CL-LoRA: Continual Low-Rank Adaptation for Rehearsal-Free Class-Incremental Learning

Supplementary Material

7. Extended Illustration of Methodology

7.1. Overview of Training Algorithm

Algorithm 1 presents the training process of our CL-LoRA for class-incremental learning. Given a sequence of tasks $\{\mathcal{T}_t\}_{t=1}^{T}$, our method learns each task sequentially while leveraging both task-shared and task-specific adapters. For the first task ($t = 1$), we initialize the shared adapter with a fixed random orthogonal down-projection matrix $\mathbf{B}_s$ and zero-initialized up-projection matrix $\mathbf{A}_s$ following Eq. 6. For each subsequent task, we copy the shared adapter from the previous task ($\mathbf{A}_s^t \leftarrow \mathbf{A}_s^{t-1}$) and initialize new components including: (1) task-specific adapters $\mathbf{A}_t, \mathbf{B}_t$ where $\mathbf{A}_t$ is initialized as zero and $\mathbf{B}_t$ follows Gaussian initialization as in [20], and (2) block weights $\mathbf{U}_t$ where each scaling factor μ_t is randomly initialized from the uniform distribution $\mathcal{U}(0, 2)$ to allow flexible modulation of task-specific adaptation. We also initialize a temporary local FC classifier h_ϕ^t for the current task's classes, which will be discarded after training as we transition to prototype-based inference. During the forward pass, we employ our dual-adapter architecture where shared adapters are applied to the first l blocks and task-specific adapters to the remaining blocks. The model is trained with three objectives: (1) a local cross-entropy loss $\mathcal{L}_{ce}$ for current task classification, (2) an early exit knowledge distillation loss $\mathcal{L}_{kd}$ at the transition point l to preserve cross-task knowledge when $t > 1$, and (3) an orthogonality loss $\mathcal{L}_{orth}$ between block weights to prevent interference. Notably, we introduce gradient reassignment based on the L_2 norm of previous shared adapter weights to effectively maintain essential knowledge during class-incremental learning. After training each task, we discard its task-specific classifier h_ϕ^t and compute class prototypes that will be used during inference.

Algorithm 1 Class-Incremental Learning with CL-LoRA

Require:
Backbone f_θ
Task sequence $\{\mathcal{T}_t\}_{t=1}^{T}$
Position of shared adapter l
for $t \leftarrow 1$ to T **do**
  if $t = 1$ **then** ▷ initial task
    $\mathbf{B}_s \leftarrow$ Random Orthogonal, $\mathbf{A}_s^0 \leftarrow \mathbf{0}$ ▷ Eq. 6
  else
    $\mathbf{A}_s^t \leftarrow \mathbf{A}_s^{t-1}$ ▷ Copy weights
  end if
  Initialize $\{\mathbf{A}_t, \mathbf{B}_t, \mathbf{U}_t, h_\phi^t\}$
  for $(x, y) \in \mathcal{T}_t$ **do**
    $z_t^0 \leftarrow x$
    for $i \leftarrow 1$ to N **do**
      if $i \leq l$ **then**
        $z_t^i \leftarrow f_\theta^i(z_t^{i-1}) + \mathbf{A}_s^i\mathbf{B}_s^i z_t^{i-1}$ ▷ Eq. 8
      else
        $z_t^i \leftarrow f_\theta^i(z_t^{i-1}) + \mu_t^i\mathbf{A}_t^i\mathbf{B}_t^i z_t^{i-1}$ ▷ Eq. 11
      end if
    end for
    $\mathcal{L}_{ce} \leftarrow \mathcal{L}_{ce}(h_\phi^t(z_t^N[CLS]), y)$ ▷ Eq. 2
    if $t > 1$ **then**
      $z_{t-1}^l \leftarrow f_{1:l}^{t-1}(x)$ ▷ Early exit
      $\mathcal{L}_{kd} \leftarrow z_t^l[CLS], z_{t-1}^l[CLS]$ ▷ Eq. 9
      $\mathcal{L}_{orth} \leftarrow \|\mathbf{U}_t^\top \mathbf{U}_{1:t-1}\|_2$ ▷ Eq. 12
      $\|\mathbf{a}_s^{t-1}\|_2 \leftarrow L_2(\mathbf{A}_s^{t-1})$
    end if
    $\mathcal{L} \leftarrow \mathcal{L}_{ce} + \lambda_1\mathcal{L}_{kd} + \lambda_2\mathcal{L}_{orth}$
    $\nabla_{\mathbf{A}_s^t}\mathcal{L}_{kd}^* \leftarrow \nabla_{\mathbf{A}_s^t}\mathcal{L}_{kd} \odot \sigma(\|\mathbf{a}_s^{t-1}\|_2)$ ▷ Eq. 10
    Update parameters via gradient descent
  end for
  Freeze and store $(\mathbf{A}_t, \mathbf{B}_t)$ and $\mathbf{U}_t$
  Discard h_ϕ^t
end for

7.2. Use of Random Orthogonal Matrix

Motivation: Our use of random orthogonal matrices in the shared adapter's down-projection is motivated by recent theoretical findings in [61], which demonstrates that fine-tuning the up-projection matrix is inherently more effective than tuning the down-projection matrix in LoRA. This asymmetric property suggests that a fixed down-projection matrix can maintain comparable performance to a fully trainable one while providing additional stability benefits for class-incremental learning. Indeed, our empirical results in Section 5.3 validate this insight where using untrained random orthogonal matrices for down-projection achieves better performance than regular trainable down-projection matrices. This suggests that fixing the low-dimensional projection space with the orthogonal structure provides a more stable foundation for cross-task knowledge accumulation in class-incremental learning.

Random Orthogonal v.s. Random: While a simple random matrix could serve as the fixed down-projection, we specifically choose orthogonal matrices because they preserve the geometric structure of the input space in the projected low-dimensional representation. Since $\mathbf{B}_s\mathbf{B}_s^\top =$

$\mathbf{I}$, orthogonal matrices maintain distances and angles between vectors during projection, ensuring that similar input patterns remain distinguishable in the lower-dimensional space. This property is essential for stable knowledge accumulation across tasks, as it prevents information collapse and maintains the discriminative power of learned features. Table 4 demonstrates the crucial importance of using random orthogonal matrices rather than standard random matrices for down-projection $\mathbf{B}_s$ in our task-shared adapter.

Down-Projection	CIFAR-100		ImageNet-R	
	A_T	$\overline{A}$	A_T	$\overline{A}$
Random $\mathbf{B}_s$	8.99	20.66	0.77	2.39
Random Orthogonal $\mathbf{B}_s$	85.96	91.85	78.85	84.77

Table 4. Comparison between random matrices and random orthogonal matrices for down-projection $\mathbf{B}_s$ on CIFAR-100 ($T = 10$) and ImageNet-R ($T = 20$). Results show last step accuracy A_T and average accuracy $\overline{A}$ (%).

In this work, as described in Section 4.1, we use SVD decomposition to generate the random orthogonal matrix $\mathbf{B}_s$ by first generating a random matrix $\mathbf{M} \sim \mathcal{N}(0, 1)$ followed by $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and setting $\mathbf{B}_s = \mathbf{U}\mathbf{V}^\top$. However, there are other efficient alternatives such as QR decomposition which could potentially offer better computational efficiency and numerical stability for generating random orthogonal matrices.

The Position of Shared Adapter: In this work, we insert the task-shared adapters in the first l blocks while inserting task-specific adapters in the last $N - l$ blocks. This design aligns with the hierarchical nature of vision transformers [37], where earlier layers tend to capture more general, transferable features while deeper layers specialize in task-specific representations. We further validate this design choice through ablation experiments where we flip the position of task-specific and task-shared adapters (*i.e.*, inserting the task-specific adapters in the first l blocks while using task-shared adapters in the last $N - l$ blocks).

As shown in Figure 5, the original configuration (**Ours**, with task-shared adapter in the first l blocks) consistently outperforms the flipped version (**Flip**, with task-specific adapter in the first l blocks) across different transition points l. Taking $l = 6$ as an example, our method achieves significantly better performance than the flipped version on ImageNet-R (79.16% v.s. 65.38%). The performance gap becomes even more obvious with smaller l values, where the flipped version shows substantial degradation (down to 58.02% when $l = 2$). This dramatic difference validates that earlier layers are more suitable for shared knowledge accumulation while later layers are better suited for task-specific adaptation. These findings align with recent studies on transformer feature hierarchies and demonstrate that

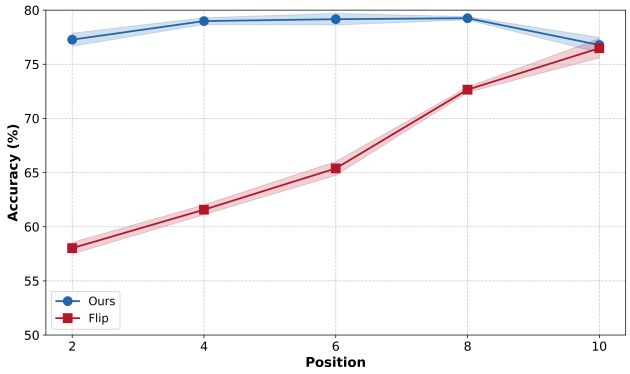

Figure 5. Performance comparison between ours design (task-shared adapters in first l blocks) and flipped configuration (task-specific adapters in first l blocks) with varying transition point $l \in \{2, 4, 6, 8, 10\}$. Results show final step accuracy A_T on ImageNet-R with $T = 20$ tasks.

respecting the natural progression from general to specific processing in vision transformers is crucial for effective class-incremental learning.

Difference with Existing Work: It is also important to note that our use of orthogonal matrices differs fundamentally from existing works like [26, 47], which employ orthogonality constraints between task-specific adapters to minimize interference, relying solely on task-specific LoRA. In contrast, our approach leverages random orthogonal matrices in the task-shared adapter to establish a fixed subspace for knowledge sharing across tasks. Specifically, by using fixed orthogonal down-projection $\mathbf{B}_s$, we ensure that $\mathbf{B}_s\mathbf{B}_s^\top = \mathbf{I}$, which creates a stable foundation for the trainable up-projection matrix $\mathbf{A}_s$ to accumulate shared knowledge while maintaining consistency in the low-dimensional space. This stability is particularly crucial for our dual-adapter architecture, as it allows the shared component to effectively preserve and transfer knowledge across tasks while the task-specific adapters handle unique characteristics through their own adaptation.

7.3. Detailed Experimental Setup

All experiments were conducted on NVIDIA A40 GPUs using PyTorch. For comprehensive evaluation and fair comparison, we run each experiment 10 times with randomly generated seeds. The experiments code implementations is based on the LAMDA-PILOT [43, 57, 59][2] and Mammoth [2, 3][3]. For existing methods that are not included in these two public codebases, we use their original official implementations with their reported best hyper-parameters. Importantly, all results reported in our paper are from our own reproduction using the same set of 10 random seeds

[2] https://github.com/sun-hailong/LAMDA-PILOT
[3] https://github.com/aimagelab/mammoth

rather than directly quoting numbers from original papers, ensuring a fair and consistent comparison across all methods. This reproduction effort helps eliminate potential variations due to different random seeds, hardware configurations, or implementation details that might exist in originally reported results.