Fast Randomized Low-Rank Adaptation of Pre-trained Language Models with PAC Regularization

Zijian Lei[1], Dong Qian [2], William K. Cheung[1]

[1]Department of Computer Science, Hong Kong Baptist University, Hong Kong
[2]Linköping University, Sweden
{cszjlei,william}@comp.hkbu.edu.hk, dong.qian@liu.se

Abstract

Low-rank adaptation (LoRA) achieves parameter efficient fine-tuning for large language models (LLMs) by decomposing the model weight update into a pair of low-rank projection matrices. Yet, the memory overhead restricts it to scale up when the model size increases. We propose Randomized LoRA (RLoRA) which adopts Randomized Walsh-Hadamard Transform to achieve significant reduction in the size of trainable parameters compared to LoRA. At the same time, it allows a PAC-Bayes regularizer to be efficiently incorporated to improve generalization. We evaluate the effectiveness of RLoRA on LLMs RoBERTa, GPT-2 and LLaMA-7B using GLUE, E2E and math reasoning benchmarks. With a much lower memory requirement, RLoRA can give similar performance as the SOTA low-rank adaptation methods for these three tasks and significantly better performance under few-shot settings.

1 Introduction

Pre-trained Language Models (PLMs), typically consisting of millions or billions of parameters, have achieved the state-of-the-art performance on NLP tasks (Devlin et al., 2018). While PLMs can be fine-tuned for specific downstream tasks, their increasing scale makes the fine-tuning and model storage formidable.

Parameter-efficient fine-tuning (PEFT) aims to fine-tune a pre-trained model efficiently by adapting only a small set of parameters. Various approaches for PEFT have been explored. i) The *adapter*-based approach (Houlsby et al., 2019; mahabadi et al., 2021; Pfeiffer et al., 2020; He et al., 2021) introduces trainable adapter modules to achieve fine-tuning with the PLM frozen. This approach however falls short of the inference latency introduced. ii) The *masking*-based approach updates only some selected subset of the PLM's parameters (Guo et al., 2020; Zaken et al., 2021; Fu et al., 2022; Sung et al., 2021). As it requires access to the PLM for updating, its application is restricted. iii) *Prefix-tuning* (Li and Liang, 2021) and *prompt-tuning* (Lester et al., 2021) prepend additional tokens to the input embeddings and only train these soft prompts where the contextual information is wrapped to the inputs (or hidden states) without modifying the PLM. This approach however converges much slower than the adapter-based one (Ding et al., 2022; Hu et al., 2021).

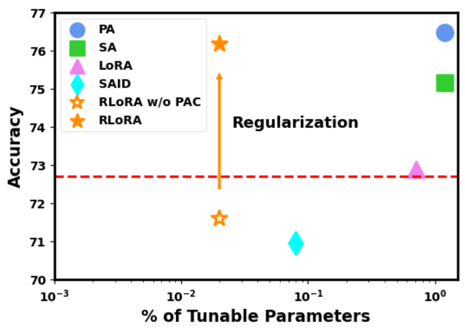


Figure 1: Performance of PEFT methods on fine-tuning Roberta-base using 200 training data points from the GLUE datasets. PA: Parallel Adapter (He et al., 2021); SA: Sequential Adapter (Houlsby et al., 2019); SAID: Intrinsic Dimension Adaptation (Aghajanyan et al., 2021); LoRA: Low-rank Adaptation (Hu et al., 2021), RLoRA: Randomized LoRA (*proposed*).

Low-rank adaptation (LoRA) (Hu et al., 2021) hypothesizes that the PLM's weight update for the fine-tuning possesses a low "intrinsic rank" dominating the desirable optimization trajectory (Li et al., 2018; Aghajanyan et al., 2021), and proposes to decompose the update of the weight (e.g., Transformer's self-attention weight metrics) into two low-rank projection matrices. This reparameteration trick has been shown particularly effective in reducing the number of trainable parameters compared to other PEFT approaches. Yet, the memory overhead is still considerably large when the size of the model increases.

To this end, inspired by the work on random feature projection for kernel machines (Rahimi and Recht, 2007; Liu et al., 2021),we propose *randomized low-rank adaptation* (RLoRA) which adopts the random matrix approximation to construct the up-projection and down-projection matrices in LoRA. In particular, we adopt the Randomized Walsh-Hadamard Transform (WHT) and a learnable scaling vector to reparameterize the low-rank projections to reduce both the time and space requirements. Using random matrices to reduce the memory requirement has also been explored in some recent work like LoRA-FA (Zhang et al., 2023b), VERA (Kopiczko et al., 2024) and NOLA (Koohpayegani et al., 2023). To contrast, RLoRA adopts the Randomized WHT which is orthogonal and recursively defined, which can further reduce the resources needed for the gradient computation w.r.t. the hidden states.

Also, the proposed RLoRA adopts PAC-Bayes regularization to enhance generalization performance, which is particularly important for adapting PLMs where only a small training set is available. Some related ideas have been explored in Lotfi et al. (2022, 2023), which empirically showed that the PAC-Bayesian theories provide a tighter bound and understanding for the role of model size and generalization ability. Achille et al. (2019); Wang et al. (2021) used the PAC-Bayesian framework to estimate the information in neural networks. The adoption of the Randomized WHT in RLoRA allows the regularization to be efficiently incorporated. To the best of our knowledge, work on incorporating the PAC-Bayes generalization bound for *regularizing* PEFT is still lacking in the literature.

Our contribution can be summarized as:

1. The number of trainable parameters in LoRA can be substantially reduced by incorporation of Randomized WHT.
2. We show theoretically how the "intrinsic subspace" of the PLM's weight update is characterized by Randomized WHT.
3. We show how a PAC-Bayes regularization term can be efficiently estimated and incorporated for fine-tuning RLoRA to boost its generalization performance.
4. We conduct comprehensive experiments to demonstrate the effectiveness of RLoRA for fine-tuning PLMs including Roberta-base, Roberta-large, GPT-2 medium and LLaMA-7B based on the GLUE, E2E and math reasoning benchmarks.

Our code will be made available for sharing via gitHub.

2 Background

In this section, we provide background and related work on low-rank adaptation, random matrix methods, and PAC-Bayes generalization bound.

2.1 Low-Rank Adaptation (LoRA)

LoRA (Hu et al., 2021) starts with the PLM $\boldsymbol{W}_0$ which is frozen and reparameterizes the PLM's weight update $\boldsymbol{\Delta W}$ using two low-rank projection matrices. The forward pass becomes:

$$\boldsymbol{h} = \boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{\Delta W}\boldsymbol{x} = \boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{W}_B\boldsymbol{W}_A\boldsymbol{x} \quad (1)$$

where $\boldsymbol{W}_B \in \mathbb{R}^{d_{out}\times r}$ and $\boldsymbol{W}_A \in \mathbb{R}^{r\times d_{in}}$ are the trainable parameters, and $r \ll \min\{d_{in}, d_{out}\}$. The memory overhead for LoRA is still an issue because of: i) high memory consumption to store the activation during the forward pass and to construct the gradient; and ii) the minimum number of trainable parameters lower-bounded by the rank-1 decomposition of the weight matrix which is considerably large as the model size increases.

2.2 LoRA with Random Matrix Methods

Random matrix theory has been applied to achieve PEFT. For instance, LoRA-FA (Zhang et al., 2023b) freezes the down-projection matrix $\boldsymbol{W}_A$ which is randomly initialized. VERA (Kopiczko et al., 2024) replaces both $\boldsymbol{W}_A$ and $\boldsymbol{W}_B$ with random matrices so that the weight update becomes: $\Delta\boldsymbol{W}_{VERA} = d\boldsymbol{W}_B b\boldsymbol{W}_A$ where only two scaling vectors d, b are learned. NOLA (Koohpayegani et al., 2023) uses multiple random seeds to generate the random basis to form $\boldsymbol{W}_A$ and $\boldsymbol{W}_B$, and learn the combination vectors α and β to compute $\Delta\boldsymbol{W}_{NOLA} = (\sum_{i=1}^{k}\alpha_i\boldsymbol{W}_{A_i}) \times \sum_{j=1}^{l}(\beta_j\boldsymbol{W}_{B_j})$.

In this paper, random projection methods (Li et al., 2018; Liu et al., 2021) are further explored to reduce the time and storage requirements while maintaining the effectiveness. In particular, we focus on the Randomized Walsh-Hadamard Transform (WHT) as it can efficiently draw a random matrix from a highly structured distribution (Tropp, 2011) (to detailed in the next section).

2.3 PAC-Bayes Generalization Bound

The PAC-Bayes bound is a theoretical framework that provides a probabilistic guarantee on the generalization performance of a learning algorithm.

It is based on the principles of probably approximately correct (PAC) learning and Bayesian inference. Given the true risk R and the empirical risk $\hat{R}_S$, the PAC-Bayes bound can be defined as:

$$R - \hat{R}_S \leq \mathbb{E}_{\mathcal{S}} \left\{ \mathbb{E}_{h \sim \mathcal{H}} \left[\mathrm{KL} \left(P(h) \middle| \frac{\sum_{h' \in \mathcal{H}} P(h'|\mathcal{S})}{|\mathcal{H}|} \right) \right] \right\} \tag{2}$$

where $\mathcal{S}$ represents the training data, $\mathcal{H}$ is the hypothesis space, $P(h)$ is the prior distribution over hypotheses, and $P(h'|\mathcal{S})$ is the posterior distribution over hypotheses given the data. The risk difference is upper bounded by the expected KL divergence between the prior and the data-dependent posterior distributions over hypotheses. This implies the PAC-Bayes framework rewards the model has strong prior on it parameters aligned with the data (Lotfi et al., 2022). The proposed RLoRA makes use of a PAC-Bayes bound to guide its fine-tuning process.

3 Methodology

In this section, we present the details about how the proposed RLoRA. In particular, Section 3.1 presents the Randomized WHT and how it is integrated into RLoRA to gain parameter-efficiency and reduction in computational cost during training. In Section 3.2, we show theoretically how the posterior distribution of the PLM's weight update based on the reparameterization is associated with a lower-dimensional "intrinsic subspace" characterized by the Randomized WHT. We derive the PAC-Bayes generalization bound in Section 3.3 and explain how the reparameterization by Randomized WHT allows the bound to be efficiently incorporated into RLoRA to regularize the fine-tuning. Fig. 2 shows an overall illustration of RLoRA.

3.1 LoRA with Randomized WHT Incorporated

RLoRA builds upon LoRA. It replaces $\boldsymbol{W}_A$ and $\boldsymbol{W}_B$ in LoRA with two WHT_A and WHT_B approximated by Random WHT as well as two trainable scaling vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, so that the low-rank decomposition of $\Delta \boldsymbol{W}$ becomes:

$$\Delta \boldsymbol{W} = [\mathrm{diag}(\boldsymbol{\beta})\, WHT_B][\mathrm{diag}(\boldsymbol{\alpha})\, WHT_A], \tag{3}$$

where $\mathrm{diag}(\boldsymbol{\alpha})_{ii} = \alpha_i$.

3.1.1 Efficient generation of projection matrices

The projection matrices WHT_A and WHT_B can be efficiently *generated* by i) first setting a random seed and ii) applying the WHT without the need to explicitly store the projection matrix.

Specifically, according to (Choromanski et al., 2017), a $d \times d$ Gaussian random matrix $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ can be approximated by a product of Randomized Walsh-Hadamard matrice $\boldsymbol{H}$ and diagonal matrices $\boldsymbol{S}$ and $\boldsymbol{B_i}$, given as:

$$\boldsymbol{R} = \frac{1}{\sigma \sqrt{d}} \boldsymbol{S} \prod_{i=1}^{k} \boldsymbol{H} \boldsymbol{B_i}. \tag{4}$$

$\boldsymbol{S}$ is a random scaling and subsample matrix. $\boldsymbol{B_i}$ has elements of independent random signs $\{-1, 1\}$ along its diagonal. The memory cost of $\boldsymbol{S}$ and $\boldsymbol{B_i}$ is both $O(d)$. The value of k determines how many independent random signals $\boldsymbol{B_i}$ should be multiplied. Increasing the value of k can improve the randomness and the orthogonality of the resulting matrix $\boldsymbol{R}$. Empirical findings suggest that setting $k = 2$ or 3 is sufficient for practical applications.

For the matrix $\boldsymbol{H} \in \mathbb{R}^{d \times d}$, it can be efficiently generated due to its recursive definition: $\boldsymbol{H}_d = \begin{bmatrix} \boldsymbol{H}_{d/2} & \boldsymbol{H}_{d/2} \\ \boldsymbol{H}_{d/2} & -\boldsymbol{H}_{d/2} \end{bmatrix}$ with $\boldsymbol{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. The generation iterates two simple steps: i) dividing the data into two halves, and ii) performing a simple $\pm$ operation. Fig. 2(c) shows an example of an 8-dimensional WHT. This fast WHT allows $\boldsymbol{Hx}$ to be computed in $O(d \log d)$ time and the transformation basis to be orthogonal.

3.1.2 Efficient matrix multiplication during backpropagation

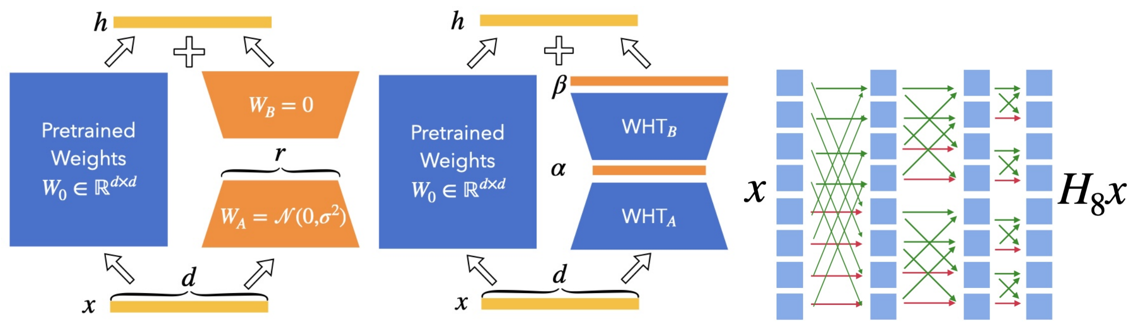
Assume that the hidden state is denoted as $\boldsymbol{x} \in \mathbb{R}^{n \times d_{in}}$ and the weight as $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ where n is the batch size and $d_{in}(d_{out})$ is the dimension of the input(output) hidden state. The information propagates between layer l and layer $l+1$ as:

$$\begin{gathered} \boldsymbol{x}_{l+1} = \boldsymbol{x}_l \boldsymbol{W}_l \\ g(\boldsymbol{W}) = g(\boldsymbol{x}_{l+1}) \boldsymbol{x}_l \quad g(\boldsymbol{x}_l) = g(\boldsymbol{x}_{l+1}) \boldsymbol{W}_l \end{gathered} \tag{5}$$

where $g(\boldsymbol{W})$ is the gradient used to update the trainable parameters and $g(\boldsymbol{x}_l)$ is for propagating the gradient to other layers. Given the Randomized WHT adopted, Eq.5 becomes:

$$\begin{aligned} \boldsymbol{x}_{l+1} &= \boldsymbol{\alpha}_l\, WHT_l(\boldsymbol{x}_l) \\ g(\boldsymbol{\alpha}_l) &= g(\boldsymbol{x}_{l+1})\, WHT_l(\boldsymbol{x}_l) \\ g(\boldsymbol{x}_l) &= WHT_l^{-1}(g(\boldsymbol{x}_{l+1})) \end{aligned} \tag{6}$$

where $WHT_l(\boldsymbol{x}_l)$ is the multiplication for layer l which can be efficiently computed, $WHT_l^{-1}()$ is the reverse projection to layer $l$and $\boldsymbol{\alpha}_l$ is the corresponding scaling vector.


h
Pretrained Weights $W_0 \in \mathbb{R}^{d \times d}$
$W_B = 0$
r
$W_A = \mathcal{N}(0,\sigma^2)$
d
x
h
β
Pretrained Weights $W_0 \in \mathbb{R}^{d \times d}$
WHT_B
α
WHT_A
d
x
x
$H_8 x$

(a) LoRA (b) RLoRA (c) Walsh-Hadamard Transform

Figure 2: Illustration of LoRA, RLoRA and the Walsh-Hadamard Transform. In Fig.2(c), the green and red lines indicate the "add" and "minus" operations respectively. The whole process iteratively divides the data into two halves and applies simple $\pm$ operations. A simple WHT only needs $\mathcal{O}(d \log d)$ to embed a single block.

3.2 Intrinsic Subspace and Posterior Distribution of Weight Update

To better understand the intrinsic subspace of the PLM's weight update, we first assume that the posterior distribution of the weight update $\Delta \boldsymbol{W}$ follows a matrix-variate Gaussian. Then, we show that the distribution after the reparameterization is associated with a low-dimensional "intrinsic subspace" characterized by the Randomized WHT matrices.

With reference to the analysis in (Rossi et al., 2020), we assume that the posterior distribution of the weight update $\Delta \boldsymbol{W} = \boldsymbol{W}_A \boldsymbol{W}_B \in \mathbb{R}^{d_{in} \times d_{out}}$ follows a matrix-variate Gaussian distribution, given as $\Delta \boldsymbol{W} \sim \mathcal{MN}(\boldsymbol{M}, \boldsymbol{U}, \boldsymbol{V})$ where $\boldsymbol{M}$ is the mean, and $\boldsymbol{V} \in \mathbb{R}^{d_{out} \times d_{out}}$ and $\boldsymbol{U} \in \mathbb{R}^{d_{in} \times d_{in}}$ denote the covariance matrices among the rows and columns of $\Delta \boldsymbol{W}$ respectively. The vectorized version of $\Delta \boldsymbol{W}$ can be expressed equivalently as:

$$\text{vect}(\Delta \boldsymbol{W}) \sim \mathcal{N}(\text{vect}(\boldsymbol{M}), \boldsymbol{V} \otimes \boldsymbol{U}), \quad (7)$$

where $\otimes$ denotes the Kronecker product.

3.2.1 Intrinsic subspace of ΔW

We rewrite $\Delta \boldsymbol{W}$ by replacing $\boldsymbol{W}_A$ and $\boldsymbol{W}_B$ using the Randomized WHT according to Eqs. 4 and 3. For the convenience of the analysis, we assume $k = 1$[1] and drop the scalar factor $\frac{1}{\sigma\sqrt{d}}$. Also, we denote the matrix selecting the first r rows (columns) from W_A (W_B) as $\boldsymbol{S}_{W_A}$ ($\boldsymbol{S}_{W_B}$). Then, it gives:

$$\Delta \boldsymbol{W} = \text{diag}(\boldsymbol{\beta}) \boldsymbol{S}_{W_B} \boldsymbol{H} \boldsymbol{B}_{W_B} \boldsymbol{S}_{W_A}^T \text{diag}(\boldsymbol{\alpha}) \boldsymbol{S}_{W_A} \boldsymbol{H} \boldsymbol{B}_{W_A}. \quad (8)$$

[1]Note that the result of this analysis can be easily generalized to cases with other values of k. We leave that in the Appendix.

By putting $\boldsymbol{H}\boldsymbol{B}_{W_A} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d]$ where $\boldsymbol{v}_i = \boldsymbol{H}_{(i,:)} \boldsymbol{B}_{W_A(i,i)}$, vect$(\Delta \boldsymbol{W})$ can be expressed as:

$$\text{vect}(\Delta \boldsymbol{W}) = \boldsymbol{G}\boldsymbol{\alpha}$$
$$= \begin{bmatrix} \text{diag}(\boldsymbol{\beta}) \boldsymbol{S}_{W_B} \boldsymbol{H} \boldsymbol{B}_{W_B} \boldsymbol{S}_{W_A}^T \boldsymbol{S}_{W_A} \text{diag}(\boldsymbol{v}_1) \\ \text{diag}(\boldsymbol{\beta}) \boldsymbol{S}_{W_B} \boldsymbol{H} \boldsymbol{B}_{W_B} \boldsymbol{S}_{W_A}^T \boldsymbol{S}_{W_A} \text{diag}(\boldsymbol{v}_2) \\ \vdots \\ \text{diag}(\boldsymbol{\beta}) \boldsymbol{S}_{W_B} \boldsymbol{H} \boldsymbol{B}_{W_B} \boldsymbol{S}_{W_A}^T \boldsymbol{S}_{W_A} \text{diag}(\boldsymbol{v}_d) \end{bmatrix} \boldsymbol{\alpha} \quad (9)$$

where $\boldsymbol{G} \in \mathbb{R}^{d_{in} d_{out} \times r}$, $\boldsymbol{\alpha} \in \mathbb{R}^r$, $\boldsymbol{\beta} \in \mathbb{R}^{d_{out}}$. The factor $\boldsymbol{S}_{W_A}^T \boldsymbol{S}_{W_A}$ is essentially randomly selecting r basis vectors from $\boldsymbol{G}$ to construct the projection from $\boldsymbol{\alpha}$ to vect$(\Delta \boldsymbol{W})$.

3.2.2 Posterior distribution of ΔW

Assume that RLoRA's trainable parameters $\boldsymbol{\alpha}, \boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{\mu}_\alpha, \boldsymbol{\Sigma}_\alpha), \mathcal{N}(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta)$, it gives

$$\text{vect}(\Delta \boldsymbol{W}) \sim \mathcal{N}(\boldsymbol{\mu}_\alpha \boldsymbol{G}, \boldsymbol{G}^T \boldsymbol{\Sigma}_\alpha \boldsymbol{G}). \quad (10)$$

This implies that the posterior distribution of $\Delta \boldsymbol{W}$ is in fact lying on the d_α-dimensional subspace of $d_{in}d_{out}$-dimensional space embedded by the orthogonal projector $\boldsymbol{G}$. The orientation of the space $\Delta \boldsymbol{W}$ is controlled by the random directions $\boldsymbol{B}_{W_B}$ $\boldsymbol{B}_{W_A}$ and the learnable $\boldsymbol{\beta}$, which could be regarded as the learnable intrinsic subspace of the network (Aghajanyan et al., 2021).

In addition, we can show that Eq. 10 is equivalent to a matrix-variate Gaussian for $\Delta \boldsymbol{W}$ with the parameters $\boldsymbol{M}, \boldsymbol{U}, \boldsymbol{V}$ given as:

$$\boldsymbol{M} = \text{diag}(\boldsymbol{\mu}_\beta) \boldsymbol{S}_{W_B} \boldsymbol{H} \boldsymbol{B}_{W_B} \boldsymbol{S}_{W_A}^T \text{diag}(\boldsymbol{\mu}_\alpha) \boldsymbol{S}_{W_A} \boldsymbol{H} \boldsymbol{B}_{W_A}$$
$$\boldsymbol{U}^{\frac{1}{2}} = \text{diag}(\boldsymbol{\beta}) \boldsymbol{S}_{W_B} \boldsymbol{H} \boldsymbol{B}_{W_B} \boldsymbol{S}_{W_A}^T \text{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}) \boldsymbol{S}_{W_B}^T \text{diag}(\boldsymbol{\beta})$$
$$\boldsymbol{V}^{\frac{1}{2}} = \frac{1}{\sqrt{\text{tr}(\boldsymbol{U})}} \boldsymbol{B}_{W_A} \boldsymbol{H} \boldsymbol{S}_{W_A}^T \text{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}) \quad (11)$$

which can be estimated based on the Randomized WHT. See Appendix B for detailed derivation.

3.3 PAC-Bayes Regularization

We derive an approximation of the PAC-Bayesian generalization bound to guide the learning process for better generation. According to Section 3.2, if $\boldsymbol{\alpha}$ follows the Gaussian distribution, the weight update matrix $\Delta\boldsymbol{W}$ will follow the matrix-variate Gaussian. For the clarity reason, we use $\boldsymbol{w}$ to denote the trainable model parameters (i.e., α and β for our case) in the sequel.

3.3.1 PAC-Bayes generalization bound

Suppose the posterior of the model parameters $\boldsymbol{w}$ given the training data S of size N for a specific fine-tuning task is $p(\boldsymbol{w}|S) \sim \mathcal{N}(\boldsymbol{\mu}_S, \boldsymbol{\Sigma}_S)$ and the oracle prior $p(\boldsymbol{w}) = \mathbb{E}_{p(S)}[p(\boldsymbol{w}\,|S)] \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The expectation of the KL divergence between the posterior and the prior of $\boldsymbol{w}$ over the task-specific data S becomes:

$$
\begin{aligned}
I(\boldsymbol{w}) &= \mathbb{E}_{p(S)}\left[KL\left(p(\boldsymbol{w}|S), p(\boldsymbol{w})\right)\right] \\
&\propto \mathbb{E}_{p(S)}\left[\log\frac{|\boldsymbol{\Sigma}_S|}{|\boldsymbol{\Sigma}|} + \mathrm{tr}\left(\frac{\boldsymbol{\Sigma}}{\boldsymbol{\Sigma}_S}\right)\right. \\
&\qquad\left. +(\boldsymbol{\mu}_S-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_S-\boldsymbol{\mu})\right]
\end{aligned}
\tag{12}
$$

where $|\boldsymbol{A}|$ and $\mathrm{tr}(\boldsymbol{A})$ are the determinant and trace of matrix $\boldsymbol{A}$, respectively. By assuming the covariance of prior and posterior to be proportional, which is common for building PAC-Bayesian bound (Dziugaite and Roy, 2018), the log and trace terms become constant. We have

$$
\begin{aligned}
I(\boldsymbol{w}) &\propto \mathbb{E}_{p(S)}\left[(\boldsymbol{\mu}_S-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_S-\boldsymbol{\mu})\right] \\
&= \mathbb{E}_{p(S)}\left[\boldsymbol{\mu}_S^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_{\boldsymbol{S}}\right] - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}.
\end{aligned}
\tag{13}
$$

To achieve a small KL with the data-dependent posterior, the prior should essentially be a good predictor for the posterior, which will be hard without access to the data distribution $p(S)$.

3.3.2 Estimating prior covariance matrix

As the pre-trained network has already converged to a stable solution, we assume that the weight update introduced by fine-tuning for the downstream task are small perturbations to refine this prior. So we first assume $\Delta\boldsymbol{W}$ is $\boldsymbol{0}$ mean matrix-variate Gaussian. To achieve this, we initialize $\boldsymbol{\alpha}, \boldsymbol{\beta}$ as $\boldsymbol{\alpha} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_\alpha)$ and $\boldsymbol{\beta} = \boldsymbol{0}$. Then, to obtain the bound, we only need to focus on estimating the prior covariance:

$$
\boldsymbol{\Sigma} = \mathbb{E}_{p(S)}\left[(\boldsymbol{\mu}_S-\boldsymbol{\mu})(\boldsymbol{\mu}_S-\boldsymbol{\mu})^T\right] \tag{14}
$$

As mentioned, the exact estimation of $\boldsymbol{\Sigma}$ in requires knowledge of the distribution $p(S)$. Inspired by Wang et al. (2021), we can bootstrap (resample with replacement) from the training data S for N_S times to construct the bootstrap datasets $\{S_i\}_{i=1}^{N_S}$.

$$
\boldsymbol{\Sigma} \approx \frac{1}{N_S}\sum_{i=1}^{N_S}\left[(\boldsymbol{\mu}_S-\boldsymbol{\mu}_{S_i})(\boldsymbol{\mu}_S-\boldsymbol{\mu}_{S_i})^T\right] \tag{15}
$$

However, to get $\boldsymbol{\mu}_{S_i}$, we still need to optimize over the bootstrap datasets $\{S_i\}_{i=1}^{N_S}$, which is not practical in neural network training. Instead, we approximate this using the group influence effect which studies the change to the model if we remove a group of data.

Lemma 3.1. ***Group Influence Function*** *(Basu et al., 2020) Assume that when all samples in a group U are up-weighted by $\epsilon \to 0$, the parameter difference can be approximated as:*

$$
\boldsymbol{\mu}_{S-U} - \boldsymbol{\mu}_S \triangleq -\frac{1}{1-p}\frac{1}{|S|}H_{\mu_S}^{-1}\sum_{z\in U}\nabla l(h_{\mu_S}(z)) \tag{16}
$$

where S is the training sample set, $\nabla l(h_{\mu_S}(z))$ and $H_\mu = \nabla_\mu^2 l(h_{\mu_S}(z))$ are the gradient and Hessian of the loss function l defined on the model $h_{\mu_S}()$, and $\boldsymbol{\mu}_S$ refers to the optimal parameters trained by dataset S. By taking U as a subset resampled from the training data S, we can show that the oracle prior covariance in Eq. 15 can be estimated by

$$
\begin{aligned}
\boldsymbol{\Sigma} &\propto H_\mu^{-1}\sum_{z\in U}\nabla l(h_\mu)\sum_{z\in U}\nabla l(h_\mu)^T(H_\mu^{-1})^T \\
&\propto H_\mu^{-1}\boldsymbol{F}_{\mu,U}(H_\mu^{-1})^T \propto \boldsymbol{F}_{\mu,U}^{-1}
\end{aligned}
\tag{17}
$$

where $\boldsymbol{F}_{\mu,U}$ is the Fisher information matrix estimated using the resampled dataset U. See Appendix B for detailed derivation.

3.3.3 Estimating the bound for regularization

Based on Eq. 17, the generalization bound in Eq. 13 can be estimated as:

$$
\begin{aligned}
I(\boldsymbol{w}) &= \mathbb{E}_{p(S)}\left[KL(p(\boldsymbol{w}|S), p(\boldsymbol{w}))\right] \\
&\propto \mathbb{E}_{p(S)}\left[(\boldsymbol{\mu}_S-\boldsymbol{\mu})^T\boldsymbol{F}_\mu(\boldsymbol{\mu}_S-\boldsymbol{\mu})\right] \\
&\simeq I(\boldsymbol{w}; S) = \sum_{t\in U}\left[\Delta\boldsymbol{\mu}^T\nabla_\mu l_t(\boldsymbol{\mu}_S)\right]^2
\end{aligned}
\tag{18}
$$

where $I(\boldsymbol{w})$ is further approximated by $I(\boldsymbol{w}; S)$ in the third step as the U is resampled from S. The overall objective function is:

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}; S) + \lambda I(\boldsymbol{w}; S) \tag{19}$$

where $L(\boldsymbol{w}; S) = -\sum_{i=1}^{N} \log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{w})$ is the likelihood with $\boldsymbol{y}_i$ the output labels in S. To utilize this regularization in practice, we can estimate the information every few steps (e.g. at the end of each epoch) and use this as the regularization term until the next estimation of $I(\boldsymbol{w}; S)$.

4 Experiment

We evaluate the performance of RLoRA using RoBERTa-base/RoBERTa-large (Liu et al., 2019) and GPT-2-medium (Radford et al., 2019). Our experiments encompass a diverse range of tasks, including natural language understanding (NLU) and natural language generation (NLG). Specifically, we evaluate the models on the GLUE benchmark (Wang et al., 2018) for RoBERTa-base/RoBERTa-large. For evaluation on GPT-2 medium, to facilitate direct comparison, we adopt the experimental setup outlined in the original work of LoRA (Hu et al., 2021). Additional details regarding the datasets used can be found in Appendix C. All the experiments were conducted using the NVIDIA Tesla V100.

4.1 Baselines

- Full Fine-tuning (FT): The model is initialized with pre-trained weights and all the parameters are trained.
- Adapter (Adpt) (Houlsby et al., 2019): It inserts adapter layers (two-layer fully connected with bias) between the self-attention and the MLP modules.
- BitFit (Zaken et al., 2021): It freezes the pre-trained weight matrix and trains the biases.
- Low-Rank Adaptation (LoRA) (Hu et al., 2021): It represents the weight update using a low-rank decomposition.
- Vector-based Random Adaptation (VeRA) (Kopiczko et al., 2024): It replaces the low-rank matrices in LoRA with random matrices and learn only scaling vectors.

4.2 Natural Language Understanding

We first evaluate our approach on the General Language Understanding Evaluation (GLUE) benchmark which consists of multiple tasks including paraphrase detection (MRPC), sentiment classification (SST-2), natural language inference (RTE, QNLI) and linguistic acceptability (CoLA). Since the original test sets are not publicly available, we use the original validation set as the test set. For estimating the information term in Eq. 18, we randomly resample 100 training data at the end of each epoch for analysis.

Table 1 shows the fine-tuning results on RoBERTa-base and RoBERTa-large using the GLUE benchmark. For RoBERTa-base model, due to the budget limitation, we did not use the *MNLI trick*[2] to fine-tune the MRPC, RTE and STSB. According to the results on RoBERTa-large, RLoRA can achieve the best results on 6 datasets just like LoRA and VeRA. At the same time, similar to VeRA, RLoRA has the number of trainable parameters subtantially reduced by ~ 13 times compare to LoRA due to the use of the random WHT for the low-rank projection. Additionally, it is worth noting that while RLoRA can achieve comparable performance as LoRA and VeRA on different datasets, its standard deviation is the smallest among them. This indicates that the PAC-Bayes regularization allows RLoRA to be more robust against the effect due to the random seeds and improve the stability of training results under different random seed settings.

4.2.1 Effectiveness of PAC-Bayes Regularization

To further evaluate the effectiveness of incorporating the PAC-Bayes term for enhancing the generalization performance, we compare the few-shot performance among LoRA, VeRA and RLoRA. We need to re-train LoRA and VeRA to evaluate their performance on the few-shot settings. We also include a version of RLoRA without the PAC-Bayes regularization term for the ablation study to underscore the impact of the regularization. We select five tasks from the GLUE benchmark for this few-shot evaluation and train the models with only 100 and 200 data samples. The performance is evaluated on the original evaluation set. As shown in Table 2, we can observe that only utilizing the Randomized WHT can reduce the number of trainable parameters to 0.02% of the original model (1/35 of LoRA) while achieving comparable performance. Furthermore, incorporating the PAC-Bayes regu-

[2] Hu et al. (2021) used the checkpoint fine-tuned with MNLI task instead of the pre-trained checkpoint to initialize the model for MRPC, RTE and STSB.

Table 1: Results of different PEFT methods on the GLUE benchmark. We report Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for the remaining tasks. In all cases, higher values indicate better performance. Results of the baselines are sourced from prior work (Kopiczko et al., 2024). **Bold** font and the Underline font indicate the best and the second-best performance for each dataset.

	Method	# Trainable Parameters	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg.
BASE	FT	125M	94.8	90.2	63.6	92.8	78.7	91.2	85.2
	BitFit	0.1M	93.7	**92.7**	62.0	91.8	81.5	90.8	85.4
	Adpt^{D}	0.3M	$94.2_{\pm 0.1}$	$88.5_{\pm 1.1}$	$60.8_{\pm 0.4}$	$93.1_{\pm 0.1}$	$71.5_{\pm 2.7}$	$89.7_{\pm 0.3}$	83.0
	Adpt^{D}	0.9M	$94.7_{\pm 0.3}$	$88.4_{\pm 0.1}$	$62.6_{\pm 0.9}$	$93.0_{\pm 0.2}$	$75.9_{\pm 2.2}$	$90.3_{\pm 0.1}$	84.2
	LoRA	0.3M	$\mathbf{95.1}_{\pm 0.2}$	$89.7_{\pm 0.7}$	$63.4_{\pm 1.2}$	$\mathbf{93.3}_{\pm 0.3}$	$\mathbf{86.6}_{\pm 0.7}$	$\mathbf{91.5}_{\pm 0.2}$	**86.6**
	VeRA	**0.031M**	$94.5_{\pm 0.3}$	$89.7_{\pm 0.8}$	$\mathbf{64.1}_{\pm 1.7}$	$91.9_{\pm 0.2}$	$75.8_{\pm 1.8}$	$90.3_{\pm 0.2}$	84.4
	RLoRA	**0.031M**	$94.6_{\pm 0.16}$	$90.2_{\pm 0.28}$	$61._{\pm 0.45}$	$91.8_{\pm 0.04}$	$77.8_{\pm 0.18}$	$90.9_{\pm 0.14}$	84.5
LARGE	Adpt^{P}	3M	$96.1_{\pm 0.3}$	$90.2_{\pm 0.7}$	$\mathbf{68.3}_{\pm 1.0}$	$\mathbf{94.8}_{\pm 0.2}$	$83.8_{\pm 2.9}$	$92.1_{\pm 0.7}$	87.6
	Adpt^{P}	0.8M	$\mathbf{96.6}_{\pm 0.2}$	$89.7_{\pm 1.2}$	$67.8_{\pm 2.5}$	$\mathbf{94.8}_{\pm 0.3}$	$80.1_{\pm 2.9}$	$91.9_{\pm 0.4}$	86.8
	Adpt^{H}	6M	$96.2_{\pm 0.3}$	$88.7_{\pm 2.9}$	$66.5_{\pm 4.4}$	$94.7_{\pm 0.2}$	$83.4_{\pm 1.1}$	$91.0_{\pm 1.7}$	86.8
	Adpt^{H}	0.8M	$96.3_{\pm 0.5}$	$87.7_{\pm 1.7}$	$66.3_{\pm 2.0}$	$94.7_{\pm 0.2}$	$72.9_{\pm 2.9}$	$91.5_{\pm 0.5}$	84.9
	LoRA	0.8M	$96.2_{\pm 0.5}$	$90.2_{\pm 1.0}$	$\underline{68.2}_{\pm 1.9}$	$\mathbf{94.8}_{\pm 0.3}$	$85.2_{\pm 1.1}$	$\mathbf{92.3}_{\pm 0.5}$	**87.8**
	VeRA	**0.061M**	$96.1_{\pm 0.1}$	$\mathbf{90.9}_{\pm 0.7}$	$68.0_{\pm 0.8}$	$94.4_{\pm 0.2}$	$\mathbf{85.9}_{\pm 0.7}$	$91.7_{\pm 0.8}$	**87.8**
	RLoRA	**0.061M**	$\underline{96.4}_{\pm 0.1}$	$\underline{90.8}_{\pm 0.4}$	$67.5_{\pm 0.2}$	$94.1_{\pm 0.2}$	$\underline{85.7}_{\pm 0.5}$	$\underline{92.1}_{\pm 0.2}$	**87.8**

Table 2: Training on GLUE datasets using only 100 and 200 training data samples. The percentage of trainable parameters with reference to the full model is indicated in the bracket next to each method.

	# of train data	Method	RTE	MRPC	STS-B	SST-2	QNLI	Avg
BASE	100	LoRA (0.71%)	$54.87_{\pm 1.28}$	$76.66_{\pm 0.16}$	$7.61_{\pm 3.89}$	$86.62_{\pm 0.38}$	$57.05_{\pm 0.09}$	56.56
		VeRA (0.02%)	$57.16_{\pm 2.07}$	$76.62_{\pm 0.09}$	$21.23_{\pm 1.79}$	$76.38_{\pm 2.42}$	$53.93_{\pm 0.22}$	57.06
		RLoRA (0.02%)	$57.25_{\pm 1.12}$	$76.63_{\pm 0.17}$	$24.07_{\pm 0.96}$	$84.06_{\pm 2.11}$	$54.64_{\pm 0.94}$	**59.36**
		w/o PAC-Reg	$57.16_{\pm 2.25}$	$76.72_{\pm 0.25}$	$22.39_{\pm 1.40}$	$82.30_{\pm 2.74}$	$54.61_{\pm 0.69}$	58.64
	200	LoRA (0.71%)	$54.87_{\pm 0.56}$	$79.25_{\pm 0.94}$	$66.74_{\pm 5.23}$	$88.49_{\pm 0.66}$	$75.02_{\pm 1.73}$	72.88
		VeRA (0.02%)	$54.39_{\pm 0.90}$	$80.78_{\pm 1.32}$	$72.88_{\pm 1.41}$	$89.14_{\pm 0.14}$	$77.69_{\pm 0.62}$	74.98
		RLoRA (0.02%)	$56.08_{\pm 0.68}$	$80.34_{\pm 0.98}$	$79.05_{\pm 1.29}$	$89.33_{\pm 1.29}$	$76.08_{\pm 2.64}$	**76.18**
		w/o PAC-Reg	$53.79_{\pm 0.88}$	$79.95_{\pm 1.32}$	$61.86_{\pm 1.29}$	$89.18_{\pm 0.66}$	$76.38_{\pm 1.35}$	72.23

larization can further improve the generalization performance indicating that the model manages to capture the task-related information from the input data. Fig.1 shows an overall performance comparison between RLoRA and other baselines for fine-tuning large models given a small training set. The advantage of RLoRA with PAC regularization can be clearly observed.

To illustrate the importance of using the generalization bound as the information measure for regularizing the fine-tuning process instead of using it only for sub-network selection (like the mask-based approach), we perform experiments to monitor the information change based on Eq. 18 in each layer of Roberta-base during the fine-tuning on RTE and MRPC datasets for 10 epochs. We estimate the layer-wise information at the end of each epoch. Fig. 3 show the temporal dynamics of the information flow during the fine-tuning process. We can observe that at different stages of training, the information "flows" through the layers. This implies that identifying and fixing some sparse masks on the network could be a too restrictive approach for regularizing the entire training process. In particular, a wrongly estimated mask might steer the whole optimatization process astray.

4.3 NLG Benchmark

For the performance of RLoRA on Natural Language Generation (NLG) models, we make use of GPT-2 medium (Radford et al., 2019). To make a direct comparison, we keep our experiment setup as close as possible to the approach in (Hu et al., 2021). Due to the space limit, we will only present our results on the E2E NLG Challenge.

For fine-tuning LoRA and VeRA, we use the values of the hyperparameters provided in their papers and tune the learning rate from $\{5 \times 10^{-1}, 1 \times 10^{-1}, 5 \times 10^{-2}, ..., 5 \times 10^{-5}\}$.

Table 3: Results of different PEFT methods on the E2E benchmark using the GPT-2 medium. Results for methods with asterisk (*) are taken from prior work (Hu et al., 2021). For all metrics, the higher the better. **Bold** fonts and the Underline fonts indicate the best and the second-best performance for each dataset.

Method	# Trainable Parameters	BLEU	NIST	METEOR	ROUGE-L	CIDEr
FT*	354.92M	68.2	8.62	46.2	71.0	2.47
Adpt^L*	0.37M	66.3	8.41	45.0	**69.8**	**2.40**
LoRA	0.35M	67.48	8.61	**46.2**	69.2	2.35
VeRA	0.098M	67.22	8.55	44.92	67.35	2.27
RLoRA	0.098M	**68.16**	**8.66**	**46.2**	68.97	2.37
w/o PAC-Reg	0.098M	67.19	8.60	45.69	67.63	2.34

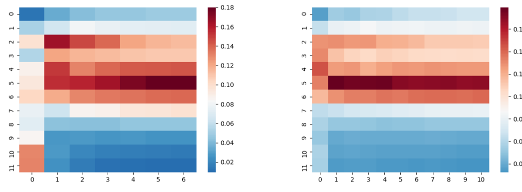

(a) RTE (b) MRPC

Figure 3: Dynamics of the layer-wise information in network during the fine-tuning process. The x-axis shows the training progress from the initial epoch on the left to the final epoch on the right. The y-axis is the index of the layer from the input layer on the top to the output layer at the bottom. The degree of redness indicates the density of information.

We further search for the best regularization strength λ from $10^{-4} \sim 10^{-6}$. Detailed hyperparameter settings can be found in Appendix C.

As shown in Table 3, for generating text using GPT-2 medium fine-tuned with E2E datasets, RLoRA as compared to LoRA, VeRA, and Adpt achieves the best or second-best results across most of the metrics and is competitive with full fine-tuning. Furthermore, by referring to the last rows in Table 3, we can clearly observe the improvement brought by the PAC-Bayes regularization.

4.4 Arithmetic Reasoning

To evaluate the performance comparison on more recent large model, we finetune the LLaMA$_{7B}$ and the corresponding consumption of computational resources.

By following the experiment settings in (Hu et al., 2023), we evaluate the performance of fine-tuning LLaMA$_{7B}$ model for math reasoning tasks where a math reasoning dataset with 10K training data is used and 4 different tasks were being evaluated. Results with * in the following table are taken from the original paper.

The results in Table.4 show that RLoRA can attain performance comparable to other baseline methods (Prefix, Adapter and LoRA) with the fewest number of parameters. The last 4 rows reveal that given similar memory requirements, RLoRA exhibits superior performance than LoRA. Also, comparing with LoRA(r=32) and Adapter, RLoRA can achieve comparable performance with $8\times$ smaller space required for storing the model. The more downstream tasks we need to tackle, the more obvious our storage advantage becomes. For 100 tasks, LoRA would require 20G space, while RLoRA needs only 2G storage for similar performance. In addition, the smallest possible model for LoRA is the one with rank=1 which highly likely will not give good adaptation performance. Yet, its size may still restrict it to be applied under some extreme memory resource constraint situations.

To evaluate the reduction in memory utilization, we compared the resource consumption with minimum requirements for training LoRA. For RLoRA, it is much more computationally efficient due to the adoption of Walsh-Hadamard Transform (WHT). Table.5 shows the minimum Flops and CUDA memory usage needed by LoRA and RLoRA. RLoRA consumes less computational resources than LoRA, and we set the batchsize=1 to get this result.

4.5 Comparison: RLoRA vs VeRA

In comparison with the contemporaneous work VeRA which also generates a random projection matrix as well, RLoRA implements the structured Randomized WHT which can make the weight mul-

Table 4: Accuracy comparison of LLMs with different adapters on 4 math reasoning datasets. Results for methods with asterisk (*) are taken from prior work

Method	Param(%)	Storage (MB)	MultiArith	AddSub	SingleEq	SVAMP	Avg
Adapter*	0.99	256MB	92.8	80.0	83.5	52.3	77.13
LoRA (r=32)*	0.83	215MB	95.0	83.3	84.4	52.1	78.65
Prefix*	0.11	28.5MB	63.2	57.0	55.3	38.1	53.4
LoRA (r=4)	0.10	27MB	94.5	83.3	84.1	46.7	77.07
RLoRA	0.10	27MB	96.5	82.8	83.9	47.2	77.6
LoRA (r=1)	0.026	6.8M	92.5	77.4	77.2	44.2	72.8
VeRA	0.017	4.1MB	92.6	71.1	69.8	38.8	69.08
RLoRA	0.017	4.1MB	93.8	76.7	79.7	44.8	73.95

Table 5: Impact on GPU memory

Method	Max. CUDA Memory
RLoRA(rank=64)	16537 MB
LoRA (rank=64)	17619 MB

tiplication more efficient ($\mathcal{O}(d \log d)$). Based on our current implementation (not yet optimized), it can enhance the training from processing 14.7 samples per sec. for VeRA to 15.3 samples per sec. for RLoRA [3]. Also, the use of Randomized WHT can effectively scale the PAC-Bayes regularization to large models, leading to improved performance evidenced by our experiment results.

5 Limitations and Future Work

In this paper, we proposed Randomized Walsh-Hadamard low-rank adaptation to reduce the fine-tuning resource consumption and scale the PAC-Bayesian regularization to larger model. The current estimation of the PAC-Bayes generation bound requires additional gradient computation on the resampled data. Better ways to integrate the estimation of the information into the training process is worth pursuing. In addition, how to select a proper rank for low-rank adaptation is another research issue where methods like AdaLoRA (Zhang et al., 2023c) and IncreLoRA (Zhang et al., 2023a) propose different parameter importance scores for the rank selection. This issue is not yet considered in this paper.

6 Conclusion

We propose the randomized low-rank adaptation RLoRA which integrates LoRA with the Randomized Walsh-Hadamard Transform to gain parameter-efficiency for fine-tuning and PAC-Bayes regularization to further boost generalization capability. Via comprehensive experiments, we demonstrates that RLoRA can achieve comparable performance as LoRA where the number of trainable parameters required is only 0.02% of the original model (and 1/35 of LoRA). Also, under few-shot settings, the PAC-Bayes regularization makes RLoRA outperforms LoRA and other SOTA low-rank adaptation methods. For future work, more effective and efficient ways for estimating the information-based regularization is worth pursuing. In addition, how to selecting the optimal rank is another open research issue.

[3]We train the RoBERTa-base on RTE datasets for 1 epoch to get this result.

Acknowledgements

This research is partially supported by the Research Matching Grant Scheme RMGS2021_8_06 from the Hong Kong Government.

References

Alessandro Achille, Giovanni Paolini, and Stefano Soatto. 2019. Where is the information in a deep neural network? *arXiv preprint arXiv:1905.12213*.

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

Samyadeep Basu, Xuchen You, and Soheil Feizi. 2020. On second-order group influence functions for black-box predictions. In *International Conference on Machine Learning*, pages 715–724. PMLR.

Krzysztof M Choromanski, Mark Rowland, and Adrian Weller. 2017. The unreasonable effectiveness of structured random orthogonal embeddings. *Advances in neural information processing systems*, 30:219–228.

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*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Gintare Karolina Dziugaite and Daniel M Roy. 2018. Data-dependent pac-bayes priors via differential privacy. *Advances in neural information processing systems*, 31:8430–8441.

Chin-Lun Fu, Zih-Ching Chen, Yun-Ru Lee, and Hung-yi Lee. 2022. Adapterbias: Parameter-efficient token-dependent representation shift for adapters in nlp tasks. *arXiv preprint arXiv:2205.00305*.

Hunter Glanz and Luis Carvalho. 2018. An expectation–maximization algorithm for the matrix normal distribution with an application in remote sensing. *Journal of Multivariate Analysis*, 167:31–48.

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

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

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. 2021. Lora: Low-rank adaptation of large language models. *ArXiv preprint*, abs/2106.09685.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

Soroush Abbasi Koohpayegani, KL Navaneet, Parsa Nooralinejad, Soheil Kolouri, and Hamed Pirsiavash. 2023. Nola: Networks as linear combination of low rank random basis. *arXiv preprint arXiv:2310.02556*.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2024. ELoRA: Efficient low-rank adaptation with random matrices. In *The Twelfth International Conference on Learning Representations*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*.

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 (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Fanghui Liu, Xiaolin Huang, Yudong Chen, and Johan AK Suykens. 2021. Random features for kernel approximation: A survey on algorithms, theory, and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):7128–7148.

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*.

Sanae Lotfi, Marc Finzi, Sanyam Kapoor, Andres Potapczynski, Micah Goldblum, and Andrew G Wilson. 2022. Pac-bayes compression bounds so tight that they can explain generalization. *Advances in Neural Information Processing Systems*, 35:31459–31473.

Sanae Lotfi, Marc Finzi, Yilun Kuang, Tim GJ Rudner, Micah Goldblum, and Andrew Gordon Wilson. 2023. Non-vacuous generalization bounds for large language models. *arXiv preprint arXiv:2312.17173*.

Rabeeh Karimi mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems*.

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*.

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.

Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20.

Simone Rossi, Sebastien Marmin, and Maurizio Filippone. 2020. Walsh-hadamard variational inference for bayesian deep learning. *Advances in Neural Information Processing Systems*, 33:9674–9686.

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.

Joel A Tropp. 2011. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(01n02):115–126.

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

Zifeng Wang, Shao-Lun Huang, Ercan E Kuruoglu, Jimeng Sun, Xi Chen, and Yefeng Zheng. 2021. Pac-bayes information bottleneck. *arXiv preprint arXiv:2109.14509*.

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*.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. 2023a. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv preprint arXiv:2308.12043*.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023b. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023c. Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

A Matrix-variate Posterior Distribution Approximated by WHT

The matrix $\widetilde{\boldsymbol{W}} = \mathrm{diag}(\boldsymbol{\beta})\mathbf{S_1HB_1S_2^T}\mathrm{diag}(\boldsymbol{\alpha})\mathbf{S_2HB_2} = \boldsymbol{W}_B\mathbf{HB_1S_2^T}\mathrm{diag}(\boldsymbol{\alpha})\mathbf{S_2HB_2}$ in Eq. 8, where $\boldsymbol{W}_B = \mathrm{diag}(\boldsymbol{\beta})\boldsymbol{S}_1$. The covariance matrices are:

$$\begin{aligned} \boldsymbol{U} &= \frac{1}{\mathrm{tr}(\boldsymbol{V})}\mathbb{E}\left[(\boldsymbol{W}-\boldsymbol{M})(\boldsymbol{W}-\boldsymbol{M})^T\right] \\ \boldsymbol{V} &= \frac{1}{\mathrm{tr}(\boldsymbol{U})}\mathbb{E}\left[(\boldsymbol{W}-\boldsymbol{M})^T(\boldsymbol{W}-\boldsymbol{M})\right] \end{aligned} \tag{20}$$

As the covariance matrix is non-identifiable (Glanz and Carvalho, 2018) which means that for any scale factor

$$\mathcal{MN}_{m\times n}(\boldsymbol{W} \mid \boldsymbol{M}, \boldsymbol{U}, \boldsymbol{V}) = \mathcal{MN}_{m\times n}(\boldsymbol{W} \mid \boldsymbol{M}, s\boldsymbol{U}, \tfrac{1}{s}\boldsymbol{V}), \tag{21}$$

we can constrain $\mathrm{tr}(\boldsymbol{V}) = 1$, and $\boldsymbol{M} = \boldsymbol{S_1HB_1S_2^T}\mathrm{diag}(\boldsymbol{\mu}_\beta)\boldsymbol{S_2HB_2}$

$$\begin{aligned} \boldsymbol{U} &= \mathbb{E}\left[\boldsymbol{HB_1S_2^T}\mathrm{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}\epsilon)\boldsymbol{S_2HB_2B_2HS_2^T}\mathrm{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}\epsilon)\boldsymbol{S_2B_1H}\right] \\ &= \mathbb{E}\left[\boldsymbol{HB_1S_2^T}\mathrm{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}\epsilon)\boldsymbol{S_2S_2^T}\mathrm{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}\epsilon)\boldsymbol{S_2B_1H}\right] \\ &= \boldsymbol{HB_1S_2^T}\mathbb{E}\left[diag(\boldsymbol{\Sigma}_\alpha)\right]\boldsymbol{S_2B_1H} \\ \boldsymbol{V} &= \frac{1}{\mathrm{tr}(\boldsymbol{U})}\mathbb{E}\left[\boldsymbol{B_2HS_2^T}\mathrm{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}\epsilon)\boldsymbol{S_2B_1HHB_1S_2^T}\mathrm{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}\epsilon)\boldsymbol{S_2HB_2}\right] \\ &= \frac{1}{\mathrm{tr}(\boldsymbol{U})}\boldsymbol{B_2HS_2^T}\mathbb{E}\left[\mathrm{diag}(\boldsymbol{\Sigma}_\alpha)\right]\boldsymbol{S_2HB_2} \end{aligned} \tag{22}$$

Then, the root of $\boldsymbol{U}, \boldsymbol{V}$ can be found

$$\begin{aligned} \boldsymbol{U}^{1/2} &= \boldsymbol{HB_1S_2^T}diag(\boldsymbol{\Sigma}_\alpha^{1/2}) \\ \boldsymbol{V}^{1/2} &= \frac{1}{\sqrt{\mathrm{tr}(\boldsymbol{U})}}\boldsymbol{B_2HS_2^T}\mathrm{diag}(\boldsymbol{\Sigma}_\alpha^{1/2}) \end{aligned} \tag{23}$$

Lemma A.1. *Suppose $\boldsymbol{W} \sim \mathcal{MN}_{m\times n}(\boldsymbol{M}^T, \boldsymbol{V}, \boldsymbol{U})$. Let $\boldsymbol{D} \in \mathbb{R}^{r\times m}$ $r \leq m$ and $\boldsymbol{C} \in \mathbb{R}^{n\times p}, p \leq n$ are both full rank matrices. Then a linear transform of the $\boldsymbol{W}$, i.e., $\boldsymbol{DWC}$ also follows matrix-variate Gaussian distribution.*

$$\boldsymbol{DWC} \sim \mathcal{MN}_{r\times s}(\boldsymbol{DMC}, \boldsymbol{DUD}^T, \boldsymbol{C}^T\boldsymbol{VC}). \tag{24}$$

So, $\boldsymbol{U}_{\widetilde{W}} = \mathrm{diag}(\boldsymbol{\beta})\boldsymbol{S}_1\boldsymbol{U}\boldsymbol{S}_1^T\mathrm{diag}(\boldsymbol{\beta})$ and $\boldsymbol{V}_{\widetilde{W}} = \boldsymbol{V}$.

B PAC-Bayesian Estimation

When U_ψ is bootstrapping from S and $\boldsymbol{\Psi}$ is the selection matrix and $\psi_i = 1$ means sample point i will be removed. $\boldsymbol{\Psi}$ is the resampled data index and we use $\boldsymbol{g} \in \mathbb{R}^{d\times n}$ to denotes the gradient matrix.

$$\begin{aligned} \boldsymbol{\mu}_{S-U} - \boldsymbol{\mu}_S &\triangleq -\frac{1}{1-p}\frac{1}{|S|}\boldsymbol{H}_{\mu_S}^{-1}\sum_{z\in U}\nabla l(h_{\mu_S}(z)) \\ &= \frac{1}{1-p}\frac{1}{|S|}\boldsymbol{H}_{\mu_S}^{-1}\sum_{i=1}^{n}\psi_i\nabla l(h_{\mu_S}(z_i)) \\ \text{so}\ \ \boldsymbol{\mu}_U - \boldsymbol{\mu}_S &\triangleq \frac{1}{1-p}\frac{1}{|S|}\boldsymbol{H}_{\mu_S}^{-1}\sum_{i=1}^{n}(1-\psi_i)\nabla l(h_{\mu_S}(z_i)) \\ &\simeq \boldsymbol{H}_{\mu_S}^{-1}\boldsymbol{g}(\mathbf{1}-\boldsymbol{\Psi}). \end{aligned} \tag{25}$$

Then we have

$$\begin{aligned}
&\mathbb{E}\left[(\boldsymbol{\mu}_U - \boldsymbol{\mu}_S)(\boldsymbol{\mu}_U - \boldsymbol{\mu}_S)^T\right] \\
&= \mathbb{E}\left[\boldsymbol{H}_{\mu_S}^{-1}\boldsymbol{g}(\boldsymbol{1}-\boldsymbol{\Psi})(\boldsymbol{H}_{\mu_S}^{-1}\boldsymbol{g}(\boldsymbol{1}-\boldsymbol{\Psi}))^T\right] \\
&= \boldsymbol{H}_{\mu_S}^{-1}\boldsymbol{g}\mathbb{E}\left[(\boldsymbol{1}-\boldsymbol{\Psi})(\boldsymbol{1}-\boldsymbol{\Psi})^T\right] g^T(\boldsymbol{H}_{\mu_S}^{-1})^T
\end{aligned} \tag{26}$$

while $\boldsymbol{\Psi} \sim$ Poisson$(k, \frac{1}{n}, ..., \frac{1}{n})$,

$$\begin{aligned}
&\mathbb{E}\left[(\boldsymbol{1}-\boldsymbol{\Psi})(\boldsymbol{1}-\boldsymbol{\Psi})^T\right]_{ij} \\
&= \begin{cases} 1 - 2\frac{k}{n} + \frac{k(1-n)}{n} + (\frac{k}{n})^2 & i = j \\ 1 - 2\frac{k}{n} + \frac{k^2}{n^2} & i \neq j \end{cases}
\end{aligned} \tag{27}$$

In practice as n increases, $(\mathbb{E}\left[(\boldsymbol{1}-\boldsymbol{\Psi})(\boldsymbol{1}-\boldsymbol{\Psi})^T\right]_{ii}) \rightarrow 2$ and $(\mathbb{E}\left[(\boldsymbol{1}-\boldsymbol{\Psi})(\boldsymbol{1}-\boldsymbol{\Psi})^T\right]_{ij}) \rightarrow 1$. So

$$\mathbb{E}\left[(\boldsymbol{1}-\boldsymbol{\Psi})(\boldsymbol{1}-\boldsymbol{\Psi})^T\right] \simeq (\boldsymbol{I} + \boldsymbol{1}\boldsymbol{1}^{\boldsymbol{T}}) \tag{28}$$

and Eq. 26 can be rewritten as:

$$\begin{aligned}
&\boldsymbol{H}_{\mu_S}^{-1}\boldsymbol{g}\mathbb{E}\left[(\boldsymbol{1}-\boldsymbol{\Psi})(\boldsymbol{1}-\boldsymbol{\Psi})^T\right] g^T(\boldsymbol{H}_{\mu_S}^{-1})^T \\
&= \boldsymbol{H}_{\mu_S}^{-1}\boldsymbol{g}(\boldsymbol{I} + \boldsymbol{1}\boldsymbol{1}^{\boldsymbol{T}})g^T(\boldsymbol{H}_{\mu_S}^{-1})^T \\
&= \boldsymbol{H}_{\mu_S}^{-1}\boldsymbol{g}\boldsymbol{I}\boldsymbol{g}^T(\boldsymbol{H}_{\mu_S}^{-1})^T \\
&\propto \boldsymbol{H}_{\mu}^{-1}F_{\mu,U}(\boldsymbol{H}_{\mu}^{-1})^T
\end{aligned} \tag{29}$$

C Experiment Details

C.1 GLUE full datasets

The detailed hyperparameter settings of our proposed RLoRA can be found in Table 6.

Table 6: Hyperparameter settings for RLoRA

	Hyperparameter	SST2	MRPC	CoLA	QNLI	RTE	STS-B
BASE	Max sequence length	128					
	Rank r	512					
	Batch size	64	32	32	64	32	32
	Epoch	10	30	30	10	30	30
	RLoRA lr	5E-2	5E-2	1E-3	1E-2	5E-2	1E-2
	Classifier lr	5E-4	1E-3	1E-3	5E-4	1E-3	1E-3
	λ	1E-4	1E-4	1E-4	5E-5	5E-5	1E-4
LARGE	Max sequence length	128					
	Rank r	256					
	Batch size	128	64	64	128	64	64
	Epoch	60	80	80	25	160	80
	RLoRA lr	5E-2	1E-2	1E-2	5E-2	1E-2	5E-2
	Classifier lr	1E-4	5E-5	1E-4	1E-4	5E-5	1E-4
	λ	1E-3	1E-3	1E-4	5E-5	5E-3	1E-4

C.2 GLUE datasets for few-shot experiments

For our few-shot experiments, we make of GLUE and train the model for 20 epoch with batch size = 32 in all experiments. Detailed hyperparameter settings for our proposed RLoRA can be found in Table 7.

C.3 E2E benchmark

The hyperparameter to reproduce the results for the E2E benchmark is shown in Table 8.

Table 7: Hyperparameter settings for RLoRA in few-shot experiments

	# of train data	Method	RTE	MRPC	STS-B	SST-2	QNLI
BASE	100	RLoRA lr	5E-4	1E-3	5E-3	1E-2	1E-2
		Classifier lr	5E-4	1E-2	1E-2	1E-3	1E-2
		RLoRA λ	1E-4	1E-4	5E-3	1E-4	1E-2
		Classifier λ	1E-4	1E-4	1E-2	1E-4	1E-2
	200	RLoRA lr	1E-4	5E-2	5E-2	5E-2	5E-2
		Classifier lr	5E-2	5E-4	5E-3	5E-3	1E-4
		RLoRA λ	5E-3	1E-3	5E-4	5E-3	1E-2
		Classifier λ	0	5E-3	5E-4	1E-4	5E-4

Table 8: Hyperparameter configurations for RLoRA, VeRA and LoRA on the E2E benchmark.

Hyperparameter	RLoRA	VeRA	LoRA
# GPUs		1	
Optimizer		AdamW	
Learning Rate Schedule		Linear	
Batch Size		8	
Epochs		5	
Warmup Steps		500	
Label Smooth		0.1	
LoRA α		32	
regularization λ	1E-6	0	0
Rank	1024	1024	8
Learning Rate	1E-1	1E-1	5E-4