Replay-Free Continual Low-Rank Adaptation with Dynamic Memory
Supplementary Material

Huancheng Chen[1,2], Jingtao Li[1], Weiming Zhuang[1], Chen Chen[1], Lingjuan Lyu[1,*]
[1]Sony AI, [2]University of Texas at Austin
huanchengch@utexas.edu, {jingtao.li, weiming.zhuang, ChenA.Chen,lingjuan.lv}@sony.com

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.

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

where

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

$$
\mathcal{B} = \mathcal{S}\left(\frac{\mathbf{Q}(\mathbf{W}_{t}^k + \Delta\mathbf{O}_{t+1}^k)^\top \mathbf{a}^\top}{\sqrt{d}}\right). \tag{2}
$$

Then, considering

$$
\mathcal{A} = \mathcal{S}\left(\mathbf{z} + \Delta\mathbf{z}\right) - \mathcal{S}\left(\mathbf{z}\right), \tag{3}
$$

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 $\left\|\Delta\mathbf{O}_{t+1}^k\right\|$ is proportional to learning rate η, by selecting a small learning rate, we can get $\|\Delta\mathbf{z}\| \ll \|\mathbf{z}\|$. Then considering

$$
\mathcal{A}_i = \mathcal{S}\left(\mathbf{z} + \Delta\mathbf{z}\right)_i - \mathcal{S}\left(\mathbf{z}\right)_i = \sum_{j=1}^{N}\frac{\partial S_i}{\partial \mathbf{z}_j}\Delta\mathbf{z}_j, \tag{4}
$$

where $S_i = \mathcal{S}\left(\mathbf{z}\right)_i < 1$ is the i-th component of $\mathcal{S}\left(\mathbf{z}\right)$; $\Delta\mathbf{z}_j$ 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 $\frac{\partial S_i}{\partial \mathbf{z}_j} = -S_iS_j$, otherwise, we can obtain:

$$
\begin{aligned}
\mathcal{A}_i &= S_i(1-S_i)\Delta\mathbf{z}_i - S_i\sum_{j\neq i} S_j\Delta\mathbf{z}_j \\
&= S_i\Delta\mathbf{z}_i - S_i\sum_{j=1}^{N} S_j\Delta\mathbf{z}_j \\
&\leq S_i\Delta\mathbf{z}_i - S_i\min_j(\Delta\mathbf{z}_j) \\
&= \left(S_i - \frac{\min_j(\Delta\mathbf{z}_j)}{\Delta\mathbf{z}_i}\right)\Delta\mathbf{z}_i.
\end{aligned} \tag{5}
$$

Similarly, we can get

$$
\left(S_i - \frac{\max_j(\Delta\mathbf{z}_j)}{\Delta\mathbf{z}_i}\right) \leq \frac{\mathcal{A}_i}{\Delta\mathbf{z}_i} \leq \left(S_i - \frac{\min_j(\Delta\mathbf{z}_j)}{\Delta\mathbf{z}_i}\right). \tag{6}
$$

Let $\mathcal{A}_i = \gamma_i\Delta\mathbf{z}_i$ 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 $\mathcal{A} = \Gamma\cdot\Delta\mathbf{z}$ where $\Gamma = \text{diag}\{\gamma_1,\ldots,\gamma_N\}$. Then we refocus on $\mathcal{B}$ such that

$$
\begin{aligned}
\mathcal{B} &= \mathcal{S}\left(\frac{\mathbf{Q}(\mathbf{W}_{t}^k + \Delta\mathbf{O}_{t+1}^k)^\top \mathbf{a}^\top}{\sqrt{d}}\right) \\
&= \mathcal{S}\left(\mathbf{z} + \Delta\mathbf{z}\right).
\end{aligned} \tag{7}
$$

For each component,

$$
\begin{aligned}
\mathcal{B}_i &= \mathcal{S}\left(\mathbf{z} + \Delta\mathbf{z}\right)_i \\
&= \mathcal{S}\left(\mathbf{z}\right)_i + \mathcal{A}_i \\
&= \mathcal{S}\left(\mathbf{z}\right)_i + S_i\Delta\mathbf{z}_i - S_i\sum_{j=1}^{N} S_j\Delta\mathbf{z}_j.
\end{aligned} \tag{8}
$$

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

$$\begin{aligned}\mathcal{B} \cdot \mathbf{a}\Delta\mathbf{O}_{t+1}^v &\approx \mathcal{S}\left(\mathbf{z}\right) \cdot \mathbf{a}\Delta\mathbf{O}_{t+1}^v \\ &= \mathcal{S}\left(\frac{\mathbf{Q}(\mathbf{W}_t^k)^\top \mathbf{a}^\top}{\sqrt{d}}\right) \cdot \mathbf{a}\Delta\mathbf{O}_{t+1}^v \\ &= \mathcal{S}\left(\frac{\mathbf{Q}\left(\mathbf{K}\right)^\top}{\sqrt{d}}\right) \cdot \mathbf{a}\Delta\mathbf{O}_{t+1}^v.\end{aligned} \tag{9}$$

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

$$\begin{aligned}\Delta \mathbf{h}^{(l)} \approx \Gamma \cdot \frac{\mathbf{Q}^{(l)}\left(\mathbf{a}^{(l)}\Delta\mathbf{O}_{t+1}^k\right)^\top}{\sqrt{d}} \cdot \mathbf{V}^{(l)} \\ + \mathcal{S}\left(\frac{\mathbf{Q}^{(l)}\left(\mathbf{K}^{(l)}\right)^\top}{\sqrt{d}}\right) \cdot \mathbf{a}^{(l)}\Delta\mathbf{O}_{t+1}^v.\end{aligned} \tag{10}$$

■

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 [2] and InfLoRA [1]. 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

$$\begin{aligned}\Delta \mathbf{h}^{(l)} \approx \Gamma \cdot \frac{(\mathbf{Q}^{(l)} + \Delta\mathbf{Q}^{(l)})\left(\Delta\mathbf{O}_{t+1}^k\right)^\top (\mathbf{a}^{(l)} + \Delta\mathbf{a}^{(l)})^\top}{\sqrt{d}} \cdot \mathbf{V}^{(l)} \\ + \Gamma \cdot \frac{(\mathbf{Q}^{(l)} + \Delta\mathbf{Q}^{(l)})\left(\Delta\mathbf{O}_{t+1}^k\right)^\top (\mathbf{a}^{(l)} + \Delta\mathbf{a}^{(l)})^\top}{\sqrt{d}} \cdot \Delta\mathbf{V}^{(l)} \\ + \text{softmax}\left(\frac{(\mathbf{Q}^{(l)} + \Delta\mathbf{Q}^{(l)})\left(\mathbf{K}^{(l)} + \Delta\mathbf{K}^{(l)}\right)^\top}{\sqrt{d}}\right) \cdot \mathbf{a}^{(l)}\Delta\mathbf{O}_{t+1}^v \\ + \text{softmax}\left(\frac{(\mathbf{Q}^{(l)} + \Delta\mathbf{Q}^{(l)})\left(\mathbf{K}^{(l)} + \Delta\mathbf{K}^{(l)}\right)^\top}{\sqrt{d}}\right) \cdot \Delta\mathbf{a}^{(l)}\Delta\mathbf{O}_{t+1}^v,\end{aligned} \tag{11}$$

where

$$\Delta \mathbf{a}^{(l)} = \Delta \mathbf{h}^{(l-1)} \cdot \mathbf{W}_{\text{FFN}}, \tag{12}$$

$$\Delta \mathbf{Q}^{(l)} = \Delta \mathbf{h}^{(l-1)} \cdot \mathbf{W}_{\text{FFN}} \cdot \mathbf{W}^q, \tag{13}$$

$$\Delta \mathbf{V}^{(l)} = \Delta \mathbf{h}^{(l-1)} \cdot \mathbf{W}_{\text{FFN}} \cdot (\mathbf{W}^v + \mathbf{O}^v) \tag{14}$$

$$\Delta \mathbf{K}^{(l)} = \Delta \mathbf{h}^{(l-1)} \cdot \mathbf{W}_{\text{FFN}} (\mathbf{W}^v + \mathbf{O}^k). \tag{15}$$

In the above formula, $\mathbf{W}_{\text{FFN}}$ 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

$$\Delta \mathbf{h}^{(l)} \approx \mathcal{K} + \mathcal{V}, \tag{16}$$

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),

$$\begin{aligned}&(\mathbf{Q} + \Delta\mathbf{Q})\left(\Delta\mathbf{O}^k\right)^\top (\mathbf{a} + \Delta\mathbf{a})^\top \cdot (\mathbf{V} + \Delta\mathbf{V}) = \\ &\begin{bmatrix} (\mathbf{Q}_1 + \mathbf{0}) \cdot (\Delta\mathbf{O}^k)^\top \cdot (\mathbf{a} + \Delta\mathbf{a})^\top \\ \vdots \\ (\mathbf{Q}_n + \Delta\mathbf{Q}_n) \cdot (\Delta\mathbf{O}^k)^\top \cdot (\mathbf{a} + \Delta\mathbf{a})^\top \end{bmatrix} \left(\mathbf{V}^{(l)} + \Delta\mathbf{V}^{(l)}\right),\end{aligned} \tag{17}$$

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

$$(\Delta\mathbf{O}^k)^\top \leftarrow \left(\mathbf{I} - (\mathbf{\Phi}^k)^\top \mathbf{\Phi}^k\right) \Delta(\mathbf{O}^k)^\top. \tag{18}$$

Ideally, we can get

$$\mathbf{Q}_1 \cdot \left(\mathbf{I} - (\mathbf{\Phi}^k)^\top \mathbf{\Phi}^k\right) \Delta(\mathbf{O}^k)^\top \approx \mathbf{0} \cdot \Delta(\mathbf{O}^k)^\top = \mathbf{0}. \tag{19}$$

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 $\mathbf{X} = \text{softmax}\left(\frac{(\mathbf{Q}+\Delta\mathbf{Q})(\mathbf{K}+\Delta\mathbf{K})^\top}{\sqrt{d}}\right) \in \mathbb{R}^{n \times n}$, 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

$$\mathbf{X} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} + \frac{\Delta\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} + \frac{\mathbf{Q}\Delta\mathbf{K}^\top}{\sqrt{d}}\right). \tag{20}$$

Therefore,

$$\begin{aligned}\mathbf{X}_1 &= \text{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_1\mathbf{K}^\top + \frac{1}{\sqrt{d}}\Delta\mathbf{Q}_1\mathbf{K}^\top + \frac{1}{\sqrt{d}}\mathbf{Q}_1\Delta\mathbf{K}^\top\right) \\ &= \text{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_1\mathbf{K}^\top + \frac{1}{\sqrt{d}}\mathbf{Q}_1\Delta\mathbf{K}^\top\right).\end{aligned} \tag{21}$$

According to derivative of softmax function, we get

$$\mathbf{X}_1 = \text{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_1\mathbf{K}^\top\right) + \mathbf{H}_1\frac{1}{\sqrt{d}}\mathbf{Q}_1\Delta\mathbf{K}^\top, \tag{22}$$

where $\mathbf{H}$ is the Jacobian matrix defined as

$$\mathbf{H} = \begin{bmatrix} \mathbf{p}_1(1-\mathbf{p}_1) & -\mathbf{p}_1\mathbf{p}_2 & \cdots & -\mathbf{p}_1\mathbf{p}_n \\ -\mathbf{p}_2\mathbf{p}_1 & \mathbf{p}_2(1-\mathbf{p}_2) & & \\ \vdots & & \ddots & \vdots \\ -\mathbf{p}_n\mathbf{p}_1 & -\mathbf{p}_n\mathbf{p}_2 & \cdots & \mathbf{p}_n(1-\mathbf{p}_n) \end{bmatrix}, \tag{23}$$

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:

$$\begin{aligned} \mathcal{V}_1 &= \mathbf{X}_1 \cdot (\mathbf{a} + \Delta\mathbf{a})\,\Delta\mathbf{O}^v \\ &= \text{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_1\mathbf{K}^\top\right)\mathbf{a}\Delta\mathbf{O}^v \\ &\quad + \text{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_1\mathbf{K}^\top\right)\Delta\mathbf{a}\Delta\mathbf{O}^v \\ &\quad + \mathbf{H}_1\frac{1}{\sqrt{d}}\mathbf{Q}_1\Delta\mathbf{K}^\top\mathbf{a}\Delta\mathbf{O}^v + \mathbf{H}_1\frac{1}{\sqrt{d}}\mathbf{Q}_1\Delta\mathbf{K}^\top\Delta\mathbf{a}\Delta\mathbf{O}^v, \end{aligned} \tag{24}$$

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

$$\text{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_1\mathbf{K}^\top\right)\mathbf{a}\Delta\mathbf{O}^v = \mathbf{0}. \tag{25}$$

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

$$\begin{aligned} \Delta\mathbf{h}_1^{(l)} \approx \mathcal{K}_1 + \mathcal{V}_1 = \text{softmax}\left(\frac{1}{\sqrt{d}}\mathbf{Q}_1\mathbf{K}^\top\right)\Delta\mathbf{a}\Delta\mathbf{O}^v \\ + \mathbf{H}_1\frac{1}{\sqrt{d}}\mathbf{Q}_1\Delta\mathbf{K}^\top\mathbf{a}\Delta\mathbf{O}^v. \end{aligned} \tag{26}$$

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,

$$\Delta\mathbf{h}_1^{(l)} \propto \eta^2. \tag{27}$$

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. **??**. 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; l denotes the number of layer expanding parameters. For DualPrompt, e_E and e_G denote the length of E-prompts and G-prompt, respectively; l_E and l_G denotes the number of layers instructed by E-prompt and G-prompt. r denotes rank of LoRA parameters; ϵ 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 $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ by multiplying $\mathbf{x}$ 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:

$$\textbf{FLOPs} = 3 \cdot 2 \cdot bnd^2 = 6bnd^2, \tag{28}$$

  where b is batch size, n is length of sequence and d denotes the dimension of embedding.
- Computing attention score S by multiplying $\mathbf{Q}$ and $\mathbf{K}$. FLOPs in this step can be found as :

$$\textbf{FLOPs} = 2bn^2d. \tag{29}$$

- Computing output signal $\mathbf{h}$ by multiplying $\mathbf{S}$ and $\mathbf{V}$, FLOPs in this step can be found as :

$$\textbf{FLOPs} = 2bn^2d. \tag{30}$$

- Linear projection on the same shape with $\mathbf{h}$, FLOPs in this step can be found as:

$$\textbf{FLOPs} = 2bnd^2. \tag{31}$$

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 $\mathbf{h}$ in FFN. Suppose the ratio is set to 4, the FLOPs can be found as:

$$\textbf{FLOPs} = 2 \cdot 8bnd^2 = 16bnd^2 \tag{32}$$

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 compuated as $\textbf{FLOPs} = L(24bnd^2 + 4bn^2d)$.

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:

$$\textbf{FLOPs} = 2L(24bnd^2 + 4bn^2d) \tag{33}$$

Table 1. 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)

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 [4], the FLOPs for the first phase can be found as

$$\textbf{FLOPs} = 2dm^2 + 11m^3. \tag{34}$$

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

$$\textbf{FLOPs} = L \cdot 2 \cdot 2 \cdot 2bndr = 8Lndr \tag{35}$$

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

$$\textbf{FLOPs} = L \cdot (2+1) \cdot 2 \cdot 2bndr = 12Lndr \tag{36}$$

Overall FLOPs in L2P

During the training and inference phases, L2P [6] 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

$$\begin{aligned}\textbf{FLOPs} \geq\ & 3 \cdot L\left(24b(n+ke)d^2 + 4b(n+ke)^2d\right) \\ & + L\left(24bnd^2 + 4bn^2d\right),\end{aligned} \tag{37}$$

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

$$\begin{aligned}\textbf{FLOPs} \geq\ & L\left(24b(n+ke)d^2 + 4b(n+ke)^2d\right) \\ & + L\left(24bnd^2 + 4bn^2d\right)\end{aligned} \tag{38}$$

Overall FLOPs in DualPrompt

DualPrompt [5] 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:

$$\begin{aligned}\textbf{FLOPs} \geq\ & 72Lb(n + \frac{2e_G}{L} + \frac{3ke_E}{L})d^2 \\ & + 12Lb(n + \frac{2e_G}{L} + \frac{3ke_E}{L})^2d + L\left(24bnd^2 + 4bn^2d\right),\end{aligned} \tag{39}$$

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

$$\begin{aligned}\textbf{FLOPs} \geq\ & L\left(24b(n+e_G+ke_E)d^2 + 24bnd^2 + 4bn^2d\right) \\ & + 4Lb(n + \frac{2e_G}{L} + \frac{3ke_E}{L})^2d.\end{aligned} \tag{40}$$

Overall FLOPs in CODAPrompt

CodaPrompt [3] 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:

$$\begin{aligned}\textbf{FLOPs} \geq\ & 3 \cdot l\left(24b(n + \frac{1+l}{2}ke)d^2 + 4b(n + \frac{1+l}{2}ke)^2d\right) \\ & + 3 \cdot (L-l)\left(24b(n+lke)d^2 + 4b(n+lke)^2d\right) \\ & + L\left(24bnd^2 + 4bn^2d\right).\end{aligned} \tag{41}$$

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

$$\begin{aligned}\textbf{FLOPs} \geq\ & l\left(24b(n + \frac{1+l}{2}ke)d^2 + 4b(n + \frac{1+l}{2}ke)^2d\right) \\ & + (L-l)\left(24b(n+lke)d^2 + 4b(n+lke)^2d\right) \\ & + L\left(24bnd^2 + 4bn^2d\right),\end{aligned} \tag{42}$$

Overall FLOPs in InfLoRA

To obtain the gradient subspace for each task, InfLoRA [1] 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

$$\textbf{FLOPs} = 4L\left(24bnd^2 + 4bn^2d\right) + 8Lndr + 13Ld^3, \tag{43}$$

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

$$\textbf{FLOPs} = L\left(24bnd^2 + 4bn^2d\right) + 8Lndr, \tag{44}$$

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

$$\begin{aligned}\textbf{FLOPs} = & 3L\left(24bnd^2 + 4bn^2d\right) + 12Lndr \\ & + L(2dm^2 + 11m^3),\end{aligned} \tag{45}$$

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

$$\textbf{FLOPs} = L\left(24bnd^2 + 4bn^2d\right) + 12Lndr, \tag{46}$$

References

[1] Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. *arXiv preprint arXiv:2404.00228*, 2024.

[2] Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762*, 2021.

[3] 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 Conference on Computer Vision and Pattern Recognition*, pages 11909–11919, 2023.

[4] Lloyd N Trefethen and David Bau. *Numerical linear algebra*. SIAM, 2022.

[5] 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.

[6] 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.