Improving Stability of Fine-Tuning Pretrained Language Models via Component-Wise Gradient Norm Clipping

Chenghao Yang[1], **Xuezhe Ma**[2]
[1]University of Chicago
[2]University of Southern California
yangalan1996@gmail.com, xuezhema@isi.edu

Abstract

Fine-tuning over large pretrained language models (PLMs) has established many state-of-the-art results. Despite its superior performance, such fine-tuning can be unstable, resulting in significant variance in performance and potential risks for practical applications. Previous works have attributed such instability to the catastrophic forgetting problem in the top layers of PLMs, which indicates iteratively fine-tuning layers in top-down manner is a promising solution. In this paper, we first point out that this method does not always work out due to different convergence speeds of different layers/modules. Inspired by this observation, we propose a simple component-wise gradient norm clipping method to adjust the convergence speed for different components. Experiment results demonstrate that our method achieves consistent improvements in terms of generalization performance, convergence speed and training stability. The codebase can be found at https://github.com/yangalan123/FineTuningStability.

1 Introduction

Fine-tuning over large pretrained language models (PLMs), which achieved remarkable performance over various benchmarks, has become the de facto paradigm for several current natural language processing (NLP) systems. However, fine-tuning can be unstable in terms of significant variance in metrics, resulting in even worse-than-random failed models (Devlin et al., 2019; Lee et al., 2019; Dodge et al., 2020; Mosbach et al., 2020).

Catastrophic forgetting (Kirkpatrick et al., 2017) during fine-tuning of PLMs is one common explanation for this instability (Lee et al., 2019), i.e., PLMs may lose their rich domain-agnostic knowledge acquired by language model pretraining in the process of fine-tuning. Through layer-replacement experiments between pretrained models and fine-tuned models, Mosbach et al. (2020) further connected the catastrophic forgetting problem to the optimization problem on top layers.

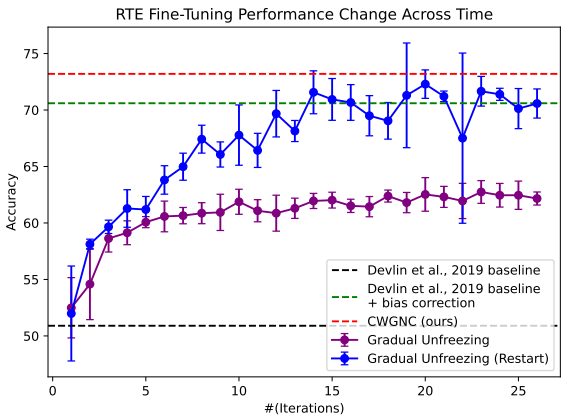


Figure 1: Fine-tuning performance over time on RTE datasets. Here we fine-tune over BERT-large-uncased model (Devlin et al., 2019). In each iteration, we train the model for 3 epochs and the errorbar is plotted based on 5 different runs.

These findings give rise to a straightforward way to enhance the fine-tuning stability: how about fine-tuning the model from top to bottom to reduce the parameter changes and hence mitigate the catastrophic forgetting problem? This is reminiscent of the gradual unfreezing (Howard and Ruder, 2018), which does layer-wise top-down fine-tuning and unfreezing new layers only when the layers above have been fine-tuned. Therefore, the newly unfrozen layer would only be tuned for a slightly easier optimization problem at each iteration, leading to much fewer changes to the parameters.

However, based on a comprehensive case study of the gradual unfreezing method, we obtained empirical observations beyond our expectations (§2). Our analysis further reveals a possible reason: *different components (e.g., feed-forward networks at different layers, fully connected matrices and biases at output layer) converge at varying speeds.* Thus, components in upper layers, which have converged to local optima, cannot easily be fine-tuned

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 4854–4859
December 7-11, 2022

with newly unfrozen parameters.

Based on this observation, we propose a simple component-wise gradient clipping method to stabilize the fine-tuning process (§3). This method achieves significant empirical improvements of fine-tuning stability in terms of the variance and the failed run percentage over three tasks (§4). In summary we make the following contributions:

1. We find that component-wise convergence speed divergences is the key challenge in fine-tuning stability, based on the case study of the gradual unfreezing method.

2. Based on our observation, we propose a new simple component-wise gradient clipping method to help stabilize fine-tuning, which achieves empirical improvements of fine-tuning stability over previous methods.

2 A Bitter Case Study: Layer-wise Gradual Unfreezing

Mosbach et al. (2020) attributed the instability problem in fine-tuning process to the catastrophic forgetting in the top layers, through a layer replacement experiment between pretrained and fine-tuned models. Following this empirical observation, the instability problem might be mitigated if we can minimize the edits to the pretrained model parameters, especially in top layers. This inspires us to mitigate the instability problem via gradual unfreezing (GU, Howard and Ruder (2018)).

Specifically, suppose we are working with a model M with L layers parameterized by $\{\theta^{(i)}, 1 \leq i \leq L\}$. GU tunes M for L iterations. At k-th (k start from 0) iteration, we only tune a subset of parameters $R^{(k)} = \{\theta^{(i)}, L-k \leq i \leq L\}$, where $\theta^{(i)}(L-k+1 \leq i \leq L)$ is also tuned in $k-1$-th iteration. In each iteration, we tune the parameter for E epochs, where E is large enough for convergence.[1] Detailed algorithm is shown in Algorithm 1 at Appendix A.

Failure of Gradual Unfreezing From Fig. 1, the accuracy of gradual unfreezing is significantly worse than full fine-tuning,[2] although it indeed achieves smaller update to pretrained model parameters compared with full fine-tuning (Fig. 2a).

Convergence Racing between Parameters To investigate the reason behind this unsatisfying performance of GU, we plot the component-wise maximum update (measured by component-wise maximum rooted mean squared difference,[3] as different layers can have multiple components with different dimensions) at each layer in Fig. 2b. Clearly, from the very beginning, the parameter updates for both early-tuned parameters and newly-unfrozen parameters have quickly diminished in GU so not many updates happen later when new parameters join. Based on this observation, we hypothesize that the failure of GU is because the early-tuned parameters have already converged in early iterations and cannot be re-activated later to adapt for newly-unfrozen parameters.

To verify this hypothesis, we simply modify GU to stop using early-tuned parameters in the previous iteration and instead copy the weight from the pretrained BERT model. For simplicity, we term this method as "GU (restart)". Note that full fine-tuning is just the last iteration of GU (restart) when all layers are unfrozen. We plot GU (restart) performance in Fig. 1, and observe that GU (restart) achieves consistently much better performance than GU. However, GU (restart) is much more unstable than GU, and the best performance is only reached when almost all layers have been tuned.

Unbalanced Gradients across Parameters To investigate the cause of the convergence racing problem, we analyze the distribution of parameters' gradients from different components, by plotting the gradient norm in Fig. 3.[4] Here we follow the terminology in previous work (Dodge et al., 2020; Mosbach et al., 2020) that, if a fine-tuned model checkpoint can't beat the majority classifier,[5] then it is a "Failed Run" and "Success Run" otherwise. Comparing Fig. 3b and Fig. 3a, we can find that convergence racing still exists under normal full fine-tuning. The success run wins by making all parameters update roughly following the same trends and making gradient norms well-bounded.

Discussion on Fine-Tuning Stability From the case study over GU, we observe that there is a convergence racing problem between parameters of model components, which would lead to incompe-

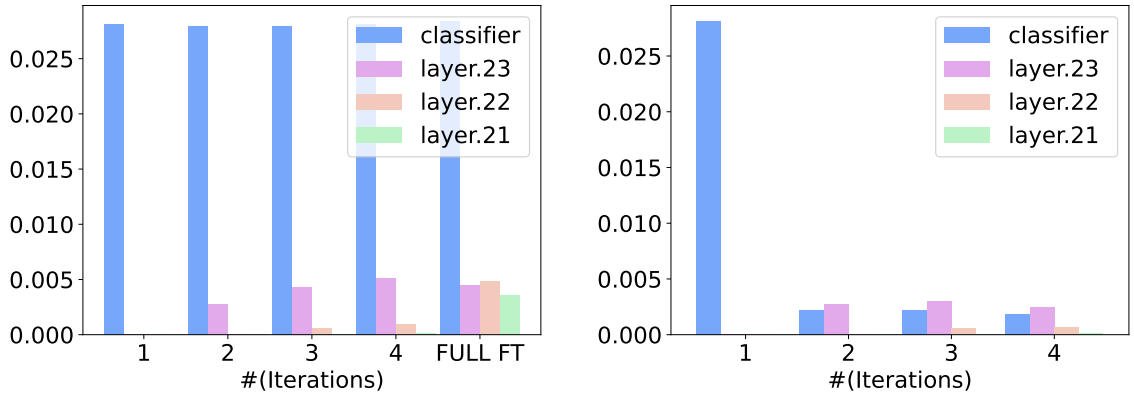
[1] We select E based on preliminary experiments.

[2] As pointed out by Mosbach et al. (2020), without bias correction, the original results in Devlin et al. (2019) paper can be pretty bad and unstable, so we add bias correction on top of Devlin et al. (2019) to make a strong baseline.

[3] We also plot the parameter change measured by cosine distance and put the figures in Appendix C

[4] In Fig. 3b and Fig. 3a, we do the fine-tuning by following the recommended setting in (Devlin et al., 2019).

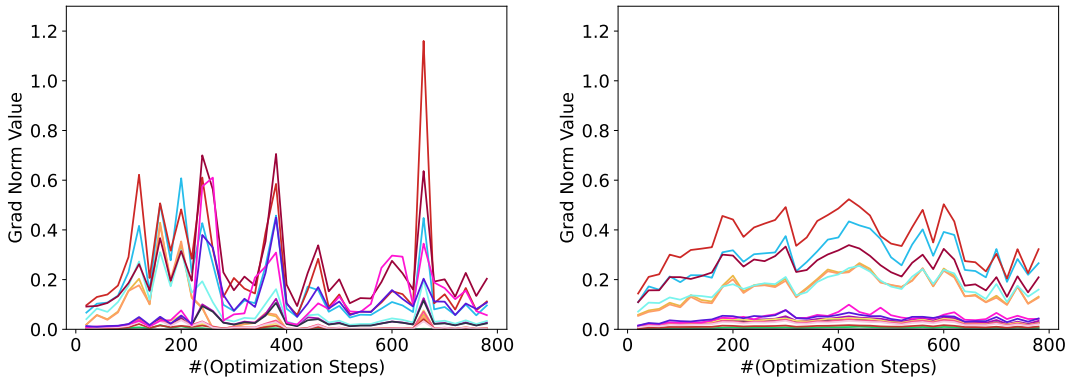
[5] Here "majority classifier" means simply using the majority labels in the training dataset as the predicted label.



(a) Compared with original BERT model

(b) Compared with previous iteration

Figure 2: Parameter update at each iteration for GU. Updates that are too small cannot be seen in this figure (e.g., for the 22-th layer, or "layer.21" in the figure update is too small to plot out in Fig. 2b). We only plot the first 4 iterations as we observe that the performance is almost stable by the fourth iteration. As different layers can have components with different dimensionalities, we show the component-wise maximum rooted mean squared difference for each layer.



(a) Failed Run

(b) Success Run

Figure 3: Gradient norm across different parameters at layer 22 on RTE dataset fine-tuning. Here the left figure ("Failed Run") refers to the case when the fine-tuned model cannot beat the majority classifier and the right figure ("Success Run") otherwise. Different colors represent different parameters. Due to space limitation, legends are omitted and we cannot show results from all layers. But in our observation, most layers have similar phenomenon.

tent or unstable performance. Based on existing studies, we argue that a robust PLM fine-tuning method should satisfy the following principles:

1. Updating the full set of parameters, not just a subset of layers.

2. No significant parameter updates to avoid *catastrophic forgetting* problem.

3. Adjusting the convergence speed for parameters to mitigate the *convergence racing*.

Mosbach et al. (2020) proposed to use small learning rate (e.g., $1e^{-5}$) to mitigate *catastrophic forgetting*, together with longer fine-tuning process. However, unbalanced gradients across different components is another factor for *catastrophic forgetting* and *convergence racing*, which is not resolved by only using small learning rates.

3 The Component-Wise Gradient Norm Clipping Method

Bearing these principles in mind, we propose the component-wise gradient norm clipping (CWGNC) method. The components are different parameters serving different functionalities. For example, in a Transformer-based neural architecture, components can be the weight matrices of Key-Query-Value in Transformer architectures, the weight and

Approach	RTE			MRPC			COLA		
	Std	Mean	Max	Std	Mean	Max	Std	Mean	Max
Devlin et al. (2019)	4.5	50.9	67.5	3.9	84.0	91.2	25.6	45.6	64.6
+ bias correction	4.0	70.6	75.5	2.1	89.2	**92.2**	11.1	59.2	64.1
Lee et al. (2019)	7.9	65.3	74.4	3.8	87.8	91.8	20.9	51.9	64.0
Mosbach et al. (2020)	2.7	67.3	71.1	0.8	90.3	91.7	1.8	62.1	**65.3**
CWGNC	**1.3**	**73.1**	**75.1**	**0.6**	**90.5**	91.5	**1.7**	**62.2**	65.0

Table 1: Experiment results for fine-tuning benchmark. Boldfaced numbers are best under each criterion.

bias terms in feed-forward layers, etc. For those optimizers which maintain first-order and/or second-order bias correction terms (e.g., Adam (Kingma and Ba, 2015) and AdamW (Loshchilov and Hutter, 2018) as the popular optimizer in PLM literature), our proposed norm clipping operation happens before the bias correction terms computation and does not interfere with normal bias correction process.

By clipping the gradient norm of each component individually, we aim to balance the distribution of gradients across different components to adjust their convergence speed, hence mitigating the convergence racing problem.

4 Experiments

To evaluate our CWGNC method, we follow Mosbach et al. (2020) to run our CWGNC method over 25 different runs and report aggregated results on the validation set of three different datasets: RTE (Acc), MRPC (F1) and COLA (MCC). We report the standard deviation (Std), averaged performance (Mean) and maximum performance (Max). Hyperparameters are tuned on held-out data. More implementation details are in Appendix B.

Baseline Setting For baselines, we first consider the original BERT paper reported results (Devlin et al., 2019) and we run original BERT fine-tuning with bias correction to make a stronger baseline following the observation in (Mosbach et al., 2020). We also compare to Mixout regularization methods (Lee et al., 2019) and "simple but hard-to-beat baseline" proposed by Mosbach et al. (2020).

Experiment Results Experiment results are shown in Table 1. Here we can see our CWGNC achieves significantly better performance in terms of averaged performance and standard deviation. Compared with previous state-of-the-art results from (Mosbach et al., 2020), our method only needs to tune 5 epochs and does not need to wait for 20 epochs even on these small datasets so our method indeed converges faster.

Our method is also robust to a wide range of the selection of gradient norm clipping thresholds. We show this in Table 2 on COLA dataset as we find that fine-tuning methods are particularly unstable on COLA dataset. Here we see that within a reasonable range of threshold (< 1), the model performance would be mostly maintained and the standard deviation is well controlled. If we use a significantly larger threshold ($>= 1$), less control is enforced by CWGNC and it will degrade to normal full model fine-tuning if we further increase the threshold.

Threshold	COLA		
	Std	Mean	Max
0.01	1.3	61.6	64.3
0.05	1.7	62.2	65.0
0.1	1.4	61.4	65.1
0.5	1.3	61.6	64.1
1	1.3	61.4	63.8
5	15.4	57.3	65.0

Table 2: CWGNC fine-tuning results on COLA under different gradient norm clipping thresholds.

5 Conclusion

In this paper, we investigate the instability problem of fine-tuning large pretrained language models. Inspired by previous works, we first experiment with gradual unfreezing methods, which should help minimize the updates in top layers and ease the catastrophic forgetting problem. However, further experiment results do not support that and we find it is because there is a convergence racing issue between different parameters, namely that early-converged parameters can limit the search space for other parameters. Based on this finding, we propose to do component-wise gradient norm clipping and achieve significant improvement on averaged performance, smaller standard deviation and

quicker convergence speed. Our method is robust to the selection of gradient norm clipping threshold. In the future, we will try to study whether the component racing problem also exists in pretrained language models with different sizes and different pretraining methods.

6 Limitations

In this paper we mainly work with one particularly popular large pretrained language model BERT (Devlin et al., 2019). While we believe our empirical investigation and conclusion is widely applicable to a wide range of current transformer-based large pretrained language models, more experiments and theoretical explanations are needed for further research. Also, due to computational resources limitation, we cannot investigate whether the most recent large models like T0 (Sanh et al., 2022) are stable under fine-tuning. We follow the evaluation protocols in previous works (Dodge et al., 2020; Lee et al., 2019; Mosbach et al., 2020) to investigate the instability of fine-tuning on small datasets including COLA, RTE and MRPC. But for real challenging low-resource situation, we believe it can be more complicated and more investigation is needed.

Acknowledgements

This material is based on research sponsored by Air Force Research Laboratory (AFRL) under agreement number FA8750-19-1-1000. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation therein. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Laboratory, DARPA or the U.S. Government.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. 2019. Mixout: Effective regularization to finetune large-scale pretrained language models. In *International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2020. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. In *Proceedings of ICLR*.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. 2022. Multitask prompted training enables zero-shot task generalization. *Proceedings of ICLR*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

A Gradual Unfreezing Algorithm

The gradual unfreezing algorithm we implemented following Howard and Ruder (2018) is shown in Algorithm 1.

Algorithm 1 Gradual Unfreezing Fine-Tuning

Require: A multi-layer Transformer PLM M with its L-layer parameters $\{\theta^{(i)}, 1 \leq i \leq L\}$. The maximum number of iterations T, the dataset D, the optimizer Opt, the number of epochs for each iteration E.

Ensure: $1 \leq T \leq L$

$\hat{R}^{(0)} \leftarrow \phi$ ▷ $\hat{R}^{(l)}$ represents to-be-tuned parameters at l-th iteration.

while $T \leq L$ **do**

$\hat{R}^{(T)} \leftarrow \hat{R}^{(T-1)} \cup \{\theta^{(L-T+1)}\}$

for $j = 1 \rightarrow E$ **do**

$\mathcal{L} \leftarrow \text{Forward}(M, D)$

$G \leftarrow \text{Backward}(\mathcal{L}, L-T+1, L)$

▷ Only needs to compute the gradient for top-T layers

$\text{Update}(Opt, G, \hat{R}^{(T)})$

▷ Apply G over $\hat{R}^{(T)}$

$\text{Replace}(M, \hat{R}^{(T)}, L-T+1, L)$

▷ Replace the updated layers back to make sure the updated parameters are involved in next epoch forward process

end for

$T \leftarrow T+1$

end while

B Implementation Details

Our codebase is based on Huggingface Transformers (Wolf et al., 2020) example fine-tuning scripts and will be released later. We tune models using our method for 5 epochs. For weight decay and warm-up steps, we follow the settings original fine-tuning method as described in (Devlin et al., 2019). We here report the result with clipping threshold 0.05 as we empirically find it works better on held-out dataset. We also show later that our method is actually pretty robust to a wide range of threshold picking in Table 2.

C Gradient Update Measured by Cosine Similarities

In the main text we measures the update via root-mean-square difference. Here we show that if measured by cosine-similarity, we would obtain similar conclusion. Here, Fig. 4 is the cosine similarity version for Fig. 2b. Fig. 5 is the cosine similarity version for Fig. 2a. Note that because smaller cosine similarity indicates more changes, in contrast to square-mean-root difference, we use the component-wise **minimum** cosine similarity to represent the update at each layer.

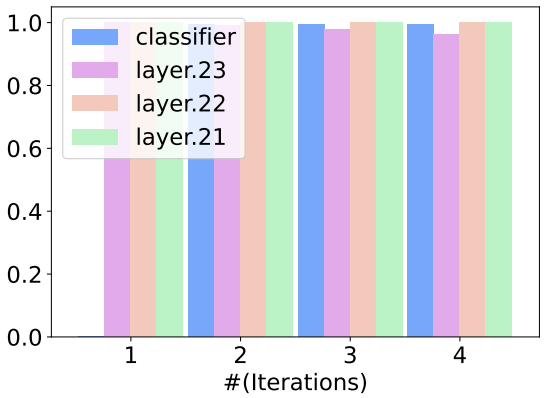


Figure 4: The scales of **incremental** parameter update at each iteration in GU (**compared with previous iteration, measured by cosine similarity**). As different layers can have different components with different dimensionalities, we show the component-wise maximum rooted mean squared difference for each layer. We only plot the first 4 iterations as we observe that the performance is almost stable by the 4-th iteration. 22-th layer ("layer.21" in the figure) update is too small to plot out in this figure.

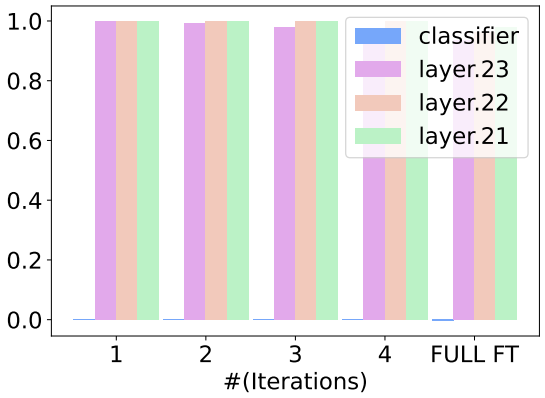


Figure 5: The scales of parameter update at each iteration in GU (**compared with original BERT model, measured by cosine similarity**). Too small updates cannot be seen in this figure. We only plot the first 4 iterations as we observe that the performance is almost stable in 4-th iteration.