers, 2024.

[4] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 60006010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[7] Pekka Malo, Ankur Sinha, Pyry Takala, Pekka Korhonen, and Jyrki Wallenius. Good debt or bad debt: Detecting semantic orientations in economic texts, 2013.

[8] Kian Long Tan, Chin Poo Lee, and Kian Ming Lim. A survey of sentiment analysis: Approaches, datasets, and future research. *Applied Sciences*, 13(7), 2023.

[9] Peter Turney. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 417–424, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[10] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 252–259, 2003.

[11] Peter D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl, 2002.

[12] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[13] Tomas Mikolov, Martin Karafiát, Luká Burget, Jan Honza ernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[15] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[17] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

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

[19] Ruibin Xiong et al. Layer normalization. *arXiv preprint arXiv:2002.04780*, 2020.

[20] A AbhishekV, S Binny, R JohanT, Nithin Raj, and Vishal Thomas. Federated learning: Collaborative machine learning without centralized training data. *international journal of engineering technology and management sciences*, 2022.

[21] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[22] Yuxin Wu and Kaiming He. Group normalization, 2018.

[23] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks, 2016.

[24] Sam Shleifer, Jason Weston, and Myle Ott. Normformer: Improved transformer pretraining with extra normalization, 2021.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[26] Siddharth Krishna Kumar. On weight initialization in deep neural networks, 2017.

[27] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers, 2023.

[28] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?, 2019.

[29] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. Learning deep transformer models for machine translation, 2019.

A Main Theorem Proof

A.1 Proof of Pre-LayerNorm is better than Post-LayerNorm from gradient exploding and vanishing perspective

The forward propagation of Post-LayerNorm is $x_{l+1} = LN(x_l + F(x_l))$, where LN is the LayerNorm operator and F is the Feed-Forward Network. Suppose the loss function is denoted as $\mathcal{L}$. We have,

$$\frac{\partial \mathcal{L}}{\partial x_l} = \frac{\partial \mathcal{L}}{\partial x_{l-k}} * \prod_{i=l}^{l-k-1} \frac{\partial LN(y_i)}{\partial y_i} * \prod_{i=l}^{l-k-1} (1 + \frac{\partial F(x_k)}{\partial x_k})$$

where $y_k = x_l + F(x_l)$.

The forward propagation of Post-LayerNorm is $x_{l+1} = x_l + F(LN(x_l))$. Wang et al. used recursion to get (29),

$$\frac{\partial \mathcal{L}}{\partial x_l} = \frac{\partial \mathcal{L}}{\partial x_{l-k}} * (1 + \sum_{i=l}^{l-k-1} \frac{\partial F(LN(x_k))}{\partial x_l})$$

Due to the difference of product and sum, Pre-LayerNorm has the advantage to alleviate gradient exploding and vanishing.

B Main Experimental Results

B.1 Convergence Condition

Learning Rate	LayerNorm	BatchNorm	No Normalization
10	Not Converged	Not Converged	Not Converged
1	Not Converged	Not Converged	Not Converged
$5 * 10^{-1}$	Converged	Converged	Not Converged
10^{-1}	Converged	Converged	Converged
10^{-2}	Converged	Converged	Converged
10^{-3}	Converged	Converged	Converged
10^{-4}	Converged	Converged	Converged

Table 1: Convergence results for three 1-layer narrow models under different learning rates. Green indicates convergence, while red indicates failure to converge.

Learning Rate	LayerNorm	BatchNorm	No Normalization
10	Not Converged	Not Converged	Not Converged
1	Not Converged	Not Converged	Not Converged
$5 * 10^{-1}$	Converged	Converged	Not Converged
10^{-1}	Converged	Converged	Converged
10^{-2}	Converged	Converged	Converged
10^{-3}	Converged	Converged	Converged
10^{-4}	Converged	Converged	Converged

Table 2: Convergence results for three 1-layer normal-width models under different learning rates. Green indicates convergence, while red indicates failure to converge.

Learning Rate	LayerNorm	BatchNorm	No Normalization
10	Not Converged	Not Converged	Not Converged
1	Not Converged	Not Converged	Not Converged
$5 * 10^{-1}$	Converged	Not Converged	Not Converged
10^{-1}	Converged	Converged	Converged
10^{-2}	Converged	Converged	Converged
10^{-3}	Converged	Converged	Converged
10^{-4}	Converged	Converged	Converged

Table 3: Convergence results for three 1-layer wide models under different learning rates. Green indicates convergence, while red indicates failure to converge.

Learning Rate	LayerNorm	BatchNorm	No Normalization
10	Not Converged	Not Converged	Not Converged
1	Not Converged	Not Converged	Not Converged
$5 * 10^{-1}$	Converged	Not Converged	Not Converged
10^{-1}	Converged	Not Converged	Not Converged
10^{-2}	Converged	Not Converged	Not Converged
10^{-3}	Converged	Converged	Converged
10^{-4}	Converged	Converged	Converged

Table 4: Convergence results for three 5-layer narrow models under different learning rates. Green indicates convergence, while red indicates failure to converge.

Learning Rate	LayerNorm	BatchNorm	No Normalization
10	Not Converged	Not Converged	Not Converged
1	Not Converged	Not Converged	Not Converged
$5 * 10^{-1}$	Converged	Not Converged	Not Converged
10^{-1}	Converged	Not Converged	Not Converged
10^{-2}	Converged	Not Converged	Not Converged
10^{-3}	Converged	Converged	Converged
10^{-4}	Converged	Converged	Converged

Table 5: Convergence results for three 5-layer normal-width models under different learning rates. Green indicates convergence, while red indicates failure to converge.

Learning Rate	LayerNorm	BatchNorm	No Normalization
10	Not Converged	Not Converged	Not Converged
1	Not Converged	Not Converged	Not Converged
$5 * 10^{-1}$	Converged	Not Converged	Not Converged
10^{-1}	Converged	Not Converged	Not Converged
10^{-2}	Converged	Not Converged	Not Converged
10^{-3}	Converged	Not Converged	Not Converged
10^{-4}	Converged	Converged	Converged

Table 6: Convergence results for three 5-layer wide models under different learning rates. Green indicates convergence, while red indicates failure to converge.