PRILoRA: Pruned and Rank-Increasing Low-Rank Adaptation

Nadav Benedek
Tel Aviv University
nadavbenedek@mail.tau.ac.il

Lior Wolf
Tel Aviv University
wolf@cs.tau.ac.il

Abstract

With the proliferation of large pre-trained language models (PLMs), fine-tuning all model parameters becomes increasingly inefficient, particularly when dealing with numerous downstream tasks that entail substantial training and storage costs. Several approaches aimed at achieving parameter-efficient fine-tuning (PEFT) have been proposed. Among them, Low-Rank Adaptation (LoRA) stands out as an archetypal method, incorporating trainable rank decomposition matrices into each target module. Nevertheless, LoRA does not consider the varying importance of each layer. To address these challenges, we introduce PRILoRA, which linearly allocates a different rank for each layer, in an increasing manner, and performs pruning throughout the training process, considering both the temporary magnitude of weights and the accumulated statistics of the input to any given layer. We validate the effectiveness of PRILoRA through extensive experiments on eight GLUE benchmarks, setting a new state of the art.

1 Introduction

The current paradigm for natural language processing tasks is to exploit pre-trained models, which were trained using large amounts of data and expensive resources, and fine-tune them to various downstream tasks (Brown et al., 2020; Liu et al., 2019; Radford et al., 2019; He et al., 2021b; Devlin et al., 2019). Such fine-tuning was traditionally conducted by gradient update of all parameters of the model (Dodge et al., 2020; Raffel et al., 2020; Qiu et al., 2020). With the ever increasing size of models, such as Llama 7B-65B (Touvron et al., 2023), Palm 540B (Chowdhery et al., 2022), and others, trained with resources consisting of hundreds of GPUs in parallel, which are available only to some institutions and corporations, full fine-tuning can become prohibitive, lengthy, and with high carbon footprint (Luccioni et al., 2022). Additionally, fully fine-tuning this way requires storing all parameters of the fine-tuned model for every downstream task.

To tackle the aforementioned challenges, a few research directions for Parameter-Efficient Fine-Tuning (PEFT) were proposed. These directions aim to maintain or even improve the accuracy of a full fine-tuning approach, while training only a small fraction of the parameters. One approach is to add small modules to the base model, which is kept frozen throughout the training process. Such adapter tuning techniques (Rebuffi et al., 2017; Houlsby et al., 2019; Pfeiffer et al., 2020; He et al., 2022) add modules between the layers. The implication, due to increased model depth, is longer training time and higher latency during inference. Alternatively, prompt and prefix tuning (Lester et al., 2021; Li and Liang, 2021) attach trainable tokens to the beginning of layers in the model, thus potentially reducing its effective maximal token length.

LoRA (Hu et al., 2022) fine-tunes linear layers by viewing each layer as a matrix of weights W_0, freezing it, and adding to it a small rank matrix, with the same shape as the original weight matrix, that is obtained as a product of two low-rank matrices A and B. The low-rank r is chosen to be much smaller than the input dimension to the layer, thereby significantly reducing the number of trainable parameters. During LoRA training, only the two low-rank matrices are updated, which are usually 0.01% to 1.00% of the original parameter count, depending on the low-rank of the two matrices. In addition to being efficient and often exceeding the performance of full fine-tuning (Hu et al., 2022), this method has the advantage of being able to be merged back to the original matrix during inference, without increasing latency. LoRA has been used in various downstream tasks successfully (Schwartz et al., 2022; Lawton et al., 2023;

Findings of the Association for Computational Linguistics: EACL 2024, pages 252–263

Dettmers et al., 2023)

One limitation of LoRA is that the low-rank r is an arbitrarily set parameter, and in the original LoRA it is set to be fixed across layers and weights.

Efforts were made to address the issue of the fixed rank of LoRA. AdaLoRA (Zhang et al., 2023) starts from an initial parameter budget, which is slightly higher than the final budget, and then gradually reduces it until matching the target by removing weights based on SVD.

In this work, we encourage the usage of linearly increasing the rank from one layer to the next while concurrently adhering to the same budget of parameters. As we show, this strategy provides a distribution of the learned parameters that is better than a uniform placement, or even the learned alternatives.

A second contribution is obtained by pruning matrix A. This is done by considering both the elements of A and an exponential moving average over the layer's input. Although we prune, in most cases, half of the elements of A, the main metric we seek to improve by pruning is the overall accuracy obtained after pruning.

We conduct extensive experiments over eight different General Language Understanding Evaluation (Wang et al., 2019) benchmarks, and present evidence that the proposed method outperforms LoRA and its recent variants, that both the linear distribution of ranks and the specific pruning approach are beneficial, and that the method does not require more GPU memory or training time than the conventional LoRA, unlike recent extensions of LoRA.

2 Related Work

In recent years, Parameter Efficient Fine-Tuning (PEFT) has garnered increasing interest among researchers as a means to reduce both the expenses associated with fine-tuning and storing large-scale pre-trained models and the time required for training. Various approaches have emerged, each exhibiting distinct characteristics pertaining to memory utilization, storage requirements, and computational overhead during inference. These approaches can be classified into two primary categories, namely, selective and additive PEFT methods, based on whether the original model parameters undergo fine-tuning during the training phase.

Selective methods involve the selection and modification of a model based on its original parameters. An early instance of this concept was observed in the fine-tuning of only a subset of the top layers of a network, as demonstrated by Donahue et al. (2014), and by more recent work (Gheini et al., 2021). In more recent developments, various approaches have been proposed, each targeting specific layers or internal modules of the model. For instance, the BitFit method (Zaken et al., 2021) updates only the bias parameters, resulting in a substantial reduction in the number of trainable parameters, but at the cost of suboptimal performance. Other methods use a scoring function when selecting trainable parameters (Guo et al., 2020; Sung et al., 2021; Vucetic et al., 2022), while others select top parameters based on a Fisher information calculation (Sung et al., 2021).

Additive methods represent an alternative to full-parameter fine-tuning by introducing additional trainable parameters into the backbone network. Adapters are a type of trainable component initially applied in the context of multi-domain image categorization by Rebuffi et al. (2017), that were subsequently integrated into Transformer networks, specifically in the attention and feed-forward layers (Houlsby et al., 2019). Prefix-Tuning and Prompt-Tuning (Li and Liang, 2021; Lester et al., 2021) involve the addition of trainable parameters preceding the sequence of hidden states across all layers. LST (Ladder Side-Tuning) (Sung et al., 2022) operates by short-cutting hidden states from the original network into a compact trainable side network, eliminating the need for backpropagating gradients through the backbone network.

LoRA (Hu et al., 2022) emulates the adjustment of the weight matrix in the model through the multiplication of two low-rank matrices. Notably, the trained parameters resulting from this process can be incorporated seamlessly into the original network during the inference phase without incurring additional computational overhead.

Recently, hybrid approaches have emerged, combining the selective and additive methods and presenting a unified framework (Chen et al., 2023; He et al., 2022; Mao et al., 2021). Other methods are based on the hypothesis that parameter redundancy exists in PEFT modules, therefore pruning the trainable parameters to achieve superior fine-tuning performance (Bai et al., 2022).

Network pruning methods (Molchanov et al., 2016; Hassibi et al., 1993; Frankle and Carbin, 2019; Liu et al., 2018; Han et al., 2015b) reduce the size of the network by removing or shrinking

matrices from the network, which effectively is equivalent to setting them to zero. Such methods require further full re-training, or other computationally intensive iterations.

Magnitude Pruning (Han et al., 2015a; Gale et al., 2019) removes individual parameter weights when the magnitude is below a certain threshold. The threshold is determined either based on the relative magnitude to other weights in the same parameter or layer (Zhu and Gupta, 2018), or for the whole network (Liu et al., 2018).

3 Background

Transformer Models. Transformer (Vaswani et al., 2017) is a sequence-to-sequence architecture that makes use of self-attention. Typically, it consists of several stacked blocks, where each block contains two sub-modules: a multi-head attention (MultiHead) and a fully connected feed-forward network (FFN). Given the input sequence $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ of n tokens of dimension d, MultiHead performs the attention function using h heads, allowing each segment of the d space to attend to a different value projection of another token:

$$\text{MultiHead}\,(\boldsymbol{X}) = [\text{head}_1, .., \text{head}_h]\boldsymbol{W}_o \in \mathbb{R}^{n \times d}$$

$$\text{head}_i = \text{Softmax}\left(\frac{\boldsymbol{X}\boldsymbol{W}_{q_i}(\boldsymbol{X}\boldsymbol{W}_{k_i})^\top}{\sqrt{d_h}}\right)(\boldsymbol{X}\boldsymbol{W}_{v_i})$$

where the square brackets denote a concatenation along the second dimension, $\boldsymbol{W}_o \in \mathbb{R}^{d \times d}$ and $\boldsymbol{W}_{q_i}, \boldsymbol{W}_{k_i}, \boldsymbol{W}_{v_i} \in \mathbb{R}^{d \times d_h}$ are parameters of head i, per block, and the softmax is applied to each row. d_h is typically set to $\frac{d}{h}$. The output of the MultiHead is fed into the FFN, consisting of two linear transformations with a ReLU non-linearity in between:

$\text{FFN}(X) = \text{ReLU}(\boldsymbol{X}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$, where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$ are parameters of the block. Lastly, a residual connection is applied and a layer normalization (Ba et al., 2016).

Adapters. (Houlsby et al., 2019; Pfeiffer et al., 2020) The adapter technique injects a module between the transformer layers, such that the input is down-projected to a lower-dimensional space using $\boldsymbol{W}_{down} \in \mathbb{R}^{d \times r}$, followed by non-linearity σ, and up-projected using $\boldsymbol{W}_{up} \in \mathbb{R}^{r \times d}$, combined with a residual connection:

$$\boldsymbol{h} = \boldsymbol{x} + \sigma(\boldsymbol{x}\boldsymbol{W}_{down})\boldsymbol{W}_{up} \quad (1)$$

Low Rank Adaptation. LoRA (Hu et al., 2022) freezes the pre-trained model weights and injects two trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for fine-tuning tasks. For a linear layer $\boldsymbol{h} = W_0\boldsymbol{x}$, the LoRA-modified forward function is:

$$\boldsymbol{h} = \boldsymbol{W}_0\boldsymbol{x} + \Delta\boldsymbol{W}\boldsymbol{x} = \boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{B}\boldsymbol{A}\boldsymbol{x} \quad (2)$$

where $\boldsymbol{W}_0, \Delta\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$, $\boldsymbol{A} \in \mathbb{R}^{r \times d_2}$ and $\boldsymbol{B} \in \mathbb{R}^{d_1 \times r}$ with $r \ll \{d_1, d_2\}$. $\boldsymbol{A}$ is Gaussian initialized and $\boldsymbol{B}$ is zero initialized, in order to have $\Delta\boldsymbol{W} = 0$ at the beginning of the fine-tuning training. Hu et al. (2022) apply LoRA to the query and value parameters (i.e, $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$) in the multi-head attention, without modifying the other weights. He et al. (2022) extend it to other weight matrices of the feed-forward network, for an increased performance.

4 Method

Our proposed method, PRILoRA (Pruned and Rank-Increasing Low-Rank Adaptation), is comprised of two main components that integrate with the LoRA fine-tuning: (i) Linear distribution of low ranks across the layers in the network, and (ii) Ongoing pruning of the $\boldsymbol{A}$ matrix of the LoRA, based on the layer's input activations and the weights of the LoRA $\boldsymbol{A}$ matrix.

4.1 Linear Distribution of Ranks

While LoRA distributes the learned parameters uniformly, one can distribute these differently. For example, one can assign a lower rank to some of the layers and a higher rank to others.

Recall that the trainable parameters in LoRA are the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. Each has one dimension that is fixed according to the layer's structure, and one dimension that is the low rank r. Since both the time complexity (train or test) and the memory complexity of a layer are linear in both the input and the output dimensions of each layer, and since only one dimension of $\boldsymbol{A}$ and $\boldsymbol{B}$ depends on r, the overall complexity of LoRA is linearly dependent on the sum of the ranks in all modified layers.

The way that we distribute the learned parameters is motivated by the results provided by (Zhang et al., 2023), which demonstrate that the top layers require more adaptation. Considering that one cannot focus only on the top layers, since the other

layers also need to adapt (see Sec. 6), and to promote simplicity, we employ a linear distribution of ranks.

In the linear distribution of ranks, we allocate a different low-rank for every layer in the model, in a linearly increasing manner. Specifically, for the DeBERTaV3-base model, we start from the first layer, applying a low-rank of $r_s = 4$, and growing linearly, up to the twelfth layer, where we apply $r_f = 12$, such that the average rank across layers is 8. We allocate the same low-rank to all weights in a given layer, regardless of the matrix type (query, key, value, etc.). This makes the total number of parameters identical to the LoRA method.

4.2 Ongoing Importance-Based A-weight Pruning

We employ pruning as a form of dynamic feature selection, which allows the fine-tuning process to focus on some of the layer's input at each bottleneck index at every pruning iteration. The intuition is that since the capacity of the update matrix $\boldsymbol{BA}$ is low, it would be beneficial to attend only to the important input dimensions.

4.2.1 Importance Matrix

Each transformer layer, whether it is a projection associated with key, query, or value, or one of the FFN layers has some weight matrix $\boldsymbol{W}$. It also has some input $\mathbf{X} \in \mathbb{R}^{b \times n \times d}$, where b is the batch size, n is the number of tokens, and d is the dimension. We abuse the notation slightly and also write $\mathbf{X}$ for the second layer of the FFN, although, in this case, the dimension is d_m, which is typically larger than d. In our framework we maintain, throughout the training process, an Exponential Moving Average of the $L2$ norm of the rows of each such input $\mathbf{X}$, as depicted in Figure 1.

For each batch, we consider the tensor that has a dimension of $b \times n \times d$, square all elements, sum across the first and second dimensions, obtaining a vector of size d, and take the square root of each vector element, to get $\boldsymbol{x}$.

The exponential moving average $\bar{\boldsymbol{x}}$ is updated between batches by the following rule

$$\bar{\boldsymbol{x}} = 0.9\bar{\boldsymbol{x}} + 0.1\boldsymbol{x} \tag{3}$$

We next compute, for every weight matrix $\boldsymbol{W}$, or, more specifically, for $\boldsymbol{A} \in \mathbb{R}^{r \times d_2}$, which is the associated half-decomposition of $\Delta\boldsymbol{W}$, an importance matrix $\boldsymbol{S}$ of the same size as $\boldsymbol{A}$. $\boldsymbol{S}$ is inspired by Wanda (Sun et al., 2023), and is the element-wise multiplication of the absolute value of $\boldsymbol{A}$ with the relevant moving average vector $\bar{\boldsymbol{x}}$ (recall that there is one $\bar{\boldsymbol{x}}$ to each weight matrix $\boldsymbol{W}$):

$$\boldsymbol{S}_{ij} = |\boldsymbol{A}_{ij}|\bar{\boldsymbol{x}}_j \tag{4}$$

Note that all values of $\bar{\boldsymbol{x}}$ are positive, since they represent a mean norm. Therefore, all elements of $\boldsymbol{S}$ are positive, too.

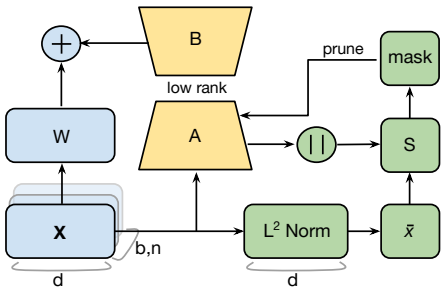

Figure 1: The schematics of PRILoRA on a single layer. The blue path demonstrates a frozen linear layer. We omitted the bias for simplicity. The yellow path depicts LoRA; dropout and scaling were omitted for simplicity. In the green path of PRILoRA, the input tensor $\mathbf{X}$ of the layer is fed into $L2$ norm calculation. Then, the exponential moving average vector $\bar{\boldsymbol{x}}$ is updated and kept as a state of the layer. When it is time for pruning, the absolute value of the elements of $\boldsymbol{A}$ is calculated, and together with $\bar{\boldsymbol{x}}$, the importance matrix $\boldsymbol{S}$ is computed. In every row of $\boldsymbol{S}$, the lowest elements, as defined by the *prune ratio*, are being selected to form the mask. The mask is used to zero out elements in the $\boldsymbol{A}$ matrix.

4.2.2 Pruning

Every 40 steps in the training process, we prune each of the $\boldsymbol{A}$-matrices, in accordance with the associated importance matrix $\boldsymbol{S}$. To do so, we consider the n lowest elements of every row $i = 1 \ldots r$ of $\boldsymbol{S}$ and create a binary mask $\boldsymbol{M} \in \mathbb{R}^{r \times d_2}$. Each mask element $\boldsymbol{M}_{ij}$ indicates whether $\boldsymbol{S}_{ij}$ is among the n lowest values of row i of $\boldsymbol{S}$. n is determined by the *prune ratio*; a higher ratio means more weights are being zeroed out. We then zero out the elements in $\boldsymbol{A}$ using the mask $\boldsymbol{M}$.

Note that zeroing out an element of $\boldsymbol{A}$ does not prevent this element from becoming non-zero immediately in the next training step. However, pruning this way changes the training dynamics and encourages $\boldsymbol{A}$ to be sparse. Figure 2 shows five random weights during training of different datasets. It can be seen that some weights can survive pruning, some weights remain in the pruning region since

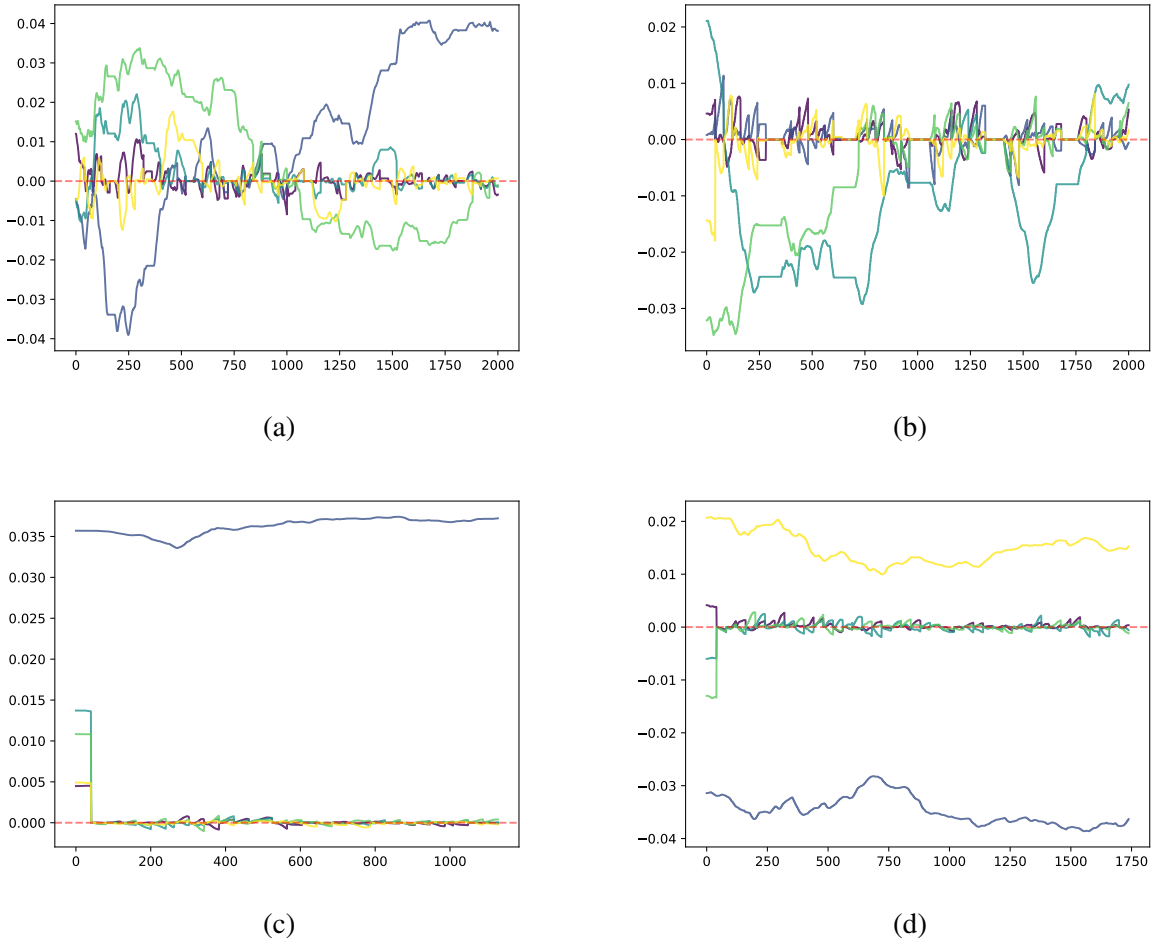

Figure 2: Five weights values over time on four different GLUE tasks: (a) RTE task, in layer 5, value_proj parameter; (b) MRPC task, in layer 6, query_proj parameter; (c) SST-2 task, in layer 7, key_proj; (d) CoLA task, in layer 8, attention.output parameter.

they cannot escape fast enough, and some weights avoid being pruned completely.

5 Experiments

We apply PRILoRA to DeBERTaV3-base (He et al., 2021a) (184 million parameters), and evaluate the method on eight natural language understanding benchmarks included in the General Language Understanding Evaluation - GLUE (Wang et al., 2019). Summary of the GLUE benchmarks can be found in Table 6. We use PyTorch (Paszke et al., 2019) and Hugging Face Transformers (Wolf et al., 2019) to implement the algorithms. All the experiments are conducted on NVIDIA GeForce RTX 2080 Ti GPUs. Due to limited GPU memory size, we leave similar analysis of large-scale models, such as T5-3B, Llama, and others, to future research.

5.1 Baselines

Full fine-tuning: In the fine-tuning stage, the model is initialized with the pre-trained parameters, and all model parameters go through gradient updates.

Bitfit: (Zaken et al., 2021) A sparse fine-tuning method where only the bias-terms of the model (or a subset of them) are being modified.

HAdapter: (Houlsby et al., 2019) Inserts adapter layers between the self-attention module, the FFN module, and the subsequent residual connection. There are two fully connected layers with biases in an adapter layer with a non-linearity in between.

PAdapter: (Pfeiffer et al., 2020) Inserts the adapter after the FNN module and LayerNorm.

LoRA: (Hu et al., 2022) Adds trainable pairs of rank decomposition matrices in parallel to existing weight matrices. The number of trainable param-

eters is determined by the rank r and the shape of the original parameters.

AdaLoRA: (Zhang et al., 2023) Parameterizes the incremental updates in the form of singular value decomposition, for a given parameter.

5.2 Implementation details

In our research, we experimented with different distributions while keeping the total number of parameters invariant and found that the configuration $\{r_s = 4, r_f = 12\}$ was optimal, together with the hyper-parameters which are specified in Table 7. The fact that higher layers require more parameters for LoRA fine-tuning may indicate that higher layers in Transfomer-based models capture deeper levels of understanding, and therefore when fine-tuning a pre-trained language model, more focus must be put on deeper layers than on lower layers that require less modification or adaptation to the downstream task in question.

5.3 Main results

We compare PRILoRA with the baseline methods. Table 1 shows our results on the GLUE development set (Appendix A). PRILoRA achieves best average score, best result in six out of the eight datasets, and in all datasets better results than HAdapter, PAdapter and LoRA, with approximately the same number of parameters.

Note that when counting the number of parameters, we do not discount for pruned parameters. However, with a pruning ratio of 0.5 in most benchmarks, a quarter of the learned parameters (half the parameters of the A matrices) are zero. A more precise count of parameters would, therefore, be closer to one million parameters and not 1.33M.

5.3.1 Ablation Study

In table 2 we present an ablation study for PRILoRA, on four GLUE tasks: SST-2, CoLA, RTE and MRPC. We aim to analyze both the rank distribution across layers and the pruning method.

For the rank distribution study we: (i) remove the linear distribution component of our method, retaining the pruning component alone with identical rank at each layer; (ii) replace the 4→12 distribution by 12→4; (iii) attach LoRA adapter to only the last layer, with a higher rank of 24 (Concentrated Distribution).

For the pruning method study we: (i) remove the importance pruning component, retaining increasing rank distribution $4 \rightarrow 12$; (ii) prune the rows of $\boldsymbol{B}$ matrix instead of $\boldsymbol{A}$, by collecting an exponential moving average of $\boldsymbol{B}$ input norm, instead of the input to $\boldsymbol{A}$ (or the layer); (iii) similarly, prune $\boldsymbol{B}$ columns instead of rows; (iv) prune the columns of $\boldsymbol{A}$ randomly, instead of PRILoRA method, but with the same *prune ratio*. During all ablation tests, per benchmark, we keep the same hyper-parameters and change only a single component. For all cells in the table, the same single seed is used.

Rank Distribution As can be seen, removing the linear distribution of the low-rank and fixing a constant rank across all layers, such that the total number of parameters stays the same as in LoRA, but applying pruning, reduces the results in all tests. Removing the linear distribution nonetheless outperforms LoRA results, signalling that pruning is indeed an essential component of the method. For example, PRILoRA with no linear distribution on the SST-2 benchmark reaches 96.10, while LoRA is 94.95, and on CoLA it is 72.17 versus 69.82.

Interestingly, changing the order of the rank allocation, to be 12→4, reduces the performance significantly; for example, a decrease of $73.08 \rightarrow 69.73$ on the CoLA benchmark, and $93.14 \rightarrow 91.91$ on the MRPC benchmark. Inverting the rank allocation order diminishes performance below fixed-rank allocation across layers. This provides additional support in the need to allocate more parameters to the top layers.

Lastly, attaching LoRA only to the last layer yields the lowest average results across the rank distribution ablation study, for example 89.95 versus 93.14 on MRPC when the full method is used.

Pruning Method Ablating pruning completely, reduces the performance. For instance, on CoLA it is reduced $73.08 \rightarrow 71.31$. This is higher than LoRA (69.82), pointing to the positive effect of the rank-increasing distribution. When we prune matrix $\boldsymbol{B}$ instead of $\boldsymbol{A}$, we obtain results similar to no pruning at all, suggesting that pruning $\boldsymbol{B}$ did not yield any discernible benefits.

A plausible argument is that the input activation shape of $\boldsymbol{A}$ and $\boldsymbol{B}$ is very different, for example 768 versus 8, in the case of most weights in DeBERTaV3-base model, and a low-rank of 8. Choosing to row-prune matrix $\boldsymbol{B}$ with a *prune ratio* of 0.5, essentially means eliminating 4 out of 8 cells in every $\boldsymbol{B}$ row, which can be too aggressive. Additionally, doing the same process on $\boldsymbol{B}$ columns can create situations where a complete row of $\boldsymbol{B}$ is zeroed out, which means that the cor-

Table 1: Results with DeBERTaV3-base on GLUE development set. The best results on each dataset are shown in **bold**. We report the average correlation for STS-B (Pearson, Spearman). We report matched accuracy for MNLI. *Full FT*, *HAdapter* and *PAdapter* represent full fine-tuning, Houlsby adapter, and Pfeiffer adapter, respectively. We report the mean and standard deviation of three runs using different random seeds. We report the baseline results from Zhang et al. (2023). Higher is better for all metrics.

Method	**#Param**	**MNLI** Acc	**SST-2** Acc	**CoLA** Mcc	**QQP** Acc	**QNLI** Acc	**RTE** Acc	**MRPC** Acc	**STS-B** Corr	**All** Avg.
Full FT	184M	89.90	95.63	69.19	92.40	94.03	83.75	89.46	91.60	88.25
BitFit	0.1M	89.37	94.84	66.96	88.41	92.24	78.70	87.75	91.35	86.20
HAdapter	1.22M	90.13	95.53	68.64	91.91	94.11	84.48	89.95	91.48	88.28
PAdapter	1.18M	90.33	95.61	68.77	92.04	94.29	85.20	89.46	91.54	88.41
LoRA$_{r=8}$	1.33M	90.65	94.95	69.82	91.99	93.87	85.20	89.95	91.60	88.50
AdaLoRA	1.27M	**90.76**	96.10	71.45	92.23	**94.55**	88.09	90.69	91.84	89.46
PRILoRA	1.33M	90.75	**96.21**	**72.79**	**92.45**	94.44	**89.05**	**92.49**	**91.92**	**90.01**
[PRILoRA SD]		±0.03	±0.30	±1.28	±0.05	±0.14	±1.04	±0.57	±0.14	±0.44

Table 2: Ablation study results on the same single seed.

	SST-2	CoLA	RTE	MRPC
PRILoRA	**96.44**	**73.08**	**90.25**	**93.14**
Fixed distribution	96.10	72.17	88.81	92.16
Inverted distribution	95.99	69.73	88.09	91.91
Concentrated dist.	95.07	69.92	87.73	89.95
No pruning	96.22	71.31	89.89	92.09
Prune B rows	96.10	71.41	89.89	91.67
Prune B cols.	96.22	71.46	88.81	91.42
Prune A rand cols.	94.84	70.75	88.09	89.22

responding output cell of LoRA will be zero as well. Furthermore, the compressed low-rank latent input to matrix $\boldsymbol{B}$ already encapsulates the essential information, so pruning it deteriorates the performance.

Finally, performing a random pruning of columns in $\boldsymbol{A}$ with the same *prune ratio*, produces the lowest results in the Pruning Method ablation study.

5.3.2 Pruning Ratio Study for PRILoRA

We would like to learn how aggressive pruning should be, that is, how much sparsity should be injected into the LoRA weights in order to reach peak performance. We chose four GLUE tasks, and for each task and for each *prune ratio* in {0.25, 0.50, 0.75} we ran the fine-tuning three times, each time with a different seed. We report the average result and standard deviation across the different seeds.

Table 3 shows that for the selected tasks, the optimal pruning ratio is 0.5. However, specifically for the STS-B task, a random hyper-parameter search yielded an optimal pruning ratio of 0.75, as can be seen in Table 7.

5.3.3 Training Cost Study for PRILoRA

We present the training cost comparison between PRILoRA and LoRA, using the DeBERTaV3-base model, on NVIDIA GeForce RTX 2080 Ti GPUs. For the two methods, the batch size is 32.

Table 4 shows that PRILoRA has zero increase in number of trainable parameters in comparison to LoRA, and a negligible increase in training time per epoch.

For comparison, AdaLoRA (Zhang et al., 2023) speed per batch is 11% slower than LoRA in the MNLI benchmark and 16% slower in the SST-2 benchmark, and with a slightly larger memory footprint.

However, analyzing the training time per batch does not suffice. Once we know that the training step time in PRILoRA is similar to LoRA, we want to delve deeper and analyze the number of steps required until reaching peak performance on the evaluation metric.

Table 5 presents the number of steps required

Table 3: Performance vs Pruning Ratio. Each cell in the table shows the average across three different seeds, together with the standard deviation.

	SST-2	CoLA	RTE	MRPC
Prune 0.25	96.10 ± 0.34	71.43 ± 0.30	87.73 ± 1.25	91.34 ± 0.99
Prune 0.50	$\mathbf{96.21 \pm 0.30}$	$\mathbf{72.79 \pm 1.28}$	$\mathbf{89.05 \pm 1.04}$	$\mathbf{92.49 \pm 0.57}$
Prune 0.75	95.95 ± 0.17	70.63 ± 1.56	87.73 ± 0.73	90.85 ± 0.51

Table 4: Comparison of memory consumption and time per epoch in training, between PRILoRA and LoRA on NVIDIA GeForce RTX 2080 Ti GPU, with a batch size of 32. All models have 1.33M parameters.

Dataset	Method	GPU Mem	Time/epoch
MNLI	LoRA	9.559 GB	117 min
	PRILoRA	9.559 GB	120 min
SST-2	LoRA	9.559 GB	24 min
	PRILoRA	9.559 GB	23 min
QQP	LoRA	9.559 GB	109 min
	PRILoRA	9.559 GB	110 min

Table 5: Number of steps to evaluation peak point, on four selected GLUE tasks.

	SST-2	CoLA	RTE	MRPC
PRILoRA	9875	12375	1875	1750
LoRA	6500	8000	3250	1250

for each method until reaching its peak evaluation performance. Evidently, there is no clear winner with respect to the number of steps or time required to reach peak performance. Both LoRA and PRILoRA have the same order of magnitude. Since one often trains beyond the peak point, the table does not indicate that one method is preferable to the other in this respect.

6 Discussion

Moving from one task to another requires an adaptation of both the input and the output domain. While the input domain of large language models may be comprehensive enough to support new downstream tasks, the generation of the output is very much context-and-task-dependent.

Therefore, it should not come as a surprise that fine-tuning requires more adaptation of the top layers, which are closer to the output, than of the earlier, input-processing, layers.

However, if one is to change only the top layers, as we showed in the ablation study, there would not be enough co-adaptation of the earlier layers to enable the top layers to produce the required output. It seems, therefore, that the gradual increase in the allocated resources, which we apply, is a reasonable strategy.

7 Conclusions

In this paper, we introduced PRILoRA, a novel, yet simple and parameter-efficient method for improving low-rank adaptation during fine-tuning. Our extensive experiments encompass eight GLUE benchmarks across multiple seeds, illustrating the effectiveness of PRILoRA. Notably, we achieve superior performance compared to state-of-the-art metrics while maintaining the same number of trainable parameters, reducing the non-zero parameters by a quarter on most benchmarks, and adhering to the same memory constraints and running time per epoch.

8 Limitations

Our work has some limitations. We pushed the limits of our computational resources, utilizing NVIDIA GeForce RTX 2080 Ti GPUs, to conduct the experiments presented in this study across the eight GLUE benchmarks. We employed the PRILoRA-modified DeBERTaV3-base model, which consists of 184 million parameters.

These experiments are of the same scale as the most related work (Zhang et al., 2023). However, the full potential of the method could be realized on larger models trained on more extensive datasets, and by using larger batches that can fit into GPU memory, allowing examination of the method on additional downstream tasks, such as question answering and text summarization.

Acknowledgments

This work was supported by a grant from the Tel Aviv University Center for AI and Data Science (TAD).

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Yue Bai, Huan Wang, Xu Ma, Yitian Zhang, Zhiqiang Tao, and Yun Fu. 2022. Parameter-efficient masking networks. *Advances in Neural Information Processing Systems*, 35:10217–10229.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. 2023. Parameter-efficient fine-tuning design spaces. *arXiv preprint arXiv:2301.01821*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

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*.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*.

Mozhdeh Gheini, Xiang Ren, and Jonathan May. 2021. Cross-attention is all you need: Adapting pretrained transformers for machine translation. *arXiv preprint arXiv:2104.08771*.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015b. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143.

Babak Hassibi, David G Stork, and Gregory J Wolff. 1993. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021a. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021b. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

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. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Neal Lawton, Anoop Kumar, Govind Thattai, Aram Galstyan, and Greg Ver Steeg. 2023. Neural architecture search for parameter-efficient fine-tuning of large pre-trained language models. *arXiv preprint arXiv:2305.16597*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4582–4597. Association for Computational Linguistics.

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*.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. 2022. Estimating the carbon footprint of bloom, a 176b parameter language model. *arXiv preprint arXiv:2211.02001*.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30.

Eli Schwartz, Assaf Arbelle, Leonid Karlinsky, Sivan Harary, Florian Scheidegger, Sivan Doveh, and Raja Giryes. 2022. Maeday: Mae for few and zero shot anomaly-detection. *arXiv preprint arXiv:2211.14307*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems*, 35:12991–13005.

Yi-Lin Sung, Varun Nair, and Colin A Raffel. 2021. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Danilo Vucetic, Mohammadreza Tayaranian, Maryam Ziaeefard, James J Clark, Brett H Meyer, and Warren J Gross. 2022. Efficient fine-tuning of bert models on the edge. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1838–1842. IEEE.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv preprint*, abs/1910.03771.

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

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Michael Zhu and Suyog Gupta. 2018. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net.

A GLUE Dataset

Here is a summary of the benchmarks and metrics we used from the GLUE (Wang et al., 2019) dataset.

B PRILoRA GLUE Training Details

For all benchmarks we used a linear rank distribution from 4 to 12 (4,5,6,6,7,8,8,9,10,10,11,12), such that the average rank is 8 (ranks rounded to integers). All eight benchmarks were trained using linear learning-rate scheduling, with the initial learning rate reported as *learning rate*, and the number of epochs for the scheduler as *epochs*. The runs were stopped after *stop epoch* epochs. Hyper-parameters: learning rate, batch size, # epochs, decay and prune ratio were randomly searched over the space $\{6 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 6 \times 10^{-4}, 1 \times 10^{-3}, 1.2 \times 10^{-3}, 1.5 \times 10^{-3}, 2 \times 10^{-3}, 2.3 \times 10^{-3}\}$, $\{4, 8, 16, 32\}$, $\{10, 30, 50, 60, 70\}$, $\{0, 0.1, 0.01\}$, $\{0.25, 0.50, 0.75\}$ correspondingly. For all benchmarks and methods the *max seq length* is 128.

Table 6: Summary of the GLUE dataset

Corpus	Task	#Train	#Dev	#Label	Metrics
Single-Sentence Tasks					
CoLA	Grammatical Acceptability	8.5k	1k	2	Matthews corr
SST-2	Sentiment	67.3k	872	2	Accuracy
Pairwise Text Tasks					
MNLI	NLI (Entailment)	392k	9.8k	3	Matched Accuracy
RTE	NLI (Entailment)	2.5k	277	2	Accuracy
QQP	Semantic Equivalence	364k	40k	2	Accuracy
MRPC	Semantic Equivalence	3.7k	408	2	Accuracy
QNLI	Question Answering	105k	5.5k	2	Accuracy
STS-B	Similarity	5.7k	1.5k	1	Pearson/Spearman corr

Table 7: Hyper-parameters of PRILoRA for GLUE benchmark.

Dataset	learning rate	batch size	# epochs	stop epoch	decay	prune ratio
MNLI	1×10^{-4}	32	70	5	0.01	0.50
RTE	1.2×10^{-3}	32	70	25	0.01	0.50
QNLI	1×10^{-4}	32	60	3	0.01	0.50
MRPC	1×10^{-3}	32	60	15	0.01	0.50
QQP	6×10^{-4}	32	10	10	0.01	0.50
SST-2	6×10^{-5}	32	60	5	0.01	0.50
CoLA	2×10^{-4}	4	70	6	0.01	0.50
STS-B	2.3×10^{-3}	32	30	30	0.10	0.75