The Benefits of Normalization Layers in Transformers

Kangqi Yu 121090735

The Chinese University of Hong Kong, Shenzhen

December 12, 2024

Transformer

- Transformer revolutionized Natural Language Processing (NLP) and the benefits of nomralization layer to transformers were tested in machine translation tasks [1, 2].

Transformer

- Transformer revolutionized Natural Language Processing (NLP) and the benefits of nomralization layer to transformers were tested in machine translation tasks [1, 2].
- However, sentiment analysis tasks are not so often tested these years.

Transformer

- Transformer revolutionized Natural Language Processing (NLP) and the benefits of nomralization layer to transformers were tested in machine translation tasks [1, 2].
- However, sentiment analysis tasks are not so often tested these years.
- Thus, I would like to employ Transformer Encoder on a financial news dataset to demonstrate the benefits of normalization layers through **the convergence condition, the initial gradients magnitude, the initialization needs, Lipschitz of loss function and smoothness of loss function**.

- Feature engineering by human: n-grams [3], part-of-speech tags [4], and sentiment lexicons [5] etc.
- Extracting features through deep neural netwroks: Convolutional Neural Networks (CNNs) [6] and Recurrent Neural Networks (RNNs) [7] etc.
- Transformer based models, like BERT (Bidirectional Encoder Representations from Transformers) [8]

- Batch Normalization: smoother training [9], reduced covariate shift, implicit regularization [10].
- Layer Normalization: Batch-size independence, effective for RNNs [11].
- Instance Normalization: Focus on style transfer, disentangling content and style [12].
- Group Normalization: Works with small batches, balances between BN and LN [13].

Dataset and Tokenizer

- Financial PhraseBank dataset is a well-developed human-labeled financial sentiment analysis dataset [14]. 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.

Model Structure

The model utilized in this study is the original Transformer Encoder [2], with Post-LayerNorm strucuture rather than Pre-LayerNorm structure.

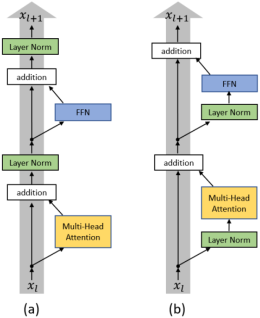

Figure 1: Example of Post-LayerNorm (a) and Pre-LayerNorm (b). Reproduced from [1].

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 [15],

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

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

The reasons to choose Post-LayerNorm

- The models in this study are not so deep, which means the advantages in controlling gradient exploding and vanishing are unimportant.
- The performance of Post-LayerNorm is better than Pre-LayerNorm due to potential degradation proposed by Shleifer et al. [16].

Initialization

Except the feed-forward networks with relu as activation function take Kaiming Initialization [17], all other layers take Xavier Initialization [18].

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.

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.

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	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.

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	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.

Initial Gradients Magnitude - Shallow Models

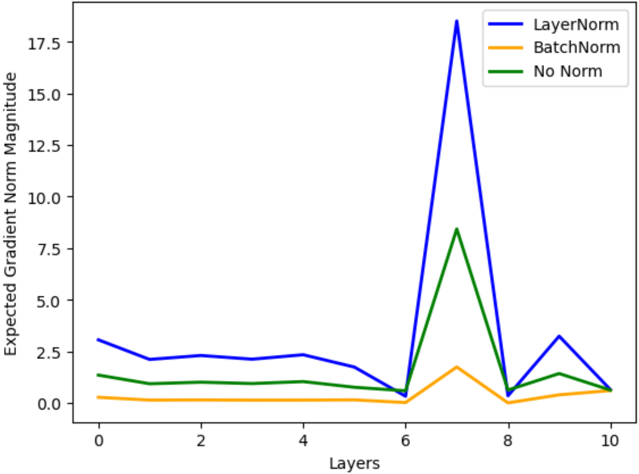

Figure 2: Expected Gradient Magnitude Norm - Shallow Models

Initial Gradients Magnitude - Deep Models

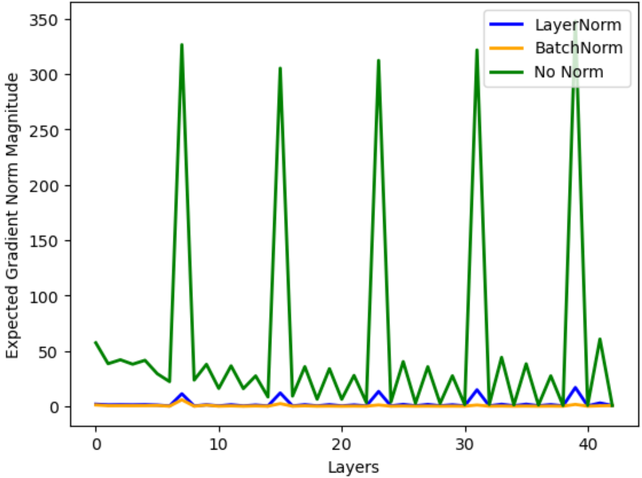

Figure 3: Expected Gradient Magnitude Norm - Deep Models

Initialization Needs

Replacing Kaiming Initialization and Xavier Initialization to Uniform Initialization.

- small scale ($U(-0.1, 0.1)$): work well for all models

Initialization Needs

Replacing Kaiming Initialization and Xavier Initialization to Uniform Initialization.

- small scale ($U(-0.1, 0.1)$): work well for all models
- large scale ($U(-100, 100)$): destructive for all models

Loss Plot

In the following experiment, this study chooses wide and 1-layer Transformer Encoder and set learning rate to make sure all models could converge.

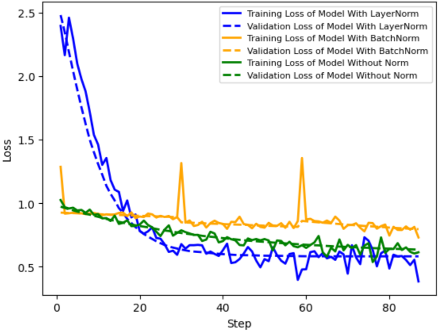

Figure 4: Loss Pots (dash lines for validation and full lines for train)

Lipschitz of Loss function

Lipschitz of loss function is defined as (lr refers to learning rate) [9]

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

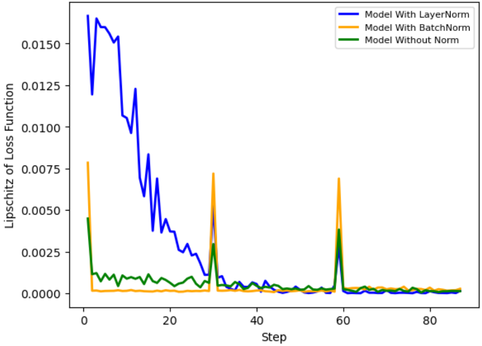

Figure 5: Lipschitz of Loss Function

Smoothness of Loss function

Smoothness of loss function is defined as [9]

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

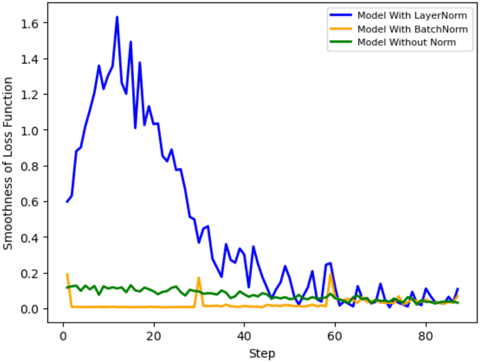

Figure 6: Smoothness of Loss Function

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.

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

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

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

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

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

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

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

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

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

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

[15] 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.

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

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

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

Thanks!