SAME: Sparse and Anchored Model Editing for Heterogeneous Incremental Learning under Limited Data

Supplementary Material

7. Practical Motivation of Heterogeneous Incremental Learning

Heterogeneous Incremental Learning arises in practical systems where data streams mix in-domain and cross-domain tasks without task identifiers. For example, a vision-language assistant may continuously receive product images (in-domain) and later medical or satellite images (cross-domain), with only few-shot labels and no domain annotation. In such cases, many existing CIL and TIL methods, which explicitly or implicitly rely on domain homogeneity or cross-domain task priors, become difficult to apply in practice, motivating HIL as a more realistic setting.

8. Detailed Algorithm and Implementation

8.1. Implementation Notation

To provide a clearer understanding of the optimization process, Table 6 summarizes the key notations, tensor shapes, and their corresponding roles within the proposed SAME framework.

8.2. Core Solver and Implementation Snippet

We provide an implementation-oriented view of the SAME solver in Fig. 5. We assume that the required second-order statistics, $K_1^\top K_1$, $K_1^\top V_1$, $K_0^\top K_0$, and $K_0^\top V_0$, have already been computed. The code focuses on the closed-form resolution step in Eq. 8.

9. Theoretical and Complexity Analysis

9.1. Theoretical Consistency

We analyze the statistical consistency of SAME. Let

$$\begin{aligned} \widehat{A} &= \widehat{\Sigma}_{1,KK} + \lambda_k \widehat{\Sigma}_{0,KK} + \lambda_p I, \\ \widehat{B} &= \widehat{\Sigma}_{1,KV} + \lambda_k \widehat{\Sigma}_{0,KV} + \lambda_p W_{\text{out}}, \end{aligned} \tag{9}$$

where both the task-specific statistics $(\widehat{\Sigma}_{1,KK}, \widehat{\Sigma}_{1,KV})$ and the anchor statistics $(\widehat{\Sigma}_{0,KK}, \widehat{\Sigma}_{0,KV})$ are empirically estimated from data. Let their population counterparts be

$$\begin{aligned} A^\star &= \Sigma_{1,KK} + \lambda_k \Sigma_{0,KK} + \lambda_p I, \\ B^\star &= \Sigma_{1,KV} + \lambda_k \Sigma_{0,KV} + \lambda_p W_{\text{out}}, \end{aligned} \tag{10}$$

and define

$$\widehat{W} = \widehat{A}^{-1}\widehat{B}, \qquad W^\star = (A^\star)^{-1} B^\star. \tag{11}$$

Assuming bounded i.i.d. samples, standard matrix concentration yields

$$\|\widehat{A} - A^\star\|_F = O_p(N^{-1/2}), \qquad \|\widehat{B} - B^\star\|_F = O_p(N^{-1/2}). \tag{12}$$

Moreover, since $\lambda_p > 0$, we have

$$\widehat{A} \succeq \lambda_p I, \qquad \|\widehat{A}^{-1}\|_2 \le \frac{1}{\lambda_p}. \tag{13}$$

Using the perturbation identity,

$$\widehat{W} - W^\star = \widehat{A}^{-1}(\widehat{B} - B^\star) + \widehat{A}^{-1}(A^\star - \widehat{A})W^\star, \tag{14}$$

we obtain

$$\begin{aligned} \|\widehat{W} - W^\star\|_F &\le \|\widehat{A}^{-1}\|_2 \, \|\widehat{B} - B^\star\|_F \\ &\quad + \|\widehat{A}^{-1}\|_2 \, \|A^\star - \widehat{A}\|_F \, \|W^\star\|_2 \\ &= O_p(N^{-1/2}). \end{aligned} \tag{15}$$

Therefore, SAME is a consistent estimator: as $N \to \infty$, the empirical solution $\widehat{W}$ converges to the population minimizer $W^\star$ at rate $O_p(N^{-1/2})$.

9.2. Computational Complexity Analysis

The core editing step of SAME solves a symmetric linear system for each FFN layer. The dominant cost comes from matrix inversion, which requires $\mathcal{O}(d^3)$ operations per layer, where d is the FFN width. Across ℓ layers, the total complexity is $\mathcal{O}(\ell \cdot d^3)$.

Importantly, this closed-form computation is performed only once after aggregating the key–value statistics, and is independent of the number of sequential tasks. As shown in Table 7, the additional cost of SAME (5.27×10^{11} FLOPs) is negligible compared to the main fine-tuning stage (1.13×10^{16} FLOPs), resulting in minimal overhead and improved efficiency over prior methods.

10. Additional Experimental Results

10.1. Scalability to Diverse Backbones

To evaluate the scalability of SAME, we conduct experiments on CLIP models with different Transformer scales, including ViT-B/16 and ViT-L/14. We focus on Transformer backbones, as SAME operates on the FFN layers.

For these experiments, both backbones share the identical training budget and hyperparameters. As shown in Table 8, our method consistently yields significant improvements over the baselines (No Editing) across different model sizes, demonstrating its strong scalability and robustness.

Table 6. Detailed notation, tensor shapes, and their roles in the SAME framework.

Component	Symbol	Shape	Role
		Dimensions & Features	
Batch & Sequence	B, L	Scalar	Batch size and sequence length (number of tokens)
Hidden Dims	d_k, d_v	Scalar	Dimensions of key and value representations
Token Features	k, v	$B \times L \times d_k$, $B \times L \times d_v$	Extracted token-wise key and value features
Batch Features	k_b, v_b	$(BL) \times d_k$, $(BL) \times d_v$	Reshaped 2D batch-level key and value matrices
		Statistics & Anchors	
Pre-trained Cov	$K_0^\top K_0$	$d_k \times d_k$	Knowledge anchor covariance matrix from the pre-trained model
Global Key Cov	$\overline{K}_1^\top \overline{K}_1$	$d_k \times d_k$	Aggregated global key covariance from all current tasks
Global Cross-Cov	$\overline{K}_1^\top \overline{V}_1$	$d_k \times d_v$	Aggregated global cross-covariance from all current tasks
System Matrix	H	$d_k \times d_k$	Symmetric positive definite matrix for the closed-form solution
		Editing Parameters	
Target Weight	W_{out}	$d_k \times d_v$	FFN Output Projection Layer
Optimal Weight	W^*	$d_k \times d_v$	Edited optimal projection matrix
Binary Mask	M	$d_k \times d_v$	Gradient Filter ($\nabla \mathcal{L} \odot M$)
Anchor Weights	λ_k, λ_p	Scalar	Hyperparameters for Knowledge and Parameter constraints
Sparsity Ratio	p	Scalar	Proportion of parameters frozen during fine-tuning

```python
import torch

@torch.no_grad()
def solve_same(W0, K1TK1, K1TV1, K0TK0, K0TV0, lambda_k, lambda_p):
    """
    Closed-form SAME solver:
    H = K1^T K1 + lambda_k K0^T K0 + lambda_p I
    Solve: H W* = K1^T V1 + lambda_k K0^T V0 + lambda_p W0
    """

    d = W0.shape[0]
    I = torch.eye(d, device=W0.device, dtype=W0.dtype)

    H = K1TK1 + lambda_k * K0TK0 + lambda_p * I
    B = K1TV1 + lambda_k * K0TV0 + lambda_p * W0

    W_star = torch.linalg.solve(H, B)
    return W_star
```

Figure 5. Closed-form solver of SAME given pre-computed second-order statistics.

10.2. Forgetting and Transfer Analysis

To provide a more comprehensive evaluation of our method's ability to retain past knowledge and facilitate future learning, we further evaluate the Backward Transfer (BWT) and Forward Transfer (FWT) metrics.

As shown in Table 9, SAME demonstrates competitive performance on both BWT and FWT compared to existing continual learning baselines. Notably, while maintaining comparable transfer metrics, our method achieves the highest Last Accuracy (66.0%), indicating a better trade-off

Table 7. Comparison of theoretical computational cost (FLOPs) and practical runtime (seconds) across different continual learning methods.

Method	**FLOPs**	**Runtime (s)**
Full Finetune	2.20×10^{16}	2996
Wise-FT	2.20×10^{16}	3267
ZSCL	6.48×10^{16}	7499
GNSP	8.61×10^{16}	12431
SAME (Ours)	$1.13 \times 10^{16} + \mathbf{5.27 \times 10^{11}}$	**2961**

Table 8. Scalability evaluation on different CLIP Transformer backbones.

Backbone	**No Editing**	**Ours**	**Oracle**
CLIP ViT-B/16	57.1	66.0	68.9
CLIP ViT-L/14	66.4	70.4	73.1

between mitigating catastrophic forgetting and adapting to heterogeneous tasks.

Table 9. Evaluation of Backward Transfer (BWT), Forward Transfer (FWT), and Last Accuracy.

Metric	FT	WiSE-FT	ZSCL	GNSP	Ours
BWT ↑	-21.7	-14.7	-4.9	1.5	-1.9
FWT ↑	-16.9	-25.9	5.8	-3.6	-2.8
Last Acc ↑	46.0	50.8	56.1	58.7	**66.0**

10.3. Sensitivity Analysis for Anchor Weights

To further evaluate the robustness of our framework, we conduct a sensitivity analysis on the knowledge anchor weight λ_k, keeping all other hyperparameters fixed. As shown in Table 10, our proposed SAME framework achieves stable and consistently high performance across a wide range of values for λ_k.

Table 10. Sensitivity analysis on the knowledge anchor weight λ_k. The average accuracy remains stable across varying values.

λ_k	0.05	0.1	0.2	0.5	1.0	2.0
Avg Acc	65.7	**66.0**	65.6	64.9	64.4	62.6