Adaptive Plasticity Improvement for Continual Learning

Yan-Shuo Liang and Wu-Jun Li*
National Key Laboratory for Novel Software Technology,
Department of Computer Science and Technology, Nanjing University, P. R. China
liangys@smail.nju.edu.cn, liwujun@nju.edu.cn

1. More Details of DualGPM and API

1.1. Relationship between Input Space and Gradient Space

According to the finding of the existing works [3, 8], the following conclusion holds:

Proposition 1. *The gradient update of the linear or convolution layer lies in the span of inputs.*

Specifically, there are two conclusions:

- The gradient update of the linear layer lies in the span of input.
- The gradient update of convolution filters lies in the space spanned by patch vectors.

Linear Layer For the linear layer, we denote its forward propagation as

$$\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l^T \boldsymbol{h}_{l-1} + \boldsymbol{b}_l), \tag{1}$$

where σ_l denotes activation function. $\boldsymbol{W}_l \in \mathbb{R}^{d_I \times d_O}$, $\boldsymbol{h}_{l-1} \in \mathbb{R}^{d_I}$, and $\boldsymbol{h}_l \in \mathbb{R}^{d_O}$. d_I and d_O denote input and output dimension, respectively. We further denote the loss function as L. Through the chain rule, we can get the gradient of $\boldsymbol{W}_l$:

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{W}_l} &= \frac{\partial L}{\partial \boldsymbol{h}_l}\frac{\partial \boldsymbol{h}_l}{\boldsymbol{W}_l} = \left(\frac{\partial L}{\partial \boldsymbol{h}_l} \odot \sigma_l^{'}\right)\boldsymbol{h}_{l-1}^T \\ &= \left[a_1\boldsymbol{h}_{l-1}, a_2\boldsymbol{h}_{l-1}, ..., a_{d_O}\boldsymbol{h}_{l-1}\right], \end{aligned}\tag{2}$$

where $\odot$ denotes element-wise multiplication. $[a_1, a_2, ..., a_{d_O}]^T$ denotes the vector $\frac{\partial L}{\partial \boldsymbol{h}_l} \odot \sigma_l^{'}$. Through (2), we can find that each column of $\frac{\partial L}{\partial \boldsymbol{W}_l}$ can be represented as input $\boldsymbol{h}_{l-1}$ multiplied by a real value a_k $(1 \leq k \leq d_O)$. Therefore, in the linear layer, each column of the gradient $\frac{\partial L}{\partial \boldsymbol{W}_l}$ lies in the span of input. The input matrix $\boldsymbol{R}_{l,t}$ in the linear layer is got by computing $[\boldsymbol{h}_{l-1}^1, \boldsymbol{h}_{l-1}^2, ..., \boldsymbol{h}_{l-1}^N]$.

*Wu-Jun Li is the corresponding author.

Convolution Layer For the convolution layer, we denote its forward propagation as

$$\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l * \boldsymbol{h}_{l-1} + \boldsymbol{b}_l), \tag{3}$$

where $\boldsymbol{W}_l \in \mathbb{R}^{C_O \times C_I \times k \times k}$, $\boldsymbol{h}_{l-1} \in \mathbb{R}^{C_I \times h_I \times w_I}$, and $\boldsymbol{h}_l \in \mathbb{R}^{C_O \times h_O \times w_O}$. C_I and C_O denote input and output channels, respectively. k denotes kernel size. (h_I, w_I) and (h_O, w_O) denote the input and output size of the feature, respectively. Through the reshaping process, we can get

$$\boldsymbol{W}_l \in \mathbb{R}^{C_O \times C_I \times k \times k} \Rightarrow \hat{\boldsymbol{W}}_l \in \mathbb{R}^{(C_I \times k \times k) \times C_O}, \tag{4}$$

$$\boldsymbol{h}_{l-1} \in \mathbb{R}^{C_I \times h_I \times w_I} \Rightarrow \hat{\boldsymbol{h}}_{l-1} \in \mathbb{R}^{(C_I \times k \times k) \times (h_O \times w_O)}, \tag{5}$$

$$\boldsymbol{h}_l \in \mathbb{R}^{C_O \times h_O \times w_O} \Rightarrow \hat{\boldsymbol{h}}_l \in \mathbb{R}^{C_O \times (h_O \times w_O)}. \tag{6}$$

Then the convolution operation $*$ can be transformed into matrix multiplication, that is

$$\hat{\boldsymbol{h}}_l = \sigma_l(\hat{\boldsymbol{W}}_l^T \hat{\boldsymbol{h}}_{l-1} + \boldsymbol{b}_l). \tag{7}$$

Based on (7), the conclusion in the linear layer can be applied to the convolution layer. Since each column of $\hat{\boldsymbol{h}}_{l-1}$ is a patch vector with patch size $k \times k$ (equal to the kernel size), we can get the conclusion in the convolution layer. The input matrix $\boldsymbol{R}_{l,t}$ in the convolution layer is got by computing $[\hat{\boldsymbol{h}}_{l-1}^1, \hat{\boldsymbol{h}}_{l-1}^2, ..., \hat{\boldsymbol{h}}_{l-1}^N]$.

Please note that these two conclusions are for weight $\boldsymbol{W}_l$ and not for bias $\boldsymbol{b}_l$, so no bias units are used. Since the parameters of the neural network mainly exist in the weight of each layer, the plasticity of the neural network will be affected little without defining biases $\boldsymbol{b}_l$.

1.2. Space Transformation

We show that we can get orthogonal bases of $\mathcal{M}_{l,t}^{\perp}$ by performing SVD on the orthogonal bases of $\mathcal{M}_{l,t}$. Specifically, we can prove the following theorem.

Theorem 2. *For a subspace $\mathcal{M}$ with a set of orthogonal bases $\boldsymbol{M} = [\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_m] \in \mathbb{R}^{d \times m}$, if we perform SVD* *of the matrix $\boldsymbol{M}$ ($\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$), then the columns of $\boldsymbol{U}$ which correspond to the zero singular values form a set of orthogonal bases of $\mathcal{M}^\perp$. Here, $\mathcal{M}^\perp$ is defined as*

$$\mathcal{M}^\perp = \{\boldsymbol{u}^\perp \in \mathbb{R}^d | \,\forall\, \boldsymbol{u} \in \mathcal{M},\ (\boldsymbol{u}^\perp)^T \boldsymbol{u} = 0\} \qquad (8)$$

Proof. Let matrix $\boldsymbol{U} = [\boldsymbol{U}_m, \boldsymbol{U}_{d-m}]$. The first m columns $\boldsymbol{U}_m$ correspond to non-zero singular values, and the last $d-m$ columns $\boldsymbol{U}_{d-m}$ correspond to zero singular values. Then the SVD of $\boldsymbol{M}$ can be rewritten as

$$\boldsymbol{M} = \left[\boldsymbol{U}_m, \boldsymbol{U}_{d-m}\right] \begin{bmatrix} \boldsymbol{\Sigma}_m \\ \boldsymbol{O} \end{bmatrix} \boldsymbol{V}^T = \boldsymbol{U}_m(\boldsymbol{\Sigma}_m \boldsymbol{V}^T), \qquad (9)$$

where $\boldsymbol{O}$ represents a matrix of all zero elements.

On the one hand, since $\boldsymbol{U}$ is an orthogonal matrix, each column in $\boldsymbol{U}_m$ is orthogonal to each column of $\boldsymbol{U}_{d-m}$. Therefore, we can get $\boldsymbol{U}_m^T \boldsymbol{U}_{d-m} = \boldsymbol{O}$. According to (9), we have

$$\boldsymbol{M}^T \boldsymbol{U}_{d-m} = \boldsymbol{V}\boldsymbol{\Sigma}_m \boldsymbol{U}_m^T \boldsymbol{U}_{d-m} = \boldsymbol{O}. \qquad (10)$$

Equation (10) shows that each column of $\boldsymbol{M}$ is orthogonal to each column of $\boldsymbol{U}_{d-m}$. Therefore, each column of $\boldsymbol{U}_{d-m}$ lies in the space $\mathcal{M}^\perp$.

On the other hand, since columns of $\boldsymbol{U}$ form a set of orthogonal bases of $\mathbb{R}^d$, any vector $\boldsymbol{v} \in \mathcal{M}^\perp$ can be denoted as $\boldsymbol{v} = \boldsymbol{U}\boldsymbol{a}$, where $\boldsymbol{a} \in \mathbb{R}^d$. Furthermore, due to the definition of $\mathcal{M}^\perp$ (see (8)), we also have

$$\boldsymbol{M}^T \boldsymbol{v} = \boldsymbol{0} \quad \Rightarrow \quad \boldsymbol{V}\boldsymbol{\Sigma}_m \boldsymbol{U}_m^T \boldsymbol{v} = \boldsymbol{0}. \qquad (11)$$

Since matrix $\boldsymbol{V}\boldsymbol{\Sigma}_m$ has full column rank, we can get $\boldsymbol{U}_m^T \boldsymbol{v} = \boldsymbol{0}$ from (11). Based on these, we have

$$\begin{aligned} \boldsymbol{U}_m^T \boldsymbol{v} = \boldsymbol{0} \quad \Rightarrow \quad \boldsymbol{U}_m^T \boldsymbol{U}\boldsymbol{a} &= \boldsymbol{U}_m^T \left[\boldsymbol{U}_m, \boldsymbol{U}_{d-m}\right] \begin{bmatrix} \boldsymbol{a}_m \\ \boldsymbol{a}_{d-m} \end{bmatrix} \\ &= \boldsymbol{a}_m = \boldsymbol{0}, \end{aligned} \qquad (12)$$

where $\boldsymbol{a}_m$ and $\boldsymbol{a}_{d-m}$ denote the components of $\boldsymbol{a}$ corresponding to $\boldsymbol{U}_m$ and $\boldsymbol{U}_{d-m}$, respectively. Therefore, $\boldsymbol{v} = \boldsymbol{U}_{d-m}\boldsymbol{a}_{d-m}$. Thus, any vector $\boldsymbol{v} \in \mathcal{M}^\perp$ is the linear combination of the columns of $\boldsymbol{U}_{d-m}$. Since we have proved that each columns of $\boldsymbol{U}_{d-m}$ lies in the subspace $\mathcal{M}^\perp$, columns of $\boldsymbol{U}_{d-m}$ form a set of orthogonal bases of subspace $\mathcal{M}^\perp$. □

1.3. Removing Subspace

In DualGPM, we need to remove subspace $\mathcal{Y}$ from $\mathcal{M}_{l,t}^\perp$ to get space $\mathcal{M}_{l,t+1}^\perp$, where $\mathcal{Y}$ denotes the subspace of $\mathcal{M}_{l,t}^\perp$ containing the gradient of the t-th task. We achieve this by the following theorem.

Theorem 3. *Let two spaces $\mathcal{M}$, $\mathcal{N}$ satisfy $\mathcal{N} \subseteq \mathcal{M} \subseteq \mathbb{R}^d$, where $\dim(\mathcal{M}) = m$, $\dim(\mathcal{N}) = n$ and $n < m \leq d$. Assume $\boldsymbol{M} = [\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_m] \in \mathbb{R}^{d\times m}$ denotes the orthogonal bases of $\mathcal{M}$, $\boldsymbol{N} = [\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n] \in \mathbb{R}^{d\times n}$ denotes the orthogonal bases of $\mathcal{N}$. Let $\hat{\boldsymbol{M}} = \boldsymbol{M} - \boldsymbol{N}(\boldsymbol{N})^T\boldsymbol{M}$. If we perform SVD of the matrix $\hat{\boldsymbol{M}}$ ($\hat{\boldsymbol{M}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$), then the column vectors of $\boldsymbol{U}$ which correspond to the non-zero singular values form a set of orthogonal bases of subspace $\mathcal{O}$, where*

$$\mathcal{O} = \{\boldsymbol{u} \in \mathcal{M} | \,\forall \boldsymbol{v} \in \mathcal{N}, \boldsymbol{v}^T\boldsymbol{u} = 0\}. \qquad (13)$$

Proof. Let matrix $\boldsymbol{U} = [\boldsymbol{U}_m, \boldsymbol{U}_{d-m}]$. The first m columns $\boldsymbol{U}_m$ correspond to non-zero singular values, and the last $d-m$ columns $\boldsymbol{U}_{d-m}$ correspond to zero singular values. Then the SVD of $\hat{\boldsymbol{M}}$ can be rewritten as

$$\hat{\boldsymbol{M}} = \left[\boldsymbol{U}_m, \boldsymbol{U}_{d-m}\right] \begin{bmatrix} \boldsymbol{\Sigma}_m \\ \boldsymbol{O} \end{bmatrix} \boldsymbol{V}^T = \boldsymbol{U}_m(\boldsymbol{\Sigma}_m \boldsymbol{V}^T), \qquad (14)$$

where $\boldsymbol{O}$ represents a matrix of all zero elements.

On the one hand, it is easy to verify that

$$(\hat{\boldsymbol{M}})^T \boldsymbol{N} = \boldsymbol{M}^T\boldsymbol{N} - \boldsymbol{M}^T\boldsymbol{N}\boldsymbol{N}^T\boldsymbol{N} = \boldsymbol{0}. \qquad (15)$$

With (14) and (15), we have

$$(\hat{\boldsymbol{M}})^T \boldsymbol{N} = \boldsymbol{V}\boldsymbol{\Sigma}_m^T \boldsymbol{U}_m^T \boldsymbol{N} = \boldsymbol{0}. \qquad (16)$$

Since matrix $\boldsymbol{V}\boldsymbol{\Sigma}_m$ has full column rank, we can get $\boldsymbol{U}_m^T\boldsymbol{N} = 0$ from (16). Therefore, each column of $\boldsymbol{U}_m$ lies in the subspace $\mathcal{O}$.

On the other hand, for any vector $\boldsymbol{u} \in \mathcal{O} \subseteq \mathcal{M}$, there exists a set of coefficients $\boldsymbol{a} \in \mathbb{R}^m$ such that $\boldsymbol{u} = \boldsymbol{M}\boldsymbol{a}$. Therefore, we have

$$\boldsymbol{u} = \boldsymbol{M}\boldsymbol{a} = \hat{\boldsymbol{M}}\boldsymbol{a} + \boldsymbol{N}(\boldsymbol{N})^T\boldsymbol{M}\boldsymbol{a} = \left[\boldsymbol{U}_m, \boldsymbol{N}\right] \begin{bmatrix} \boldsymbol{\Sigma}_m \boldsymbol{V}^T\boldsymbol{a} \\ (\boldsymbol{N})^T\boldsymbol{M}\boldsymbol{a} \end{bmatrix}. \qquad (17)$$

This means $\boldsymbol{u}$ can be denoted as a linear combination of the columns of matrix $[\boldsymbol{U}_m, \boldsymbol{N}]$. Since $\boldsymbol{u}$ is orthogonal to the columns of $\boldsymbol{N}$ (see Definition (13)), $\boldsymbol{N}^T\boldsymbol{M}\boldsymbol{a}$ must be zero. Therefore, any vector $\boldsymbol{u} \in \mathcal{O}$ can be denoted as a linear combination of the columns of matrix $\boldsymbol{U}_m$. We have proved that each column of $\boldsymbol{U}_m$ lies in $\mathcal{O}$. Therefore, columns of $\boldsymbol{U}_m$ form a set of orthogonal bases of subspace $\mathcal{O}$. □

1.4. Algorithm for Computing Gradient Retention Ratio

We give the process of computing Gradient Retention Ratio (GRR) in Algorithm 1. Please note that this process requires only one more epoch computation.

2. More Details of Experimental Setup

2.1. Architecture Details

LeNet like Architecture LeNet-5 is used in existing work GPM [3] and ADP [7]. Besides the output layer, this network also consists of 2 convolution layers and 2 linear layers. The number of filters of the convolution layers from

Table 1. Statistics of three datasets

	Split CIFAR100	CIFAR100-sup	Split Mini-Imagenet
Task Number	20	20	20
Input Size	$3 \times 32 \times 32$	$3 \times 32 \times 32$	$3 \times 84 \times 84$
Classes per Task	5	5	5
Training Samples	2375	2375	2375
Valid Samples	125	125	125
Testing Samples	500	500	500

Table 2. Statistic of 5-Datasets, the input size is set as $3 \times 32 \times 32$.

	CIFAR10	MNIST	SVHN	Fashion MNIST	notMNIST
Classes	10	10	10	10	10
Number of Samples (train)	4750	57000	69595	57000	16011
Number of Samples (val)	2500	3000	3662	3000	842
Number of Samples (test)	10000	10000	26032	10000	1873

Algorithm 1 Gradient Retention Ratio

```
1: Input: Current Task $\{\mathcal{D}_t\}_{t=1}^T$, network model $f(\cdot, \mathbf{\Theta})$ with $L$ layers, $\mathbf{\Theta} = \{\boldsymbol{W}_{l,t-1}\}_{l=1}^L$, orthogonal bases memory $\{\boldsymbol{M}_{l,t}^*\}_{l=1}^L$.
2: Output: Gradient Retention Ratio value $\{\mathrm{GRR}(l,t)\}_{l=1}^L$.
3: Initialize $\{\mathrm{GRR}(l,t)\}_{l=1}^L$ value: $\mathrm{GRR}(l,t) \leftarrow 0$;
4: $b \leftarrow 0$;
5: for $\mathcal{B}_t$ sampled from $\mathcal{D}_t$ do
6:     Compute the loss $L(\mathcal{B}_t; \mathbf{\Theta})$ over $\mathcal{B}_t$ and get gradient $\boldsymbol{g}_t = [\boldsymbol{g}_{1,t}, \boldsymbol{g}_{2,t} \cdots, \boldsymbol{g}_{L,t}]$;
7:     Using $\boldsymbol{M}_{l,t}^*$ to perform operation by (3) or (4) of the paper and get $\hat{\boldsymbol{g}}_{l,t}$;
8:     $\mathrm{GRR}(l,t) \leftarrow \mathrm{GRR}(l,t) + \frac{||\hat{\boldsymbol{g}}_{l,t}||_2^2}{||\boldsymbol{g}_{l,t}||_2^2}$;
9:     $b \leftarrow b + 1$;
10: end for
11: $\mathrm{GRR}(l,t) \leftarrow \mathrm{GRR}(l,t)/b$;
```

bottom to up is 10, 20. Both two layers have kernel size 5×5. The number of units of two linear layers is 800 and 500, respectively. After each convolution layer, 3×3 max-pooling with stride size 2 is applied.

AlexNet like Architecture This architecture is the same as several existing works [3, 4]. Specifically, the network consists of 3 convolution layers plus 2 linear layers. The number of filters of the convolution layers from bottom to up is 64, 128, and 256 with kernel sizes 4×4, 3×3, and 2×2, respectively. The number of units of two linear layers is 2048. Rectified function is used as activations for all the layers except the classifier layer. After each convolution layer, 2×2 max-pooling is applied. Dropout with ratio 0.2 is used for the first two layers and 0.5 for the rest layers.

Reduced ResNet18 Architecture This architecture is the same as existing work [3] where the parameters of ResNet18 are reduced. Specifically, each layer of reduced ResNet18 is three times fewer features than the original architecture. The average-pooling before the classifier layer is set as 2×2.

2.2. Datasets Statistic

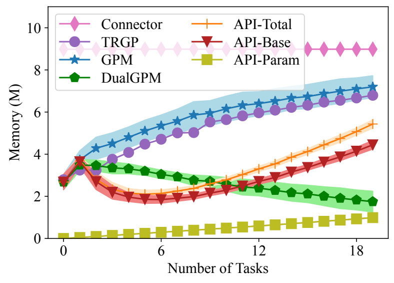
We give detailed statistics of datasets in this section. Specifically, Table 1 shows the detailed information of three datasets, including CIFAR100-sup, Split CIFAR100, and Split Mini-Imagenet. These datasets are constructed by one dataset (CIFAR100 or Mini-Imagenet). Table 2 shows the detailed information of 5-Datasets. This dataset is constructed by five different datasets and each dataset forms a task of 5-Datasets.

2.3. Threshold Setting for Orthogonal Bases Updating

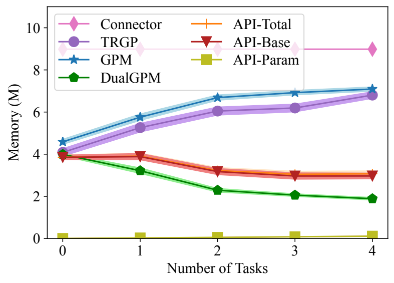
The set of threshold ϵ_{th} follows existing work GPM [3]. For the experiment of Split CIFAR100 with AlexNet architecture, threshold ϵ_{th} is set as 0.97 for all the layers and increased by 0.0015 for each new task. For the experiment of CIFAR100-sup with LeNet, ϵ_{th} is set as 0.98 for the first layer and increased by 0.001 for each new task. For the experiment of Split Mini-Imagenet with ResNet18, ϵ_{th} is set as 0.985 for the first layer and increased by 0.0003 for each

Table 3. List of hyper-parameters for different methods

Methods	Hyper-Parameters
EWC	lr: 0.03 (5-Datasets), 0.05 (SCIFAR100, CIFAR100-sup), 0.1 (mini) regular: 300 (SCIFAR100, CIFAR100-sup), 2000 (mini), 5000 (5-Datasets)
ER-Res	lr: 0.05 (SCIFAR100, CIFAR100-sup), 0.1 (5-Datasets, mini) memory: 2000 (SCIFAR100, CIFAR100-sup, mini), 3000 (5-Datasets)
GPM	lr: 0.01 (SCIFAR100, CIFAR100-sup), 0.1 (5-Datasets, mini)
GEM	lr: 0.05 (SCIFAR100, CIFAR100-sup) memory strength: 0.5 (SCIFAR100, CIFAR100-sup) memory: 2000 (SCIFAR100, CIFAR100-sup)
A-GEM	lr: 0.05 (SCIFAR100, CIFAR100-sup), 0.1 (5-Datasets, mini) memory: 2000 (SCIFAR100, CIFAR100-sup, mini), 3000 (5-Datasets)
FS-DGPM	lr, η_3: 0.01 (SCIFAR100, CIFAR100-sup) lr for sharpness, η_1: 0.001 (SCIFAR100), 0.01 (CIFAR100-sup) lr for DGPM, η_2: 0.01 (SCIFAR100, CIFAR100-sup) memory: 900 (SCIFAR100), 1100 (CIFAR100-sup)
TRGP	lr: 0.01 (SCIFAR100, CIFAR100-sup), 0.1 (5-Datasets, mini) K: 2 (SCIFAR100, CIFAR100-sup, 5-Datasets, mini) ϵ: 0.5 (SCIFAR100, CIFAR100-sup, 5-Datasets, mini)
API	lr: 0.01 (SCIFAR100, CIFAR100-sup), 0.1 (5-Datasets, mini) K: 10 (SCIFAR100, CIFAR100-sup, 5-Datasets, mini) ρ: 0.5 (SCIFAR100, CIFAR100-sup, 5-Datasets, mini)



(a) Variation of memory usage



(b) Variation of memory usage

Figure 1. (a) Variation of memory usage for Split Mini-Imagenet. (b) Variation of memory usage for 5-Dataset.

new task. For the experiment of 5-Dataset with ResNet18 architecture, threshold ϵ_{th} is set as 0.965 for all the layers.

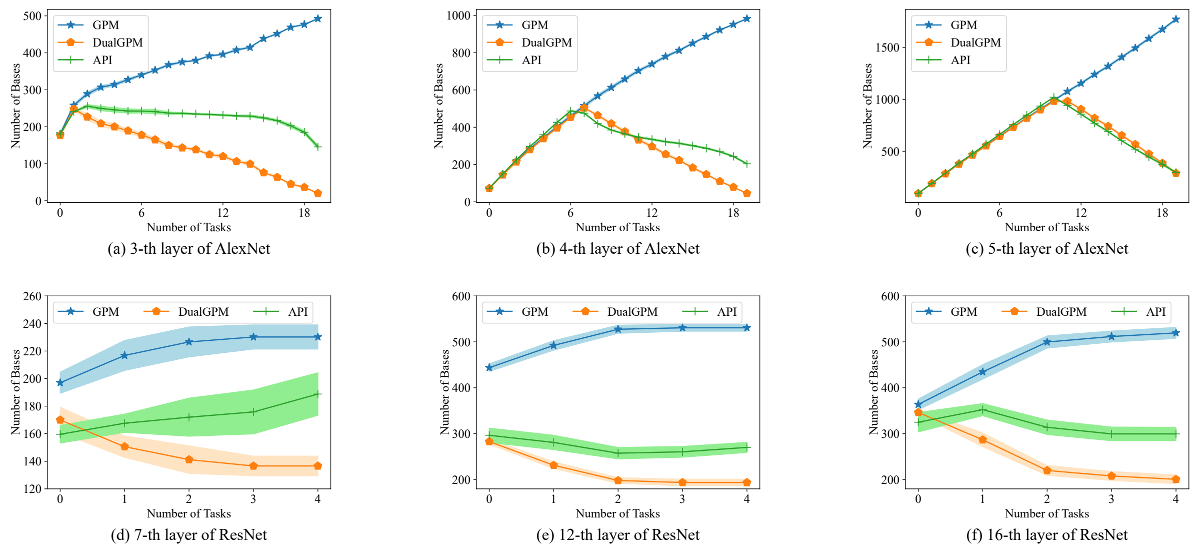
2.4. Hyper-Parameters

We give the hyper-parameters for each method in Table 3. SCIFAR100 denotes Split CIFAR100 and mini denotes Split Mini-Imagenet. FS-DGPM, GPM and TRGP are implemented by their official codes under MIT License. EWC, GEM, ER-Res are implemented by the code provided by FS-DGPM [1] under MIT License.

3. Additional Experimental Results

3.1. Memory Usage

Variation of Memory We show the variation of memory usage in the experiment of Split Mini-Imagenet and 5-Datasets. The memory usage of GPM increases all the time. The memory usage of DualGPM increases first and decreases later. However, the memory usage of API on Split Mini-Imagenet differs from that of API on Split CIFAR100 and 5-Datasets. Specifically, API-Base first increases, then decreases, and finally increases again. The second increase in API-Base is because the dimension of bases increases with the expansion of parameters $\boldsymbol{W}_{l,t}$ (see Equation (9) of



(a) 3-th layer of AlexNet (b) 4-th layer of AlexNet (c) 5-th layer of AlexNet

(d) 7-th layer of ResNet (e) 12-th layer of ResNet (f) 16-th layer of ResNet

Figure 2. Variation of bases in different layers of AlexNet and ResNet.

the paper).

Variation of Bases We give the variation of bases in different layers. Specifically, we show the variation of bases in the 3-rd, 4-th, and 5-th layers of AlexNet when the model learns on Split CIFAR100. We also show the variation of bases in the 7-th, 12-th, and 16-th layers of ResNet when the model learns on 5-Datasets. We can find that GPM increases the bases all the time, while DualGPM and API do not increase bases all the time and keep much fewer bases than GPM in these layers.

3.2. Time Consumption

API expands the model's parameters in each layer, and the new task has access to more parameters to learn the model than the old task. However, this increases the time consumption in both the training and inference phases. Figure 3 gives the time consumption for different methods. We can find that API does consume more time than GPM in both the training and inference phases. However, the time consumption of API is comparable with other methods. Specifically, the training time of API is much less than FS-DGPM on Split CIFAR100 and CIFAR100-sup. The training time of API is slightly less than that of TRGP on Split CIFAR100 and 5-Datasets, and slightly larger than that of TRGP on CIFAR100-sup. The average inference time of API is less than TRGP on Split-CIFAR100 and 5-Datasets, and slightly larger than TRGP on CIFAR100-sup.

3.3. More Comparison with Expansion-Based Methods

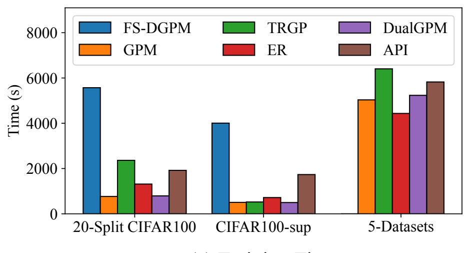
We choose calibrating CNNs for lifelong learning (CCLL) [6] and rectification-based knowledge retention (RKR) [5] to compare with our API. Other expansion-based methods have different experimental settings from ours, and many hyperparameters are difficult to tune. Besides, some methods that define a search space and use some methods to search for expansion strategies incur high computational costs. On the contrary, CCLL and RKR are simple and easy to implement, and they have demonstrated that they outperform many expansion-based methods.

We compare API with CCLL and RKR on Split CIFAR100 and Split Mini-Imagenet. The experimental setting is the same as that of the experiments in the paper. We tune the hyperparameter K in these two methods so that their model capacity is not smaller than that of our method. Please note that larger K in these two methods means larger model capacity and higher accuracy. The results are given in Table 4. We can find that API also gets the best results with the smallest model capacity.

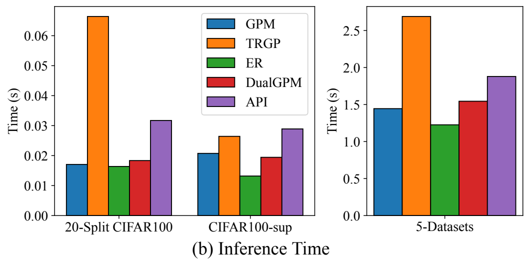
3.4. More Comparison with Memory-Based Methods

Recursive gradient optimization (RGO) [2] is also an memory-based method that rectifies new task gradient layer by layer to overcome catastrophic forgetting. This method directly maintains projection matrix $\boldsymbol{P}_l$ in memory. Please note that GPM maintains orthogonal bases $\boldsymbol{M}_l$ in memory and DualGPM maintains orthogonal bases $\boldsymbol{M}_l$ or $\boldsymbol{M}_l^{\perp}$ in memory. When learning the new task, GPM and DualGPM get projection matrix $\boldsymbol{P}_l = \boldsymbol{M}_l\boldsymbol{M}_l^T$.

Compared with RGO, GPM and DualGPM use less memory. Specifically, RGO keeps $\boldsymbol{P}_l$ in memory and requires d_l^2 parameters. GPM keeps $\boldsymbol{M}_l$ in memory and requires $\dim(\mathcal{M}_l)d_l$ parameters. DualGPM keeps $\boldsymbol{M}_l$ or



(a) Training Time



(b) Inference Time

Figure 3. (a) Training time of different methods on three different datasets. (b) Average inference time of different methods on three different datasets.

Table 4. The performance for different expansion-based methods on Split CIFAR100 dataset and Split Mini-Imagenet.

	Split CIFAR100		Split Mini-Imagenet	
Methods	ACC (%)	Capacity (%)	ACC (%)	Capacity (%)
RKR	77.5 ± 0.5	107	63.7 ± 2.0	132
CCLL	75.4 ± 0.3	105	65.7 ± 0.5	132
API	**81.4 ± 0.4**	**104**	**65.9 ± 0.6**	**127**

$\boldsymbol{M}_l^{\perp}$ in memory and requires $\min(\dim(\mathcal{M}_l), \dim(\mathcal{M}_l^{\perp}))d_l$ parameters in memory. Since

$$\min(\dim(\mathcal{M}_l), \dim(\mathcal{M}_l^{\perp})) \leq \dim(\mathcal{M}_l) \leq d_l, \qquad (18)$$

GPM and DualGPM use less memory than RGO. Furthermore, since $\dim(\mathcal{M}_l)$ increases with the increase of tasks, DualGPM uses much less memory than RGO and GPM when the number of tasks is large.

We compare RGO with our methods on Split CIFAR100 and Split Mini-Imagenet. We use the official implementation of RGO and keep its architecture consistent with our methods. Table 5 gives the comparison between RGO and our methods. 'AVG-MEMORY' denotes the average memory used when the model learns each new task. RGO gets similar accuracy to API on Split CIFAR100 but uses much more memory than GPM and our method. On Split Mini-Imagenet, RGO gets lower accuracy than our methods and GPM, and its average memory usage is more than GPM, DualGPM and API.

3.5. Variation of AGRR

We show the variation of AGRR and average gradient norm during the learning of Split CIFAR100 in Figure 4. We can find that both AGRR and average gradient norm decrease with the increase of tasks. However, our method API adaptively improves the model's plasticity. Therefore, AGRR and average gradient norm are larger than GPM during the whole learning process.

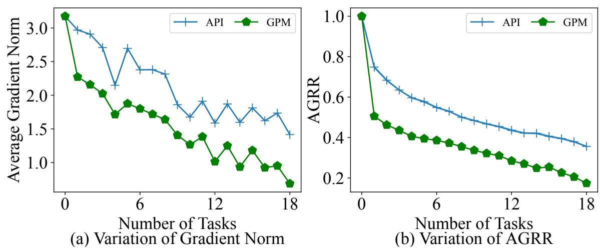


Figure 4. (a) Variation of average gradient norm on Split CIFAR100. (b) Variation of AGRR on Split CIFAR100.

3.6. More Plasticity Evaluation Results

In Section 3.2 of the paper, we present the relationship between the model's performance and AGRR when the model is trained on task 2 of Split-CIFAR100 with DualGPM (non-expandable parameters). Here, we give the details and more results of this experiment.

All experimental settings are the same as that in Section 4 of the paper, except that ϵ_{th}^l (see (6) and (8) of the paper) is adjusted to 6 different values, including 0.97, 0.975, 0.98, 0.985, 0.99, 0.995. For each value of ϵ_{th}^l, we perform experiments three times. Therefore, there are 18 points in Figure 3 of the paper. Average gradient norm denotes the average of the norm of the gradient used for updating parameters during the learning of a task. Specifically, we use $\frac{1}{S}\sum_{i=1}^{S}||\hat{\boldsymbol{g}}_i||_2$ to denote the average gradient norm, where S denotes the update times. Obviously, the larger the average gradient norm is, the larger the model updates the pa-

Table 5. The performance for different algorithm-based methods on Split CIFAR100 dataset and Split Mini-Imagenet.

	Split CIFAR100		Split Mini-Imagenet	
Methods	ACC (%)	AVG-Memory (M)	ACC (%)	AVG-Memory (M)
RGO	**81.4** $\pm$ **0.2**	6.3	58.3 $\pm$ 2.5	9.0
GPM	78.9 $\pm$ 0.2	2.9	61.2 $\pm$ 0.6	5.7
DualGPM	78.5 $\pm$ 0.4	**1.7**	61.2 $\pm$ 0.6	**2.6**
API	**81.4** $\pm$ **0.4**	2.1	**65.9** $\pm$ **0.6**	3.3

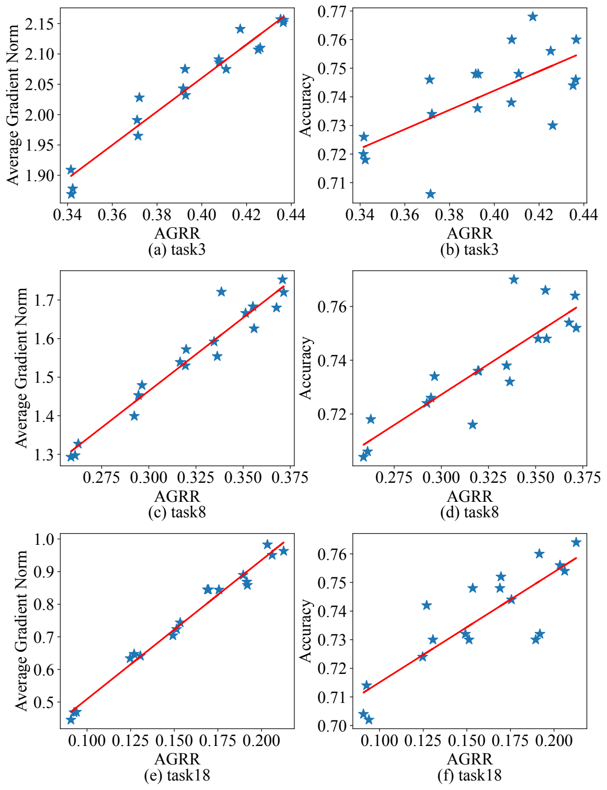


Figure 5. DualGPM with non-expandable parameters learns on Split CIFAR100. (a), (c) and (e) show the correlation between AGRR and average gradient norm for learning different tasks. (b), (c) and (d) show the correlation between AGRR and the accuracy of different tasks.

rameters.

In Figure 5, we give the results on more tasks. Specifically, when the constraint increases (AGRR decreases), the model becomes more conservative in updating parameters, and the accuracy of the model shows a downward trend.

References

[1] Danruo Deng, Guangyong Chen, Jianye Hao, Qiong Wang, and Pheng-Ann Heng. Flattening sharpness for dynamic gradient projection memory benefits continual learning. *arXiv preprint arXiv:2110.04593*, 2021. 4

[2] Hao Liu and Huaping Liu. Continual learning with recursive gradient optimization. In *International Conference on Learning Representations*, 2022. 5

[3] Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2021. 1, 2, 3

[4] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557, 2018. 3

[5] Pravendra Singh, Pratik Mazumder, Piyush Rai, and Vinay P. Namboodiri. Rectification-based knowledge retention for continual learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 15282–15291, 2021. 5

[6] Pravendra Singh, Vinay Kumar Verma, Pratik Mazumder, Lawrence Carin, and Piyush Rai. Calibrating cnns for lifelong learning. *Advances in Neural Information Processing Systems*, 33:15579–15590, 2020. 5

[7] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. In *International Conference on Learning Representations*, 2020. 2

[8] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, pages 107–115, 2021. 1