Literature Review on Advanced LoRA methods

Low-rank Adaptation (LoRA) Problem:

The problem with LoRA is the optimal selection of ranks: Setting a huge rank wastes training time and resources, while a small one can degrade performance. Therefore, dynamic adjustment of ranks is necessary.

Several studies have attempted to improve the performance of LoRA. In this report, we will review these studies and their approaches. Specifically, we will look at:

1. SoRA
2. ALORA
3. AdaLoRA
4. SaLoRA

SoRA: Sparse Low-rank Adaptation

Key Idea:

SoRA dynamically adjusts the intrinsic rank during training using a sparse gating unit optimized via the proximal gradient method. It builds on the low-rank decomposition framework for its effectiveness and parameter efficiency.

Module Structure:

- Pre-define a maximum acceptable rank $r_{\max}$ based on practical or research needs.
- Each SoRA module inherits two matrices from LoRA:

$\mathbf{W}_d \in \mathbb{R}^{r_{\max} \times q}$ for down projection and $\mathbf{W}_u \in \mathbb{R}^{p \times r_{\max}}$ for up projection.

- Introduce a gating unit $\mathbf{g} \in \mathbb{R}^{r_{\max}}$ between projection matrices to control the effective rank.
- Forward propagation in SoRA:

$$\mathbf{h} \xleftarrow{\text{down projection}} \mathbf{W}_d \mathbf{x};$$

$$\mathbf{h}' \xleftarrow{\text{gating}} \mathbf{g} \odot \mathbf{h};$$

$$\mathbf{z} \xleftarrow{\text{up projection}} \mathbf{W}_u \mathbf{h}';$$

Optimization:

- Optimize Wd and Wu with stochastic gradient methods
- Update the gating unit g using a sparsity-promoting method:

$$\mathbf{g}_{t+1} \leftarrow \mathcal{T}_{\eta_t \cdot \lambda}(\mathbf{g}_t - \eta_t \nabla_{\mathbf{g}} \mathcal{L}_0(\mathbf{\Delta}_t)),$$

Post-pruning:

- After training, prune SoRA weights to eliminate zeroed-out ranks, reducing the module to the LoRA form.
- For the k-th SoRA module:
    - Idensity zero entries in the gating vector g(k)
    - Drop corresponding rows in Wd(k) and columns in Wu(k)
    - Adjust the gate vector.
- The pruned module operates as a standard LoRA module with reduced rank during inference.

Reference:

3.1 Sparse Low-rank Adaptation

Module Structure. At the start of building a SoRA module, we pre-define a maximum acceptable rank $r_{\max}$ according to practical or research concerns. Then, each SoRA module will inherit two matrices $\mathbf{W}_d \in \mathbb{R}^{r_{\max} \times q}$ and $\mathbf{W}_u \in \mathbb{R}^{p \times r_{\max}}$ from LoRA for down projection and up projection. The maximum rank $r_{\max}$ is set to be relatively large, but we will show in the subsequent paragraph how to tame it efficiently in a sparse sense. In fact, this is realized by injecting a gating unit $\mathbf{g} \in \mathbb{R}^{r_{\max}}$ between the projection matrices, which imitates the formulation of SVD. The forward propagation of the SoRA module proceeds as follows:

$$\mathbf{h} \xleftarrow{\text{down projection}} \mathbf{W}_d \mathbf{x}; \tag{6}$$

$$\mathbf{h}' \xleftarrow{\text{gating}} \mathbf{g} \odot \mathbf{h}; \tag{7}$$

$$\mathbf{z} \xleftarrow{\text{up projection}} \mathbf{W}_u \mathbf{h}'; \tag{8}$$

or, more compactly,

$$\mathbf{z} \leftarrow \mathbf{W}_u \left(\mathbf{g} \odot \left(\mathbf{W}_d \mathbf{x} \right) \right). \tag{9}$$

Optimization. We optimize down-projection and up-projection matrices with stochastic gradient

methods as in LoRA, while each gate $\mathbf{g}$ is updated in a different sparsity-promoting way:

$$\mathbf{g}_{t+1} \leftarrow \mathcal{T}_{\eta_t \cdot \lambda}(\mathbf{g}_t - \eta_t \nabla_{\mathbf{g}} \mathcal{L}_0(\mathbf{\Delta}_t)), \tag{10}$$

in which $\mathcal{L}_0(\cdot)$ is the original loss function of the language model, $\mathbf{\Delta}$ denotes the complete tunable parameter (including the gates), $\eta_t > 0$ stands for the step-size at the t-th iteration, and $\lambda > 0$ works as the regularization strength hyperparameter that promotes sparsity. Besides, $\mathcal{T}_{\eta_t \cdot \lambda}(\cdot)$ in the above expression stands for the element-wise broadcast of the following soft-thresholding function:

$$\mathcal{T}_\xi(x) := \begin{cases} x - \xi, & x > \xi \\ 0, & -\xi < x \leq \xi \\ x + \xi, & x \leq -\xi \end{cases} \tag{11}$$

with $\xi = \eta_t \cdot \lambda$ being the threshold. In practice, the true gradient $\nabla_{\mathbf{g}} \mathcal{L}_0$ in (10) is approximated by its mini-batch stochastic counterpart.

Post-pruning. When training is completed, we further prune the SoRA weights to drop the zeroed-out ranks and reduce the module back to the LoRA form. To be specific, for the k-th SoRA module, let

$$\mathcal{I}^{(k)} = \left\{ i \in [1 : r_{\text{max}}] \mid \mathbf{g}_i^{(k)} = 0 \right\} \tag{12}$$

be the index of zero entry in the k-th gating vector $\mathbf{g}^{(k)}$. We drop the $\mathcal{I}^{(k)}$-th rows of down-projection $\mathbf{W}_d^{(k)}$ to obtain $\widetilde{\mathbf{W}}_d^{(k)}$, the $\mathcal{I}^{(k)}$-th columns of up-projection $\mathbf{W}_u^{(k)}$ to obtain $\widetilde{\mathbf{W}}_u^{(k)}$, as well as the $\mathcal{I}^{(k)}$-th entry of gate $\mathbf{g}^{(k)}$ to obtain $\widetilde{\mathbf{g}}^{(k)}$. In this way, during inference time the k-th SoRA module will proceed as a usual LoRA module of rank $r_{\text{max}} - |\mathcal{I}^{(k)}|$ with down-projection matrix $\widetilde{\mathbf{W}}_d^{(k)}$ and up-projection matrix $\widetilde{\mathbf{W}}_u^{(k)} \cdot \text{diag}(\widetilde{\mathbf{g}}^{(k)})$.

Limitations of SoRA:

1. **Limited Evaluation:** Tested only on traditional NLP tasks, not on cross-modal or instruction-tuning scenarios.
2. **Unknown Sparsity Behavior:** Effectiveness of sparsity in new domains is unclear.
3. **Scheduler Complexity:** Difficult to explain and assess the sparsifying scheduler's adaptation process.
4. **Lack of Theoretical Explanation:** No thorough theoretical understanding of the sparsifying process and its impact.

Reference:

Limitations

Despite the encouraging results demonstrated by SoRA, there are certain limitations in our current study that are worth acknowledging. This paper only evaluates the effectiveness of SoRA on traditional natural language processing tasks. However, recent studies demonstrate that parameter-efficient methods could be applied to cross-modal or instruction-tuning scenarios. In those cases, how the sparsity of SoRA is displayed is still unknown and worth investigating. Our sparsifying scheduler could provide insights on the adaptation process of language models, but it is still challenging to rigorously explain the procedure and more efficiently to assess the difficulty of an adaptation process.

ALoRA: Allocating Low-Rank Adaptation

Key idea:

Unlike traditional LoRA, which uses a fixed intrinsic rank, ALoRA allows for dynamic adjustments to the rank during the adaptation process.

The methodology involves two main steps:

1. AB-LoRA: A novel method that estimates the importance score of each LoRA rank.
2. Rank Pruning and Allocation: Guided by the scores from AB-LoRA, the method prunes less important and negatively impacting ranks and reallocates these resources to more critical Transformer modules that require higher ranks.

AB-LoRA:

The key idea is to calculate the importance score of each LoRA rank by evaluating its contribution to the performance of the super-network, rather than relying on architecture weights.

The super-network refers to the entire neural network configuration that includes all possible LoRA ranks enabled by the gate units, meaning all potential ranks are considered active and contributing to the model.

Approach:

1. Super-network training: Train the complete super-network M until convergence on the training set.
2. Modified super-networks: Create a modified super-network $M_{\setminus r}$ by zeroing out a single LoRA rank while keeping all other ranks. Create another modified super-network M_r where only the LoRA rank is kept, and all other ranks are zeroed out.
3. Importance score calculation: Evaluate all versions of the super-network on the same validation data batch.
   Importance score of LoRA rank:

$$\mathrm{IS}(r) = S(M) - S(M_{\setminus r}) + S(M_r).$$

Intuition:

- A significant performance drop when a rank is zeroed out indicates its importance.
- A rank that maintains most of the performance when acting alone is also considered important.

ALoRA:

Dynamically adjust the LoRA ranks in a Transformer model to optimize performance based on the importance scores calculated using the AB-LoRA method.

Approach:

1. Train the super-network on the training set for K1 epochs with all LoRA ranks enabled. Freeze the architectural parameters, training only the model parameters.
2. Importance Score Evaluation: For each LoRA rank, calculate its importance score using a batch of samples from the development set.
3. Pruning and Allocation: Prune the nA LoRA ranks with the lowest importance scores by zeroing out their corresponding gate units. Allocate the parameter budgets from pruned ranks to Transformer modules that have not had ranks pruned and are deemed important based on high importance scores.
4. Performance Recovery: Fine-tune the altered super-network for K2 epochs to recover any performance loss due to pruning.
5. Repeat the above steps for NA iterations to gradually optimize the LoRA rank allocation without significant performance degradation.

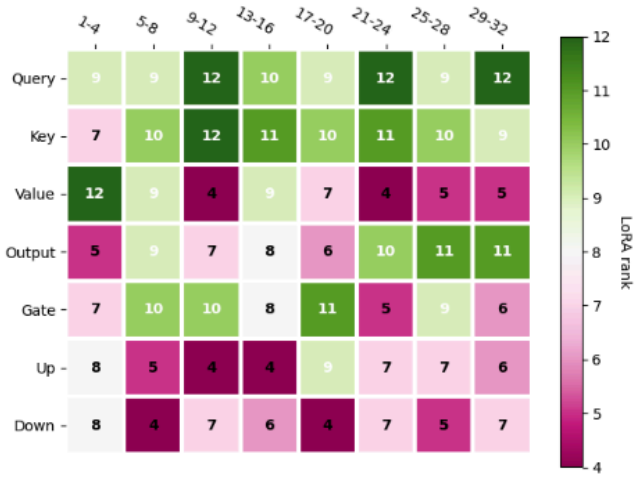

Figure 3: The final rank allocations of ALoRA after fine-tuning the LlaMA-2 7B model on the E2E task.

References:

3.3 Our novel AB-LoRA method

Under the DNAS setting, it is natural to consider the architecture weight $\alpha_i^{'}$ as the importance score for LoRA rank i, and one can use these scores to guide the pruning of abundant LoRA ranks. However, as pointed out by the literature (Zhang et al., 2023f; Chen et al., 2019), and as will be demonstrated in the experiment, the architecture weights are not reliable indicators for the final LoRA allocation's performance. This observation motivates us to propose a simple yet effective modification to the DNAS-style architecture search. Instead of relying on the architecture weights' values to keep the best LoRA ranks, we propose directly evaluating the LoRA rank's superiority by its contribution or influence on the super-network's performances. Since our method mimics conducting ablation studies of a certain LoRA rank from the super-network, we refer to our method as the ablation-based LoRA (AB-LoRA).

We now introduce the core of our AB-LoRA method: calculating each LoRA rank's importance score, defined as how much it contributes to the performance of the super-network. Denote the complete super-network as M. Super-network M is

trained till convergence on the training set. We now consider a modified super-network obtained by zeroing out a single LoRA rank r while keeping all other LoRA ranks. This new super-network is denoted as $M_{\setminus r}$. We also consider another modified super-network M_r in which only LoRA rank r is kept while all other LoRA ranks are zeroed out. We evaluate the three versions of super-networks on the same batch of validation data B_{val}. Denote the metric score as a function of a model M, $S(M)$, with the validation data fixed. Then, the importance score of LoRA rank r is given by

$$\text{IS}(r) = S(M) - S(M_{\setminus r}) + S(M_r). \quad (6)$$

In the above equation, $S(M)$ can be treated as a constant term. Thus the above equation can be simplified to $\text{CS}(o) = -S(M_{\setminus r}) + S(M_r)$. Intuitively, the LoRA rank that results in a significant performance drop upon zeroing out must play an important role in the super-network. Similarly, the one keeping most of the performance when acting alone contains important task-related knowledge and should be considered important. In the experiments, different from Chen and Hsieh (2020), we set $S()$ as the negative of the cross-entropy (CE) loss since the widely applied metrics like accuracy or F1: (a) may not vary if the super-network only

masks out a single operation, and (b) is not suitable for generative language model fine-tuning.

3.4 The complete process of ALoRA

With the guidance of the importance score in Equation 6, we can now formally define the whole working process of our ALoRA framework (Figure 1). Our working flow of allocating the LoRA ranks builds upon the following intuitions: (a) the pruning and allocation of LoRA ranks is conducted gradually to avoid performance degradation. (b) if the LoRA ranks in a Transformer module receive relatively high importance scores and are not pruned, this module is deemed important. It may need more LoRA ranks for adaptation so that the LoRA parameters can better learn the task knowledge.

The framework of ALoRA is centered on our AB-LoRA method, which requires the super-network to be trained for K_1 epochs on the train set. We freeze the architectural parameters and train only the model parameters on the train set. No bi-level optimization is required, thus saving training time costs. Then, for each LoRA rank, we evaluate the importance score on a batch of samples B_{val} from the development set. Then, n_A LoRA ranks with the lowest scores are pruned by zeroing out their corresponding gate units. Moreover, if some Transformer modules do not have pruned LoRA ranks, we allocate the parameter budgets to them to enhance the adaptation further. [23] After the pruning and adding operations, we tune the altered super-network for $K_2 > 0$ epochs to recover the lost performance. The above steps are repeated for N_A times. Formally, we summarize the above process in Algorithm 1.

Limitations:

AdaLoRA: Adaptive Low-Rank Adaptation

Key Idea:

AdaLoRA (Adaptive Low-Rank Adaptation) dynamically allocates the parameter budget among weight matrices during fine-tuning based on their importance. Instead of uniformly distributing incremental updates, AdaLoRA uses singular value decomposition (SVD) to parameterize these updates. This approach adjusts the rank of incremental matrices, assigning higher ranks to critical matrices to capture fine-grained, task-specific information, and pruning the singular values of less important ones to lower ranks, preventing overfitting and saving computational resources.

Approach:

2 Key Components:

1. Formulates the incremental updates of pre-trained weight matrices using singular value decomposition (SVD).
2. Prunes redundant singular values based on a newly-designed importance metric.

SVD-based Adaptation:

1. The pre-trained weight matrices are updated incrementally using SVD. This involves decomposing the updates into left and right singular vectors and a diagonal matrix of singular values.
2. The diagonal matrix is initialized to zero, while the singular vectors adopt a random Gaussian initialization, ensuring the updates start from zero.
3. A regularizer is used to enforce the orthogonality of the singular vectors, maintaining their proper form.
4. Instead of computing SVD for every update, which is computationally expensive, the parameterization used by AdaLoRA avoids intensive SVD computation, reducing overhead significantly.

Importance-aware Rank Allocation:

1. SVD-based adaptation is applied to every weight matrix, including those in transformer layers.

2. Singular values are pruned iteratively based on their importance score during training.

Importance Metric:

Magnitude-based: The simplest way to measure importance is by the magnitude of singular values. However, this might not fully capture the parameters' contribution to model performance.

Sensitivity-based: Another approach is to use the sensitivity of parameters to the training loss. This method involves combining the sensitivity of individual parameters within a triplet to quantify its overall importance.

Smoothed Sensitivity: To reduce the variability and uncertainty of importance estimation, smoothed sensitivity and uncertainty quantification are used.

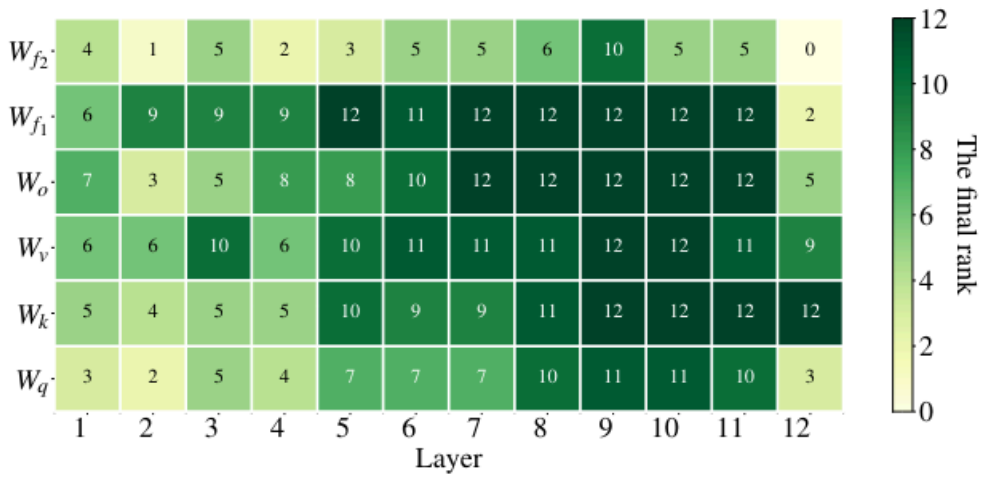

Figure 3: The resulting rank of each incremental matrix when fine-tuning DeBERTaV3-base on MNLI with AdaLoRA. Here the x-axis is the layer index and the y-axis represents different types of adapted weight matrices.

Reference:

3 AdaLoRA Method

Our method contains two important components: (i) SVD-based adaptation, which formulates the incremental matrices in the form of singular value decomposition; (ii) Importance-aware rank allocation, which prunes redundant singular values based on our newly-designed importance metric.

3.1 SVD-Based Adaptation

As mentioned in Section 1, we propose to parameterize the incremental updates of the pre-trained weight matrices in the form of singular value decomposition:

$$W = W^{(0)} + \Delta = W^{(0)} + P\Lambda Q, \tag{3}$$

where $P \in \mathbb{R}^{d_1 \times r}$ and $Q \in \mathbb{R}^{r \times d_2}$ represent the left/right singular vectors of Δ and the diagonal matrix $\Lambda \in \mathbb{R}^{r \times r}$ contains the singular values $\{\lambda_i\}_{1 \le i \le r}$ with $r \ll \min(d_1, d_2)$. We further denote $\mathcal{G}_i = \{P_{*i}, \lambda_i, Q_{i*}\}$ as the triplet containing the i-th singular value and vectors. In practice, since Λ is diagonal, we only need to save it as a vector in $\mathbb{R}^r$. Λ is initialized with zero while P and Q adopt a random Gaussian initialization to ensure $\Delta = 0$ at the beginning of training. To enforce the orthogonality of P and Q, i.e., $P^\top P = QQ^\top = I$, we utilize the following regularizer[2]:

$$R(P, Q) = \|P^\top P - I\|_\mathsf{F}^2 + \|QQ^\top - I\|_\mathsf{F}^2. \tag{4}$$

In our method, Λ is iteratively pruned to adjust the rank after each gradient decent step. As mentioned in Section 1, one can directly compute SVD for every Δ to manipulate singular values. The computational complexity, however, is $O(\min(d_1, d_2)d_1d_2)$. It becomes extremely expensive to iteratively apply SVD for a large number of high-dimensional incremental matrices. In contrast, our parameterization avoids intensive SVD computation, greatly releasing the computational overhead.

We remark that one can also apply structured pruning to LoRA to control the rank (i.e., prune BA doublet-wise in (1)), whereas it has the following disadvantages. First, when a doublet is measured as unimportant, we have to prune all of its elements. It makes scarcely possible to reactivate the pruned doublets as their entries are all zeroed out and not trained. In contrast, AdaLoRA only masks out the singular values based on (3) while the singular vectors are always maintained. It preserves the potential of future recovery for the triplets dropped by mistake. Second, A and B of LoRA are not orthogonal, meaning the doublets can be dependent with each other. Discarding the doublets can incur larger variation from the original matrix than truncating the smallest singular values. Therefore, the incremental matrices are often altered dramatically after each step of rank allocation, which causes training instability and even hurts generalization. To demonstrate this point, we present an ablation study in Section 4.4, which compares AdaLoRA with structured pruning for LoRA.

3.2 IMPORTANCE-AWARE RANK ALLOCATION

We apply the SVD-based adaptation (3) to every weight matrix including W_q, W_k, W_v, W_{f_1} and W_{f_2} of each transformer layer. In order to control the budget, we iteratively prune singular values in correspondence to their importance score during the training. For clear reference, we use k to index the incremental matrix, i.e., $\Delta_k = P_k \Lambda_k Q_k$ for $k = 1, \dots, n$, where n is the number of adapted weight matrices. We denote the i-th triplet of Δ_k as $\mathcal{G}_{k,i} = \{P_{k,*i}, \lambda_{k,i}, Q_{k,i*}\}$ and its importance score as $S_{k,i}$. We further denote the parameter sets $\mathcal{P} = \{P_k\}_{k=1}^n$, $\mathcal{E} = \{\Lambda_k\}_{k=1}^n$, $\mathcal{Q} = \{Q_k\}_{k=1}^n$ and training cost as $\mathcal{C}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$. With the regularization (4), the training objective is given by $\mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q}) = \mathcal{C}(\mathcal{P}, \mathcal{E}, \mathcal{Q}) + \gamma \sum_{k=1}^n R(P_k, Q_k)$, where $\gamma > 0$ is the regularization coefficient. At the t-th step, we first take a stochastic gradient step to update $P_k^{(t)}$, $\Lambda_k^{(t)}$ and $Q_k^{(t)}$ for $k = 1, \dots, n$. Specifically, for $\Lambda_k^{(t)}$

$$\tilde{\Lambda}_k^{(t)} = \Lambda_k^{(t)} - \eta \nabla_{\Lambda_k} \mathcal{L}(\mathcal{P}^{(t)}, \mathcal{E}^{(t)}, \mathcal{Q}^{(t)}), \tag{5}$$

where $\eta > 0$ is learning rate. Then, given importance score $S_k^{(t)}$, the singular values are pruned following

$$\Lambda_k^{(t+1)} = \mathcal{T}(\tilde{\Lambda}_k^{(t)}, S_k^{(t)}), \text{ with } \mathcal{T}(\tilde{\Lambda}_k^{(t)}, S_k^{(t)})_{ii} = \begin{cases} \tilde{\Lambda}_{k,ii}^{(t)} & S_{k,i}^{(t)} \text{ is in the top-} b^{(t)} \text{ of } S^{(t)}, \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $S^{(t)} = \{S_{k,i}^{(t)}\}_{1 \le k \le n, 1 \le i \le r}$ contains the importance score of all triplets. Here $b^{(t)}$ is the budget of remaining singular values at the t-th step, which we explain more in Section 3.3. In this way, we leave more budget to the incremental matrices of higher priority by pruning the singular values of less important ones. In the sequel, we introduce several options to design the importance score.

Magnitude of singular values is the most straightforward way to quantify the importance of every triplet, i.e., $S_{k,i} = |\lambda_{k,i}|$. In this way, only the least significant singular values are discarded. It minimizes the deviation from the original matrix and further stabilizes the training. Many existing methods use this criterion to control the rank of matrix (Cai et al., 2010; Koltchinskii et al., 2011; Toh & Yun, 2010). However, we remark that such a simple metric cannot properly quantify the contribution of parameters to model performance.

Sensitivity-based importance is another option for importance scoring, which quantifies the sensitivity of parameters to the training loss (Molchanov et al., 2019; Sanh et al., 2020; Liang et al., 2021; Zhang et al., 2022). The prior work, however, leverages the sensitivity to quantify the importance of single entries and applies it for unstructured pruning that prunes weights element-wise. When it turns to our case, we have to design a new metric as the triplets are discarded group-wise. Every entry's sensitivity ought to be considered and properly combined to quantify the overall contribution of the triplet to model performance. Therefore, we propose a newly-designed importance metric in account of both the singular value and vectors in triplet $\mathcal{G}_{k,i}$:

$$S_{k,i} = s(\lambda_{k,i}) + \frac{1}{d_1}\sum_{j=1}^{d_1} s(P_{k,ji}) + \frac{1}{d_2}\sum_{j=1}^{d_2} s(Q_{k,ij}), \tag{7}$$

where we calculate the mean importance of $P_{k,*i}$ and $Q_{k,i*}$ such that $S_{k,i}$ does not scale with the number of parameters in $\mathcal{G}_{k,i}$. Here $s(\cdot)$ is a specific importance function for single entries. We can adopt the sensitivity for $s(\cdot)$, which is defined as the magnitude of the gradient-weight product:

$$I(w_{ij}) = |w_{ij} \nabla_{w_{ij}} \mathcal{L}|, \tag{8}$$

where w_{ij} is any trainable parameter. (8) essentially approximates the change in loss when a parameter is zeroed out. If the removal of a parameter has a large influence, then the model is sensitive to it and we should retain it (Molchanov et al., 2019; Liang et al., 2021; Zhang et al., 2022).

However, Zhang et al. (2022) point out that the sensitivity in (8) is not yet a reliable importance indicator. Such a score is estimated on the sampled mini batch. The stochastic sampling and complicated training dynamics incur high variability and large uncertainty for estimating the sensitivity with (8). Therefore, Zhang et al. (2022) propose to resolve this issue by sensitivity smoothing and uncertainty quantification:

$$\overline{I}^{(t)}(w_{ij}) = \beta_1 \overline{I}^{(t-1)}(w_{ij}) + (1-\beta_1) I^{(t)}(w_{ij}) \tag{9}$$

$$\overline{U}^{(t)}(w_{ij}) = \beta_2 \overline{U}^{(t-1)}(w_{ij}) + (1-\beta_2) \left| I^{(t)}(w_{ij}) - \overline{I}^{(t)}(w_{ij}) \right|, \tag{10}$$

where $0 < \beta_1, \beta_2 < 1$. $\overline{I}^{(t)}$ is the smoothed sensitivity by exponential moving average and $\overline{U}^{(t)}$ is the uncertainty term quantified by the local variation between $I^{(t)}$ and $\overline{I}^{(t)}$. Then they define the importance as the product between $\overline{I}^{(t)}$ and $\overline{U}^{(t)}$, which can be another option for $s(\cdot)$:

$$s^{(t)}(w_{ij}) = \overline{I}^{(t)}(w_{ij}) \cdot \overline{U}^{(t)}(w_{ij}). \tag{11}$$

We present a detailed ablation study in Section 4.4 to compare the performance of different importance metrics. We find the proposed metric (7) based on the sensitivity variant (11) generally performs best. We summarize the detailed algorithm in Algorithm 1.

Limitations:

Heuristic Importance Score:

1. The sparsity selection criterion in AdaLoRA relies on a newly proposed importance score based on the moving average of the weight-gradient product.
2. This approach is largely heuristic and lacks a strong theoretical foundation.

Additional Computation Cost:

1. The moving average operation of importance scores adds to the computational burden.
2. The gradients of the orthogonality regularization also contribute to the increased computational cost, making the training process more resource-intensive.

In spite of the effectiveness demonstrated through experiments, there are still two problems in AdaLoRA that demand rethinking of the methodology and wait for further improvements. First, the

sparsity selection criterion in AdaLoRA is based on their newly proposed importance score relied on the moving average of weight-gradient product. Despite its effectiveness in empirical study, this criterion is largely heuristic, lacking theoretical motivation. Second, both the moving average operation of importance scores and the gradients of orthogonality regularization (5) add up to additional computation cost. Compared to AdaLoRA with the aforementioned limitations, our approach, SoRA, serves as an amelioration with highly simplified updating rules and is backed up by the theory of sparsity regularization and proximal gradient methods. Detailed methodology of SoRA will be elaborated in the next section.

SaLoRA

Key idea:

SaLoRA dynamically learns the intrinsic rank of each incremental matrix by selectively removing non-critical components using a binary gate mechanism and a differentiable relaxation method. This approach assigns higher ranks to crucial matrices to capture task-specific information while pruning less significant ones to prevent overfitting. To enhance training stability and generalization, orthogonality regularization is applied to the factor matrices.

Approach:

1. Intrinsic Rank Adaptation: Introduces a gate matrix to control the rank of each incremental matrix dynamically. Uses a binary gate mechanism to identify and remove non-critical components, ensuring that only the most essential parts of the matrix are retained.
2. L0 Norm Regularization: Applies L0 norm regularization to the gate matrix to encourage the deactivation of less important components. Uses a differentiable relaxation method to make the optimization computationally feasible.
3. Orthogonal Regularization: Ensures training stability and better generalization by applying orthogonal regularization to the factor matrices, preventing significant deviations from the original matrix during pruning.
4. Controlled Budget Using Lagrangian Relaxation: Introduces a density constraint to control the overall parameter budget of the model. Utilizes Lagrangian relaxation to balance the optimization of the parameter budget with maintaining model performance.
5. Optimization Strategy: Combines gradient-based optimization with a min-max game to ensure the model meets the desired parameter budget while maintaining performance and stability.

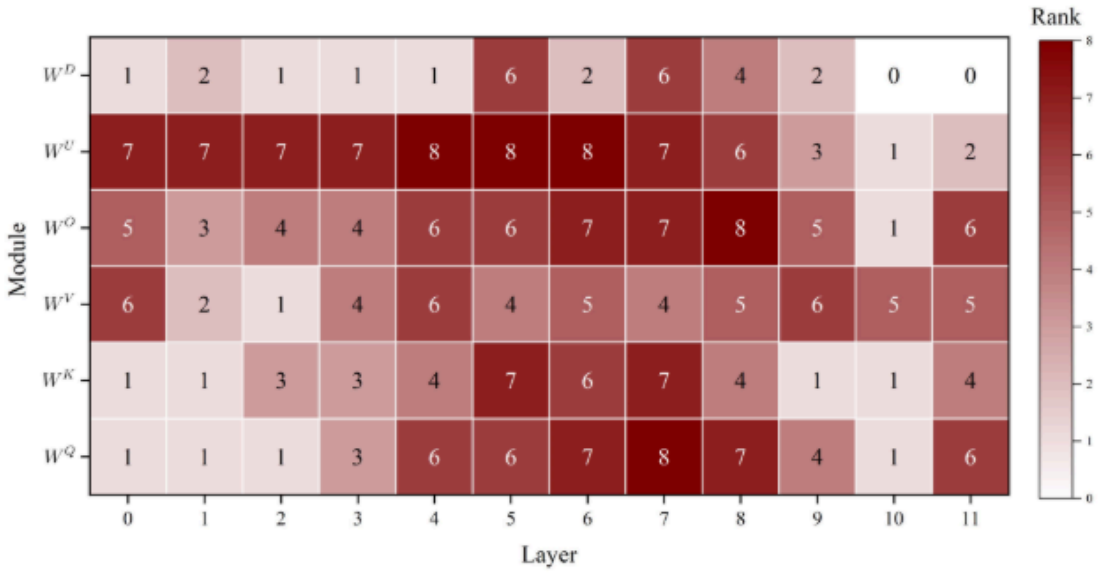

Figure 7. The resulting rank of each incremental matrix obtained from fine-tuning RoBERTa-base on MRPC with SaLoRA. The initial rank is set at 8, and the target sparsity is 0.5. The x-axis is the layer index and the y-axis represents different types of modules.

Reference:

3. Method

In this section, we will first give a brief introduction to parameter-efficient fine-tuning, and then discuss our proposed model based on the problem definition.

3.1. Problem Formalization

We consider the general problem of efficiently fine-tuning LLMs for specific downstream tasks. Firstly, let us introduce some notations. Consider a training corpus $\mathcal{D} = (x_i, y_i)_{i=1}^{N}$, where N represents the number of samples. Each sample consists an input, x_i, and its corresponding output, y_i. We use the index i to refer to the incremental matrix, i.e., $\Delta \boldsymbol{W}_i = \boldsymbol{B}_i \boldsymbol{A}_i$ for $i = 1, \ldots, K$, where K is the number of incremental matrices. However, LoRA's assumption of identical ranks for each incremental matrix overlooks structural relationships and the varying importance of weight matrices across different modules and layers during fine-tuning. This oversight can potentially impact overall model performance. Our objective is to determine the optimal $\{rank^*(\Delta \boldsymbol{W}_i)\}_{i=1}^{K}$ on the fly. The optimization objective can be formulated as follows:

$$\begin{aligned} &\min_{\mathcal{W}} \mathcal{R}(\mathcal{W}) \triangleq \frac{1}{N}(\sum_{i=1}^{N} \mathcal{L}(f(x_i; \mathcal{W}), y_i)) \\ &s.t. \quad rank(\Delta \boldsymbol{W}_i) \leq r, k = 1, \ldots, K. \end{aligned} \tag{4}$$

where $\mathcal{W} = \{\Delta \boldsymbol{W}_i, \ldots, \Delta \boldsymbol{W}_K\}$ represents the sets of trainable parameters and $\mathcal{L}$ corresponds to a loss function, such as cross-entropy for classification. Note that $rank(\Delta \boldsymbol{W}_i) \in \{0, 1, \ldots, r\}$ is an unknown parameter that needs to be optimized.

3.2. Structure-Aware Intrinsic Rank Using L_0 Norm

To find the optimal $\{rank^*(\Delta \boldsymbol{W}_i)\}_{i=1}^{K}$ on the fly, with minimal computational overhead during training, we introduce a gate matrix $\boldsymbol{G}$ to define the structure-aware intrinsic rank:

$$\Delta \boldsymbol{W} = \boldsymbol{B}\boldsymbol{G}\boldsymbol{A} = \sum_{j=1}^{r} g_j \boldsymbol{B}_{*j} \boldsymbol{A}_{j*} \tag{5}$$

where the $g_j \in \{0, 1\}$ serves as a binary "gate", indicating the presence or absence of the j-th rank. The gate matrix $\boldsymbol{G} = \text{diag}(g_1, \ldots, g_r)$ is a diagonal matrix consisting of the pruning variables. By learning the variable g_j, we can control the rank of each incremental matrix individually, rather than applying the same rank to all matrices. To deactivate non-critical rank-0 components, the ideal approach would be to apply L_0 norm regularization to the gate matrix $\boldsymbol{G}$:

$$||\boldsymbol{G}||_0 = \sum_{j=1}^{r} g_j \tag{6}$$

where r is the rank of incremental matrices. The L_0 norm measures the number of non-zero triplets; thus, optimizing L_0 would encourage the model to deactivate less important incremental matrices.

Unfortunately, the optimization objective involving $||\boldsymbol{G}||_0$ is computationally intractable due to its non-differentiability, making it impossible to directly incorporate it as a regularization term in the objective function. Instead, we use a stochastic relaxation approach, where the gate variables g are treated as continuous variables distributed within the interval $[0, 1]$. We leverage the reparameterization trick [30,31] to ensure that g remains differentiable. Following prior studies [17,19], we adopt the Hard-Concrete (HC) distribution as a continuous surrogate for random variables g, illustrated in Figure 3. The HC distribution applies a hard-sigmoid rectification to s, which can easily be sampled by first sampling $u \in U(0, 1)$ and then computing as follows:

$$\begin{aligned} s &= Sigmod\left(\frac{\log \frac{u}{1-u} + \log \theta}{\tau}\right) \times (\zeta - \gamma) + \gamma \\ g &= \min(1, \max(0, s)) \end{aligned} \tag{7}$$

where θ is the trainable parameter of the distribution and τ is the temperature. The interval (γ, ζ), with $\gamma < 0$ and $\zeta > 1$, enables the distribution to concentrate probability mass at the edge of the support. The final outputs g are rectified into $[0, 1]$. By summing up the probabilities of the gates being non-zero, the L_0 norm regularization can be computed via a closed form, as follows:

$$\begin{aligned} \mathbb{E}[||\boldsymbol{G}||_0] &= \sum_{j=1}^{r} \mathbb{E}\left[g_j > 0\right] \\ &= \sum_{j=1}^{r} Sigmod\left(\log \theta_j - \tau \log \frac{-\gamma}{\zeta}\right) \end{aligned} \tag{8}$$

As g now represents the output of the parameterized HC distribution function and serves as an intermediate representation for the neural network, gradient-based optimization methods can perform gradient updates for $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_r\}$. For each training batch, we sample the gate mask and then share it across the training examples within the batch to enhance sampling efficiency.

3.3. Enhanced Stability Using Orthogonal Regularization

In deep networks, orthogonality plays a crucial role in preserving the norm of the original matrix during multiplication, preventing signal vanishing or exploding [32]. However, in LoRA, where $\boldsymbol{B}$ and $\boldsymbol{A}$ are not orthogonal, the dependence can lead to larger variations when removing certain columns or rows through L_0 regularization. This, in turn, leads to training instability and the potential for negative effects on generalization [16]. For this, we turn to orthogonal regularization, which enforces the orthogonality condition:

$$\mathcal{R}_{orth}(\boldsymbol{B}, \boldsymbol{A}) = ||\boldsymbol{B}^T\boldsymbol{B} - \boldsymbol{I}||_F^2 + ||\boldsymbol{A}\boldsymbol{A}^T - \boldsymbol{I}||_F^2 \tag{9}$$

where $\boldsymbol{I}$ is the identity matrix.

Now, let us substitute Equations (8) and (9) into Equation (4) to derive the new training objective:

$$\min_{\boldsymbol{\mathcal{W}},\boldsymbol{\Theta}} \mathcal{R}(\boldsymbol{\mathcal{W}}, \boldsymbol{\Theta}) \triangleq \frac{1}{N}(\sum_{i=1}^{N} \mathcal{L}(f(x_i; \boldsymbol{\mathcal{W}}), y_i)) + \lambda \sum_{i=1}^{K} \mathbb{E}[||\boldsymbol{G}_i||_0] + \beta \sum_{i=1}^{K} \mathcal{R}_{orth}(\boldsymbol{B}_i, \boldsymbol{A}_i) \tag{10}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i, \ldots, \boldsymbol{\theta}_K\}$ represents the sets of trainable parameters, and λ and β are two constant hyperparameters.

3.4. Controlled Budget Using Lagrangian Relaxation

If we only rely on Equation (10) to learn the intrinsic rank for each incremental matrix, the resulting parameter budget cannot be directly controlled. This limitation becomes problematic in many real-world applications that require a specific model size or parameter budget. To address this issue, we further introduce an additional density constraint on $\mathcal{R}(\boldsymbol{\mathcal{W}}, \boldsymbol{\Theta})$ to guide the network towards achieving a specific desired budget.

$$\begin{aligned} \min_{\boldsymbol{\mathcal{W}}} \mathcal{R}(\boldsymbol{\mathcal{W}}) \triangleq \frac{1}{N}(\sum_{i=1}^{N} \mathcal{L}(f(x_i; \boldsymbol{\mathcal{W}}), y_i)) + \beta \sum_{i=1}^{K} \mathcal{R}_{orth}(\boldsymbol{B}_i, \boldsymbol{A}_i) \\ s.t. \quad C(\boldsymbol{\Theta}) \triangleq \sum_{i=1}^{K} \frac{\mathbb{E}[||\boldsymbol{G}_i||_0] \times (d_i + k_i)}{\#(\boldsymbol{B}_i) + \#(\boldsymbol{A}_i)} = b \end{aligned} \tag{11}$$

where b represents the target density and $\#(x)$ counts the total number of parameters in matrix x. $\Delta \boldsymbol{W}_i = \boldsymbol{B}_i \boldsymbol{A}_i$, where $\boldsymbol{B}_i$ is of $d_i \times r_i$, and $\boldsymbol{A}_i$ is of $r_i \times k_i$. However, lowering the density constraint poses a challenging and (not necessarily strictly) constrained optimization problem. To tackle this challenge, we leverage Lagrangian relaxation as an alternative approach, along with the corresponding min-max game:

$$\max_{\lambda} \min_{\boldsymbol{\mathcal{W}}, \boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\mathcal{W}}, \boldsymbol{\Theta}, \lambda) \triangleq \mathcal{R}(\boldsymbol{\mathcal{W}}, \boldsymbol{\Theta}) + \lambda(C(\boldsymbol{\Theta}) - b)^2 \tag{12}$$

where $\lambda \in \mathbb{R}$ is the Lagrangian multiplier, which is jointly updated during training. The updates to λ would increase the training loss unless the equality constraint is satisfied, resulting in the desired parameter budget. We optimize the Lagrangian relaxation by simultaneously performing gradient descent on $(\boldsymbol{\mathcal{W}}, \boldsymbol{\Theta})$ and projected gradient ascent (to $\mathbb{R}^+$) on λ, as demonstrated in previous works [19,33]. During the experiments, we observed that the term $\lambda(C(\boldsymbol{\Theta}) - b)^2$ converged quickly. To enhance training efficiency, we only optimize $(\boldsymbol{\Theta}, \lambda)$ between T_{start} and T_{end} time steps. We provide a summarized algorithm in Algorithm 1.

Limitations: The optimization process involves complex techniques like Lagrangian relaxation and orthogonal regularization, which may increase the computational resources.