InfLoRA: Interference-Free Low-Rank Adaptation for Continual Learning

Supplementary Material

A. Details of GPM and DualGPM

GPM and DualGPM are established on the fact that the gradient updates lie in the span of input data points [8].

For a linear layer, we denote its forward propagation as

$$\boldsymbol{e} = \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}, \tag{1}$$

$\boldsymbol{W} \in \mathbb{R}^{d_I \times d_O}$, $\boldsymbol{h} \in \mathbb{R}^{d_I}$, and $\boldsymbol{e} \in \mathbb{R}^{d_O}$. d_I and d_O denote input and output dimension, respectively. We further denote the loss function as $\mathcal{L}$. Through the chain rule, we can get the gradient of $\boldsymbol{W}$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\boldsymbol{W}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{e}} \boldsymbol{h}^T = \begin{bmatrix} a_1 \boldsymbol{h}^T, \\ a_2 \boldsymbol{h}^T, \\ ..., \\ a_{d_O} \boldsymbol{h}^T \end{bmatrix}. \tag{2}$$

$[a_1, a_2, ..., a_{d_O}]^T$ denotes the vector $\frac{\partial \mathcal{L}}{\partial \boldsymbol{e}}$. Through (2), we can find that each column of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$ can be represented as input $\boldsymbol{h}$ multiplied by a real value a_k ($1 \leq k \leq d_O$). Therefore, in the linear layer, each column of the gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$ lies in the span of input.

A.1. Gradient Projection Memory

GPM learns a subspace $\mathcal{M}_t$ with orthogonal bases $\boldsymbol{M}_t$ to approximate the gradient space of the old tasks. Here, the columns of $\boldsymbol{M}_t$ contribute a set of orthogonal bases in $\mathcal{M}_t$. GPM expands the bases of $\mathcal{M}_t$ to the bases of $\mathcal{M}_{t+1}$ after learning the t-th new task. Specifically, GPM computes the inputs matrix $\boldsymbol{H}_t$ such that each column of $\boldsymbol{H}_t$ represents an input of this layer. Then, the part of $\boldsymbol{H}_t$ that has already in $\mathcal{M}_t$ is removed by

$$\hat{\boldsymbol{H}}_t = \boldsymbol{H}_t - \boldsymbol{M}_t(\boldsymbol{M}_t)^T \boldsymbol{H}_t = \boldsymbol{H}_t - \boldsymbol{H}_{t,proj}. \tag{3}$$

Please note that when $t = 1$, $\dim(\mathcal{M}_t) = 0$ and hence $\boldsymbol{H}_{t,proj}$ is a zero matrix. After that, singular value decomposition (SVD) is performed on $\hat{\boldsymbol{H}}_t = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^T$. Then, u new orthogonal bases are chosen from the columns of $\hat{\boldsymbol{U}}$ for a minimum of u satisfying the following criteria for given threshold ϵ_{th}:

$$||(\hat{\boldsymbol{H}}_t)_u||_F^2 + ||\boldsymbol{H}_{t,proj}||_F^2 \geq \epsilon_{th}||\boldsymbol{H}_t||_F^2. \tag{4}$$

Here, $(\hat{\boldsymbol{H}}_t)_u = [\boldsymbol{u}_1, ..., \boldsymbol{u}_u]$ denotes the components of $\hat{\boldsymbol{H}}_t$ that correspond to top-u singular values. Then, subspace $\mathcal{M}_{t+1}$ is obtained with the bases $\boldsymbol{M}_{t+1} = [\boldsymbol{M}_t, \boldsymbol{u}_1, ..., \boldsymbol{u}_u]$.

A.2. Dual Gradient Projection Memory

Different from GPM that learns a subspace $\mathcal{M}_t$ with orthogonal bases $\boldsymbol{M}_t$ to approximate the gradient space of the old tasks, DualGPM either learns a subspace $\mathcal{M}_t$ with orthogonal bases $\boldsymbol{M}_t$ to approximate the gradient of the old tasks, or learns a subspace $\mathcal{M}_t^\perp$ with orthogonal bases $\boldsymbol{M}_t^\perp$ to approximate orthogonal complement of the gradient space of the old tasks.

DualGPM decides whether to keep $\boldsymbol{M}_t$ or $\boldsymbol{M}_t^\perp$ in memory according to $\dim(\mathcal{M}_t)$ and $\dim(\mathcal{M}_t^\perp)$. Specifically, during the learning of the first several tasks, $\dim(\mathcal{M}_t) \leq \dim(\mathcal{M}_t^\perp)$. At this time, DualGPM maintains $\boldsymbol{M}_t$, and expands $\boldsymbol{M}_t$ to $\boldsymbol{M}_{t+1}$ after each task. When $\dim(\mathcal{M}_t)$ increases and exceeds $\dim(\mathcal{M}_t^\perp)$, DualGPM obtains $\boldsymbol{M}_t^\perp$ through some transformations on $\boldsymbol{M}_t$. After that, DualGPM only maintains $\boldsymbol{M}_t^\perp$ in memory, and reduces $\boldsymbol{M}_t^\perp$ to $\boldsymbol{M}_{t+1}^\perp$ after each task. Through this way, the number of bases kept for each layer is $\min\{\dim(\mathcal{M}_t), \dim(\mathcal{M}_t^\perp)\}$.

There are three key problems in DualGPM: expanding the bases of $\mathcal{M}_t$, obtaining the bases of $\mathcal{M}_t^\perp$ through the bases of $\mathcal{M}_t$, and reducing the bases of $\mathcal{M}_t^\perp$.

Expanding the Bases of $\mathcal{M}_t$ The expansion of $\mathcal{M}_t$ is the same as that in GPM.

Transforming $\mathcal{M}_t$ to $\mathcal{M}_t^\perp$ DualGPM transforms $\mathcal{M}_t$ to $\mathcal{M}_t^\perp$ by performing SVD to the matrix $\boldsymbol{M}_t$. Specifically, let $\boldsymbol{M}_t = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, the column vectors of $\boldsymbol{U}$ which correspond to the zero singular values form a set of orthogonal bases of $\mathcal{M}_t^\perp$. Please refer to the paper of DualGPM [2] for this explanation.

Reducing the Bases of $\mathcal{M}_t^\perp$ DualGPM reduces space $\mathcal{M}_t^\perp$ by removing the part of $\mathcal{M}_t^\perp$ which contains the gradient of the t-th task. Specifically, DualGPM first computes the input matrix $\boldsymbol{R}_t$. Then, the part of $\boldsymbol{R}_t$ which lies in $\mathcal{M}_t^\perp$ can be computed through

$$\hat{\boldsymbol{R}}_t^\perp = \boldsymbol{M}_t^\perp(\boldsymbol{M}_t^\perp)^T \boldsymbol{R}_t = \boldsymbol{R}_{t,proj}^\perp. \tag{5}$$

After that, SVD is performed on $\hat{\boldsymbol{R}}_t^\perp = \hat{\boldsymbol{U}}^\perp\hat{\boldsymbol{\Sigma}}^\perp(\hat{\boldsymbol{V}}^\perp)^T$. Then, k new orthogonal bases are chosen from the columns of $\hat{\boldsymbol{U}}^\perp$ for a maximum of k satisfying the following criteria for the given threshold ϵ_{th} (the same as ϵ_{th} in (4)):

$$||(\hat{\boldsymbol{R}}_t^\perp)_k||_F^2 \leq (1 - \epsilon_{th})||\boldsymbol{R}_t||_F^2. \tag{6}$$

Let $\boldsymbol{Z} = (\hat{\boldsymbol{R}}_t^\perp)_k = [\boldsymbol{u}_1^\perp, ..., \boldsymbol{u}_k^\perp]$, $\mathcal{Z} = \text{span}\{\boldsymbol{u}_1^\perp, ..., \boldsymbol{u}_k^\perp\}$. Here, $\mathcal{Z}$ is the subspace of $\mathcal{M}_t^\perp$ that contains the gradient of

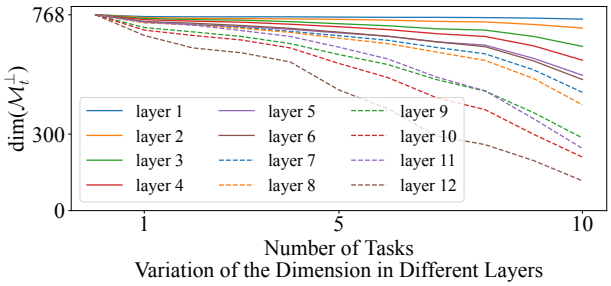


Figure 1. Change of the dimension of subspace $\mathcal{M}_t^\perp$ throughout the whole learning process.

the t-th task. DualGPM removes $\mathcal{Z}$ from $\mathcal{M}_t^\perp$ to get $\mathcal{M}_{t+1}^\perp$. Specifically, let $\hat{\boldsymbol{M}}_t^\perp = \boldsymbol{M}_t^\perp - \boldsymbol{Z}(\boldsymbol{Z}^T)\boldsymbol{M}_t^\perp$. DualGPM performs the second SVD on $\hat{\boldsymbol{M}}_t^\perp = \widetilde{\boldsymbol{U}}^\perp\widetilde{\boldsymbol{\Sigma}}^\perp(\widetilde{\boldsymbol{V}}^\perp)^T$. The columns of $\widetilde{\boldsymbol{U}}^\perp$ which correspond to the non-zero singular values form the bases $\boldsymbol{M}_{t+1}^\perp$. Please refer to the paper of DualGPM [2] for this explanation.

A.3. Approximation Error in DualGPM

DualGPM either learns a subspace $\mathcal{M}_t$ to approximate the gradient space of the old tasks or learns a subspace $\mathcal{M}_t^\perp$ to represent the orthogonal complement of the gradient of the old tasks. From Seciton A.2, we can find that the approximation error is related to the hyperparameter ϵ_{th} in (4) and (6). Specifically, when the value of ϵ_{th} in (4) and (6) increases, the approximation error decreases. As a result, the dimension of subspace $\mathcal{M}_t$ becomes larger, while the dimension of $\mathcal{M}_t^\perp$ becomes smaller. Note that our InfLoRA constrains the update of the model to lie within the subspace $\mathcal{N}_t \cap \mathcal{M}_t^\perp \subseteq \mathcal{M}_t^\perp$. Therefore, we can adjust the value of ϵ_{th} to adjust the space for learning the new task. Here, for all the experiments, we set

$$\epsilon_{th} = \epsilon + \frac{(1-\epsilon) * t}{T}, \tag{7}$$

where t denotes the task id and T denotes the total number of tasks. In other words, we gradually increase the value of ϵ_{th} as the number of tasks increases throughout the whole learning process. Table 1 shows the setting of ϵ in our InfLoRA.

Figure 1 illustrates the variation of the dimension of the subspace $\mathcal{M}_t^\perp$ in different Transformer layers of ViT-B/16. We can find that the dimension of the subspace $\mathcal{M}_t^\perp$ in different Transformer layers of ViT-B/16 is always much larger than zero, which means the space for learning the new task always exists throughout the whole learning process.

B. More Experimental Details

B.1. Training Details

For all the methods in all the experiments except for the comparison with SLCA, the batch size is set to 128 to follow many existing continual learning methods based on PEFT [4, 5]. Hyperparameters for different methods are selected based on the experimental settings in existing works [1, 4, 7] or through hyperparameter search. For example, Adam is used as the optimizer with running averages of gradient and its square ($\beta_1 = 0.9$, $\beta_2 = 0.999$). The learning rate is searched among [5e-4, 1e-3, 2e-3, 1e-2] for all the methods through the validation sets we split from the training sets. For the hyperparameter r in our InfLoRA, we search it among [1, 5, 10, 20, 30] through the validation sets we split from the training sets. Table 1 shows the hyperparameters of different methods.

When compared with SLCA, our method is combined with classifier alignment (CA). At this time, we follow SLCA to train the expanded LoRA branches and classifiers using the SGD optimizer. Each task is trained for 50 epochs on ImageNet-R, 20 epochs on CIFAR100 and 5 epochs on DomainNet. The batch size is set to 128.

B.2. Expanded Parameters

For L2P [7], the expanded parameters consist of the inserted prompts and their corresponding keys. Let d denote the embedding dimension, e denote the prompt length, p denote the number of prompts, and l denote the number of layers in which prompts are inserted. To compute the total number of expanded parameters, the formula used is $dlp(e+1)$.

For DualPrompt [6], the expanded parameters also consist of the inserted prompts and corresponding keys. However, DualPrompt contains expert prompts and shared prompts. Let d denote the embedding dimension, T denote the number of tasks, e_E denote the expert prompt length, e_S denote the shared prompt length, l_E denote the number of layers in which expert prompts are inserted and l_S denote the number of layers in which shared prompts are inserted. To compute the total number of expanded parameters, the formula used is $d[Tl_E(e_E+1)+e_Sl_S]$.

For CODA-Prompt [4], the expanded parameters consist of the inserted prompts, corresponding keys and attention parameters. Let d denote the embedding dimension, e denote the prompt length, p denote the number of prompts, and l denote the number of layers in which prompts are inserted. To compute the total number of expanded parameters, the formula used is $dlp(e+2)$.

For LAE [1], we implement it with LoRA. Therefore, the expanded parameters in this method consist of the inserted LoRA modules and the corresponding ensemble modules. Let d denote the embedding dimension, r denote the rank, and l denote the number of layers in which LoRA modules are inserted. Since LAE inserts LoRA modules into key and value projection in multi-head attention, the number of expanded parameters is $8ldr$.

For C-LoRA [3], the expanded parameters in this method consist of the inserted LoRA modules. Let d denote the

Table 1. List of hyper-parameters for different methods. The meaning of different hyperparameters is given in Section B.2. The hyperparameter ϵ in InfLoRA is explained in Section A.3

Methods	Hyper-Parameters
L2P	lr: 0.001 (ImageNet-R, DomainNet, CIFAR100) l: 1 (ImageNet-R, DomainNet, CIFAR100) p: 30 (ImageNet-R, DomainNet, CIFAR100) e: 20 (ImageNet-R, DomainNet, CIFAR100)
DualPrompt	lr: 0.001 (ImageNet-R, DomainNet, CIFAR100) l_E: 3 (ImageNet-R, DomainNet, CIFAR100) l_S: 2 (ImageNet-R, DomainNet, CIFAR100) e_E: 20 (ImageNet-R, DomainNet, CIFAR100) e_S: 6 (ImageNet-R, DomainNet, CIFAR100)
CODA-P	lr: 0.001 (ImageNet-R, DomainNet, CIFAR100) l: 5 (ImageNet-R, DomainNet, CIFAR100) p: 100 (ImageNet-R, DomainNet, CIFAR100) e: 8 (ImageNet-R, DomainNet, CIFAR100)
LAE	lr: 0.001 (ImageNet-R, DomainNet, CIFAR100) r: 5 (ImageNet-R, DomainNet, CIFAR100)
C-LoRA	lr: 0.001 (ImageNet-R, DomainNet, CIFAR100) r: 64 (ImageNet-R, DomainNet, CIFAR100) λ: 0.5 (ImageNet-R, DomainNet, CIFAR100)
InfLoRA-b5	lr: 0.001 (CIFAR100), 0.0005 (ImageNet-R, DomainNet) r: 10 (ImageNet-R, CIFAR100), 20 (DomainNet) ϵ: 0.99 (ImageNet-R), 0.95 (CIFAR100, DomainNet)
InfLoRA	lr: 0.0005 (ImageNet-R, DomainNet, CIFAR100) r: 10 (ImageNet-R, DomainNet, CIFAR100) ϵ: 0.98 (ImageNet-R), 0.95 (CIFAR100, DomainNet)

embedding dimension, r denote the rank, and l denote the number of layers in which LoRA modules are inserted. Since C-LoRA inserts LoRA modules into query, key and value projection in multi-head attention, the number of expanded parameters is $6ldr$.

For our methods, since we integrate the branches of the old tasks when the model learns a new task, the number of expanded parameters equals the number of parameters in a single branch. Let d denote the embedding dimension, r denote the rank, and l denote the number of layers in which our InfLoRA modules are inserted. Since we also insert InfLoRA modules into key and value projection in multi-head attention, the number of expanded parameters is $4ldr$.

C. More Experimental Results

C.1. Compare with More Methods

We compare with SeqLoRA, which initials LoRA modules and finetunes these modules on multiple tasks sequentially without any operation to overcome forgetting. The results are given in Table 2, Table 3 and Table 4. We can find that our method outperforms this method.

A recent continual learning PEFT method hierarchical decomposition prompt (HiDe-Prompt) [5] proposes to perform continual learning hierarchically. This method maintains a set of task-specific prompts for each task and contains two stages during training and inference. Specifically, given an input sample, Hide-Prompt infers the prompt index and then uses the corresponding prompt to infer its label. We also compare our method with this method, and the results are also given in Table 2, Table 3 and Table 4. We can find that our method outperforms this method. Furthermore, this method shows comparable performance to our method in terms of final accuracy ACC_T on ImageNet-R. However, there is a notable gap between this method and our method in terms of averaged accuracy $\overline{ACC}_T$. Note that averaged accuracy $\overline{ACC}_T$ is more important than final accuracy ACC_T since $\overline{ACC}_T$ represents the performance of the model over the whole learning process.

C.2. Hyperparameter Analysis

We perform the hyperparameter analysis for our method InfLoRA. There are two specific hyperparameters in our method InfLoRA. The first hyperparameter is r, which con-

Table 2. The comparison between our InfLoRA and more methods on ImageNet-R.

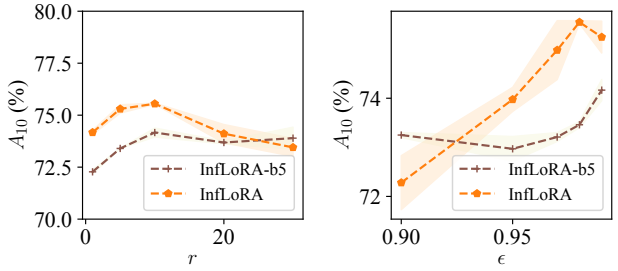
Tasks	5		10		20	
Method	ACC_5 (↑)	$\overline{ACC}_5$ (↑)	ACC_{10} (↑)	$\overline{ACC}_{10}$ (↑)	ACC_{20} (↑)	$\overline{ACC}_{20}$ (↑)
SeqLoRA	70.96 ± 0.25	79.14 ± 0.32	64.32 ± 0.09	74.78 ± 0.29	56.98 ± 0.29	69.29 ± 0.26
HiDe-Prompt [5]	76.82 ± 0.91	77.19 ± 0.34	75.06 ± 0.12	76.60 ± 0.01	66.88 ± 1.29	76.71 ± 0.23
InfLoRA	$\mathbf{77.52 \pm 0.37}$	$\mathbf{82.01 \pm 0.12}$	$\mathbf{75.65 \pm 0.14}$	$\mathbf{80.82 \pm 0.24}$	$\mathbf{71.01 \pm 0.45}$	$\mathbf{77.28 \pm 0.45}$

Table 3. The comparison between our InfLoRA and more methods on DomainNet.

	ACC_5 (↑)	$\overline{ACC}_5$ (↑)
SeqLoRA	71.69 ± 0.13	78.68 ± 0.12
HiDe-Prompt [5]	71.48 ± 0.10	76.15 ± 0.05
InfLoRA	$\mathbf{74.53 \pm 0.23}$	$\mathbf{79.57 \pm 0.57}$

Table 4. Results (%) of different methods on ImageNet-R (10 tasks) using various self-supervised pre-trained models. Here, DINO-1k and iBOT-1k indicate that the ViT is pre-trained on ImageNet-1k using these respective methods.

	Method	ACC_{10} (↑)	$\overline{ACC}_{10}$ (↑)
DINO-1k	SeqLoRA	60.67 ± 0.11	66.29 ± 0.21
	HiDe-Prompt [5]	68.11 ± 0.18	71.70 ± 0.01
	InfLoRA	$\mathbf{68.31 \pm 0.28}$	$\mathbf{76.15 \pm 0.05}$
iBOT-1k	SeqLoRA	66.87 ± 0.40	71.80 ± 0.28
	HiDe-Prompt [5]	71.33 ± 0.21	73.62 ± 0.13
	InfLoRA	$\mathbf{71.84 \pm 0.09}$	$\mathbf{78.29 \pm 0.09}$



(a) ImageNet-R (b) ImageNet-R

Figure 2. (a) Analysis of the hyperparameter r. (b) Analysis of the hyperparameter ϵ.

trols the expanded parameters in InfLoRA. The second hyperparameter is ϵ, which is not the specific hyperparameter of our InfLoRA but the hyperparameter introduced by DualGPM. This hyperparameter controls the component maintained in the matrix $\boldsymbol{M}_t$.

Figure 2 shows the results of our method with different values of r or ϵ. We can find that the performance of InfLoRA increases first and then decreases with the increase of r and ϵ.

Table 5. Results of DomainNet for domain incremental setting.

Method	ACC_6 (↑)	$\overline{ACC}_6$ (↑)
L2P [7]	34.15 ± 0.10	49.84 ± 0.03
DualPrompt [6]	35.24 ± 0.12	48.44 ± 0.13
CODA-P [4]	56.89 ± 0.04	57.56 ± 0.03
C-LoRA [3]	44.96 ± 0.01	52.95 ± 0.08
InfLoRA	$\mathbf{68.44 \pm 0.04}$	$\mathbf{67.46 \pm 0.03}$

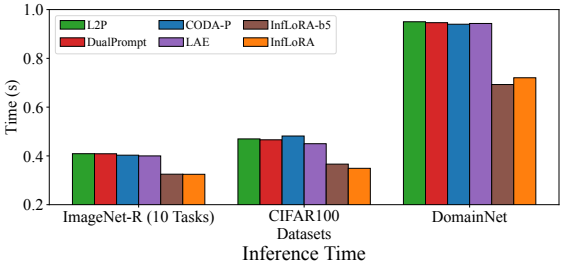


Figure 3. The time of inferring one task for different methods.

C.3. Domain Incremental Setting

InfLoRA can be extended to the domain incremental setting. Specifically, DomainNet contains six domains and InfLoRA can learn on these domains sequentially. Table 5 shows that InfLoRA outperforms other baselines.

C.4. Inference Efficiency

Existing methods often involve multiple forward propagations through the pre-trained backbone. Specifically, prompt-based continual learning methods, including L2P, DualPrompt, and CODA-P, require an extra forward propagation to generate instance-specific prompts. LAE requires an extra forward propagation for ensembling. In contrast, our InfLoRA only requires a single forward propagation through the pre-trained backbone. Figure 3 provides a comparison of the time consumed by different methods during inference. We can find that our method consistently outperforms existing methods in terms of time efficiency.

References

[1] Qiankun Gao, Chen Zhao, Yifan Sun, Teng Xi, Gang Zhang, Bernard Ghanem, and Jian Zhang. A unified continual learning framework with general parameter-efficient tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11449–11459, 2023. 2

[2] Yan-Shuo Liang and Wu-Jun Li. Adaptive plasticity improvement for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7816–7825, 2023. 1, 2

[3] James Seale Smith, Yen-Chang Hsu, Lingyu Zhang, Ting Hua, Zsolt Kira, Yilin Shen, and Hongxia Jin. Continual diffusion: Continual customization of text-to-image diffusion with c-lora. *CoRR*, 2023. 2, 4

[4] 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. 2, 4

[5] Liyuan Wang, Jingyi Xie, Xingxing Zhang, Mingyi Huang, Hang Su, and Jun Zhu. Hierarchical decomposition of prompt-based continual learning: Rethinking obscured sub-optimality. *arXiv preprint arXiv:2310.07234*, 2023. 2, 3, 4

[6] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer G. Dy, and Tomas Pfister. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *Proceedings of the European Conference on Computer Vision*, pages 631–648, 2022. 2, 4

[7] 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. 2, 4

[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.