# Low-Rank Adaptation in Multilinear Operator Networks for Security-Preserving Incremental Learning

Supplementary Material

## A. Proofs

**Proposition 1** (Restatement of Proposition 1)**.** *Fine-tuning $B_t$ is equivalent to fine-tuning the pre-trained weight $W_{t-1}$ within the subspace span$\{a_{t,1}, \ldots, a_{t,r}\}$, where $a_{t,i}$ $(1 \leq i \leq r)$ denotes the $i$-th row vector of $A_t$. This equivalence holds when learning the $t$-th task with forward propagation represented by:*

$$z = W_{t-1}x + \alpha_{\text{lora}} B_t A_t x. \tag{8}$$

*Here, all symbols have the same meaning as in Sec. 4.*

*Proof of Proposition 1.* We have the weight of the concatenated channel projection of the first $(t-1)$ tasks before task $t$-th and the weight of pre-trained channel projection represented by a matrix $W_{t-1} \in \mathbb{R}^{o \times c}$ and $W \in \mathbb{R}^{o \times c}$, respectively, the input tensor $x \in \mathbb{R}^{c \times h \times w}$. Then, $z \in \mathbb{R}^{o \times h \times w}$. With $i, j \in \mathbb{N},\ 1 \leq i \leq h,\ 1 \leq j \leq w$, the pixel-wise feature vector $z_{i,j}$ of the tensor $z$ is calculated as:

$$z_{i,j} = W_{t-1}x_{i,j} + \alpha_{\text{lora}} B_t A_t x_{i,j} \tag{9}$$

where $x_{i,j}$ is the corresponding pixel-wise feature vector of the tensor $x$. With $\mathcal{L}$ as the loss function, the gradient of $\mathcal{L}$ with respect to the weight matrix $W_t$ is as follows:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{i,j \in \mathbb{N},\ 1 \leq i \leq h,\ 1 \leq j \leq w} \frac{\partial \mathcal{L}}{\partial z_{i,j}} \cdot \frac{\partial z_{i,j}}{\partial W} \tag{10}$$

$$= \sum_{i,j \in \mathbb{N},\ 1 \leq i \leq h,\ 1 \leq j \leq w} \frac{\partial \mathcal{L}}{\partial z_{i,j}} \cdot x_{i,j}^\top \tag{11}$$

We will now compute the gradient of $\mathcal{L}$ with respect to the weight matrix $B_t$:

$$\frac{\partial \mathcal{L}}{\partial B_t} = \sum_{1 \leq i \leq h,\ 1 \leq j \leq w} \frac{\partial \mathcal{L}}{\partial z_{i,j}} \cdot \frac{\partial z_{i,j}}{\partial B_t} \tag{12}$$

$$= \sum_{1 \leq i \leq h,\ 1 \leq j \leq w} \frac{\partial \mathcal{L}}{\partial z_{i,j}} \cdot x_{i,j}^\top A_t^\top \tag{13}$$

$$= \left( \sum_{1 \leq i \leq h,\ 1 \leq j \leq w} \frac{\partial \mathcal{L}}{\partial z_{i,j}} \cdot x_{i,j}^\top \right) A_t^\top \tag{14}$$

$$= \frac{\partial \mathcal{L}}{\partial W_t} A_t^\top \tag{15}$$

Then, the changes in $W_t$ with respect to a change in $B_t$ is as follows:

$$\Delta_{B_t} W_t = [W_{t-1} + \alpha_{\text{lora}}(B_t + \Delta B_t) A_t] \tag{16}$$

$$- (W_{t-1} + \alpha_{\text{lora}} B_t A_t) \tag{17}$$

$$= \alpha_{\text{lora}} \Delta B_t A_t \tag{18}$$

$$= -\alpha \alpha_{\text{lora}} \frac{\partial \mathcal{L}}{\partial B_t} A_t \quad (\alpha \text{ is the step size}) \tag{19}$$

$$= -\alpha \alpha_{\text{lora}} \frac{\partial \mathcal{L}}{\partial W_t} A_t^\top A_t \tag{20}$$

Since $W_t \in \mathbb{R}^{o \times c}$, we have $\frac{\partial \mathcal{L}}{\partial W_t} \in \mathbb{R}^{o \times c}$. With $l$ is a row vector of $\frac{\partial \mathcal{L}}{\partial W_t}$, $lA_t^\top \in \mathbb{R}^{1 \times r}$, then $lA_t^\top A_t$ is the linear combination of all row vectors in $A_t$. So, $A_t A_t^\top$ projects each row vector of $\frac{\partial \mathcal{L}}{\partial W_t}$ into the subspace spanned by $\{a_{t,1}, \ldots, a_{t,r}\}$. This completes the proof of Proposition 1.

□

## B. Further Insights into Multilinear Operator Network

**Inference process of Multilinear Operator Network** Fig. 2b in the main paper shows the inference process of MONet.

Multiple pairs of Basic Blocks and Downsamplings are stacked above one another, followed by the Global Average Pooling. This inference process ends with a fully connected layer, combined with Softmax, to return the confidence levels for all classes that are used to predict the label of an image.

The notable difference between MONet and CNN, ViT lies in their initial image processing step, where MONet uses a **pyramid patch embedding** technique. Initially, it extracts non-overlapping patches of size $p \times p$ from the input image using a convolutional layer with a kernel and stride matching the patch size. These patches are then passed through a second convolution with a $q \times q$ kernel and stride of $q$, further compressing their spatial resolution. This method captures embeddings at smaller scales and progressively extracts new patch embeddings on top, allowing the model to capture finer feature details without increasing parameters or computational load.

**Modifications to MONet with PoLoRA-M** Fig. 9 illustrates a modified version of the standard Mu-Layers, in which a skip connection is replaced by a channel projec-

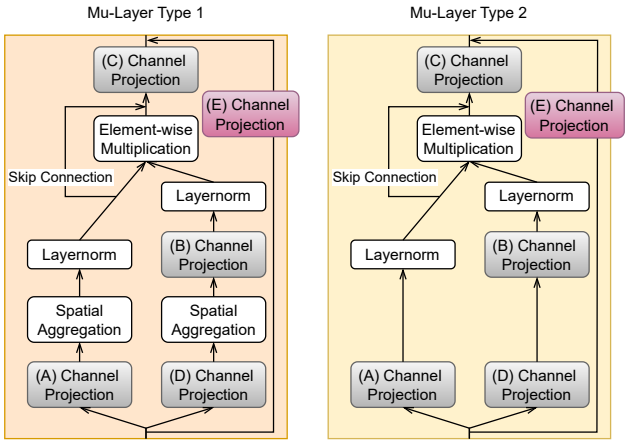



Figure 9. The architecture of a modified Mu-Layer.

tion, called $E$. Names of other channel projections are kept as in Sec. 3.1.

Here, denoted as in [5], $A \in \mathbb{R}^{m\times c}$, $B \in \mathbb{R}^{m\times l}$, $C \in \mathbb{R}^{c\times m}$, and $D \in \mathbb{R}^{l\times c}$ have parameters for tuning. Then, the additional channel projection $E \in \mathbb{R}^{c\times c}$. It is apparent that PoLoRA-M is similar to applying the low-rank update for $E$. The original MU-Layers are special cases of these modified Mu-Layers when $E$ is the identity matrix.

Let $r$ denote the rank of the low-rank adaptations matrix. Then, in each Mu-Layer, the number of trainable parameters for PoLoRA-M is $r\times 2c$, lower than that for PoLoRA-C, which is $r\times(3m+3c+2l)$. This, combining with the decent performance demonstrated in Sec. 5, makes PoLoRA-M an effective choice for incremental learning with a few trainable parameters.

## C. Gradient Projection Memory

Gradient Projection Memory (GPM) is a technique designed to mitigate catastrophic forgetting by Saha et al. [32]. It achieves this by projecting gradients orthogonally to the subspaces representing important information from prior tasks, ensuring minimal interference.

We denote the gradient space of the $(t-1)$ old tasks as $\mathcal{G}_{\text{old},t}$. GPM uses the matrix $G_{\text{old},t}$, the columns of which form the orthonormal basis of $\mathcal{G}_{\text{old},t}$, to approximate the gradient of old tasks. GPM constructs the input matrix $H_t$, where each column represents an input vector of the $t$-th task. The part of $H_t$ which has already been in $\mathcal{G}_{\text{old},t}$ is discarded by:

$$\widehat{H}_t = H_t - G_{\text{old},t}G_{\text{old},t}^T H_t = H_t - H_{\text{proj},t}. \tag{21}$$

Singular value decomposition is then used on the input matrix $\widehat{H}_t$ as follows: $\widehat{H}_t = U\Sigma V^T$. However, GPM does not use all column vectors of $U$ as new orthogonal bases to combine with the existing orthogonal bases in $G_{\text{old},t}$ to form the updated orthonormal basis system for the gradients of learned tasks. Instead, GPM selects only $u$ columns corresponding to the highest singular values, where the number of columns $u$ is the smallest number satisfying:

$$\|(\widehat{H}_t)_u\|_F^2 + \|H_{\text{proj},t}\|_F^2 \geq \lambda\|H_t\|_F^2 \tag{22}$$

Here, $(\widehat{H}_t)_u = [h_1, h_2, \ldots, h_u]$ represents the components of $\widehat{H}_t$ associated with the top-$u$ singular values; $\lambda$ is a specified threshold whose value can be modified to control the capacity allocated to learn the new task. In other words, when $\lambda$ increases, the dimension of subspace $\mathcal{G}_{\text{old},t}$ tends to increase. In practice, $\lambda$ is set to increase from $\lambda_{min}$ to $\lambda_{max}$ at the last task $T$-th. At task $t$-th ($t \leq T$), the value of $\lambda$ is:

$$\lambda_t = \lambda_{min} + \frac{(\lambda_{max} - \lambda_{min}) * t}{T} \tag{23}$$

GPM can work with MONet ($H_t$ just needs input information of the channel projection to control the gradient updates) due to Observation 1.

**Observation 1.** *For the channel projection in MONet, the gradient updates are in the span of the input pixel-wise feature vectors.*

We denote the loss function as $\mathcal{L}$, the forward propagation of a channel projection as:

$$z = Wx \tag{24}$$

where the weight of a channel projection is represented by a matrix $W \in \mathbb{R}^{o\times c}$, the input tensor $x \in \mathbb{R}^{c\times h\times w}$. Then, $z \in \mathbb{R}^{o\times h\times w}$, with $1 \leq i \leq h,\ 1 \leq j \leq w$, we have:

$$z_{i,j} = [z_{1,i,j}; z_{2,i,j}; \ldots; z_{o,i,j}]^\top \tag{25}$$
$$= W[x_{1,i,j}; x_{2,i,j}; \ldots; x_{c,i,j}]^\top \tag{26}$$
$$= Wx_{i,j}. \tag{27}$$

With the chain rule, we have:

$$\frac{\partial\mathcal{L}}{\partial W} = \sum_{1\leq i\leq h,\ 1\leq j\leq w} \frac{\partial\mathcal{L}}{\partial z_{i,j}} \cdot \frac{\partial z_{i,j}}{\partial W} \tag{28}$$
$$= \sum_{1\leq i\leq h,\ 1\leq j\leq w} \frac{\partial\mathcal{L}}{\partial z_{i,j}} \cdot x_{i,j}^\top. \tag{29}$$

Therefore, each row of $\frac{\partial\mathcal{L}}{\partial W}$ is the result of $x_{i,j}^\top$ multiplied by a real value. This completes the proof of Observation 1.

## D. Experimental Details

### D.1. Dataset Processing

As stated in Sec. 5, our experiments are conducted on three popular datasets in incremental learning: CIFAR-100 [20], ImageNet-R [14], and PACS [23]. For CIFAR-100

and ImageNet-R, classes are randomly divided into separate groups to form different tasks. In PACS, tasks are defined by domains, namely *photo*, *art painting*, *cartoon*, and *sketch*, in order.

CIFAR-100 and ImageNet-R provide predefined training and validation splits, while for PACS, we allocate 80% of the data for training and the remaining 20% for validation.

### D.2. Hyperparameter Settings

As illustrated in Sec. 5, we set the batch size to 64 and trained every model for 20 epochs on each task in all our experiments. For IPoLoRA methods, we utilize Adam optimizer with the learning rate of 0.002, and $(\beta_1, \beta_2) = (0.9, 0.999)$ for all datasets.

The only difference between our hyperparameters for the two IPoLoRA frameworks is the value of GPM scaling factor $\alpha_{\text{GPM}}$. For IPoLoRA-M, we set $\alpha_{\text{GPM}} = 200$, whereas for IPoLoRA-C, $\alpha_{\text{GPM}} = 2$. Unless explicitly mentioned otherwise, we use the default hyperparameters as in Tab. 6. For a fair comparison, we use identical hyperparameters across all approaches under the same experimental settings.

## E. Additional Results

### E.1. Task prediction accuracy

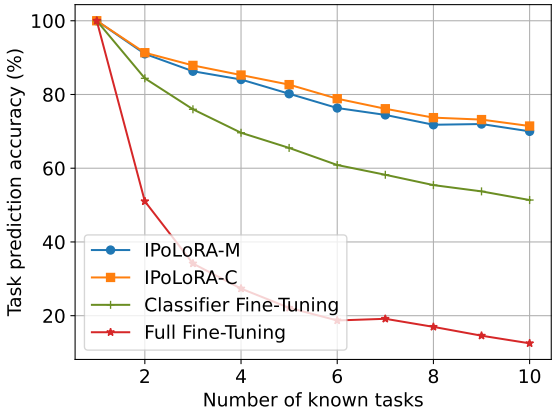



Figure 10. Task prediction accuracy with different numbers of known tasks for each method in CIFAR-100.

Fig. 10 and Fig. 11 present the task prediction accuracy throughout the training process across methods on CIFAR-100 and PACS, respectively. In these experiments, if a model predicts a sample to belong to a specific class, we consider it as predicting the task associated with that class.

From both figures, we observe that the task prediction accuracy of *Full Fine-tuning* is consistently the lowest, and significantly decreases as the number of known tasks increases. This is attributed to the observation that *Full Fine-tuning* tends to overfit the latest task and make biased predictions toward its classes, therefore significantly exacerbating forgetting of previously learned tasks, as demonstrated in Fig. 6, Sec. 5.

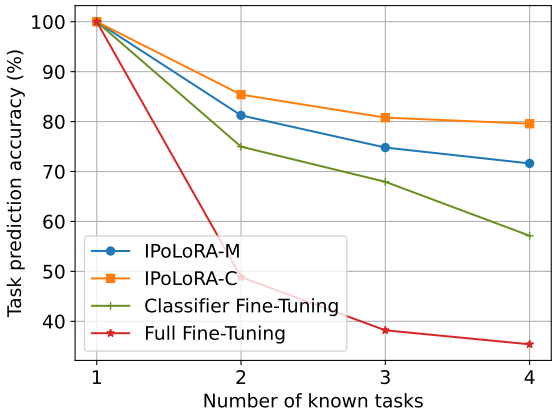



Figure 11. Task prediction accuracy with different numbers of known tasks for each method in PACS.

*Classifier Fine-tuning* performs better but still falls short compared to the IPoLoRA methods. Among these, IPoLoRA-C outperforms all other approaches across both datasets and task numbers. However, the performance gap between IPoLoRA-C and IPoLoRA-M is minimal in CIFAR-100, indicating that IPoLoRA-M performs well when tasks are closely related.

### E.2. Performance on Real-world Datasets

We further evaluate our proposed methods on a real-world medical image dataset comprising three subdatasets from MedMNIST [40]—OrganA, OrganC, and OrganS. Each subdataset represents a different anatomical view of computed tomography (CT) scans: axial (OrganA), coronal (OrganC), and sagittal (OrganS), and is used for multi-class classification of 11 body organs.

For our incremental learning setup, we construct a three-task scenario with a total of 33 classes, where each class corresponds to a specific organ in a given view. Each task consists of one subdataset with 11 classes. The results are presented in Tab. 7.

### E.3. Additional Ablation Study

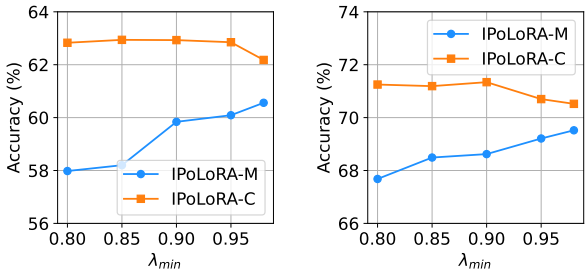
**Scenarios of different $\lambda_{min}$** Fig. 12 shows that our IPoLoRA-M and IPoLoRA-C work well under a wide range of GPM threshold $\lambda_{min}$ values. Notably, IPoLoRA-C exhibits high stability, with its final average accuracy varying by less than 1% (from 62.18% to 62.94% on ImageNet-R, and from 70.52% to 71.34% on CIFAR-100) as $\lambda_{min}$ changes, whereas IPoLoRA-M seems to perform better with a higher threshold $\lambda_{min}$, with its final average accuracy improving from 57.98% to 60.56% on ImageNet-R, and from

| Hyperparameter | ImageNet-R | CIFAR-100 | PACS |
|---|---|---|---|
| Batch size | 64 | 64 | 64 |
| Number of epochs per task | 20 | 20 | 20 |
| Adam optimizer ($\beta_1$ & $\beta_2$) | 0.9, 0.999 | 0.9, 0.999 | 0.9, 0.999 |
| Initial learning rate | 0.002 | 0.002 | 0.004 |
| Learning rate decay | 0.1 | 0.1 | 0.1 |
| PoLoRA rank $r$ | 10 | 10 | 10 |
| GPM threshold $\lambda_{min}$ | 0.95 | 0.95 | 0.98 |
| GPM threshold $\lambda_{max}$ | 1.0 | 1.0 | 1.0 |
| PoLoRA scaling factor $\alpha_{\text{lora}}$ | 0.5 | 0.5 | 0.5 |
| GPM scaling factor $\alpha_{\text{GPM}}$ (for IPoLoRA-M) | 200 | 200 | 20 |
| GPM scaling factor $\alpha_{\text{GPM}}$ (for IPoLoRA-C) | 2 | 2 | 2 |

Table 6. Choices of hyper-parameters for IPoLoRA.

| No. of tasks | 1 | 2 | 3 |
|---|---|---|---|
| Full FT | 92.51 ± 1.30 | 31.87 ± 1.83 | 18.90 ± 5.32 |
| Classifier FT | 77.76 ± 0.12 | 61.08 ± 0.74 | 49.70 ± 0.49 |
| PoLoRA-M | 92.50 ± 0.63 | 54.64 ± 3.52 | 40.06 ± 1.53 |
| PoLoRA-C | **94.92** ± 0.25 | 68.99 ± 1.10 | 53.18 ± 1.38 |
| IPoLoRA-M | 92.48 ± 0.44 | 72.41 ± 0.74 | 59.32 ± 1.19 |
| IPoLoRA-C | 94.85 ± 0.19 | **72.49** ± 1.67 | **60.15** ± 0.89 |

Table 7. Average accuracy with different numbers of known tasks on the MedMNIST dataset.




(a) On ImageNet-R (10 tasks) (b) On CIFAR-100 (10 tasks)

Figure 12. Impact of GPM threshold $\lambda_{min}$ on the performance (%) of IPoLoRA.

| PoLoRA s.f. | | 0.1 | 0.2 | 0.5 | 1 | 2 |
|---|---|---|---|---|---|---|
| ImR | IP-M | 56.78 | 58.83 | **60.09** | 59.04 | 58.71 |
| | IP-C | 59.87 | 61.42 | **62.85** | 62.08 | 56.49 |
| CI | IP-M | 67.24 | 68.85 | **69.21** | 68.74 | 68.32 |
| | IP-C | 68.76 | 70.46 | **70.70** | 69.68 | 65.67 |

Table 8. Impact of PoLoRA scaling factor on the performance (%) of IPoLoRA. ImR: ImageNet-R (10 tasks), CI: CIFAR-100 (10 tasks), IP: IPoLoRA.

67.68% to 69.52% on CIFAR-100 when $\lambda_{min}$ increases from 0.80 to 0.98.

Authors of GPM, Saha et al. [32], stated that a low value of $\lambda_{min}$ allows the optimizer to modify weights in directions strongly influenced by past data, which can significantly disrupt the correlation between past inputs and weights, potentially causing catastrophic interference. In contrast, a high value of $\lambda_{min}$ helps preserve these correlations but may hinder the learning of new tasks due to the increased constraints in the gradient space. Fig. 12 verifies the impact of this phenomenon on IPoLoRA-M and IPoLoRA-C. IPoLoRA-C achieves the best performance when $\lambda_{min}$ is sufficiently large but not excessively high. Meanwhile, IPoLoRA-M tends to work better even when $\lambda_{min}$ increases, without difficulties in learning new tasks. This might be due to the fact that training PoLoRA-M is similar to applying the low-rank update for additional channel projections while keeping the weights of other existing channel projections, which is discussed in Sec. 4 and Appendix B.

**Impact of PoLoRA scaling factor** We examine the sensitivity of the PoLoRA scaling factor $\alpha_{\text{lora}}$ on ImageNet-R (10 tasks) and CIFAR-100 (10 tasks) scenarios, testing values in the range of [0.1, 0.2, 0.5, 1, 2]. As shown in Tab. 8, the results remain stable across a wide range of $\alpha_{\text{lora}}$ values, with the optimal configuration being $\alpha_{\text{lora}} = 0.5$.

Additionally, we also compare the final average accuracy $\overline{Acc_T}$ of IPoLoRA-M using the same set of GPM scaling factor, $\{1, 2, 5, 25, 100, 200, 300\}$, in two scenarios:

(1) Using a fixed value for the PoLoRA scaling factor $\alpha_{\text{lora}}$, for every value of $\alpha_{\text{GPM}}$.

(2) Calculating the PoLoRA scaling factor as $\alpha_{\text{lora}} = C_\alpha \cdot \alpha_{\text{GPM}}$, where $C_\alpha$ is a constant.

Tab. 9 shows that altering $\alpha_{\text{GPM}}$ while keeping PoLoRA scaling factor unchanged ($\alpha_{\text{lora}} = 0.5$) can degrade the

| GPM s.f. ($\alpha_{\text{GPM}}$) | | 1 | 2 | 5 | 25 | 100 | 200 | 300 |
|---|---|---|---|---|---|---|---|---|
| ImR | $\alpha_{\text{lora}} = 0.5$ | 6.98 | 23.36 | 45.88 | 57.20 | 59.66 | 60.09 | 60.10 |
| | $C_\alpha = 0.0025$ | 59.74 | 60.32 | 59.97 | 59.96 | 60.04 | 60.09 | 60.22 |
| CI | $\alpha_{\text{lora}} = 0.5$ | 19.05 | 31.72 | 52.41 | 66.53 | 68.73 | 69.21 | 69.02 |
| | $C_\alpha = 0.0025$ | 69.49 | 69.23 | 69.38 | 69.35 | 69.43 | 69.21 | 69.63 |

Table 9. The performance (%) of IPoLoRA-M with adjusted $\alpha_{\text{lora}}$ w.r.t different $\alpha_{\text{GPM}}$ values. ImR: ImageNet-R (10 tasks), CI: CIFAR-100 (10 tasks).

model performance when the value of $\alpha_{\text{GPM}}$ decreases, as discussed in Sec. 5. However, when the ratio of the two scaling factors remains constant at $C_\alpha = 0.0025$, the changes in $\overline{Acc_T}$ are negligible. This means when changing $\alpha_{\text{GPM}}$, we can change $\alpha_{\text{lora}}$ so that the performance remains almost unchanged. This indicates that $\alpha_{\text{lora}}$ and $\alpha_{\text{GPM}}$ can be fine-tuned interchangeably, offering a versatile approach to fine-tuning.