SparseGrad: Parameter-Efficient Fine-Tuning via Sparse Gradient Transformations

Artificial Intelligence Research Institute
Skolkovo Institute of Science and Technology

Abstract

The performance of Transformer models has been enhanced by increasing the number of parameters and the length of the processed text. Consequently, fine-tuning the entire model becomes a memory-intensive process. High-performance methods for parameter-efficient fine-tuning (PEFT) typically work with Attention blocks and often overlook MLP blocks, which contain about half of the model parameters. We propose a new selective PEFT method, namely SparseGrad, that performs well on MLP blocks. We transfer layer gradients to a space where only about 1% of the layer's elements remain significant. By converting gradients into a sparse structure, we reduce the number of updated parameters. We apply SparseGrad to fine-tune BERT and RoBERTa for the NLU task and LLaMa2 for the Question-Answering task. In these experiments, with identical memory requirements, our method outperforms LoRA and MeProp, robust popular state-of-the-art PEFT approaches.

1 Introduction

Due to the tendency to increase the size of transformer models with each new generation, we need efficient ways to fine-tune such models on downstream task data. The usual practice is fine-tuning a large pre-trained foundational 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 is high-performance methods for parameter-efficient fine-tuning (PEFT). Typically, such methods as 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 **??**), we propose to focus instead on MLP blocks. We introduce a novel selective PEFT approach called SparseGrad. Our method is based on finding a special sparsification transformation that allows us to fine-tune about 1% of the dense MLP layer parameters and still show good performance in downstream tasks.

We validate our approach on BERT (Devlin et al., 2019) and RoBERTa (Zhuang et al., 2021) models on GLUE (Wang et al., 2019) benchmark and in both cases obtain results better than LoRA (Hu et al., 2021) and MeProp (Sun et al., 2017) methods. We also fine-tune LLaMa-2 (Touvron et al., 2023) 2.7B on the OpenAssistant dataset (Köpf et al., 2023) and also achieve performance higher than LoRA and MeProp.

Table 1: Number of parameters for different layers in models based on the Transformer.

Blocks/Model	BERT	RoBERTa	LLaMa-2
Full model	109 M	100%	125 M
MLP	57 M (52%)	[MISSING]	57 M
Embeddings	24 M (22%)	[MISSING]	40 M
Attention	28 M (25%)	[MISSING]	28 M

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 adapters 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. In reparametrization-based approaches low-rank representations of trainable parameters are used. For example, LoRA (Hu et al., 2021) parameterizes the weight update by a trainable low-rank matrix decomposition. In the original paper, LoRA is applied to self-attention modules, but not to MLP ones. In the selective methods, parts of the model or sets of the parameters are chosen for fine-tuning using some heuristics.

Such methods include, for example, Bit Fit (Zaken et al., 2021) or MeProp (Sun et al., 2017), where only top-k parameters are updated during backpropagation. The approach proposed in this paper is related to selective methods.

3 Method

Our aim is to reduce the amount of trainable parameters at the fine-tuning stage. Taking into account that

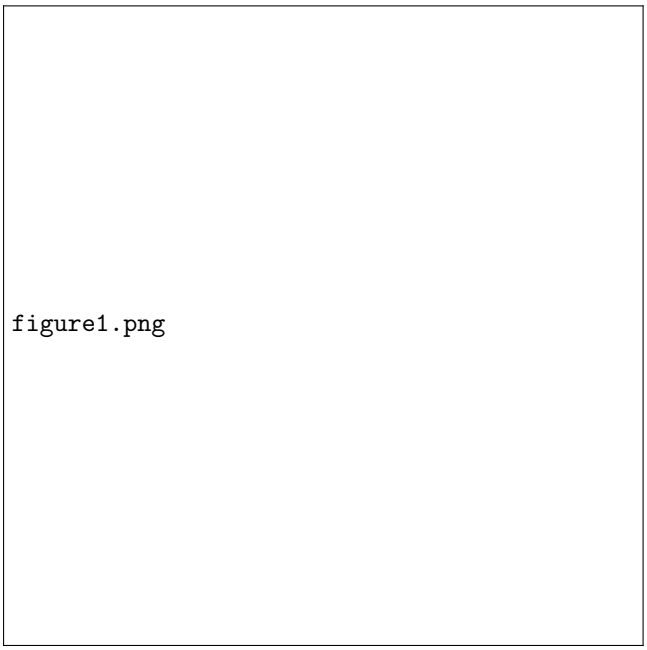

Figure 1: The first row illustrates signal propagation in the original Linear Layer, while the second row illustrates propagation with the proposed SparseGradLinear layer.

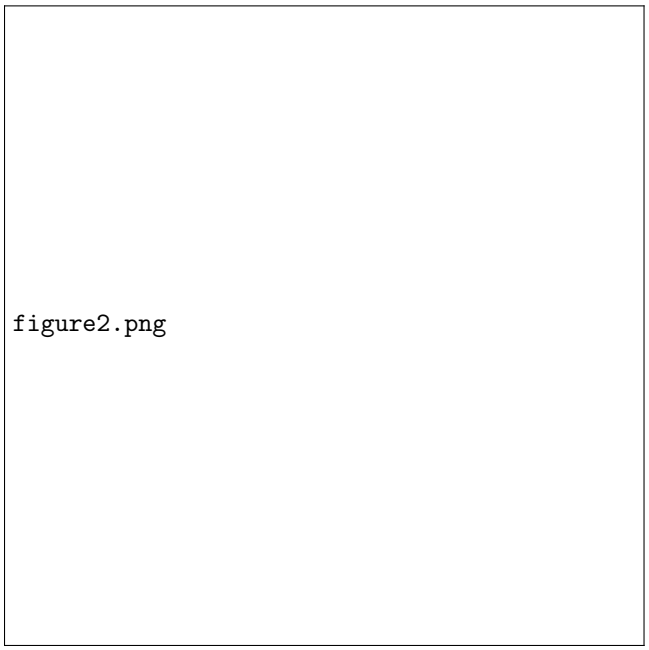

Figure 2: Gradients on the 5-th BERT MLP: $U\frac{\partial L}{\partial W^T}V^T$ (right) is more sparse than the original $\frac{\partial L}{\partial W^T}$ (left).

fine-tuning data is restricted to a limited scope, we assume there is a basis where the weight gradient matrix is very close to being sparse. To identify this basis, we applied a decomposition technique to the stacked weight gradient matrices. As a result, we introduce a new PyTorch layer class, SparseGradLinear, which transitions weights to this sparse gradient space, accumulates gradients in sparse form, and enables the reverse transition back to the original space.

3.1 Preliminary Phase: Finding Transition Matrices

To obtain transition matrices, an initial procedure is necessary. During this, we perform *n_steps* of standard backpropagation by freezing the entire model and unfreezing only the linear layers in MLP blocks. We do it to obtain the set of weights gradient matrices $\frac{\partial L}{\partial W} \in \mathcal{R}^{D\cdot\text{in}\times D\cdot\text{out}}$. Stacking these matrices over *n_blocks* - the number of all blocks in the model - and over *n_steps* , we obtain a 3D tensor of size $D\cdot\text{in}\times D\cdot\text{out}\times(n\_steps * n\_blocks)$.

Applying Higher Order SVD (HOSVD) (Cichocki et al., 2016) to this tensor yields matrices $U \in \mathcal{R}^{D\cdot\text{in}\times D\cdot\text{in}}$, corresponding to the dimension $D\cdot\text{in}$ and $V^T \in \mathcal{R}^{D\cdot\text{out}\times D\cdot\text{out}}$, corresponding to $D\cdot\text{out}$. In this way, we get two orthogonal transition matrices U, V^T which are shared across all blocks of the model. Multiplying the layer's weight matrix on the left by U and on the right by V^T transforms it into a new space. In this transformed space, the gradient matrix exhibits greater sparsity compared to the original space. Examples of $\frac{\partial L}{\partial W^T}$ with and without transition to the new space are shown in Fig. **??**.

3.2 Signal Propagation in SparseGradLinear Layer

Given a Transformer Linear layer with a weight matrix W^T , input activation X , and output $Y = XW^T$, we define the gradients of the output, input, and weights as $\frac{\partial L}{\partial Y}$, $\frac{\partial L}{\partial X}$, and $\frac{\partial L}{\partial W^T}$, respectively. To create the corresponding SparseGradLinear layer, we represent the weights in the U, V^T basis, such that the new weights are $\tilde{W}^T = UW^TV^T$. Since the modules following SparseGradLinear remain unchanged in both forward and backward passes, it is crucial to maintain consistency between outputs of the Original Linear Layer Y and the SparseGradLinear layer $\tilde{Y}$, as well as their input gradients $\frac{\partial L}{\partial X}$ and $\frac{\partial L}{\partial X}$.

Table **??** outlines these adjustments and illustrates the correspondence of variables in Torch Autograd for Linear and SparseGrad layers.

Thus, SparseGradLinear is equivalent to 3 linear layers: first with frozen weights U^T , defined by the HOSVD, second with trainable new weights $\tilde{W}^T = UW^TV^T$, third with frozen weights V , defined by the HOSVD. Fig. **??** shows the propagation of the signal in this structure.

Table 2: Correspondence of variables in Torch Autograd for a regular Linear layer and SparseGradLinear.

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

3.3 Sparse-by-Dense Matrix Multiplication

We provide the SparseGradLinear class with updated Forward and Backward procedures. However, the addition of multiplications by U, V into them increased the execution time and affected peak memory in the training loop.

The sparsity of the gradient tensor $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y}X$ results in some of the multiplications being sparse. We explore the structure of each component in this formula and figure out that $\frac{\partial L}{\partial Y}$ has a sparsity approximately equal to $\frac{\partial L}{\partial W}$. Histograms of the percent of its non-zero elements are presented in Fig. **??**. It also shows that the sparsity is "strided" - most of the rows are completely filled with zeros. These rows can be excluded from the multiplication procedure, thus optimizing it.

figure3.png

Figure 3: Strided structure of $\frac{\partial L}{\partial Y}$ (left) and visualizations of % nonzero elements in $\frac{\partial L}{\partial Y}$ throughout training (right).

More precisely, to multiply the sparse matrix $A \in \mathcal{R}^{b\times c}$ by a dense matrix $B \in \mathcal{R}^{c\times d}$ we select rows and cols - indices of rows and columns of A which contain nonzero elements and multiply as follows:

$$C = A(rows, :)(:, cols)B(cols, :). \quad (1)$$

We employ C either for further multiplications, or convert it into COO format and send it to SparseAdam optimizer. Indexes in COO format are defined by restoring indexes of A :

$$C_{coo}(rows(k), cols(l)) = C(k, l). \quad (2)$$

As it is shown in the Table **??**, such procedure significantly speeds up the harnessing of SparseGradLinear.

4 Time and Memory Consumption per Training Iteration

We measure the peak memory allocated during training using the CUDA memory allocator statistics. Table **??** demonstrates this statistic on average for all GLUE datasets for the RoBERTa base model. The comprehensive Tables **??** and **??**, which outline metrics for each dataset separately, can be found in Appendix A. Among all methods, LoRA presents the most efficient memory usage, preserving 30% of the peak memory. SparseGrad, while using slightly more memory, still achieves a 20% savings. The increase in peak memory with SparseGrad is attributed to the maintenance of matrices U and V and their multiplication by the dense objects, such as Input X .

Table 3: Training speed and memory requirements averaged on the GLUE benchmark. The last two rows of the Table report the results for the SparseGrad method with Sparse-by-Dense (SD) and Regular (Reg) matrix multiplication, respectively.

Method	Steps/Sec.	Memory, MB
Regular FT	4.1	11345
LoRA	4.7	944
SparseGradsp (SD)	4.3	1016
SparseGradreg (Reg)	0.9	1210

In terms of training time, LoRA demonstrates the fastest training, followed by SparseGrad, and then standard fine-tuning. Table **??** shows that Sparse-by-Dense multiplication saves approximately 12% memory, leading to an almost fivefold increase in speed.

5 Experiments

We conducted experiments on three transformer-based encoder models, BERT and RoBERTa base and large, on the GLUE (Wang et al., 2019) benchmark, and the

LLaMa-2 decoder model on the OpenAssistant Conversations corpus (Kopf et al., 2023). We compared the fine-tuning of the full model (Regular FT scheme) with three PEFT methods, namely LoRA, MeProp and SparseGrad, applied to MLP blocks. To harness LoRA, we use an official repository code. For the MeProp method, we kept the largest elements in the $\frac{\partial L}{\partial W}$ matrix. The proposed SparseGrad involves replacing layers in MLP blocks with its SparseGrad-Linear equivalents.

5.1 Natural Language Understanding with BERT and RoBERTa

We explore the acceptable sparsity level of the gradient matrices in the "sparse" space, $\frac{\partial L}{\partial W}$. By varying the number of remaining parameters in the Linear Layer from $100 \cdot 10^3$ to $18 \cdot 10^3$, we fine-tuned the model on the GLUE benchmark and identified the point at which performance begins to degrade. This occurs when the number of trainable parameters reaches 22×10^3 , corresponding to 1% of the total weights. Full experimental results can be found in Appendix C.

Guided by this heuristic, in our experiments we leave the top 1% of the largest elements and set the rest to zero. To deal with SparseGradients, we use the SparseAdam optimizer - the masked version of the Adam algorithm. The remaining model parameters are trained with the standard AdamW optimizer.

We fine-tune BERT, RoBERTa $_{base}$ and RoBERTa $_{large}$ (Zhuang et al., 2021) using Regular FT, LoRA, MeProp and SparseGrad schemes for 20 epochs with early stopping for each task in the GLUE. We varied the batch size and learning rate using the Optuna framework (Akiba et al., 2019). The learning rate ranged from $1e^{-6}$ to $1e^{-1}$, and the batch size is selected from the set {8, 16, 32}. Optimal training parameters for each task are available in the Appendix D. In LoRA we take the rank 10 for RoBERTa $_{large}$ and rank 7 for BERT and RoBERTa $_{base}$. For SparseGrad and MeProp we keep the same number of parameters - approximately 1% of each Linear layer.

The average scores for all GLUE tasks for BERT and RoBERTa $_{base}$ are in the Table **??**; per-task results are placed in the Appendix B. Table **??** depicts the scores for the RoBERTa $_{large}$ model. Our results indicate that SparseGrad outperforms LoRA with an equivalent number of trainable parameters across all models. For BERT, SparseGrad even exceeds the performance of Regular FT. This may be attributed to the changing basis of the weights in SparseGrad acting as a form of regularization. Concerning MeProp, it provides weaker results than SparseGrad in all cases except the RoBERTa $_{large}$ on CoLA. This could be explained by the fact that our approach first transforms the elements into a special "sparse" space, while MeProp operates on gradients in the original space. In the original space, the histogram of elements is flatter (see Fig. **??**), which suggests that, with the same cut-off threshold, MeProp may remove more significant elements compared to SparseGrad.

Table 4: Comparative results of RoBERTa $_{large}$ for 20-epoch task-specific fine-tuning.

Method	#Trainable params	Model	AVG	STSB	C
SST2					
Regular FT	355 mln		84.7	91.9±4	6
[MISSING]					
LoRA	168 mln. (0.05 mln)		83.7	92.1±3	6
[MISSING]					
SparseGrad	168 mln. (0.05 mln)		85.4	92.4±2	6
[MISSING]					
MeProp	168 mln. (0.05 mln)		84.3	92.3±1	6
[MISSING]					

Table 5: Average scores over the GLUE benchmark for BERT and RoBERTa $_{base}$ models.

Model	BERT	RoBbase
Regular FT	109 mln	82.5
LoRA	54 mln	81.6
SparseGrad	54 mln	82.6
MeProp	54 mln	82.1

5.2 Conversations with LLaMa-2

We apply the SparseGrad method to fine-tune LLaMa-2 7B (Touvron et al., 2023) model on the OpenAssistant conversational dataset (Kopf et al., 2023). Fine-tuning was performed on a single GPU NVIDIA A40 during 1 epoch with learning rate $9e^{-4}$. For Regular FT, we unfroze up_proj and down_proj layers in the MLP modules with a block index divisible by 3 (0, 3, 6, ...). We apply LoRA with rank 32 to the selected blocks, leaving the rest of the model untrainable. In the SparseGrad and MeProp methods, we also consider selected MLP modules in the transformer and leave $\approx 100,000$ 0.2% nonzero elements in the gradient matrix. For LLaMA-2, we conducted a similar ablation study as we did for BERT and RoBERTa. We varied the number of remaining parameters in the MLP block and identified the point where the model's performance began to decline.

We validate obtained models on the question set MT-Bench Inf from InflectionBenchmarks (Zheng et al., 2023). We followed the guidelines outlined in this work, called "Single Protocol" or "Single Answer Grading". We got the answers by using the FastChat platform and then evaluating them using GPT-4. GPT-4 rates the answers on a scale of 1 to 10, with the evaluation prompt taken from (Zheng et al., 2023).

The resulting losses and average GPT-4 scores are presented in Table **??**. While the models perform sim-

ilarly overall, SparseGrad slightly outperforms LoRA, MeProp, and regular fine-tuning. Examples of responses to Inflection-Benchmark samples are provided in Appendix E. These examples illustrate that, although all models produce good answers, the LoRA-trained model occasionally overlooks important nuances. In the examples given, it fails to recognize that presentations can be stressful for introverts or that hierarchy plays a significant role in Japanese corporate culture.

Table 6: Comparative results for LLaMa-2 on the OpenAssistant-1 dataset.

Method	#Train params	Valid Loss	I-Bench Score
Regular FT	22%	1.250 ±0.03	4.407
LoRA	0.5%	1.249 ±0.05	5.025
SparseGrad	0.5%	1.247 ±0.03	5.132
MeProp	0.5%	1.259 ±0.04	4.261

6 Conclusion

We propose a new selective PEFT method called SparseGrad, which identifies a space where the gradients exhibit a sparse structure and updates only its significant part. SparseGrad is validated through experiments conducted on the BERT, RoBERTa and LLaMa-2 model models, demonstrating its superiority over the additive LoRA and selective MeProp methods.

Leveraging the sparsity property significantly accelerated the calculations in SparseGrad. Our method runs faster than standard fine-tuning but slower than LoRA, while yielding better performance than LoRA; the same trend applies to memory usage. In summary, our method serves as an alternative to LoRA in situations where the performance of the final model takes precedence over the execution time. The source code as well as links to pretrained models are available at repository.2

7 Acknowledgements

The work was supported by the Analytical center under the RF Government (subsidy agreement 000000D730321P5Q0002, Grant No. 70-2021-00145 02.11.2021).

8 Limitations

The main limitation of our method is the additional memory requirements during the Preliminary Phase. The extra memory is assessed as follows: we need to unfreeze the MLP layers, which hold approximately half of the training parameters in Transformers (see Table **??**), store and decompose a large tensor. For instance, 30 steps in the preliminary phase result in a tensor of approximately 276 MB for BERT and ROBERTA models, and 5.2 GB for LLaMa-2.7B models. The decomposition part can be the most memory-consuming, as it involves reshaping a 3-dimensional tensor into a matrix with a dimension size equal to the product of two dimension sizes of the tensor (Cichocki et al., 2016).

However, this part is executed only once during the entire fine-tuning process and can be computed on the CPU in a short time. The Higher Order SVD decomposition of such objects takes approximately 78 seconds for BERT and RoBERTa base layers and about 668 seconds for LLaMa on an Intel Xeon Gold 6342 CPU processor.

9 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% of 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

A Appendix A

Table 7: The training step execution speed, measured in steps per second (where a higher value indicates faster execution), is reported for the RoBERTa base model. The last two rows describe the SparseGradMethod with Sparse-by-Dense multiplication and with Regular matrix multiplication.

Method / Dataset	AVG	STSB	CoLA	MNL
Regular FT	4.1	12.9	4.3	4.2
LoRA	4.7	2.8	5.8	6.2
SparseGrad, Sparse-by-Dense	4.3	3.8	1.8	3.9
SparseGrad, Regular	0.9	0.4	0.3	0.4

Table 8: Peak memory measurement in MB for training loop for the model RoBERTa base.

Method / Dataset	AVG	STSB	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST2
Regular FT	1345	1344	1358	1350	1362	1369	1333	1314	1339
LoRA	944	969	978	986	998	938	935	902	855
SparseGrad, Sparse-by-Dense	1016	997	1082	1017	1110	1019	981	960	980
SparseGrad, Regular	1210	1283	1212	1250	1183	1245	1172	1116	1209

Table 9: Comparative results of BERT model for 20-epoch task-specific fine-tuning.

Method	#Trainable Parameters Model	AVG	STSB	CoLA	MNLI	MRPC	QNLI	QQP	SST2
Regular FT [MISSING]	109 mln (3 mln)	82.5	89.3±4	59.0±1.9	84.0±3	86.2±1.1	89.3±1.3	91.1±0	67
LoRA [MISSING]	53 mln (0.03 mln)	81.6	89.2±4	58.4±2.3	84.2±2	83.8±4	89.3±4	91.0±0	64
SparseGrad [MISSING]	53 mln (0.03 mln)	82.6	89.2±4	58.4±0	84.6±1.3	86.6±4	89.4±1.6	90.9±3	69
MeProp [MISSING]	53 mln (0.03 mln)	82.1	88.9±4	58.4±4	83.3±3	84.2±4	89.6±3	90.4±4	6

Table 10: Comparative results of ROBERTA for 20-epoch task-specific fine-tuning.

Method	#Trainable parameters Model	AVG	STSB	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST2
Regular FT [MISSING]	125 mln. (3 mln.)	84.2	90.4±3	59.7±1.4	87.7±1	90.0±4	90.6±4	91.5±1	68.8±2.	
LoRA [MISSING]	68 mln. (0.03 mln.)	83.1	90.5±4	60.6±1.7	87.5±1	88.4±4	90.0±4	91.4±1	63.1±2.	
SparseGrad [MISSING]	68 mln. (0.03 mln.)	83.6	90.8±4	60.0±1.6	87.5±1	89.6±1	91.5±4	91.5±1	65.6±2.	
MeProp [MISSING]	68 mln. (0.03 mln.)	82.5	90.7±4	59.2±1.3	85.9±1	89.1±0	89.4±4	90.5±1	61.5±1.	

Table 11: GLUE score as a function of the weight gradient sparsity in BERT

Method	% of remained params in Linear Layers	AVG	STSB	CoLA	MNLI	MRPC	QNLI
SparseGrad 92.4±1	100	82.0	89.2±4	58.8±0	84.0±1.3	86.6±1.3	89.4±1.
SparseGrad 18k 92.4±1.6	0.8	81.9	89.1±1.5	59.1±1.5	83.8±1.1	84.6±1.8	89.4±1.
SparseGrad 22k 92.3±1.3	1	82.2	89.7±1.6	60.0±1.4	83.9±1.1	84.6±1.5	88.8±1.
SparseGrad 30k 92.2±1.4	1.2	82.0	89.2±1.4	59.1±1.5	84.1±1.3	85.4±1.4	89.3±1.
SparseGrad 100k 92.4±1.3	4.2	82.2	89.3±1.3	60.0±1.3	83.8±1.2	85.1±1.2	88.9±1.

Table 12: GLUE score as a function of the weight gradient sparsity in ROBERTA

Method	% of remained params in Linear Layers	A
SparseGrad 94.2±1.1	100	
SparseGrad 18k 94.0±1.4	0.8	
SparseGrad 22k 94.6±1.2	1	
SparseGrad 30k 94.2±1.1	1.2	
SparseGrad 100k 94.1±1.3	1.4	

B Appendix B

C Appendix C

D Appendix D

Best training parameters for all models. In all experiments, we repeat fine-tuning 3 times over different seeds and report the average score.

Table 13: Best training parameters on GLUE benchmark for BERT model.

Dataset	batch size, learning rate
STSB	32, 1.24e-4
CoLA	32, 3.15e-5
MNLI	32, 6.07e-6
MRPC	32, 1.22e-5
QNLI	16, 1.94e-5
QQP	32, 1.41e-5
RTE	16, 6.81e-5
SST2	32, 1.47e-5

E Appendix E

Responses from the models to an example from Inflection-Benchmarks are shown. While all models perform fairly well, the LoRA-trained model overlooks the fact that public speaking can be stressful for an introvert when answering the first question.

[Question 1]

Write a persuasive email to convince your introverted friend, who dislikes public speaking, to volunteer as a guest speaker at a local event. Use compelling argu-

Table 14: Best training parameters on GLUE benchmark for RoBERTa model.

Dataset	batch size, learning rate
STSB	16, 2.70e-5
CoLA	16, 1.01e-5
MNLI	32, 1.51e-5
MRPC	32, 1.9e-5
QNLI	16, 1.91e-5
QQP	16, 5.11e-6
RTE	32, 3.05e-5
SST2	16, 1.33e-5

Table 15: Best training parameters on GLUE benchmark for RoBERTa-large model.

Dataset	batch size, learning rate
STSB	32, 7.71e-5
CoLA	16, 1.8e-5
MNLI	16, 1.15e-6
MRPC	32, 2.47e-5
QNLI	16, 8.83e-6
QQP	32, 7.2e-6
RTE	32, 1.02e-5
SST2	32, 1.02e-5

ments and address potential objections. Please be concise.

LLaMa2-7B Regular FT:
Here's a persuasive email to convince your introverted friend to volunteer as a guest speaker at a local event: Subject: Join us as a guest speaker at our local event Dear (Friend's Name), I hope this email finds you well. I hope you're doing well! I'm writing to invite you to be a guest speaker at our upcoming local event. As a friend, I know how passionate you are!

LLaMa2-7B LoRA:
Subject Line: Please, Volunteer as a Guest Speaker! Dear John, I hope you are doing well. I am writing to you because I know you are an introverted person and do not like public speaking. However, I have a great opportunity for you to volunteer as a guest speaker at the event!

LLaMa2-7B Sparse Grad:
Subject: Volunteer as a Guest Speaker at a Local Event

Subject: Volunteer as a Guest Speaker at a Local EventDear [Name], I hope this email finds you well. I wanted to reach out to you about an exciting opportunity to volunteer as a guest speaker at a local event. As you know, public speaking can be a nerve-wracking experience, but it's also a great way to share your knowledge and expertise with others.

[Question 2]

What are some business etiquette norms when doing business in Japan?
LLaMa2-7B Regular FT:
If you're doing business in Japan, it's important to follow their business etiquette norms. Here are some key things to keep in mind:

1. Hierarchy: In Japan, it's important to respect the hierarchy in the workplace. Address your Japanese colleagues with the appropriate title, and don't speak to them unless they initiate the conversation.

2. Punctuality: Being punctual is considered very important in Japanese culture, so make sure to arrive on time for meetings and events.

3. Business Cards: Business cards are an important part of doing business in Japan. Make sure to have your own business card made beforehand and carry it with you at all times.

LLaMa2-7B LoRA:
Of course! Here are some business etiquette norms when doing business in Japan:

1. Business Cards: Business cards are a vital part of business interactions in Japan. Always have a few business cards with you and exchange them with others during introductions.

2. Punctuality: Time is sacred in Japan, and being punctual is essential for business meetings. Arrive on time or even a few minutes early.

3. Gifts: Gifts are common in business interactions in Japan, especially for special occasions. Small personalized gifts such as chocolates or flowers are appropriate.

LLaMa2-7B Sparse Grad:
Here are some business etiquette norms to keep in mind when doing business in Japan:

1. Respect for seniority: In Japanese culture, age and seniority are highly respected, so it is important to show respect to those who are older or in a higher position than you.

2. Business cards: Business cards are an important part of doing business in Japanese culture, and it is customary to exchange business cards at the beginning of a meeting.

3. Gifts: Gifts are often given as a sign of appreciation or gratitude in Japanese culture. It is important to give gifts that are appropriate for the occasion and not too extravagant.