Replay-Free Continual Low-Rank Adaptation with Dynamic Memory

Huancheng Chen University of Texas at Austin

Jingtao Li SonyAI

Weiming Zhuang SonyAI

huanchengch@utexas.edu

jingtao.li@sony.com

weiming.zhuang@sony.com

Chen Chen SonyAI

ChenA.Chen@sony.com

Lingjuan Lyu* SonyAI

lingjuan.lv@sony.com

Abstract

We revisit continual learning (CL), which enables pretrained vision transformers (ViTs) to sequentially fine-tune on new downstream tasks over time. However, as the scale of these models increases, catastrophic forgetting remains a more serious challenge. Recent studies highlight a crossover between CL techniques and parameter-efficient fine-tuning (PEFT), which focuses on fine-tuning only a small set of trainable parameters to adapt to downstream tasks, such as low-rank adaptation (LoRA). While LoRA achieves faster convergence and requires fewer trainable parameters, it has seldom been explored in the context of continual learning. To address this gap, we propose a novel PEFT-CL method called Dual Low-Rank Adaptation (Dual-LoRA), which introduces both an orthogonal LoRA adapter and a residual LoRA adapter parallel to pre-trained weights in each layer. These components are orchestrated by a dynamic memory mechanism to strike a balance between stability and plasticity. Additionally, we propose a scheme to predict task identity with confidence and calibrate the model's outputs accordingly. On ViT-based models, we demonstrate that DualLoRA offers significant advantages in accuracy, inference speed, and computation efficiency in training over existing CL methods across multiple benchmarks.

1. Introduction

Continual learning (CL) [22], which involves training a model on a sequence of tasks, often faces the challenge of *catastrophic forgetting*—a significant decline in performance on previously learned tasks when learning new ones. This issue persists even in the continual training of vision foundation models [25, 26], despite their strong generalization capabilities and robustness. Existing replay-based CL schemes [1] typically mitigate forgetting by storing a

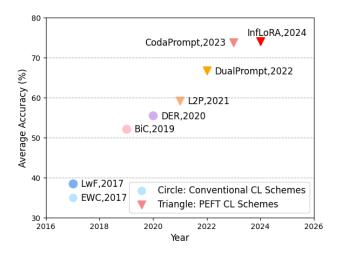


Figure 1. PEFT-based continual learning schemes dominate the ImageNet-R dataset in recent years.

subset of data from previous tasks as *exemplars*. However, this approach can be impractical due to storage constraints and data retention policies. Meanwhile, architecture-based CL schemes [14] allocate isolated parameters for each task, using a task identifier during training to prevent interference across tasks. Nevertheless, modifying the architecture of vision foundation models for fine-tuning on downstream tasks can compromise their pre-trained capabilities, as the pre-trained weights are learned within a fixed architecture. Furthermore, these methods often assume that task identity is known during inference, which is unrealistic in real-world scenarios.

Recently, parameter-efficient fine-tuning (PEFT) techniques have gained significant attention for their ability to adapt foundation models to downstream tasks by updating or adding only a small number of parameters. Moreover, PEFT methods have demonstrated notable robustness in mitigating catastrophic forgetting during fine-tuning on

sequential tasks. In particular, prompt-tuning [20, 25, 26] and low-rank adaptation (LoRA) [12] are two widely used PEFT approaches that have achieved remarkable success in continual learning, significantly outperforming conventional CL schemes across established benchmarks. Prompt-tuning focuses on learning a *prompt pool*, which enables the matching of an image with a set of prompt vectors and the alignment of image features with patch tokens. However, existing prompt-based CL methods require using the original pre-trained encoder as the query function. Consequently, image tokens must be processed through the network **twice**, resulting in substantial computational overhead and increased inference latency.

In contrast, LoRA methods require fewer trainable parameters to achieve comparable performance on domain-specific tasks and enable faster inference speeds than prompt-tuning. Nevertheless, vanilla LoRA [7] struggles to preserve satisfactory performance in continual learning settings due to significant interference across different tasks. Inspired by gradient projection techniques [19], InfLoRA [12] takes the first step in mitigating such interference by initializing LoRA adapter parameters for new tasks within a subspace orthogonal to the gradient subspace of previously learned tasks. However, InfLoRA requires passing through the encoder **twice** for each sample in a task: the first pass extracts the gradient subspace for LoRA initialization, while the second pass updates the parameters, leading to substantial computational overhead in the **training** phase.

To this end, we propose a novel continual learning method, dual low-rank adaptation (DualLoRA), which integrates an orthogonal adapter and a residual adapter into each layer of pre-trained vision transformers (ViTs). Specifically, the orthogonal adapter O is updated exclusively in directions orthogonal to features extracted from previously learned tasks, while the residual adapter R is updated in a task-specific subspace spanned by the residual bases extracted from previously learned tasks. This design enhances stability, i.e., robustness against forgetting in old tasks, through orthogonal adapters while improving plasticity, i.e., the ability to adapt continuously to new tasks, through residual adapters—thus achieving a balance between both objectives. Unlike InfLoRA, DualLoRA efficiently extracts orthogonal bases from the feature subspace of previously learned tasks and projects the updates of O and R using matrices constructed from these extracted bases. Moreover, these bases can be used to compute task relevance during inference, allowing the residual adapter's outputs to be adjusted based on task relevance, thereby mitigating components that could degrade test performance. This mechanism, which dynamically modulates the parameters of the residual adapter based on input test samples during inference, is referred to as dynamic memory (DM), as illustrated in Fig. 2.

Extensive experimental results demonstrate that Dual-

LoRA outperforms existing PEFT methods across various continual learning benchmarks, without incurring significant additional computational or memory overhead. The main contributions of this paper are summarized as follows:

- We introduce a novel low-rank adaptation paradigm for fine-tuning ViTs in continual learning settings. This paradigm efficiently extracts feature subspaces from previously learned tasks using singular value decomposition and mitigates catastrophic forgetting by reducing task interference through gradient projection.
- To address the challenge of limited update space due to gradient projection, we design a dual LoRA structure consisting of an orthogonal adapter and a residual adapter. This design incorporates the proposed dynamic memory mechanism, effectively balancing stability and plasticity in continual learning.
- To further enhance the performance of DualLoRA, we develop a simple and efficient method for inferring task identities of test samples during inference, leveraging the extracted feature subspaces. Extensive experimental results demonstrate the superior performance of DualLoRA compared to state-of-the-art baselines.

2. Background and Related Work

2.1. Gradient Projection in CL

Gradient projection is widely employed in continual learning to mitigate catastrophic forgetting by updating parameters in directions that minimize interference with previously learned tasks. OGD [3] was the first to implement gradient descent in the direction orthogonal to the stored gradient directions computed in previous tasks. The follow-up work GPM [19] extracts orthogonal bases of task representations from randomly selected training data via singular value decomposition (SVD). A subsequent study, TRGP [13], introduces the concept of a trust region, allowing partial reuse of selected bases from previous tasks. FSDGPM [2] further evaluates the importance of bases in GPM by assessing the sharpness of the loss landscape, assigning weights to the bases in the projection matrix according to their relative importance. Due to the computational expense of determining loss landscape sharpness, SGP [18] offers an alternative by using accumulated singular values as an importance indicator to scale the projection matrix in GPM. Additionally, several studies [9, 11, 23, 27] have explored relaxing the orthogonality constraints and optimizing the relaxation factor to enhance performance. However, these approaches are primarily developed for simpler models such as CNNs and face significant challenges when applied to advanced architectures like ViTs. For instance, the high dimensionality of feature embeddings in ViTs leads to substantial computational overhead when performing SVD. This motivates us to explore more efficient methods for extracting orthogonal

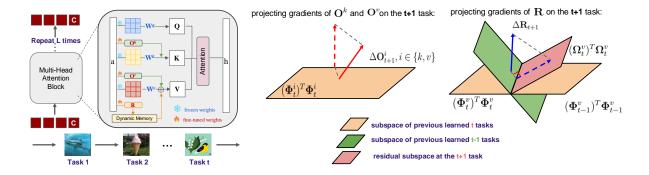


Figure 2. Illustration of our proposed DualLoRA paradigm (left) and design insights of orthogonal adapter and residual adapter (right), where the solid arrow denotes the original update and the dashed arrow denotes the projected update.

bases from feature subspaces in ViTs.

2.2. Parameter-Efficient Fine-Tuning in CL

Parameter-efficient fine-tuning (PEFT) methods enable the adaptation of pre-trained models to downstream tasks by fine-tuning only a small number of trainable parameters. Among these, prompt-tuning [10] has demonstrated strong robustness in learning sequential tasks and has achieved remarkable success in continual learning benchmarks, significantly outperforming traditional continual learning schemes. The pioneering work L2P [26] addresses continual learning challenges by introducing a prompt pool that matches the top-k relevant queries, serving as supplementary inputs to facilitate feature alignment in the pretrained model. The follow-up study, DualPrompt [25], takes a step further by employing task-invariant G-Prompts and task-specific E-Prompts to capture both shared and taskspecific knowledge across different tasks. S-Prompt [24] independently learns prompts across domains, using K-means clustering on features of the training data and a K-NN algorithm to match test data with domain-specific prompts. CodaPrompt [20] shifts the focus from fixed instancespecific prompts to a set of prompt components, selecting a weighted combination of these components for each data sample. Moreover, a series of studies [5, 16, 28] finetune expandable prompt adapters to enhance model learnability. Although prompt-tuning methods have achieved remarkable success in continual learning, the aforementioned schemes all require forwarding training and testing data to the query function (typically the original pre-trained model) for extracting features before fine-tuning and inference, which causes longer fine-tuning and inference times.

Recently, PGP [17] and VPT-NSP [15] adapt the idea of GPM [19] in prompt-tuning to mitigate the forgetting phenomenon by implementing orthogonal projection on the gradients of the prompt pool, but inheriting all drawbacks in prompt-tuning. Similarly, InfLoRA [12] marks the first attempt to apply gradient projection in low-rank adaptation

for ViT models, storing gradient directions from learned tasks during fine-tuning. However, InfLoRA introduces significant computational overhead due to performing singular value decomposition (SVD) on the high-dimensional gradient subspace, which is computationally more expensive than the original LoRA. Moreover, the orthogonal bases derived from the gradient subspace cannot be utilized to infer task identities during inference, as gradients are not available at that stage. This challenge motivates us to explore the use of gradient projection based on the orthogonal bases in the feature subspace, enabling the modulation of dynamic memory during inference for improving average test performance.

3. Preliminary

3.1. Continual Learning Problem Setting

Given a pre-trained model with backbone parameters \mathbf{W}_0 , we aim to fine-tune the model by adding adapters \mathcal{A}_l in the l-th layer of the model and the classifier \mathcal{F} to fit a sequence of domain data $\mathcal{D}_t = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{|\mathcal{D}_t|}$, where \mathbf{x}_i denotes data samples and $\mathbf{y}_i \in \mathcal{Y}_t$ denotes corresponding labels in the t-th task. In the challenging class-incremental setting, there is no intersection between the label sets from two different tasks as $\mathcal{Y}_{t_1} \cap \mathcal{Y}_{t_2} = \emptyset$ for all $t_1 \neq t_2$. When learning a new task, access to the old task data might become unavailable due to storage constraints. The objective function in continual learning is to minimize the empirical risk of the unified adapters $\mathcal{A}_{1:L}$ integrated in L layers of the pre-trained model that performs inference on the sequence of data from T various tasks as follows:

$$\min_{\{\mathcal{A}_t\}_{t=1}^L, \mathcal{F}} \frac{1}{T} \sum_{t=1}^T \mathcal{L}_{\text{task}}(\mathbf{W}_0, \mathcal{A}_{1:L}, \mathcal{F}, \mathcal{D}_t), \tag{1}$$

where \mathbf{W}_0 is the frozen backbone parameters of the pretrained model; $\mathcal{L}_{task}(\cdot)$ is the loss function depending on the specific task. The classifier \mathcal{F} consists of an expanding set of fully-connected layers $f_t: \mathbb{R}^d \to \mathbb{R}^{C_t}$, where d is the dimension of embedding and C_t is the number of classes in the t-th task. Given **no task identities** during inference, all the learned $f_t(\cdot)$ are used to predict categories of input data.

3.2. Multi-Head Attention Block

Vision transformer (ViT) models break down images into n patches and flatten these patches into patch embeddings with dimension d. The encoder of a ViT consists of a sequence of multi-head attention (MHA) blocks containing key, query and value weights \mathbf{W}^q , \mathbf{W}^k and \mathbf{W}^v for mapping input activation signals $\mathbf{a}^{(l)}$ into $\mathbf{Q}^{(l)}$, $\mathbf{K}^{(l)}$ and $\mathbf{V}^{(l)}$ and obtaining the output signals $\mathbf{h}^{(l)}$ by computing

$$\mathbf{h}^{(l)} = \operatorname{softmax} \left(\frac{\mathbf{Q}^{(l)} \left(\mathbf{K}^{(l)} \right)^{\top}}{\sqrt{d}} \right) \cdot \mathbf{V}^{(l)}, \tag{2}$$

where $\mathbf{Q}^{(l)} := \mathbf{a}^{(l)} \mathbf{W}^q$, $\mathbf{K}^{(l)} := \mathbf{a}^{(l)} \mathbf{W}^k$ and $\mathbf{V}^{(l)} := \mathbf{a}^{(l)} \mathbf{W}^v$. The output signals $\mathbf{h}^{(l)}$ are input to the feedforward network and passed through normalization before being forwarded to the next MHA block until $\mathbf{h}^{(L)}$ is directed to the classifier.

3.3. Low-Rank Adaptation

Low-rank adaptation (LoRA) is a parameter-efficient fine-tuning method that enables reducing memory consumption by assigning learnable low-rank matrices $\mathbf{A} \in \mathbb{R}^{r \times d}$ and $\mathbf{B} \in \mathbb{R}^{d \times r}$ parallel to the frozen pre-trained weights $\mathbf{W}_0 \in \mathbb{R}^{d \times d}$ into each layer of the model as follow,

$$\mathbf{W} := \mathbf{W}_0 + \mathbf{B}\mathbf{A},\tag{3}$$

where **W** denotes model weights after fine-tuning, and $r \ll d$. In this paper, we follow the strategy in [4], and only implement LoRA fine-tuning on \mathbf{W}_0^k and \mathbf{W}_0^v while keeping \mathbf{W}_0^q frozen during the whole procedure.

4. Methodology: Dual Low-Rank Adaptation

The subspace for updating the model becomes more constrained as the gradients are projected into a subspace orthogonal to all feature subspaces from the previous tasks. Subsequent studies [2, 13, 23, 27] ease the stringent constraints of orthogonality to expand the optimization subspace in a new task, considering the stability-plasticity trade-off. Inspired by the prior studies, we propose a novel low-rank adaptation structure, DualLoRA, consisting of an orthogonal adapter $\mathbf{O} := \mathbf{A}_o \mathbf{B}_o \in \mathbb{R}^{d \times d}$ and a residual adapter $\mathbf{R} := \mathbf{A}_r \mathbf{B}_r \in \mathbb{R}^{d \times d}$ that are updated in the orthogonal direction and residual direction, as shown in Fig 2. We follow the strategy in the existing PEFT continual learning schemes [12, 17, 20, 25, 26], using pre-trained vision transformers (ViTs) as backbone models throughout this paper. In the forthcoming sections, we will illustrate the process of updating both adapters and integrating dynamic memory during model inference.

4.1. Orthogonal Adapter

The milestone work GPM [19], which uses orthogonal gradient projection to mitigate forgetting, involves flattening feature maps extracted by convolutional kernels into vectors and performing singular value decomposition (SVD) on these vectors to obtain orthogonal feature bases. However, vectorizing the patch embedding of a ViT with dimensions (n,d) necessitates extensive computation in SVD, particularly when dealing with high-resolution inputs. In vision transformers, feature embeddings are often redundant for classification tasks, as only the first embedding (commonly referred to as the *class token*) is passed to the classifier for prediction.

To this end, we propose an efficient method for extracting the orthogonal bases of the class-token subspace without performing SVD on the entire high-dimensional embedding space. Specifically, given the pre-trained weights \mathbf{W}_0^q , \mathbf{W}_0^k , and \mathbf{W}_0^v , the fine-tuned *key* and *value* weights, i.e., \mathbf{W}_{t+1}^k and \mathbf{W}_{t+1}^v , for the (t+1)-th task can be derived as follows:

$$\mathbf{W}_{t+1}^{k} = \mathbf{W}_{0}^{k} + \sum_{\tau=1}^{t+1} \Delta \mathbf{O}_{\tau}^{k} = \mathbf{W}_{t}^{k} + \Delta \mathbf{O}_{t+1}^{k}, \quad (4)$$

$$\mathbf{W}_{t+1}^{v} = \mathbf{W}_{0}^{v} + \sum_{\tau=1}^{t+1} \Delta \mathbf{O}_{\tau}^{v} = \mathbf{W}_{t}^{v} + \Delta \mathbf{O}_{t+1}^{v}, \quad (5)$$

where $\Delta \mathbf{O}_{\tau}^k$, $\Delta \mathbf{O}_{\tau}^v$ are the updates of the orthogonal adapter computed from the τ -th task. According to (2), when we fine-tune the parameters on the (t+1)-th task, the change of output signal $\mathbf{h}^{(l)}$ given the same data can be formulated as

$$\Delta \mathbf{h}^{(l)} \approx \Gamma \cdot \frac{\mathbf{Q}^{(l)} \left(\mathbf{a}^{(l)} \Delta \mathbf{O}_{t+1}^{k} \right)^{\top}}{\sqrt{d}} \cdot \mathbf{V}^{(l)} + \underbrace{\operatorname{softmax} \left(\frac{\mathbf{Q}^{(l)} \left(\mathbf{K}^{(l)} \right)^{\top}}{\sqrt{d}} \right) \cdot \mathbf{a}^{(l)}}_{\mathbf{S}^{(l)}} \Delta \mathbf{O}_{t+1}^{v},$$
(6)

where Γ is a diagonal matrix (the derivation is deferred to Appendix A.1). To preserve the value of class-token in $\mathbf{h}^{(L)}$ output by the last layer, we must restrict the value of $\Delta\mathbf{h}_1^{(l)} \approx \mathbf{0}$ (the change of first row in $\mathbf{h}^{(l)}$) for each layer so the class-token of the same test sample from old tasks can be preserved after fine-tuning on the new task. Since $\mathbf{Q}^{(l)}$ is unchanged with the frozen weight \mathbf{W}_0^q , we need to project $(\Delta\mathbf{O}_{t+1}^k)^{\top}$ into the subspace orthogonal to the subspace spanned by $\mathbf{k}^{(l)} := \mathbf{Q}_1^{(l)}$, denoting the first row of $\mathbf{Q}^{(l)}$. Meanwhile, $\Delta\mathbf{O}_{t+1}^v$ must be orthogonal to the subspace of $\mathbf{v}^{(l)} := \mathbf{S}_1^{(l)}$, the first row of $\mathbf{S}^{(l)}$. Following the strategy in GPM [19], we randomly sample m data points

from the current task after fine-tuning on the t-th task and input these m samples to the model for obtaining embedding matrices $\tilde{\mathbf{K}}^{(l)} \in \mathbb{R}^{m \times d}$ consisting of $\{\mathbf{k}_i^{(l)}\}_{i=1}^m$ and $\tilde{\mathbf{V}}^{(l)} \in \mathbb{R}^{m \times d}$ consisting of $\{\mathbf{v}_i^{(l)}\}_{i=1}^m$. We update the new feature matrices $\mathbf{\Phi}_t^k$ and $\mathbf{\Phi}_t^v$ by extracting the orthogonal bases of $\tilde{\mathbf{K}}^{(l)}$ and $\mathbf{V}^{(l)}$ using SVD and concatenating them into the previous feature matrices $\mathbf{\Phi}_{t-1}^k$ and $\mathbf{\Phi}_{t-1}^v$. With the feature matrices obtained on the t-th task, we can project the updates $\Delta \mathbf{O}_{t+1}^k$ and $\Delta \mathbf{O}_{t+1}^v$ by

$$\Delta \mathbf{O}_{t+1}^{i} \leftarrow \Delta \mathbf{O}_{t+1}^{i} - (\mathbf{\Phi}_{t}^{i})^{\top} \mathbf{\Phi}_{t}^{i} \Delta \mathbf{O}_{t+1}^{i}, \quad \forall i \in \{k, v\}.$$
(7)

When we select m data points, with $m \ll d$, for extracting orthogonal bases, the complexity of SVD is $\mathcal{O}(m^2d)$, which is much more efficient than SVD with $\mathcal{O}(d^3)$ implemented in the previous work InfLoRA [12]. We emphasize that our orthogonal adapters undergo a different update process than InfLoRA, where orthogonal bases are extracted from the gradient subspace. Instead, we develop an alternative feature set, $\mathbf{S}^{(l)}$, specifically to preserve the class token, as outlined in (6). Since we only extract the orthogonal subspace from $\Delta\mathbf{h}_1^{(l)}$, we cannot guarantee that $\Delta\mathbf{h}_1^{(l)} = \mathbf{0}$ in every layer. However, reducing the magnitude of $\Delta\mathbf{h}_1^{(l)}$ can mitigate catastrophic forgetting, even if the value is not zero (details are provided in Appendix A.2).

4.2. Residual Adapter

As the feature subspaces represented by Φ^k_t and Φ^v_{t+1} expand with the accumulation of learned tasks, the majority of the components in the updates, $\Delta \mathbf{O}^k_{t+1}$ and $\Delta \mathbf{O}^v_{t+1}$, are progressively subtracted, as detailed in (7). Consequently, the update magnitudes approach zero, resulting in diminished performance during fine-tuning on new tasks. To address this issue, we introduce a residual adapter \mathbf{R}_{t+1} (initialized as 0) in parallel with \mathbf{O}^v_{t+1} , providing extra capacity for new tasks and maintaining a balance between stability and plasticity.

When the model is fine-tuned on the (t+1)-th task, the residual adapter \mathbf{R}_{t+1} is updated within the subspace \mathcal{R}_t spanned by Ψ_t (we set $\Psi_1 = \emptyset$) defined as

$$\mathbf{\Psi}_t := \mathbf{\Phi}_t^v - \mathbf{\Phi}_{t-1}^v \subseteq \mathbb{R}^d, \tag{8}$$

where Φ^v_t and Φ^v_{t-1} are obtained after fine-tuning on the t and t-1 tasks. It is worth noting that subspace \mathcal{R}_t and \mathcal{R}_{t+1} are specific to their corresponding tasks t and t+1, respectively, as $\Psi_{t+1} \cap \Psi_t = \emptyset$. Specifically, the subspace \mathcal{R}_t indicates the residual knowledge extracted from the most recent task, providing supplementary bases to enlarge the optimization subspace. With these extracted bases, we are able to project the updates $\Delta \mathbf{R}_{t+1}$ into the subspace \mathcal{R}_t by

$$\Delta \mathbf{R}_{t+1} \leftarrow \mathbf{\Psi}_{t}^{\top} \mathbf{\Psi}_{t} \Delta \mathbf{R}_{t+1}. \tag{9}$$

When we conduct **fine-tuning** on the (t+1)-task, the value matrix $\mathbf{V}^{(l)}$ in the l-th layer given activations $\mathbf{h}^{(l)}$ can be found as

$$\mathbf{V}^{(l)} = \mathbf{a}^{(l)} \left(\mathbf{W}_0^v + \mathbf{O}_{t+1}^v \right) + \mathbf{a}^{(l)} \mathbf{R}_{t+1} = \mathbf{V}_o^{(l)} + \mathbf{V}_r^{(l)}, \tag{10}$$
where \mathbf{W}_0^v is the pre-trained weights, and $\mathbf{V}_o^{(l)} := \mathbf{a}^{(l)} \left(\mathbf{W}_0^v + \mathbf{O}_{t+1}^v \right), \mathbf{V}_r^{(l)} := \mathbf{a}^{(l)} \mathbf{R}_{t+1}.$

4.3. Dynamic Memory

As previously mentioned, the residual adapter \mathbf{R}_{t+1} is updated within the subspace \mathcal{R}_t , a subset of the feature subspace spanned by Φ_t extracted from the t-th task. Consequently, the fine-tuning process may deteriorate the performance on prior tasks. To mitigate this issue, we introduce a *dynamic memory* mechanism that adjusts value of $\mathbf{V}_r^{(l)}$, which is the output of \mathbf{R}_{t+1} during **inference** on test data according to

$$\hat{\mathbf{V}}^{(l)} = \mathbf{V}_o^{(l)} + \mathbf{a}^{(l)} \mathbf{\Omega}_{t+1}^T \mathbf{\Omega}_{t+1} \mathbf{R}_{t+1} = \mathbf{V}_o^{(l)} + \hat{\mathbf{V}}_r^{(l)}, (11)$$

where $\Omega_{t+1}^{\top}\Omega_{t+1}$ is computed according to the input activation signal $\mathbf{a}^{(l)}$. Specifically, the attention score $\mathbf{S}^{(l)} \propto \operatorname{softmax}(\frac{\mathbf{Q}^{(l)}(\mathbf{K}^{(l)})^{\top}}{\sqrt{d}})$, computed by applying $\mathbf{a}^{(l)}$ to the query and key weight matrices, reflects the relevance between the input test sample and the task associated with the extracted bases used to update the residual adapter \mathbf{R}_{t+1} . We utilize the matrices $\Psi_{\tau} \in \mathbb{R}^{r_{\tau} \times d}$ ($\tau \leq t+1$), stored in memory, to multiply the first row of the attention score, $\mathbf{v}^{(l)} := \mathbf{S}_1^{(l)}$, as follows:

if
$$r_{\tau} \neq 0, \omega_{\tau} = \frac{\left\| \mathbf{\Psi}_{\tau} \cdot \mathbf{v}^{(l)} \right\|}{r_{\tau} \left\| \mathbf{v}^{(l)} \right\|}$$
, otherwise, $\omega_{\tau} = 0$, (12)

where r_{τ} indicates the rank of Ψ_{τ} . Since the task-specific residual bases remain independent across different tasks, the cosine similarity between feature vectors $\mathbf{v}^{(l)}$ extracted from input test samples and the stored residual bases Ψ_{τ} for each task can be used as a scaling factor for the corresponding components in the outputs of the residual adapter. This method assigns lower weights to components irrelevant to the current test samples, while components with high relevance to the samples are given proportionally higher weights. The resulting matrix, Ω_{t+1} , is obtained by:

$$\Omega_{t+1} = \begin{bmatrix}
\Sigma_1 & \mathbf{0} & \cdots & \mathbf{0} \\
\mathbf{0} & \Sigma_2 & & \\
\vdots & & \ddots & \vdots \\
\mathbf{0} & \mathbf{0} & \cdots & \Sigma_{t+1}
\end{bmatrix} \begin{bmatrix}
\Psi_1 \\
\Psi_2 \\
\vdots \\
\Psi_{t+1}
\end{bmatrix} \in \mathbb{R}^{r' \times d},$$
(13)

where $r' = \sum_{\tau=1}^{t+1} r_{\tau}$, $\Sigma_{\tau} \in \mathbb{R}^{r_{\tau} \times r_{\tau}}$ is a diagonal matrix with the identical value $\omega_{\tau}^{1/2}$.

Table 1. Metrics (%) computed from experiments on ImageNet-R. We report the average accuracy over 3 trials, each with different random seeds. The numeric after "±" denotes standard deviation.

Method	5-Split ImageNet-R		10-Split ImageNet-R		20-Split ImageNet-R	
Method	ACC(↑)	$FT(\downarrow)$	$ACC(\uparrow)$	$FT(\downarrow)$	$ACC(\uparrow)$	$FT(\downarrow)$
LoRA	72.33 ± 0.94	12.1 ± 1.19	61.85 ± 0.52	26.0 ± 1.35	48.59 ± 0.39	34.4 ± 0.57
L2P	61.60 ± 0.43	5.36 ± 0.27	59.21 ± 0.68	7.59 ± 0.78	56.36 ± 0.83	10.3 ± 0.72
DualPrompt	68.47 ± 0.23	3.18 ± 0.24	66.72 ± 0.30	4.15 ± 0.11	64.40 ± 0.18	5.82 ± 0.51
PGP	69.07 ± 0.28	3.41 ± 0.18	64.22 ± 4.53	4.23 ± 0.22	64.19 ± 0.38	6.50 ± 0.31
S-Prompt	51.33 ± 0.22	27.6 ± 1.18	49.80 ± 0.16	29.2 ± 0.93	55.64 ± 0.53	22.3 ± 1.85
CodaPrompt	74.91 ± 0.30	1.85 ± 0.07	73.83 ± 0.29	2.56 ± 0.31	68.96 ± 0.46	3.25 ± 0.40
InfLoRA	77.30 ± 0.49	3.05 ± 0.44	74.03 ± 0.30	6.18 ± 0.25	69.77 ± 0.31	7.98 ± 0.40
DualLoRA	78.55 ± 0.12	2.61 ± 0.25	76.23 ± 0.33	3.67 ± 0.66	71.25 ± 0.31	5.45 ± 0.27
DualLoRA+	79.88 ± 0.50	1.10 ± 0.16	81.17 ± 0.23	2.04 ± 0.05	74.73 ± 0.40	3.75 ± 0.10

Table 2. Metrics (%) computed from experiments on CIFAR100 and Tiny-ImageNet. We report the average accuracy over 3 trials, each with different random seeds. The numeric after "±" denotes standard deviation.

Method	10-Split CIFAR100		10-Split TinyImageNet		20-Split TinyImageNet	
	$ACC(\uparrow)$	$FT(\downarrow)$	$ACC(\uparrow)$	$FT(\downarrow)$	$ACC(\uparrow)$	$FT(\downarrow)$
LoRA	73.32 ± 0.38	20.4 ± 0.53	67.69 ± 0.49	23.7 ± 0.65	48.48 ± 2.36	44.4 ± 2.73
L2P	83.97 ± 0.18	6.41 ± 0.09	81.90 ± 0.42	5.39 ± 0.33	81.24 ± 0.21	5.86 ± 0.22
DualPrompt	85.85 ± 0.22	5.41 ± 0.12	85.10 ± 0.10	3.95 ± 0.22	82.77 ± 0.12	5.31 ± 0.10
PGP	85.28 ± 0.01	5.60 ± 0.34	84.83 ± 0.21	4.32 ± 0.16	83.49 ± 0.35	5.24 ± 0.31
S-Prompt	67.03 ± 0.66	24.8 ± 0.62	68.41 ± 0.26	10.41 ± 0.68	74.69 ± 0.30	7.70 ± 0.28
CodaPrompt	85.77 ± 0.69	4.07 ± 0.22	85.67 ± 0.25	3.16 ± 0.17	83.61 ± 0.47	3.34 ± 0.35
InfLoRA	85.62 ± 0.74	4.34 ± 0.06	81.28 ± 0.40	8.62 ± 0.36	75.89 ± 0.38	13.8 ± 0.11
DualLoRA	89.13 ± 0.17	4.08 ± 0.16	86.42 ± 0.07	3.87 ± 0.18	83.75 ± 0.25	5.24 ± 0.15
DualLoRA+	90.94 ± 0.15	3.20 ± 0.18	87.74 ± 0.21	2.45 ± 0.25	84.65 ± 0.07	3.61 ± 0.13

4.4. Task Identification with Confidence

As the number of fully connected layers $f_t(\cdot)$ increases with the addition of tasks during continual learning, there is a risk that an irrelevant fully connected layer may generate the maximal logit, resulting in incorrect predictions for input test samples. This motivates us to propose a task identity prediction scheme based on task relevance, computed using Ψ_{τ} as described earlier in (12).

As mentioned earlier, we sample m training data points to extract orthogonal bases after completing the t-th task. During this process, we obtain the average feature vector $\bar{\mathbf{v}}^{(L)}$ forwarded to the **final** attention block from each task and compute the similarity vector $\boldsymbol{\pi}_t = \{\omega_\tau\}_{\tau=1}^t$ based on (12). Therefore, we obtain a set $\boldsymbol{\Pi}_t = \{\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_t\}$ that can be used to distinguish the task identity during inference. Specifically, let $\boldsymbol{\pi}^*$ denote the similarity vector computed from the input test sample, then we can predict the **task identity** by

$$\hat{k} = \arg\max_{\tau} g(\boldsymbol{\pi}_{\tau}, \boldsymbol{\pi}^{\star}) := \frac{\|\boldsymbol{\pi}_{\tau} \cdot \boldsymbol{\pi}^{\star}\|}{\|\boldsymbol{\pi}_{\tau}\| \cdot \|\boldsymbol{\pi}^{\star}\|}, \quad (14)$$

$$\hat{\delta} = \lambda \left(g(\boldsymbol{\pi}_{\hat{k}}, \boldsymbol{\pi}^{\star}) - \max_{\tau \neq \hat{k}} g(\boldsymbol{\pi}_{\tau}, \boldsymbol{\pi}^{\star}) \right), \quad (15)$$

where λ is a scaling factor, \hat{k} denotes the predicted task identity and $\hat{\delta}$ indicates the confidence of this prediction. Moreover, we scale the output logits as

$$f_{\hat{k}}(\mathbf{h}^{(L)}) \leftarrow (1+\hat{\delta}) \cdot f_{\hat{k}}(\mathbf{h}^{(L)}).$$
 (16)

5. Experiments

5.1. Experimental Settings

Datasets and Metrics. We evaluate the proposed method DualLoRA on three continual learning benchmark datasets: CIFAR100, Tiny-ImageNet and ImageNet-R[6]. To generate a sequence of tasks in a class-incremental setting as illustrated in (1), we randomly split the original dataset by class ID. This process creates multiple partitions, each containing an equal number of classes. Each partition corresponds to a distinct task. Following the strategy in existing studies in continual learning [17, 20, 25], we compute the final average accuracy (denoted by ACC) and degree of forgetting (denoted by FT) for evaluating the performance of

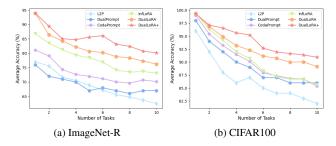


Figure 3. Figures (a) and (b) demonstrate the average accuracy of different methods during training.

our method, and these two metrics can be found as

$$ACC = \frac{1}{T} \sum_{\tau=1}^{T} acc_{\tau,T}, \tag{17}$$

$$FT = \frac{1}{T - 1} \sum_{\tau = 1}^{T - 1} acc_{\tau, best} - acc_{\tau, T},$$
 (18)

where $\mathrm{acc}_{\tau,T}$ denotes the accuracy of the τ -th task after the model learns the T-th task while $\mathrm{acc}_{\tau,\mathrm{best}}$ denotes the highest accuracy on the τ -th task during the whole fine-tuning process. Throughout the evaluation, we assume that the task identities of the testing data are **unknown**.

Baselines. Our baselines include vanilla LoRA, L2P [26], DualPrompt [25], PGP [17], S-Prompt [24], CodaPrompt [20], and InfLoRA [12]. We focus on comparing the proposed DualLoRA with the state-of-the-art PEFT-based CL schemes since they are superior to traditional CL schemes. To demonstrate the **upper-bound** performance of DualLoRA, we implemented it under the setting where all samples in a batch share the same task identity, referred to as **DualLoRA+**. In this scenario, we use average feature to compute similarity in the task identity prediction as described in Section 4.4 and facilitate more accurate task prediction, as the average feature exhibits less variance and is closer to the true mean.

Model Architecture and Hyperparameters. We use ViT-B/16 backbone pretrained on ImageNet-21K as the foundation model throughout all experiments. We use the Adam optimizer with parameters $\beta_1=0.9$ and $\beta_2=0.999$ for model fine-tuning in 5 epochs and the batchsize is set to 16 in all experiments. More details of hyperparameters are provided in Appendix B.1.

5.2. Experimental Results

ImageNet-R. As shown in Table 1, DualLoRA demonstrates comparable performance with InfLoRA, which outperforms other Prompt-based CL schemes in final

average accuracy ACC. DualLoRA exhibits a slight performance decrease compared to InfLoRA by 1.25% on the 5-split benchmark, yet it outperforms InfLoRA by 2.2\% and 1.48% on the 10-split and 20-split benchmarks, respectively. Forgetting serves as another metric to quantify the performance degradation on previous tasks, which may not consistently align with the average accuracy. CodaPrompt shows superior performance in mitigating forgetting, even though it does not achieve the same level as InfLoRA and DualLoRA in the average accuracy metric. DualLoRA+ significantly enhances both average accuracy and forgetting metrics, surpassing all other baselines except for securing the second position in the forgetting metric on the 20-split benchmark. DualLoRA+ outperforms the state-of-the-art scheme InfLoRA by 2.58%, 7.14%, and 4.96% in terms of average accuracy metric across the 5-split, 10-split, and 20-split settings. In addition, in terms of the forgetting metric, DualLoRA+ shows improvements over InfLoRA by 1.95%, 4.14%, and 3.23%, respectively, in the same settings.

CIFAR100 and Tiny-ImageNet. DualLoRA steadily shows strong performance in these two datasets, achieving the best performance in average accuracy compared to prior existing schemes on CIFAR100 and Tiny-ImageNet benchmarks while slightly underperforming CodaPrompt on Tiny-ImageNet in forgetting metrics. DualLoRA+ consistently demonstrates extraordinary performance in average accuracy, outperforming CodaPrompt (the best scheme in these baselines) by 5.17%, 2.07% and 1.04%, respectively, and also demonstrating an advantage in the forgetting metric on 10-split CIFAR100 and 10-split Tiny-ImageNet. To give more insights, we report the average accuracy computed with different numbers of learned tasks, as illustrated in Fig. 3(a) and 3(b) As shown in the figures, DualLoRA and DualLoRA+ outperform other baselines in different stages of continual learning. Furthermore, DualLoRA+ demonstrates robust resistance to forgetting as the number of learned tasks increases, suggesting the potential of DualLoRA+ fine-tuning foundational models across a wider range of tasks without significant forgetting.

5.3. Ablation Study

We perform additional experiments to confirm the effectiveness of various subroutines within the DualLoRA scheme. To be specific, we implement three variants of DualLoRA: (1) $\mathbf{LoRA} + \mathbf{O}$ stands for only using the orthogonal adapter; (2) $\mathbf{LoRA} + \mathbf{O} + \mathbf{R}$ stands for running DualLoRA with orthogonal and residual adapters but performing no task identity prediction; (3) $\mathbf{LoRA} + \mathbf{O} + \mathbf{R} + \mathbf{Task}$ ID assumes knowing true task identities. As illustrated in Table 3, the orthogonal adapter significantly improves the performance of \mathbf{LoRA} while the residual adapter further enhances both

Table 3. O stands for orthogonal adapter,	R stands for residual adapte	er, and Task ID stands for	giving true task ID during inference.

Method	10-Split ImageNet-R		10-Split CIFAR100		10-Split TinyImageNet	
Method	$ACC(\uparrow)$	$FT(\downarrow)$	$ACC(\uparrow)$	$FT(\downarrow)$	$ACC(\uparrow)$	$FT(\downarrow)$
LoRA	61.85	26.0	73.03	20.26	67.69	23.70
LoRA + O	74.11	4.65	84.03	5.67	83.92	7.74
LoRA + O + R	74.60	4.12	86.65	3.96	85.61	4.30
DualLoRA	76.23	3.67	89.13	5.08	86.42	3.87
DualLoRA+	81.17	2.04	90.94	3.20	87.74	2.45
DualLoRA + Task ID	87.66	0.46	94.39	1.05	95.71	0.84

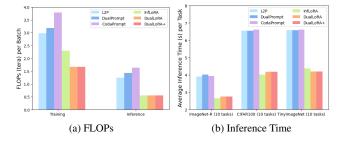


Figure 4. Figure (a) demonstrates the approximated average FLOPs during training and inference on each batch of data points. Figure (b) demonstrates the actual average running time for different schemes to perform inference on a task.

average accuracy and forgetting metrics. Moreover, Dual-LoRA and DualLoRA+ further improve the performance by using individual features and average features to perform task identity prediction. The final configuration, where true task identities are utilized, exhibits superior performance in both average accuracy and forgetting, highlighting the crucial role of task prediction.

5.4. Computation and Inference Time

To better compare DualLoRA with other baselines in terms of computation, we report the number of floating point operations (FLOPs) during the training and inference phases in Figure 4(a) and the average inference time on a task in Figure 4(b) According to the results in the figure, InfLoRA has the lowest FLOPs during inference, while DualLoRA has the lowest FLOPs during training because InfLoRA requires a double forward pass. During inference, InfLoRA and DualLoRA have similar inference time across different datasets, which is less than 50% inference time compared to the prompt-based CL schemes. Details on computing FLOPs are provided in Appendix B.2.

5.5. Varying the Pre-Trained Models

To evaluate the consistency of DualLoRA's performance, we conduct experiments using the ViT-B/16 model, pre-trained on ImageNet-1k through both supervised learning and the unsupervised SAM framework [8]. To represent

Table 4. Metrics computed from experiments on ImageNet-R (10 tasks) using various pretrained ViTs beyond ImageNet-21k. \overline{ACC} denotes the average of \overline{ACC} in every timestep.

	Method	ACC(%)	$\overline{ACC}(\%)$
	L2P	60.23	67.86
	DualPrompt	67.45	72.48
ImageNet-1k	CodaPrompt	73.26	79.49
	InfLoRA	75.58	81.92
	DualLoRA	77.28	82.63
	L2P	52.71	58.24
	DualPrompt	56.57	63.23
SAM-1k	CodaPrompt	61.13	70.79
	InfLoRA	64.62	73.66
	DualLoRA	66.44	74.70

the average performance of all schemes during the continual learning process, we use $\overline{ACC} = \frac{1}{T} \sum_{t=1}^{T} ACC_t$ as an additional metric in the table. As shown, all schemes experience performance degradation when using a ViT model pretrained on unsupervised datasets. Nonetheless, DualLoRA consistently outperforms other schemes in terms of average accuracy.

6. Conclusion

We introduce DualLoRA, a novel low-rank adaptation scheme for vision transformers (ViTs) that integrates orthogonal and residual adapters, operating in parallel with pre-trained weights. This structure achieves a balance between stability and plasticity in continual learning through a dynamic memory mechanism that leverages subspaces from previously learned tasks. Furthermore, we develop a task identity prediction scheme based on core bases extracted from each learned task to enhance DualLoRA's performance. Extensive experiments demonstrate that DualLoRA outperforms state-of-the-art continual learning methods across multiple benchmarks while requiring fewer computational resources. We see this work as an important step toward developing more efficient and effective continual learning paradigms for foundational models.

References

- [1] Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co2l: Contrastive continual learning. In *Proceedings of the IEEE/CVF International conference on computer vision*, pages 9516–9525, 2021.
- [2] Danruo Deng, Guangyong Chen, Jianye Hao, Qiong Wang, and Pheng-Ann Heng. Flattening sharpness for dynamic gradient projection memory benefits continual learning. Advances in Neural Information Processing Systems, 34: 18710–18721, 2021. 2, 4
- [3] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR, 2020. 2
- [4] Qiankun Gao, Chen Zhao, Yifan Sun, Teng Xi, Gang Zhang, Bernard Ghanem, and Jian Zhang. A unified continual learning framework with general parameter-efficient tuning. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 11483–11493, 2023. 4
- [5] Xinyuan Gao, Songlin Dong, Yuhang He, Qiang Wang, and Yihong Gong. Beyond prompt learning: Continual adapter for efficient rehearsal-free continual learning. In *European Conference on Computer Vision*, pages 89–106. Springer, 2024. 3
- [6] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In Proceedings of the IEEE/CVF international conference on computer vision, pages 8340–8349, 2021. 6
- [7] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021. 2
- [8] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Con*ference on Computer Vision, pages 4015–4026, 2023. 8
- [9] Yajing Kong, Liu Liu, Zhen Wang, and Dacheng Tao. Balancing stability and plasticity through advanced null space in continual learning. In *European Conference on Computer Vision*, pages 219–236. Springer, 2022. 2
- [10] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691, 2021. 3
- [11] Yan-Shuo Liang and Wu-Jun Li. Adaptive plasticity improvement for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7816–7825, 2023. 2
- [12] Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. *arXiv* preprint *arXiv*:2404.00228, 2024. 2, 3, 4, 5, 7, 12, 17
- [13] Sen Lin, Li Yang, Deliang Fan, and Junshan Zhang. Trgp: Trust region gradient projection for continual learning. arXiv preprint arXiv:2202.02931, 2022. 2, 4

- [14] Noel Loo, Siddharth Swaroop, and Richard E Turner. Generalized variational continual learning. arXiv preprint arXiv:2011.12328, 2020. 1
- [15] Yue Lu, Shizhou Zhang, De Cheng, Yinghui Xing, Nannan Wang, Peng Wang, and Yanning Zhang. Visual prompt tuning in null space for continual learning. arXiv preprint arXiv:2406.05658, 2024. 3
- [16] Mark D McDonnell, Dong Gong, Amin Parvaneh, Ehsan Abbasnejad, and Anton Van den Hengel. Ranpac: Random projections and pre-trained models for continual learning. Advances in Neural Information Processing Systems, 36:12022–12053, 2023. 3
- [17] Jingyang Qiao, Xin Tan, Chengwei Chen, Yanyun Qu, Yong Peng, Yuan Xie, et al. Prompt gradient projection for continual learning. In *The Twelfth International Conference on Learning Representations*, 2023. 3, 4, 6, 7
- [18] Gobinda Saha and Kaushik Roy. Continual learning with scaled gradient projection. In *Proceedings of the AAAI Con*ference on Artificial Intelligence, pages 9677–9685, 2023. 2
- [19] Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. arXiv preprint arXiv:2103.09762, 2021. 2, 3, 4, 12
- [20] James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Con*ference on Computer Vision and Pattern Recognition, pages 11909–11919, 2023. 2, 3, 4, 6, 7, 16
- [21] Lloyd N Trefethen and David Bau. Numerical linear algebra. SIAM, 2022. 16
- [22] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. 1
- [23] Shipeng Wang, Xiaorong Li, Jian Sun, and Zongben Xu. Training networks in null space of feature covariance for continual learning. In Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition, pages 184–193, 2021. 2, 4
- [24] Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. S-prompts learning with pre-trained transformers: An occam's razor for domain incremental learning. *Advances in Neural Information Processing Systems*, 35:5682–5695, 2022. 3, 7
- [25] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *European Conference on Computer Vision*, pages 631–648. Springer, 2022. 1, 2, 3, 4, 6, 7, 16
- [26] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 139–149, 2022. 1, 2, 3, 4, 7, 16

- [27] Zhen Zhao, Zhizhong Zhang, Xin Tan, Jun Liu, Yanyun Qu, Yuan Xie, and Lizhuang Ma. Rethinking gradient projection continual learning: Stability/plasticity feature space decoupling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3718–3727, 2023. 2, 4
- [28] Da-Wei Zhou, Hai-Long Sun, and et al. Expandable subspace ensemble for pre-trained model-based class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23554–23564, 2024. 3

Appendix A.1: Approximate Change of Output Activation

In main paper, we introduce the approximate output change after orthogonal adapter $\mathbf{O}_t^{(l)}$ fine-tuning on the (t+1)-th task. In what follows we drop the layer superscript (l) and let $\mathcal{S}(\cdot)$ denote **softmax** operation.

$$\begin{split} & \Delta \mathbf{h}_{t+1} = \mathbf{h}_{t+1} - \mathbf{h}_{t} \\ & = \mathcal{S} \left(\frac{\mathbf{Q} \left(\mathbf{K}_{t+1} \right)^{\top}}{\sqrt{d}} \right) \cdot \mathbf{V}_{t+1} - \mathcal{S} \left(\frac{\mathbf{Q} \left(\mathbf{K}_{t} \right)^{\top}}{\sqrt{d}} \right) \cdot \mathbf{V}_{t} \\ & = \mathcal{S} \left(\frac{\mathbf{Q} \left(\mathbf{W}_{t+1}^{k} \right)^{\top} \mathbf{a}^{\top}}{\sqrt{d}} \right) \cdot \mathbf{V}_{t+1} - \mathcal{S} \left(\frac{\mathbf{Q} \left(\mathbf{W}_{t}^{k} \right)^{\top} \mathbf{a}^{\top}}{\sqrt{d}} \right) \cdot \mathbf{V}_{t} \\ & = \mathcal{S} \left(\frac{\mathbf{Q} \left(\mathbf{W}_{t+1}^{k} \right)^{\top} \mathbf{a}^{\top}}{\sqrt{d}} \right) \cdot \mathbf{a} \left(\mathbf{W}_{t}^{v} + \Delta \mathbf{O}_{t+1}^{v} \right) \\ & - \mathcal{S} \left(\frac{\mathbf{Q} \left(\mathbf{W}_{t}^{k} \right)^{\top} \mathbf{a}^{\top}}{\sqrt{d}} \right) \cdot \mathbf{a} \mathbf{W}_{t}^{v} \\ & = \mathcal{A} \cdot \mathbf{a} \mathbf{W}_{t}^{v} + \mathcal{B} \cdot \mathbf{a} \Delta \mathbf{O}_{t+1}^{v}, \end{split}$$

where

$$\mathcal{A} = \mathcal{S}\left(\frac{\mathbf{Q}(\mathbf{W}_t^k + \Delta \mathbf{O}_{t+1}^k)^{\top} \mathbf{a}^{\top}}{\sqrt{d}}\right) - \mathcal{S}\left(\frac{\mathbf{Q}(\mathbf{W}_t^k)^{\top} \mathbf{a}^{\top}}{\sqrt{d}}\right),\tag{19}$$

$$\mathcal{B} = \mathcal{S}\left(\frac{\mathbf{Q}(\mathbf{W}_t^k + \Delta \mathbf{O}_{t+1}^k)^{\top} \mathbf{a}^{\top}}{\sqrt{d}}\right). \tag{20}$$

Then, considering

$$\mathcal{A} = \mathcal{S}\left(\mathbf{z} + \Delta \mathbf{z}\right) - \mathcal{S}\left(\mathbf{z}\right),\tag{21}$$

where $\mathbf{z} = \frac{\mathbf{Q}(\mathbf{W}_t^k)^T\mathbf{a}^T}{\sqrt{d}}, \Delta \mathbf{z} = \frac{\mathbf{Q}(\Delta \mathbf{O}_{t+1}^k)^T\mathbf{a}^T}{\sqrt{d}}$. Since $\|\Delta \mathbf{O}_{t+1}^k\|$ is proportional to learning rate η , by selecting a small learning rate, we can get $\|\Delta \mathbf{z}\| \ll \|\mathbf{z}\|$. Then considering

$$\mathcal{A}_{i} = \mathcal{S} \left(\mathbf{z} + \Delta \mathbf{z} \right)_{i} - \mathcal{S} \left(\mathbf{z} \right)_{i} = \sum_{j=1}^{N} \frac{\partial S_{i}}{\partial \mathbf{z}_{j}} \Delta \mathbf{z}_{j}, \tag{22}$$

where $S_i = \mathcal{S}(\mathbf{z})_i < 1$ is the *i*-th component of $\mathcal{S}(\mathbf{z})$; $\Delta \mathbf{z}_j$ denotes the *j*-th component of $\Delta \mathbf{z}$. Since $\frac{\partial S_i}{\partial \mathbf{z}_j} = S_i(1 - S_j)$ if i = j and $\frac{\partial S_i}{\partial \mathbf{z}_i} = -S_iS_j$, otherwise, we can obtain:

$$\mathcal{A}_{i} = S_{i}(1 - S_{i})\Delta\mathbf{z}_{i} - S_{i}\sum_{j \neq i} S_{j}\Delta\mathbf{z}_{j}$$

$$= S_{i}\Delta\mathbf{z}_{i} - S_{i}\sum_{j=1}^{N} S_{j}\Delta\mathbf{z}_{j}$$

$$\leq S_{i}\Delta\mathbf{z}_{i} - S_{i}\min_{j}(\Delta\mathbf{z}_{j})$$

$$= \left(S_{i} - \frac{\min_{j}(\Delta\mathbf{z}_{j})}{\Delta\mathbf{z}_{i}}\right)\Delta\mathbf{z}_{i}.$$
(23)

Similarly, we can get

$$\left(S_i - \frac{\max_j(\Delta \mathbf{z}_j)}{\Delta \mathbf{z}_i}\right) \le \frac{\mathcal{A}_i}{\Delta \mathbf{z}_i} \le \left(S_i - \frac{\min_j(\Delta \mathbf{z}_j)}{\Delta \mathbf{z}_i}\right).$$
(24)

Let $A_i = \gamma_i \Delta \mathbf{z}_i$ such that $\left(S_i - \frac{\max_j(\Delta \mathbf{z}_j)}{\Delta \mathbf{z}_i}\right) \leq \gamma_i \leq \left(S_i - \frac{\min_j(\Delta \mathbf{z}_j)}{\Delta \mathbf{z}_i}\right)$, then we have $A = \Gamma \cdot \Delta \mathbf{z}$ where $\Gamma = \operatorname{diag}\{\gamma_1, \ldots, \gamma_N\}$. Then we refocus on $\mathcal B$ such that

$$\mathcal{B} = \mathcal{S} \left(\frac{\mathbf{Q} (\mathbf{W}_t^k + \Delta \mathbf{O}_{t+1}^k)^{\top} \mathbf{a}^{\top}}{\sqrt{d}} \right)$$
$$= \mathcal{S} \left(\mathbf{z} + \Delta \mathbf{z} \right).$$
(25)

For each component,

$$\mathcal{B}_{i} = \mathcal{S} \left(\mathbf{z} + \Delta \mathbf{z} \right)_{i}$$

$$= \mathcal{S} \left(\mathbf{z} \right)_{i} + \mathcal{A}_{i}$$

$$= \mathcal{S} \left(\mathbf{z} \right)_{i} + S_{i} \Delta \mathbf{z}_{i} - S_{i} \sum_{j=1}^{N} S_{j} \Delta \mathbf{z}_{j}.$$
(26)

Since $\Delta \mathbf{z}$ and $\Delta \mathbf{O}_{t+1}^v$ are small, we then get

$$\mathcal{B} \cdot \mathbf{a} \Delta \mathbf{O}_{t+1}^{v} \approx \mathcal{S}(\mathbf{z}) \cdot \mathbf{a} \Delta \mathbf{O}_{t+1}^{v}$$

$$= \mathcal{S}\left(\frac{\mathbf{Q}(\mathbf{W}_{t}^{k})^{\top} \mathbf{a}^{\top}}{\sqrt{d}}\right) \cdot \mathbf{a} \Delta \mathbf{O}_{t+1}^{v}$$

$$= \mathcal{S}\left(\frac{\mathbf{Q}(\mathbf{K})^{\top}}{\sqrt{d}}\right) \cdot \mathbf{a} \Delta \mathbf{O}_{t+1}^{v}.$$
(27)

Combining $A \cdot \mathbf{a} \mathbf{W}_t^v$ and $B \cdot \mathbf{a} \Delta \mathbf{O}_{t+1}^v$, we get

$$\Delta \mathbf{h}^{(l)} \approx \Gamma \cdot \frac{\mathbf{Q}^{(l)} \left(\mathbf{a}^{(l)} \Delta \mathbf{O}_{t+1}^{k} \right)^{\top}}{\sqrt{d}} \cdot \mathbf{V}^{(l)} + \mathcal{S} \left(\frac{\mathbf{Q}^{(l)} \left(\mathbf{K}^{(l)} \right)^{\top}}{\sqrt{d}} \right) \cdot \mathbf{a}^{(l)} \Delta \mathbf{O}_{t+1}^{v}.$$
(28)

Appendix A.2: Major Change of $\Delta \mathbf{h}_1^{(l)}$

We aim to use gradient projection to **mitigate** forgetting, similar to prior gradient projection schemes such as GPM [19] and InfLoRA [12]. In practice, strictly performing orthogonal gradient descent can preserve previously learned features but also restrict the subspace for fine-tuning on new tasks. Therefore, these schemes typically set a threshold value, such as $\epsilon = 0.95$, to extract a subset of significant core bases, selecting the top 95% of singular values to construct the feature subspaces. Although these CL schemes cannot perfectly preserve previously learned representations, a significant reduction in changes can effectively mitigate forgetting.

Similarly, our DualLoRA extracts only the feature subspace relevant to the class token and performs gradient projection to reduce the change in $\Delta \mathbf{h}_1^{(l)}$. We acknowledge that we cannot keep the class token unchanged but aim to prevent major changes to it. In the main paper, we simplified the error which can be generalized as

$$\Delta \mathbf{h}^{(l)} \approx \Gamma \cdot \frac{\left(\mathbf{Q}^{(l)} + \Delta \mathbf{Q}^{(l)}\right) \left(\Delta \mathbf{O}_{t+1}^{k}\right)^{\top} \left(\mathbf{a}^{(l)} + \Delta \mathbf{a}^{(l)}\right)^{\top}}{\sqrt{d}} \cdot \mathbf{V}^{(l)}
+ \Gamma \cdot \frac{\left(\mathbf{Q}^{(l)} + \Delta \mathbf{Q}^{(l)}\right) \left(\Delta \mathbf{O}_{t+1}^{k}\right)^{\top} \left(\mathbf{a}^{(l)} + \Delta \mathbf{a}^{(l)}\right)^{\top}}{\sqrt{d}} \cdot \Delta \mathbf{V}^{(l)}
+ \operatorname{softmax} \left(\frac{\left(\mathbf{Q}^{(l)} + \Delta \mathbf{Q}^{(l)}\right) \left(\mathbf{K}^{(l)} + \Delta \mathbf{K}^{(l)}\right)^{\top}}{\sqrt{d}}\right) \cdot \mathbf{a}^{(l)} \Delta \mathbf{O}_{t+1}^{v}
+ \operatorname{softmax} \left(\frac{\left(\mathbf{Q}^{(l)} + \Delta \mathbf{Q}^{(l)}\right) \left(\mathbf{K}^{(l)} + \Delta \mathbf{K}^{(l)}\right)^{\top}}{\sqrt{d}}\right) \cdot \Delta \mathbf{a}^{(l)} \Delta \mathbf{O}_{t+1}^{v}, \tag{29}$$

where

$$\Delta \mathbf{a}^{(l)} = \Delta \mathbf{h}^{(l-1)} \cdot \mathbf{W}_{\text{FFN}},\tag{30}$$

$$\Delta \mathbf{Q}^{(l)} = \Delta \mathbf{h}^{(l-1)} \cdot \mathbf{W}_{\text{FFN}} \cdot \mathbf{W}^{q}, \tag{31}$$

$$\Delta \mathbf{V}^{(l)} = \Delta \mathbf{h}^{(l-1)} \cdot \mathbf{W}_{\text{FFN}} \cdot (\mathbf{W}^v + \mathbf{O}^v)$$
(32)

$$\Delta \mathbf{K}^{(l)} = \Delta \mathbf{h}^{(l-1)} \cdot \mathbf{W}_{\text{FFN}}(\mathbf{W}^{v} + \mathbf{O}^{k}). \tag{33}$$

In the above formula, \mathbf{W}_{FFN} denotes feedforward network. Assuming $\Delta \mathbf{h}_1^{(l-1)} = \mathbf{0}$, then $\Delta \mathbf{a}_1^{(l)} = \Delta \mathbf{Q}_1^{(l)} = \Delta \mathbf{K}_1^{(l)} = \Delta \mathbf{V}_1^{(l)} = \mathbf{0}$. Since there are two terms on the right-hand side, we consider

$$\Delta \mathbf{h}^{(l)} \approx \mathcal{K} + \mathcal{V},$$
 (34)

where \mathcal{K} is relevant to the updates $\Delta \mathbf{O}_{t+1}^k$ while \mathcal{V} is relevant to the updates $\Delta \mathbf{O}_{t+1}^v$. For clarity, we omit the superscript l and subscript t+1 and recall that $\Delta \mathbf{h} \in \mathbb{R}^{n \times d}$, $\mathbf{a} \in \mathbb{R}^{n \times d}$, $\Delta \mathbf{O}^k \in \mathbb{R}^{d \times d}$, $\Delta \mathbf{O}^v \in \mathbb{R}^{d \times d}$, $\mathbf{V} \in \mathbb{R}^{n \times d}$, $\mathbf{Q} \in \mathbb{R}^{n \times d}$, $\mathbf{K} \in \mathbb{R}^{n \times d}$. Let consider the first term \mathcal{K} , (ignoring the constants),

$$(\mathbf{Q} + \Delta \mathbf{Q}) \left(\Delta \mathbf{O}^{k} \right)^{\top} (\mathbf{a} + \Delta \mathbf{a})^{\top} \cdot (\mathbf{V} + \Delta \mathbf{V}) =$$

$$\begin{bmatrix} (\mathbf{Q}_{1} + \mathbf{0}) \cdot (\Delta \mathbf{O}^{k})^{\top} \cdot (\mathbf{a} + \Delta \mathbf{a})^{\top} \\ \vdots \\ (\mathbf{Q}_{n} + \Delta \mathbf{Q}_{n}) \cdot (\Delta \mathbf{O}^{k})^{\top} \cdot (\mathbf{a} + \Delta \mathbf{a})^{\top} \end{bmatrix} (\mathbf{V}^{(l)} + \Delta \mathbf{V}^{(l)}),$$
(35)

where $\mathbf{Q}_i \in \mathbb{R}^{1 \times d}$ is the *i*-th row of \mathbf{Q} . We randomly select m samples for computing the layer-wise features set $\mathbf{k}^{(l)}$ including m varying \mathbf{Q}_1 vectors and extracting the core bases $\mathbf{\Phi}^k$. By projection, we update $\Delta \mathbf{O}^k$ by

$$(\Delta \mathbf{O}^k)^{\top} \leftarrow \left(\mathbf{I} - (\mathbf{\Phi}^k)^{\top} \mathbf{\Phi}^k \right) \Delta (\mathbf{O}^k)^{\top}. \tag{36}$$

Ideally, we can get

$$\mathbf{Q}_1 \cdot \left(\mathbf{I} - (\mathbf{\Phi}^k)^\top \mathbf{\Phi}^k \right) \Delta (\mathbf{O}^k)^\top \approx \mathbf{0} \cdot \Delta (\mathbf{O}^k)^\top = \mathbf{0}. \tag{37}$$

Therefore, we can constraint \mathcal{K}_1 , the first row of \mathcal{K} , in a small value close to zero. Similarly, let consider the value of \mathcal{V} . Let $\mathbf{X} = \operatorname{softmax}\left(\frac{(\mathbf{Q} + \Delta \mathbf{Q})(\mathbf{K} + \Delta \mathbf{K})^{\top}}{\sqrt{d}}\right) \in \mathbb{R}^{n \times n}$, we can ignore the higher order infinitesimal since $\Delta \mathbf{Q}(\Delta \mathbf{K})^{\top} \propto \Delta \mathbf{h}^{(l-1)}(\Delta \mathbf{h}^{(l-1)})^{\top}$, then

$$\mathbf{X} = \operatorname{softmax} \left(\frac{\mathbf{Q} \mathbf{K}^{\top}}{\sqrt{d}} + \frac{\Delta \mathbf{Q} \mathbf{K}^{\top}}{\sqrt{d}} + \frac{\mathbf{Q} \Delta \mathbf{K}^{\top}}{\sqrt{d}} \right).$$
 (38)

Therefore,

$$\mathbf{X}_{1} = \operatorname{softmax} \left(\frac{1}{\sqrt{d}} \mathbf{Q}_{1} \mathbf{K}^{\top} + \frac{1}{\sqrt{d}} \Delta \mathbf{Q}_{1} \mathbf{K}^{\top} + \frac{1}{\sqrt{d}} \mathbf{Q}_{1} \Delta \mathbf{K}^{\top} \right)$$

$$= \operatorname{softmax} \left(\frac{1}{\sqrt{d}} \mathbf{Q}_{1} \mathbf{K}^{\top} + \frac{1}{\sqrt{d}} \mathbf{Q}_{1} \Delta \mathbf{K}^{\top} \right).$$
(39)

According to derivative of softmax function, we get

$$\mathbf{X}_{1} = \operatorname{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_{1}\mathbf{K}^{\top}\right) + \mathbf{H}_{1}\frac{1}{\sqrt{d}}\mathbf{Q}_{1}\Delta\mathbf{K}^{\top},\tag{40}$$

where H is the Jacobian matrix defined as

$$\mathbf{H} = \begin{bmatrix} \mathbf{p}_{1}(1-\mathbf{p}_{1}) & -\mathbf{p}_{1}\mathbf{p}_{2} & \cdots & -\mathbf{p}_{1}\mathbf{p}_{n} \\ -\mathbf{p}_{2}\mathbf{p}_{1} & \mathbf{p}_{2}(1-\mathbf{p}_{2}) & & & \\ \vdots & & \ddots & \vdots \\ -\mathbf{p}_{n}\mathbf{p}_{1} & -\mathbf{p}_{n}\mathbf{p}_{2} & \cdots & \mathbf{p}_{n}(1-\mathbf{p}_{n}) \end{bmatrix}, \tag{41}$$

where \mathbf{p}_i is the *i*-th component of softmax $\left(\frac{1}{\sqrt{d}}\mathbf{Q}_1\mathbf{K}^{\top}\right)$. Therefore, the first row of \mathcal{V} can be found as:

$$\mathcal{V}_{1} = \mathbf{X}_{1} \cdot (\mathbf{a} + \Delta \mathbf{a}) \Delta \mathbf{O}^{v}
= \operatorname{softmax} \left(\frac{1}{\sqrt{d}} \mathbf{Q}_{1} \mathbf{K}^{\top} \right) \mathbf{a} \Delta \mathbf{O}^{v}
+ \operatorname{softmax} \left(\frac{1}{\sqrt{d}} \mathbf{Q}_{1} \mathbf{K}^{\top} \right) \Delta \mathbf{a} \Delta \mathbf{O}^{v}
+ \mathbf{H}_{1} \frac{1}{\sqrt{d}} \mathbf{Q}_{1} \Delta \mathbf{K}^{\top} \mathbf{a} \Delta \mathbf{O}^{v} + \mathbf{H}_{1} \frac{1}{\sqrt{d}} \mathbf{Q}_{1} \Delta \mathbf{K}^{\top} \Delta \mathbf{a} \Delta \mathbf{O}^{v},$$
(42)

where $\mathbf{X}_1 \in \mathbb{R}^{1 \times n}$, $\mathbf{a} \in \mathbb{R}^{n \times d}$ and $\mathbf{O}^v \in \mathbb{R}^{d \times d}$. Similarly, we can eliminate the fourth term since higher order infinitesimal $\Delta \mathbf{K}^{\top} \Delta \mathbf{a} \propto \Delta \mathbf{h}^{(l-1)} (\Delta \mathbf{h}^{(l-1)})^{\top} \approx \mathbf{0}$. According to our methodology, we collect feature set $\mathbf{s}^{(l)}$ and project $\Delta \mathbf{O}^v$ to constraint

$$\operatorname{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_{1}\mathbf{K}^{\top}\right)\mathbf{a}\Delta\mathbf{O}^{v} = \mathbf{0}.\tag{43}$$

In summary, if we guarantee the change of activation from last layer $\Delta \mathbf{h}^{(l-1)} \approx \mathbf{0}$, then

$$\Delta \mathbf{h}_{1}^{(l)} \approx \mathcal{K}_{1} + \mathcal{V}_{1} = \operatorname{softmax} \left(\frac{1}{\sqrt{d}} \mathbf{Q}_{1} \mathbf{K}^{\top} \right) \Delta \mathbf{a} \Delta \mathbf{O}^{v}$$

$$+ \mathbf{H}_{1} \frac{1}{\sqrt{d}} \mathbf{Q}_{1} \Delta \mathbf{K}^{\top} \mathbf{a} \Delta \mathbf{O}^{v}.$$

$$(44)$$

Although $\Delta \mathbf{a}$, $\Delta \mathbf{O}^v$ and $\Delta \mathbf{K}$ are not the same infinitesimal, but $\Delta \mathbf{a} \propto \eta$, $\Delta \mathbf{O}^v \propto \eta$ and $\Delta \mathbf{K} \propto \eta$, where η is the learning rate. Therefore,

$$\Delta \mathbf{h}_1^{(l)} \propto \eta^2. \tag{45}$$

If η is sufficiently small, we can have a very small change on $\Delta \mathbf{h}_1^{(l)}$.

Appendix B.1: Experimental Details

In this section, we report hyper-parameters used in different methods in Table. 1. In the table, we use η to denote learning rate; p denotes total number of prompts in the prompt pool; e denotes length of prompts and k denotes the number of prompts needed to match input images; e denotes the number of layer expanding parameters. For DualPrompt, e and e denote the length of E-prompts and G-prompt, respectively; e and e denotes the number of layers instructed by E-prompt and G-prompt. e denotes rank of LoRA parameters; e denotes the accumulated singular value for extracting bases in SVD.

Appendix B.2: FLOPs Computation

In this section, we present formulas to estimate floating point operations (FLOPs) to facilitate the comparison of computational demands across different methods. For simplicity, we concentrate on matrix multiplication and disregard operations with minor computational costs, such as addition, dropout, normalization and computing activation. For two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, the FLOPs for multiplication can be found as 2mnp.

Forward Pass in ViT

Multi-Head Attention. Computation in multi-head attention block involves in

• Obtaining Q, K and V by multiplying x with \mathbf{W}^q , \mathbf{W}^k and \mathbf{W}^v . Since $\mathbf{x} \in \mathbb{R}^{b \times n \times d}$, \mathbf{W}^q , \mathbf{W}^k , $\mathbf{W}^v \in \mathbb{R}^{d \times d}$, FLOPs in this step can be found as:

$$\mathbf{FLOPs} = 3 \cdot 2 \cdot bnd^2 = 6bnd^2,\tag{46}$$

where b is batch size, n is length of sequence and d denotes the dimension of embedding.

• Computing attention score S by multiplying Q and K. FLOPs in this step can be found as:

$$\mathbf{FLOPs} = 2bn^2d. \tag{47}$$

• Computing output signal h by multiplying S and V, FLOPs in this step can be found as :

$$\mathbf{FLOPs} = 2bn^2d. \tag{48}$$

• Linear projection on the same shape with h, FLOPs in this step can be found as:

$$\mathbf{FLOPs} = 2bnd^2. \tag{49}$$

Overall, the total FLOPs needed each MHA block for one batch is $8bnd^2 + 4bn^2d$.

The Feedforward Network (FFN). There are two linear projection for decoding and encoding output signals h in FFN. Suppose the ratio is set to 4, the FLOPs can be found as:

$$\mathbf{FLOPs} = 2 \cdot 8bnd^2 = 16bnd^2 \tag{50}$$

Since ViT model does not need to compute word embedding in the output layer for each block, we do not consider computation in this part. Suppose there are L blocks in the ViT model, the total FLOPs needed in the forward pass of ViT can be computed as $\mathbf{FLOPs} = L(24bnd^2 + 4bn^2d)$.

Table 5. List of Hyper-parameters used in different schemes.

Method	Hyper-parameters
L2P	η: 0.03 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	p: 30 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	e: 20 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	k: 5 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	l: 1 (ImageNet-R, CIFAR100, Tiny-ImageNet)
DualPrompt	η: 0.03 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	p: 5 (5 Tasks), 10 (10 Tasks), 20 (20 Tasks)
	e_E : 20 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	e _G : 6 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	k: 5 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	l_E : 3 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	$l_G: 2$ (ImageNet-R, CIFAR100, Tiny-ImageNet)
PGP	η : 0.05 (ImageNet-R, Tiny-ImageNet), 0.03 (CIFAR100)
	p: 5 (5 Tasks), 10 (10 Tasks), 20 (20 Tasks)
	e_E : 20 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	e_G : 6 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	k: 5 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	$l_E: 3$ (ImageNet-R, CIFAR100, Tiny-ImageNet)
	$l_G: 2$ (ImageNet-R, CIFAR100, Tiny-ImageNet)
SPrompt	η: 0.0005 (ImageNet-R, CIFAR100, Tiny-ImageNet)
•	p: 5 (5 Tasks), 10 (10 Tasks), 20 (20 Tasks)
	e: 10 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	k: 5 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	l: 1 (ImageNet-R, CIFAR100, Tiny-ImageNet)
CodaPrompt	η : 0.0005 (ImageNet-R, CIFAR100, Tiny-ImageNet)
•	p: 100 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	e: 8 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	k: 5 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	l: 5 (ImageNet-R, CIFAR100, Tiny-ImageNet)
InfLoRA	η : 0.0005 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	r: 10 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	ϵ : 0.95 (CIFAR100), 0.98 (ImageNet-R, Tiny-ImageNet)
	l: 12 (CIFAR100), 0.98 (ImageNet-R, Tiny-ImageNet)
DualLoRA	η : 0.0005 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	r: 10 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	ϵ : 0.95 (ImageNet-R, CIFAR100, Tiny-ImageNet)
	m: 200 (5 Tasks), 150 (10 Tasks), 100 (20 Tasks)
	λ : 2 (CIFAR100, 10-Split ImageNet-R), 1.5 (5-Split ImageNet-R, Tiny-ImageNet)
	1.2 (20-Split ImageNet-R)
	l: 12 (CIFAR100), 0.98 (ImageNet-R, Tiny-ImageNet)

Backward Pass in ViT

The FLOPs required for the backward pass are simply double those needed for the forward pass with the same model. There, the FLOPs needed for backward pass can be found as:

$$\mathbf{FLOPs} = 2L(24bnd^2 + 4bn^2d) \tag{51}$$

Singular Value Decomposition

SVD of a matrix $\mathbf{A} \in \mathbb{R}^{d \times m}$ typically involves two phases: (1) reduction to bidiagonal form and (2) performing the decomposition using the Golub-Kahan algorithm. The second phase is iterative, making it difficult to determine the exact FLOPs required. However, we focus on the FLOPs needed for the first phase, as it dominates the overall computational cost.

According to the textbook [21], the FLOPs for the first phase can be found as

$$\mathbf{FLOPs} = 2dm^2 + 11m^3. \tag{52}$$

Forward Pass in LoRA module

LoRA parameters are assigned parallel to the pre-trained weights \mathbf{W}^k and \mathbf{W}^v causing additional FLOPs to forward the signals. Since LoRA parameters for each per-trained weight consist of $\mathbf{A} \in \mathbb{R}^{d \times r}$ and $\mathbf{B}^{r \times d}$, the additional FLOPs can be found as

$$FLOPs = L \cdot 2 \cdot 2bndr = 8Lndr \tag{53}$$

Forward Pass in DualLoRA module

Compared to original LoRA, DualLoRA assigns an additional residual adapter parallel to the value weight \mathbf{W}^v . Therefore, the addintional FLOPs in DualLoRA can be found as

$$\mathbf{FLOPs} = L \cdot (2+1) \cdot 2 \cdot 2bndr = 12Lndr \tag{54}$$

Overall FLOPs in L2P

During the training and inference phases, L2P [26] first forwards image tokens to the original pre-trained encoder to obtain the key needed for matching prompt vectors in the prompt pool. Then, the selected prompt vectors are concatenated with the image tokens and forwarded into the encoder again. Therefore, the forward pass computation needs to be counted twice. For simplicity, we ignore the computation needed in the minimizing problem to select the top k prompt vectors because the FLOPs is depending on the optimization algorithm. Therefore, we can getting a lower bound for the overall FLOPs in L2P. For training phase, the overall FLOPs for a batch data can be found as

FLOPs
$$\geq 3 \cdot L \left(24b(n+ke)d^2 + 4b(n+ke)^2 d \right) + L \left(24bnd^2 + 4bn^2 d \right),$$
 (55)

where e is the length of prompt vectors. And the overall FLOPs for the inference phase can be found as:

FLOPs
$$\geq L \left(24b(n+ke)d^2 + 4b(n+ke)^2 d \right) + L \left(24bnd^2 + 4bn^2 d \right)$$
 (56)

Overall FLOPs in DualPrompt

DualPrompt [25] has a workflow similar to L2P but processes prompt vectors in only a subset of layers. For simplicity, we ignore the matching algorithm in DualPrompt and estimate the lower bound of the overall FLOPs. Therefore, the overall FLOPs can be calculated as follows:

$$\mathbf{FLOPs} \ge 72Lb(n + \frac{2e_G}{L} + \frac{3ke_E}{L})d^2
+ 12Lb(n + \frac{2e_G}{L} + \frac{3ke_E}{L})^2d + L\left(24bnd^2 + 4bn^2d\right),$$
(57)

where e_G is the length of G-prompts and e_E is the length of E-prompts. Similarly, for inference phase:

$$FLOPs \ge L \left(24b(n + e_G + ke_E)d^2 + 24bnd^2 + 4bn^2d \right) + 4Lb(n + \frac{2e_G}{L} + \frac{3ke_E}{L})^2d.$$
(58)

Overall FLOPs in CODAPrompt

CodaPrompt [20] requires significantly more FLOPs for optimizing the prompt keys and prompt pool. However, quantifying the exact number of FLOPs is challenging due to its dependence on the minimization algorithm. In addition to this computation, CodaPrompt increases the size of the matching prompt vectors for the first l layers. Therefore, the lower bound for the overall FLOPs in the training phase can be estimated as follows:

FLOPs
$$\geq 3 \cdot l \left(24b(n + \frac{1+l}{2}ke)d^2 + 4b(n + \frac{1+l}{2}ke)^2 d \right)$$

 $+ 3 \cdot (L-l) \left(24b(n+lke)d^2 + 4b(n+lke)^2 d \right)$
 $+ L \left(24bnd^2 + 4bn^2 d \right).$ (59)

And the overall FLOPs for the inference phase can be found as:

FLOPs
$$\geq l \left(24b(n + \frac{1+l}{2}ke)d^2 + 4b(n + \frac{1+l}{2}ke)^2 d \right) + (L-l) \left(24b(n+lke)d^2 + 4b(n+lke)^2 d \right) + L \left(24bnd^2 + 4bn^2 d \right),$$
(60)

Overall FLOPs in InfLoRA

To obtain the gradient subspace for each task, InfLoRA [12] requires forwarding the entire training dataset through the model and performing SVD on the collected average gradient. Additionally, InfLoRA forwards the data through the model once more for parameter updates. Therefore, there is also twice FLOPs in forward pass in the training phase but only one forward pass in the inference phase. Combing the FLOPs in forward pass, LoRA pass and SVD, the overall FLOPs in the training phase can be found as

$$FLOPs = 4L (24bnd^2 + 4bn^2d) + 8Lndr + 13Ld^3,$$
(61)

And the overall FLOPs for the inference phase can be found as:

$$\mathbf{FLOPs} = L\left(24bnd^2 + 4bn^2d\right) + 8Lndr,\tag{62}$$

Overall FLOPs in DualLoRA

Since DualLoRA does not require forwarding the training data twice during training, the overall FLOPs in the training phase can be found as

$$FLOPs = 3L (24bnd^{2} + 4bn^{2}d) + 12Lndr + L(2dm^{2} + 11m^{3}),$$
(63)

And the overall FLOPs for the inference phase can be found as:

$$\mathbf{FLOPs} = L\left(24bnd^2 + 4bn^2d\right) + 12Lndr,\tag{64}$$