Sparse Grad: a new selective method for efficient fine-tuning of MLP layers

Anonymous ACL submission

Abstract

The performance of Transformer models has been improved by increasing the number of parameters and the length of the processed text. Due to this, fine-tuning of the entire model becomes a memory-consuming process. High-performance methods for parameter-efficient fine-tuning (PEFT) work usually with Attention blocks and do not take into account the MLP blocks, which contain about a half of the model parameters. We propose a new selective PEFT method Sparse Grad that performs well on such MLP. We find transition matrices that transfer layer gradients to space, where only about 1% of its elements remain significant. By converting gradients into a sparse structure, we reduce the number of updated parameters. We applied the method in fine-tuning the BERT and RoBERTa on the NLU task and the LLaMa-2 on the Question-Answering task. In these experiments, our method provides a higher quality than LoRA - a strong state-of-the-art PEFT baseline.

1 Introduction

Due to the tendency to increase the size of transformer models of each new generation, we need efficient ways to train such models on downstream task data. The usual practice is fine-tuning a large pre-trained foundation model on a downstream task. The major problem that prevents efficient fine-tuning is a steady increase in the memory footprint. One of the best strategies to alleviate memory pressure is high-performance methods for parameter-efficient fine-tuning (PEFT). Typically, such methods (e.g., LoRA (Hu et al., 2021)) focus on attention blocks and do not consider dense MLP blocks. Since MLP blocks can take a significant fraction of the model parameters (see Table 1), we propose to focus on MLP blocks instead. We introduce a novel selective PEFT approach called Sparse Grad. Our method is based on finding a special sparsification transformation that allows us to fine-tune about a 1% of the dense MLP layer parameters and still show good performance in downstream tasks.

Table 1: Number of parameters for different layers in various Transformer architectures.

Blocks/Model	BERT		RoBERTa	
Full model	109 M	100%	125 M	100%
MLP	57 M	52%	57 M	45%
Embeddings	24 M	22%	40 M	31%
Attention	28 M	26%	28 M	22%

We validate our approach on BERT (Lan et al., 2020) and RoBERTa model on GLUE (Wang et al., 2018a) benchmark and in both cases obtain results better than LoRA (Hu et al., 2021) method. We also fine-tune Llama (Touvron et al., 2023) 2.7B on the Opeassitant dataset (Köpf et al., 2023) and also achieve performance higher than LoRA.

2 Related Work

In the last few years, many approaches to PEFT have appeared. Lialin et al. (2023) distinguishes three types of methods: additive, reparametrization-based, and selective.

In additive PEFT, small neural networks called adaptors are added to the main model to steer the outputs of its modules (Pfeiffer et al., 2020). Adapters are trainable, therefore, the main model remains unchanged. Houlsby et al. (2019) adapt this approach to NLP. The next step in expanding this type of PEFT was an attempt to present "additional" parameters in a more efficient form. In Reparametrization-based approaches we use low-rank representations of it. For example, the LoRA model (Hu et al., 2021) parameterizes the weight update that represents it by decomposition of the low-rank trainable matrix. In the original paper, LoRA is applied to self-attention modules, but not to MLP ones.

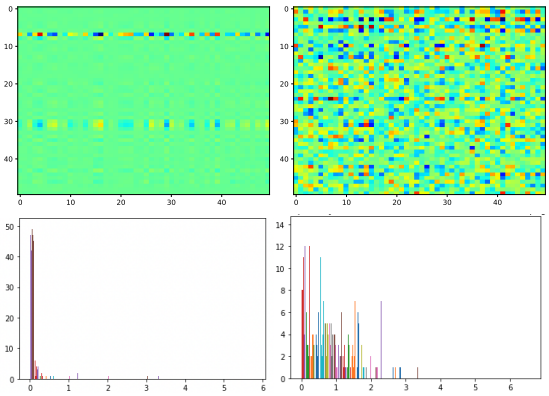

Figure 1: Example of the gradient sparsity in a new basis: footprints of $U\frac{\partial L}{\partial W}V.T$ and $\frac{\partial L}{\partial W}$ of 5-th BERT MLP block with elements histograms.

In the selective methods, we choose parts of the model or a set of parameters for fine-tuning using some heuristics. Such methods include, for example, Bit Fit (Zaken et al., 2021), where only the layers' biases are trained, and the matrices W remain unchanged. The approach proposed in this paper is related to selective methods.

3 Method

With the aim of reducing the trainable parameters in the layer, we assumed that there is a space where the weight gradient matrix has mostly zero values. We found such a space by collecting information about gradients in the usual pre-training process and appliying tensor decomposition to it. The appearance of such a "sparse gradient" space means that the layer weights should also be considered in this space - this entailed changes to the formulas in the forward and backward propagation.

3.1 Pre-training: Finding The Proper Transition Matrices

To obtain transition matrices, a pre-train procedure is necessary. During pre-training, we perform several steps of standard training by freezing the entire model and unfreezing only the linear layers in MLP blocks. By stacking its gradient matrices for the weights $W \in \mathcal{R}^{D_in \times D_out}$ over all blocks over several training steps, we obtain a 3D tensor of size $D_in \times D_out \times (n_step * n_blocks)$, where n_steps - number of pre-training steps, n_blocks - number of MLP blocks in the model. The Tucker decomposition (Tucker, 1966c) of this tensor gives three factor matrices $U \in \mathcal{R}^{D_in \times D_in}, V \in \mathcal{R}^{D_out \times D_out}, S \in \mathcal{R}^{(n_step * n_blocks) \times (n_step * n_blocks)}$, corresponding to each dimension of the tensor and identical for the entire model. S relates to the dimention over which we stack and remain unusable. U, V can transfer layer gradients (and corresponding layer weights W) into the space where they are sparse and the number of trainable parameters can be reduced.

The memory requirements of the pre-training action are assessed according to the following: we need to unfreeze the MLP layers, which maintain approximately half of the training parameters in Transformers (see Table 1), and store and decompose a large tensor. The last part is the most memory-consuming - for example, for the BERT model, 30 steps of pre-train yield tensor of approximately 6 GB size. However, this part is performed once during the entire fine-tuning and can be computed on the CPU. As additional extra parameters, we get two matrices U, V that are shared across all modules of the model. Multiplication of these matices and $\frac{\partial L}{\partial W}$ turn out to be sparse. Examples of $\frac{\partial L}{\partial W}$ with and without transmission to the new space are in Fig. 1.

3.2 Layer with Sparse Gradients

Having weight gradients $\frac{\partial L}{\partial W}$ in a new "sparse" space, defined by U, V, we also consider the layer weights W there. We created a new PyTorch layer class, **SparseGradLinear**, where the weights are defined as $\tilde{W} = UWV^T$. Since the module following **SparseGradLinear** in forward and backward passes does not change the basis, we must guarantee that the output $Y = XW^T$ and input gradients $\frac{\partial L}{\partial X}$ remain unchanged in the training procedure. We made changes to the Torch Autograd function which take into account transmision matrices in **SparseGradLinear**. Table 3 shows the correspondence of the variables in autograd functions for the linear layer and the **SparseGradLinear**.

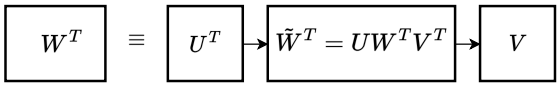

Figure 2: In the new basis, a linear layer can be represented as 3 consecutive linear layers: first with frozen weights U^T, second with trainable new weights $\tilde{W}$, third with frozen weights V.

The **SparseGradLinear** layer is equivalent to 3 linear layers: first with frozen weights $U.T$, middle with trainable weights $\tilde{W} = UWV^T$, last with

Table 2: Comparative results of BERT 1-epoch task-specific fine-tuning.

Method	Number of tunable parameters	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
Regular FT	109 mln.	0.78	0.89	0.557	0.83	0.85	0.885	0.885	0.675	0.926	0.56
LoRA	53 mln.	0.77	0.88	0.54	0.82	0.81	0.89	0.89	0.64	0.92	0.54
Sparse Grad	53 mln.	0.79	0.89	0.57	0.82	0.86	0.898	0.89	0.70	0.92	0.56

Table 3: Correspondence of variables in Torch Autograd for a regular Linear layer and SparseGradLinear

Variable / Layer	Original	Sparsified
Weights	W^T	$\tilde{W}^T = UW^TV^T$
Input	X	$\tilde{X} = XU^T$
Output	$Y = XW^T$	$\tilde{Y} = \tilde{X}\tilde{W}^T$ $= XU^TUW^TV^TV$
Grad Output	$\frac{\partial L}{\partial Y}$	$\frac{\partial L}{\partial Y}V^T$
Grad Input	$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y}W$	$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y}V^T\tilde{W}U$
Grad Weights	$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y}^T X$	$\frac{\partial L}{\partial \tilde{W}} = \frac{\partial L}{\partial \tilde{Y}}^T \tilde{X}$ $= V\frac{\partial L}{\partial Y}^T XU^T$

frozen weights V (see Fig. 2). As can be seen from the Table 3, the $\frac{\partial L}{\partial W}$ is obtained by multiplying the input and the output gradient $\frac{\partial L}{\partial Y}^T$. One of these factors, the output gradient, also turned out to be sparse on the new basis. We transfer this multiplication to sparse-dense matrix operations, which allows you to reduce the total computation time and mitigate the loss of time on multiplications to the transmission matrices.

Having gradients with a set of near-zero values, we transfer them to the sparse structure. We explore the sparse gradients matices on various BERT and RoBERTa modules. We found that approximately 0.1 to 10% of the elements in the matrix $\frac{\partial L}{\partial W}$ remain significant. Based on this heuristic, in our experiments we leave the top 1% of the largest elements and set the rest to zero. To deal with sparse gradients, we use an SparseAdam optimizer [1] - masked version of the Adam algorithm. The remaining model parameters are changed with the standard optimizer.

4 Limitations: Time-Stableness of the Sparsity Feature

The main limitation of our approach is a possible loss of sparsity in a long-term training. Transmission matrices U, V, determined at the beginning of the training, provide space, where weight gradients are sparse, but this feature may not be preserved. Fig. 3 shows that as training time progresses, $\frac{\partial L}{\partial W}$ can become less sparse. It limits the method to fine-tuning only. For long-term full training, we should consider meeting the sparsity condition.

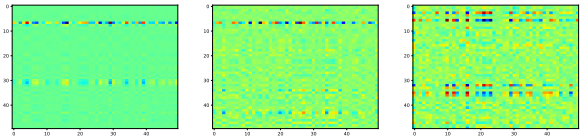

Figure 3: From left to right: gradients of the SparseGradLinear in the 5 MLP block of the BERT at 10, 500 and 3000 iterations from the beginning of training.

5 Experiments

We conduct experiments on two transformer-based encoder models, BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019), on the GLUE (Wang et al., 2018b) benchmark, and the LLaMa-2 decoder model (Touvron et al., 2023) on the OpenAssistant Conversations corpus (Köpf et al., 2023). We compared the fine-tuning of the full model (Regular FT scheme) with two PEFT methods, namely LoRA and SparseGrad. To harness LoRA, we use an official repository code and obtain a low-rank approximation of δW in the fully-connected (FC) layers in MLP blocks. The proposed SparseGrad involves replacing the FC layers in the model with its **SparseGradLinear** equivalents. In all experiments, we repeat fine-tuning 5 times over different seeds and report average score.

5.1 Natural Language Understanding with BERT

We fine-tune BERT under Regular FT, LoRA, and Sparse Grad schemes for 1 epoch for every task in the benchmark GLUE. We varied the batch size and the learning rate using the Optuna framework (Akiba et al., 2019). The learning rate ranged from $1e-3$ to $1e-5$, and the batch size

[1] https://pytorch.org/docs/stable/generated/torch.optim.SparseAdam.html

Table 4: Comparative results of ROBERTA 1 epoch task-specific fine-tuning.

Method	Trainable parameters	%	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
Regular FT	125 mln.	100	0.80	0.89	0.59	0.87	0.88	0.92	0.89	0.69	0.94	0.56
LoRA	68 mln.	54	0.73	0.89	0.43	0.86	0.84	0.91	0.63	0.58	0.94	0.51
Sparse Grad	68 mln.	54	0.78	0.88	0.54	0.86	0.86	0.90	0.87	0.63	0.94	0.56

ranged from 8 to 16. Optimal training parameters for each task are available in the Appendix 8.

We employ LoRA with rank 7 and Sparse Grad which remains approximately 1% of the layer parameters. While Regular FT had 109 million trainable parameters, both Sparse Grad and LoRA had 53 million trainable parameters. Our findings, presented in Table 2, suggest that Sparse Grad outperforms LoRA with the same number of trainable parameters. Sparse Grad sometimes also surpasses Regular FT. This could be attributed to the fact that the changing basis of the weights in the SparseGrad may act as a form of regularisation.

5.2 Natural Language Understanding with RoBERTa

In order to find optimal parameters for fine-tuning Roberta (Liu et al., 2019), we use the Optuna framework. The range of learning rate and batch size is the same as in the case of BERT. We fine-tune RoBERTa with obtained parameters for the 1 epoch towards every GLUE task. Method Regular FT provides 125 mln trainable parameters, whereas LoRA and Sparse Grad require twice smaller trainable parameters. We use the rank-7 approximation of the δW matrix in LoRA case, while Sparse Grad maintained $\approx 1\%$ non-zero elements in gradients. Table 4 shows that our method has a performance better than LoRA on almost all tasks. The exception are QNLI and STSB datasets, where SparseGrad performed slightly worse.

5.3 Conversations with LLaMa-2

We apply the Sparse Grad method to file-tune LLaMa-2 (Touvron et al., 2023) model with 7 billion parameters on the OpenAssistant conversational dataset (Köpf et al., 2023) consisting of 161,443 messages. Fine-tining was performed on a single GPU NVIDIA A40 during 1 epoch with learning rate $9e-4$. We load LLaMa-2 in *bfloat* representation; for full-parameter fine-tuning, we unfroze *up_proj* and *down_proj* layers in the MLP modules with a number divisible by 3 $(0, 3, 6, \ldots)$. We apply LoRA with rank 32 to the selected blocks, leaving the rest of the model untrainable. In the Sparse Grad method, we also consider selected MLP modules in the transformer, and leave $\approx 100,000$ (0, 2%) nonzero elements in the gradient matrix. To calculate transition matrices U, V in the pre-training stage, the accumulated gradients were converted to *float32* and then back to *bfloat*. The resulting losses are shown in Table 5: SparseGrad also surpasses LoRA, but is inferior to full-parameter fine-tuning.

Table 5: Comparative results for LLaMa-2 fine-tuning on OpenAssistant-1 dataset.

Method	Trainable parameter	Loss before FT	Loss after FT
Unfrozen MLP	22%	1.59	1.247
LoRA on MLP	0.5%	1.59	1.265
Sparse Grad on MLP	0.5%	1.59	1.254

6 Conclusion

Fine-tuning of large transformer models is prohibitively expensive and needs to be more efficient. We proposed a new selective PEFT method Sparse Grad. Sparse Grad finds a space where the gradients have a sparse structure and update only its' significant part. Using the proposed approach, we can update only $0.1-1\%$ of the layer parameters without loss of performance. The method can be applied to linear layers and works well on MLP blocks. This part of the transformers is significant in size and is usually not covered by PEFT methods.

Sparse Grad is supported by experiments on models BERT and RoBERTa, where our method performs better than the additive method LoRA. We also fine-tune the large enough LLaMa-2 model on a conversesational corpus - the proposed method works better than LoRA, but worse than the full-parameter fine-tuning.

The source code is available at anonimized repository: `https://anonymous.4open.science/r/sparse_grads-0C5E/`

7 Ethics Statement

Our proposed approach involves a novel method for fine-tuning large language models, which can be considered as cost-effective as we only update 0.1%f the weights. This type of fine-tuning is environmentally friendly as it reduces resource wastage.

We utilized pre-trained models from the Hugging Face repository and implemented updates using the Pytorch library.

We exclusively used open-source datasets to avoid any potential harm or ethical concerns.

By prioritizing ethical standards and recognizing potential risks, we strive to promote responsible and sustainable research practices.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. 97:2790–2799.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. 2023. Openassistant conversations – democratizing large language model alignment.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulic, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. *CoRR*, abs/2007.07779.

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

L. R. Tucker. 1966c. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 353–355. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018b. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *CoRR*, abs/2106.10199.

8 Appendix

Table 6: Best training parameters on GLUE benchmark for BERT model.

Dataset	batch size	learning rate	epoch
STSB	10	4.74e-5	1
CoLA	4	1.23e-5	1
MNLI	8	3.43e-5	1
MRCP	15	5.28e-5	1
QNLI	32	4.7e-5	1
QQP	8	1.5e-5	1
RTE	8	3.18e-5	1
SST2	15	4.71e-5	1
WNLI‘	12	3.33e-5	1

Table 7: Best training parameters on GLUE benchmark for RoBERTa model.

Dataset	batch size	learning rate	epoch
STSB	16	2e-5	1
CoLA	4	1.5e-5	1
MNLI	8	7e-6	1
MRCP	8	3e-5	1
QNLI	8	1e-5	1
QQP	8	1.3e-6	1
RTE	16	2.5e-5	1
SST2	32	1e-5	1
WNLI‘	32	1e-2	1