Does Continual Learning Equally Forget All Parameters?

Haiyan Zhao [1] **Tianyi Zhou** [2 3] **Guodong Long** [1] **Jing Jiang** [1] **Chengqi Zhang** [1]

Abstract

Continual learning (CL) on neural networks suffers from catastrophic forgetting due to the distribution or task shift. In this paper, we study which parts of neural nets are more prone to forgetting by investigating their training dynamics during CL. We discover that only a few modules (e.g., batch-norm, last layer, earlier convolutional layers) are more task-specific and sensitively alters between tasks, while others can be shared across tasks as common knowledge. Hence, we attribute forgetting mainly to the former and find that finetuning them on only a small buffer at the end of any CL method can bring non-trivial improvement. Due to their few parameters, such "Forgetting Prioritized Finetuning (FPF)" is efficient and only requires a small buffer to retain the previous tasks. We further develop an even simpler replay-free method that applies FPF k-times during CL to replace the costly every-step replay. Surprisingly, this "k-FPF" performs comparably to FPF and outperforms the state-of-the-art CL methods but significantly reduces their computational overhead and cost. In experiments on several benchmarks of class- and domain-incremental CL, FPF consistently improves existing CL methods by a large margin and k-FPF further excels on the efficiency without degrading the accuracy.

1. Introduction

Deep learning has achieved unprecedented promising performance on challenging tasks under the i.i.d. offline setting. However, its reliability and performance degenerates drastically in the continual learning (CL) where the data distribution in training is changing over time, because the model can quickly adapt to a new task and overwrite the previously learned weights, which leads to "catastrophic forgetting" of previously learned knowledge.

A widely studied strategy to mitigate forgetting is experience replay (ER) (Riemer et al., 2018), which store a few data from previous tasks and train the model using both the current and buffered data. However, they only bring marginal improvements when the memory is small. In contrast, multi-task learning (Caruana, 1997) usually adopts a model architecture composed of a task-agnostic backbone network and multiple task-specific adaptors on top of it. In CL, however, we cannot explicitly pre-define and separate the task-agnostic parts and task-specific parts.

In this paper, we study a fundamental but open problem in CL, i.e., are most parameters task-specific and sensitively changing with the distribution shift? In addition, how many task-specific parameters suffice to achieve promising performance on new task(s)? Is every-step-replay necessary? To answer the above questions, we conduct extensive empirical studies of the training dynamics of model parameters during CL. Over multiple datasets and different CL methods and scenarios, we consistently observe that **only a few parameters change more drastically than others between tasks**. The results indicate that most parameters can be shared across tasks and **we only need to finetune a few task-specific parameters to retain the previous tasks' performance**.

The empirical studies motivate a simple yet effective method, "forgetting prioritized finetuning (FPF)", which finetunes the task-specific parameters using buffered data at the end of CL methods. Surprisingly, on multiple datasets, FPF consistently improves several widely-studied CL methods and substantially outperforms a variety of baselines. Moreover, we extend FPF to a **replay-free CL method** "k-FPF" that saves the cost of time-consuming every-step replay by replacing such frequent replay with occasional FPF. k-FPF applies FPF only k times during CL. We show that a relatively small k suffices to enable k-FPF achieving comparable performance with that of FPF+SoTA CL methods and meanwhile significantly reduces the computational cost. In addition, we explore different groups of parameters to finetune in FPF and k-FPF by ranking their

[*]Equal contribution [1]Australian Artificial Intelligence Institute, University of Technology Sydney [2]University of Washington, Seattle [3]University of Maryland, College Park. Correspondence to: Haiyan Zhao <Haiyan.Zhao-2@student.uts.edu.au>, Tianyi Zhou <tianyizh@uw.edu>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

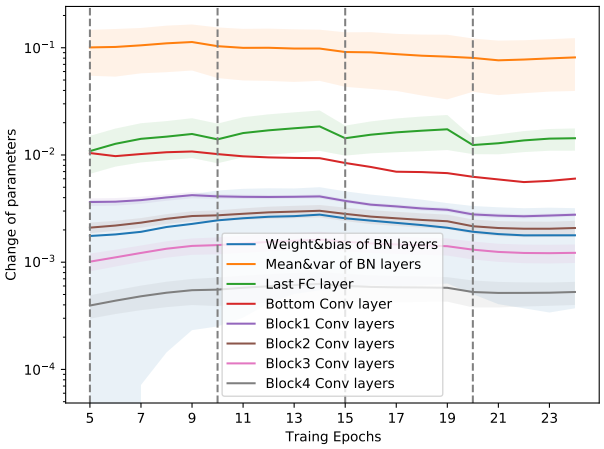

Figure 1. The training dynamics of different group of parameters which apply ER to train Seq-CIFAR-10. The parameter difference between the same epochs in two consecutive tasks during CL. Note the logarithmic scale on the y-axis.

sensitivity to task shift evaluated in the empirical studies. FPF can significantly improve existing CL methods by only finetuning batch-norm (BN) and last fully-connected (FC) layers (0.127% parameters). k-FPF achieves a promising trade-off between efficiency and performance when finetuning BN, FC, and earlier convolutional layers.

2. Problem Setup

Notations We consider the CL setting, where the model is trained on a sequence of tasks indexed by $t = 1, 2, \ldots, T$. During each task t, the training samples (x, y) (with label y) are drawn from an i.i.d. distribution D_t. Given a neural network $f(\cdot)$ of L layers with parameter $\theta = \theta_{\cdot;i}{}_{i=1:L}$, $\theta_{\cdot;i} = \theta_{\cdot;i}{}_{i=1:n_\ell}$ denote all parameters in layer-ℓ where $\theta_{\cdot;i}$ denotes parameter-i. On each task, $f(\cdot)$ is trained for N epochs. We denote all parameters and the layer-ℓ's parameters at the end of the n-th epoch of task t by θ^t_n and $\theta^t_{\cdot;n}$, $n = 1, \ldots, N$, respectively.

Settings In this paper, we mainly focus on class-incremental learning (class-IL) and domain-incremental learning (domain-IL). In class-IL, D_t are drawn from a subset of classes C_t and $C_t{}_{t=1}^T$ for different tasks are assumed to be disjoint. In domain-IL, tasks to be learnt remain the same but the input data distribution D_t changes. The model is expected to adapt to the new domain without forgetting the old ones. The goal of the class-IL and domain-IL is: $\min L(\cdot) \triangleq \sum_{t=1}^T \mathbb{E}_{(x;y)\ D_t}[l(y, f(x))]$. We conduct class-IL experiments on Seq-OrganAMNIST, Seq-PathMNIST, Seq-CIFAR-10, and Seq-TinyImageNet. Seq-OrganAMNIST and Seq-PathMnist are generated by splitting OrganAMNIST or PathMNIST from the medical dataset MedMNIST(Yang et al., 2021). Both datasets consist of 4 disjoint classification tasks. The number of classes for each task in Seq-OrganAMNIST and Seq-PathMnist are [3, 3, 3, 2] and [3, 2, 2, 2] respectively. Seq-CIFAR-10 are generated by splitting the 10 classes in CIFAR-10(Krizhevsky et al., 2009) into five binary classification tasks. Seq-TinyImageNet partitions the 200 classes of Tiny-ImageNet(Le & Yang, 2015) into 10 disjoint classification tasks with 20 classes per task. We conduct domain-IL experiments on PACS dataset (Li et al., 2017). In the Seq-PACS dataset for CL, each task only focuses on one domain and the sequence of tasks follows Sketches ! Cartoons ! Paintings ! Photos (increasing the level of realism over time) (Volpi et al., 2021). We follow the standard network architectures adopted in most previous CL works. Following (Rebuffi et al., 2017; Li et al., 2020; Derakhshani et al., 2022), we train ResNet-18 (He et al., 2016) on all the five datasets.

3. Forgetting Effects on Different Parameters: An Empirical study

A fundamental and long-lasting question in CL is how the distribution shift changes the model parameters and how these changes lead to harmful forgetting. In order to measure the forgetting effects on parameters between tasks, we explore two different metrics calculated from the training dynamics. The experimental results show patterns of these metrics consistently holding in different settings, from which we can allocate the task-specific parameters.

3.1. Measuring Forgetting via Training Dynamics

To measure and compare the forgetting effects of different parameters, we adopt an intuitive metric to compute the change of parameters and investigate their dynamics over CL. Since the major changes of parameters are caused by the task shift, we study a metric that mainly compares the difference between two consecutive tasks. It computes the change of parameters between epoch-n in two consecutive tasks, i.e., $(1/j\theta_{\cdot}j)k\theta^{t+1}_{\cdot;n} - \theta^t_{\cdot;n}k_1$. We report the new metric computed for running ER on Seq-CIFAR-10 to train ResNet-18 with buffer size of 500 and $N = 5$ epochs per task in Fig. 1. A primary discovery from this plot is: all BN layers' mean and variance, the last FC layer, and the earlier convolutional layers are much more task-specific and sensitive to the task shift. This observation makes intuitive sense because the bottom convolutional layer and last FC layer are closest to the inputs and outputs whose distributions change between tasks, and the mean and variance of BN capture the first and second order moments of the distribution for the latent representations. Hence, they change sensitively with the tasks, implying they are a main reason for the catastrophic forgetting. In addition, it is worth noting that these parameters only constitute a small portion of the whole neural net. Therefore, a few buffered data might suffice for accurate finetuning on all tasks. This observation is consistent for different scenarios, and the results are shown in appendix.

Table 1. Test accuracy (%) of CL baselines, FPF and k-FPF. "-" indicates that the algorithm is not applicable to the setting. For FPF and k-FPF, we report the best performance among all combinations of parameters in Fig. 3. k-FPF-KD applies an additional knowledge distillation loss to the finetuning objective of k-FPF-SGD. **Bold** and **Bold gray** mark the best and second best accuracy.

BUFFER	METHODS	CLASS-IL								DOMAIN-IL	
		SEQ-ORGANAMNIST		SEQ-PATHMNIST		SEQ-CIFAR-10		SEQ-TINY-IMAGENET		SEQ-PACS	
	JOINT	91.92	0.46	82.47	2.99	81.05	1.67	41.57	0.55	42.96	8.90
	SGD	24.19	0.15	23.65	0.07	19.34	0.06	7.10	0.14	31.43	6.39
	OEWC (SCHWARZ ET AL., 2018)	22.71	0.67	22.36	1.18	18.48	0.71	6.58	0.12	35.96	4.59
500	GDUMB (PRABHU ET AL., 2020)	73.29	1.82	63.55	5.62	42.18	2.05	3.67	0.25	43.29	2.53
	k-**FPF**+SGD	81.28	0.71	76.72	1.94	64.35	0.87	19.57	0.37	65.90	0.72
	k-**FPF**+KD	85.16	0.67	**79.20**	**3.89**	66.43	0.50	**20.56**	**0.32**	**66.42**	**2.21**
	ER (RIEMER ET AL., 2018)	80.45	0.99	57.54	3.05	57.64	4.27	10.09	0.34	52.72	4.01
	FPF+ER	84.07	1.26	69.83	2.87	65.47	2.64	18.61	0.70	64.27	1.91
	AGEM (CHAUDHRY ET AL., 2018)	24.00	0.18	27.33	3.93	19.47	0.03	7.14	0.10	35.29	4.94
	FPF+AGEM	79.86	0.88	73.32	3.73	57.84	1.98	17.35	0.65	62.40	1.89
	ICARL (REBUFFI ET AL., 2017)	82.95	0.47	57.67	1.13	62.26	1.09	14.81	0.37	-	
	FPF+ICARL	84.53	0.37	74.35	4.89	**67.75**	**0.67**	17.37	0.35	-	
	FDR (BENJAMIN ET AL., 2018)	76.62	1.81	40.08	4.13	43.52	1.74	11.33	0.33	48.50	4.67
	FPF+FDR	82.32	0.91	75.59	2.64	63.82	0.69	17.94	0.56	65.47	1.13
	DER (BUZZEGA ET AL., 2020)	82.52	0.52	66.71	3.40	55.98	3.35	11.54	0.70	47.63	3.85
	FPF+DER	**85.24**	**0.55**	74.80	3.45	67.52	0.83	17.60	0.50	65.69	1.66
	DER++ (BUZZEGA ET AL., 2020)	84.25	0.47	71.09	2.60	67.06	0.31	17.14	0.66	57.77	2.54
	FPF+DER++	**85.67**	**0.23**	**77.37**	**1.32**	**69.09**	**0.74**	**20.17**	**0.35**	**66.89**	**1.32**

4. Forgetting Prioritized Finetuning (FPF) Methods

The above empirical study of the parameter training dynamics immediately motivates a simple baselines for CL, i.e., "forgetting prioritized finetuning (FPF)".

FPF to improve CL performance. FPF applies lightweight finetuning to the most task-specific parameters using the buffered data before deployment of the CL model generated by arbitrary CL methods. Hence, it is complementary to any existing CL methods as a correction step to remove their biases in the task-specific parameters by finetuning them on the unbiased buffer data. Thereby, it can improve the performance of any existing CL methods without causing notably extra computation.

k-FPF to improve CL efficiency. FPF is a simple technique that brings non-trivial improvement but it is applied to an existing CL method. Unfortunately, many SoTA CL methods require time-consuming replay in every step, which doubles the total computation. We propose k-FPF that applies FPF k times during CL. Without the costly experience replay, k-FPF can still achieve comparable performance as FPF+SoTA CL methods but spend nearly half of their computation. Specifically, we can apply k-FPF with any replay-free algorithms, e.g., SGD, which updates the model solely on the stream of tasks and their incoming data, and is usually used as a lower-bound for CL methods. We still maintain a small buffer by reservoir sampling but SGD does not access it. We only apply FPF after every τ SGD steps (in total k times in $k\tau$ SGD steps) on the buffer without knowing the task boundaries.

We propose two variants of k-FPF, i.e., k-FPF-SGD and k-FPF-KD. k-FPF-SGD uses the cross-entropy loss to update the parameters during FPF. Inspired by DER (Buzzega et al., 2020), we further propose k-FPF-KD that introduces knowledge distillation (KD) (Hinton et al., 2015) to the FPF objective in k-FPF-SGD. During FPF, the current model is trained to match the buffered logits to retain the knowledge of previous models. Compared to the computation of every-step SGD in CL, the additional computation by k-FPF-KD is negligible.

5. Experiments

In this section, we applied FPF and k-FPF to multiple benchmark datasets and compare them with SoTA CL baselines in terms of test accuracy and efficiency. Besides, we also compare the performance of finetuning different parameters in FPF and k-FPF and show that finetuning a small portion of task-specific parameters suffices to improve CL. More results and analysis are show in appendix.

Implementation Details. We follow the settings in (Buzzega et al., 2020) to train various SoTA CL methods on different datasets. Each task is trained for 5 epochs. For both FPF and k-FPF, we use the same optimizer, i.e., stochastic gradient descent with the cosine-annealing learning rate schedule, and finetune the selected parameters with a batchsize of 32 for all scenarios. The finetuning steps for FPF and k-FPF are 300 and 100 respectively. We perform a grid-search on the validation set to tune the learning rate and other hyper-parameters in our experiments.We apply FPF to several SoTA CL methods listed in Table 1. We report the test accuracy of these baseline methods and the best test accuracy of FPF and k-FPF among different combinations

of task-specific parameters. We take JOINT as the upper bound for CL which trains all tasks jointly and SGD as the lower bound which trains tasks sequentially without any countermeasure to forgetting. All results reported in Table 1 are averaged over five trials with different random seeds.

FPF considerably improves the performance of all memory-based CL methods and achieves SoTA performance over all scenarios in class-IL and domain-IL in Table 1. For methods with catastrophic forgetting, like AGEM, the accuracy of FPF increases exponentially. The surge of performance illustrates that FPF can eliminate bias by finetuning task-specific parameters to adapt to all seen tasks.

k-FPF-SGD removes the costly every-step replay with efficient occasional FPF. In Table 1, the performance of k-FPF-SGD on Seq-PathMNIST, Seq-Tiny-ImageNet and Seq-PACS are better than the best CL methods and its performance on Seq-OrganAMNIST and Seq-Cifar10 are also better than most CL methods, which implies that finetuning the task-specific parameters on a small number of buffer during SGD can help retain the previous knowledge and mitigate forgetting, each-step replay is not necessary. In Fig. 2, the number of training FLOPs and accuracy of different methods are reported. Compared to the training FLOPs of several CL methods, the computation cost of FPF and k-FPF-SGD is almost negligible. The overall training FLOPs of k-FPF-SGD is still much less than SoTA CL methods while its performance are better, which show the efficiency of k-FPF.

k-FPF-KD further improves the performance of k-FPF-SGD to be comparable to FPF. k-FPF-SGD propose the efficiency of CL methods, but its performance is a bit worse than that of FPF. Inspired by DER, we propose k-FPF-KD, which introduce knowledge distillation to drive the current model to match the output of previous models on buffer data to retain the knowledge of previous tasks. The results of k-FPF-KD in Table 1 show that it is comparable to FPF in most scenarios. In Fig. 2, we can find that the FLOPs of k-FPF-KD is similar to k-FPF-SGD and much less than other CL methods and FPF, but in some cases, it outperforms FPF. k-FPF-KD shows SoTA performance in both efficiency and accuracy.

Figure 2. Comparison of FLOPs and accuracy between k-FPF, FPF and SoTA CL methods. k-FPF improves all CL methods a lot without notably extra computation. k-FPF consumes much less computation but achieves comparable performance as FPF.

Figure 3. Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF(Top) and k-FPF(Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in the x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show both FPF and k-FPF only require to finetune a small portion of task-specific parameters. k-FPF halves FPF's FLOPs.

5.1. Comparison of finetuning different parameters in FPF and k-FPF

FPF and k-FPF get the best performance when only a small portion of task-specific parameters are finetuned. In Fig. 3, the accuracy, training FLOPs and number of trainable parameters during finetuning of applying FPF or k-FPF to different task-specific parameters are compared. Over all different scenarios, k-FPF only needs about half FLOPs of FPF with better performance (indicated by Red Stars). When finetuning on different task-specific parameters, FPF get the best performance when BN+FC layers are finetuned, which is only 0.127% of all parameters (indicated by Orange Stars). This is consistent with our observations in empirical studies where BN and FC layers are the most sensitive parameters to distribution shift. And the results shows that only finetuning a small portion of task-specific parameters can mitigate catastrophic forgetting and generalize the model.

The phenomenon for k-FPF is a little different. (1) In the bottom plot of Fig. 3, when FC layer is fixed for k-FPF, the performance is much worse. This is because in class-IL learning, the output classes of tasks change, and the current output classes may dominate all other classes (Hou et al., 2019). To prove this, we apply k-FPF to domain-IL like Seq-PACS, where the output classes for different tasks are the same. Fig. 6 in Appendix shows that the performance of finetuning FC only is similar to finetuning other parameters, which proves our assumptions. (2) As the red star indicates, a little more parameters (block3 of convolutional layers) are needed to be finetuned by k-FPF to achieve comparable accuracy with FPF, which is about 18.91% of all parameters. Without experience replay during CL method SGD, the model has a larger bias on current task, so more task-specific parameters are needed to be finetuned. As shown in the figure, when block4 of convolutional layers (about 75.22% of all parameters) are finetuned, which is the least sensitive parameters shown in our empirical study, the performance of

k-FPF degrades. This indicates that the bias of task-specific parameters is the main reason for catastrophic forgetting.

6. Conclusion

We study a fundamental problem in CL, i.e., which parts of a neural network are task-specific and more prone to catastrophic forgetting. Extensive empirical studies in diverse scenarios consistently show that only a small portion of parameters is task-specific and sensitive. This discovery leads to a simple yet effective FPF that only retunes a subset of these parameters on the buffered data before model deployment. FPF is complementary to existing CL methods and can consistently improve their performance. We further replace the costly every-step replay with k times of occasional FPF during CL to improve the efficiency. Such k-FPF achieves comparable performance as FPF+SoTA CL while consumes nearly half of its computation.

References

Benjamin, A. S., Rolnick, D., and Kording, K. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

Buzzega, P., Boschini, M., Porrello, A., Abati, D., and Calderara, S. Dark experience for general continual learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*, 2020.

Caruana, R. Multitask learning. *Machine learning*, 28(1): 41–75, 1997.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Derakhshani, M. M., Najdenkoska, I., van Sonsbeek, T., Zhen, X., Mahapatra, D., Worring, M., and Snoek, C. G. Lifelonger: A benchmark for continual disease classification. arXiv preprint arXiv:2204.05737, 2022.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., Dean, J., et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Hou, S., Pan, X., Loy, C. C., Wang, Z., and Lin, D. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 831–839, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Le, Y. and Yang, X. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. M. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pp. 5542–5550, 2017.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. Sequential learning for domain generalization. In *European Conference on Computer Vision*, pp. 603–619. Springer, 2020.

Prabhu, A., Torr, P. H., and Dokania, P. K. Gdumb: A simple approach that questions our progress in continual learning. In *European conference on computer vision*, pp. 524–540. Springer, 2020.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., and Tesauro, G. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pp. 4528–4537. PMLR, 2018.

Volpi, R., Larlus, D., and Rogez, G. Continual adaptation of visual representations via domain randomization and meta-learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4443–4453, 2021.

Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., and Ni, B. Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification. *arXiv preprint arXiv:2110.14795*, 2021.

Figure 4.The training dynamics of different group of parameters which apply ER to train Seq-CIFAR-10. The parameter difference between two consecutive epochs during CL. Note the logarithmic scale on the y-axis.

A. Another metric to compute the change of parameters and investigate their dynamics over CL.

This metric calculates the difference between two consecutive epochs, e.g., for parameter θ_j it computes (1) $(1/|\theta_j|)\|\theta^t_{j,n} - \theta^t_{j,n-1}\|_1$ between epoch $(n-1)$ and epoch n within a task t and (2) $(1/|\theta_j|)\|\theta^{t+1}_{j,1} - \theta^t_{j,N}\|_1$ between the last epoch of task t and the first epoch of task $(t+1)$. The training dynamics of this metric on different groups of parameters are shown in the Fig. 4, which is collected by running ER on Seq-CIFAR-10 to train ResNet-18 with buffer size of 500 and $N = 5$ epochs per task.

In this plot, we split all parameters into several groups, i.e., the weights and bias of all batch-norm (BN) layers, the mean and variance of all BN layers, the last fully-connected (FC) layer (closest to the output), the bottom convolutional layer (closest to the input), and convolutional layers in different blocks. For each group, the mean and standard deviation over all layers are reported. In the plot, all parameters experience more changes at the epoch of task switching and quickly converge after a few epochs in the same task. Hence, the dynamic patterns of this metric can be used to detect the task boundaries. Since tasks differ on their predicted classes in class-IL, the task shift results in the greatest changes on the FC layer. Moreover, these groups of parameters show different levels of sensitivity to the task shift, indicating that netuning on a few task-speci c parameters suf ces to retain the previous tasks.

B. Forgetting of Different Parameters During CL for Different Scenarios

In the above section, we observe that only a small portion of parameters, i.e., BN mean and variance, last FC layer, and earlier convolutional layers, are much more sensitive and task-speci c than other parameters during CL. However, the experiment is limited to one dataset, a speci ed neural network, one CL method, and speci ed hyper-parameters (e.g., buffer size) in class-IL. In the following, we conduct a more extensive study in different scenarios by varying these factors in class-IL and domain-IL. We will mainly focus on the second metric, which only compute the difference between epochs in two different tasks. Although the rst metric might exhibit other patterns, e.g., the task boundaries, its dynamics are less

stable than the second one. Surprisingly, we find that the previous observation consistently hold in all the evaluated cases.

Different datasets and architectures for class-IL.We first extend the empirical study to two additional datasets of diverse types of image data, i.e., Seq-TinyImageNet and Seq-OrganAMNIST, and the results are shown in Fig. 5(a)-(b). Comparing to Seq-CIFAR-10, they differ on the number of tasks, the dataset size, and the image size and type. We then extend the empirical study of ER on ResNet-18 to other neural network architectures, e.g., VGG-11-BN and WideResNet162 in Fig. 5(c)-(d). They differ on the depth, width, total size, and basic cells. Although the ranking of the bottom convolutional layer and the last FC layer exchanges in some cases, the batch-norm layer, last FC layer, and earlier convolutional layers are still the most sensitive groups.

Different buffer sizes and CL methods for class-IL.Increasing the buffer sizes can potentially alleviate the forgetting since it enable replay on more data from previous tasks. Hence, we change the buffer size from a smaller size 50 to a larger size2000when running ER on Seq-CIFAR10. The training dynamics of different parameters are reported in Fig. 5(e)-(f). We further extend the empirical study from ER to two other CL algorithms, i.e., SGD without any buffer replay, and DER based on knowledge distillation of previous models, whose results on Seq-CIFAR10 are reported in Fig. 5(g)-(h). After changing the buffer size and replay strategies, the ranking order of the sensitivity of different groups of parameters are always consistent.

Different scenarios for domain-IL In domain-IL, the training dynamics of different parameters is in line with our observations in class-IL and only a small portion of parameters are sensitive and task-specific. That being said, some notable difference between domain-IL and class-IL can be found through the training dynamics of parameters associated with different buffer sizes in Fig. 5(i)-(j) and different CL methods in Fig. 5(k)-(l). In class-IL, the last FC layer in most cases is more sensitive than the bottom convolutional layer to the change of output classes since its update strongly relates to the outputs. In domain-IL, since the output classes stay the same across tasks and only the input domain changes, the last FC layer is equally or less sensitive than the bottom convolutional layer.

Figure 5.The parameter difference between the same epochs of adjacent tasks during the course of CL for different scenarios. We discuss different datasets(a, b), different architectures(c,d), different buffer size(e, f) and different CL methods(g,h) for class-IL as well as different buffer size(i, j) and different CL methods(k,l) for domain-IL. Note the logarithmic scale on the y-axis.

Table 2. Test accuracy (%) of CL baselines, FPF and k-FPF. "-" indicates that the algorithm is not applicable to the setting. For FPF and k-FPF, we report the best performance among all combinations of parameters in Fig. 3. k-FPF-KD applies an additional knowledge distillation loss to the finetuning objective of k-FPF-SGD. **Bold** and **Bold gray** mark the best and second best accuracy.

BUFFER	METHODS	CLASS-IL								DOMAIN-IL	
		SEQ-ORGANAMNIST		SEQ-PATHMNIST		SEQ-CIFAR-10		SEQ-TINY-IMAGENET		SEQ-PACS	
	JOINT	91.92	0.46	82.47	2.99	81.05	1.67	41.57	0.55	42.96	8.90
	SGD	24.19	0.15	23.65	0.07	19.34	0.06	7.10	0.14	31.43	6.39
	OEWC (SCHWARZ ET AL., 2018)	22.71	0.67	22.36	1.18	18.48	0.71	6.58	0.12	35.96	4.59
200	GDUMB (PRABHU ET AL., 2020)	61.78	2.21	46.31	5.64	30.36	2.65	2.43	0.31	34.16	3.45
	k-**FPF**+SGD	75.21	2.03	**72.88**	**3.22**	57.97	1.53	13.76	0.72	60.70	2.81
	k-**FPF**+KD	**80.32**	**1.16**	**74.68**	**4.72**	58.50	1.03	**14.74**	**0.94**	63.15	1.19
	ER (RIEMER ET AL., 2018)	71.69	1.71	51.66	5.86	45.71	1.44	8.15	0.25	51.53	5.10
	FPF+ER	77.66	1.93	67.34	2.68	57.68	0.76	13.13	0.63	**65.16**	**1.97**
	AGEM (CHAUDHRY ET AL., 2018)	24.16	0.17	27.93	4.24	19.29	0.04	7.22	0.15	40.54	3.43
	FPF+AGEM	73.76	2.45	67.04	4.51	55.40	1.97	13.24	0.54	57.33	0.76
	ICARL (REBUFFI ET AL., 2017)	79.61	0.56	54.35	0.94	59.60	1.06	12.13	0.20	-	
	FPF+ICARL	80.24	0.70	71.83	1.51	**63.95**	**0.84**	**17.45**	**0.38**	-	
	FDR (BENJAMIN ET AL., 2018)	68.29	3.27	44.27	3.20	41.77	4.24	8.81	0.19	45.91	3.54
	FPF+FDR	76.92	1.38	70.08	4.06	52.49	2.97	12.25	0.77	58.38	1.70
	DER (BUZZEGA ET AL., 2020)	73.28	1.33	54.45	5.92	47.04	3.03	9.89	0.58	46.93	4.94
	FPF+DER	79.63	1.21	67.29	3.75	57.25	2.19	12.62	1.08	61.49	1.37
	DER++ (BUZZEGA ET AL., 2020)	78.22	2.05	62.00	3.79	59.13	0.81	12.12	0.69	55.75	2.02
	FPF+DER++	**80.99**	**0.91**	68.78	2.99	**61.98**	**1.04**	13.78	0.57	**65.28**	**1.02**
500	GDUMB (PRABHU ET AL., 2020)	73.29	1.82	63.55	5.62	42.18	2.05	3.67	0.25	43.29	2.53
	k-**FPF**+SGD	81.28	0.71	76.72	1.94	64.35	0.87	19.57	0.37	65.90	0.72
	k-**FPF**+KD	85.16	0.67	**79.20**	**3.89**	66.43	0.50	**20.56**	**0.32**	**66.42**	**2.21**
	ER (RIEMER ET AL., 2018)	80.45	0.99	57.54	3.05	57.64	4.27	10.09	0.34	52.72	4.01
	FPF+ER	84.07	1.26	69.83	2.87	65.47	2.64	18.61	0.70	64.27	1.91
	AGEM (CHAUDHRY ET AL., 2018)	24.00	0.18	27.33	3.93	19.47	0.03	7.14	0.10	35.29	4.94
	FPF+AGEM	79.86	0.88	73.32	3.73	57.84	1.98	17.35	0.65	62.40	1.89
	ICARL (REBUFFI ET AL., 2017)	82.95	0.47	57.67	1.13	62.26	1.09	14.81	0.37	-	
	FPF+ICARL	84.53	0.37	74.35	4.89	**67.75**	**0.67**	17.37	0.35	-	
	FDR (BENJAMIN ET AL., 2018)	76.62	1.81	40.08	4.13	43.52	1.74	11.33	0.33	48.50	4.67
	FPF+FDR	82.32	0.91	75.59	2.64	63.82	0.69	17.94	0.56	65.47	1.13
	DER (BUZZEGA ET AL., 2020)	82.52	0.52	66.71	3.40	55.98	3.35	11.54	0.70	47.63	3.85
	FPF+DER	**85.24**	**0.55**	74.80	3.45	67.52	0.83	17.60	0.50	65.69	1.66
	DER++ (BUZZEGA ET AL., 2020)	84.25	0.47	71.09	2.60	67.06	0.31	17.14	0.66	57.77	2.54
	FPF+DER++	**85.67**	**0.23**	**77.37**	**1.32**	**69.09**	**0.74**	**20.17**	**0.35**	**66.89**	**1.32**

C. Analysis of FPF and k-FPF in Different Scenarios

Different training FLOPs for k-FPF In Fig. 7(a), we show the trade-off between the training FLOPs and accuracy of applying k-FPF to Seq-PathMNIST. Each point in the figure represent running k-FPF with different k and number of finetuning steps. τ in the legend refers to the interval of two consecutive finetuning. For experiments with same k, k-FPF saturates quickly as the increase of number of finetuning steps. This implies that k-FPF can achieve best performance with low FLOPs and shows its efficiency. From Fig. 7(a), for experiments with small k, e.g. k=2, although the computation required is very low, performance cannot be further improved. This implies that more finetune times are needed, so the model can see more previous samples to mitigate forgetting. When the k is large, like k=41 or 121, the accuracy increases but much more computation are required. In this scenario, as the red star in the plot indicates, apply k-FPF every 1500 training steps can get the best trade-off between computation and accuracy.

D. Performance of finetuning different parameters for FPF and k-FPF on domain-IL dataset

In Figure 6, the performance of finetuning different parameters for FPF and k-FPF on domain-IL dataset Seq-PACS are reported.

Different buffer sizes and training epochs for FPF The buffer size and the number of training epochs for each task are always crucial in memory-based CL methods. As shown in plot (b) of Fig. 7, as the buffer size or number of epochs increases, the performance of ER become better as well. The increase of buffer size brings more benefits. When the buffer size or

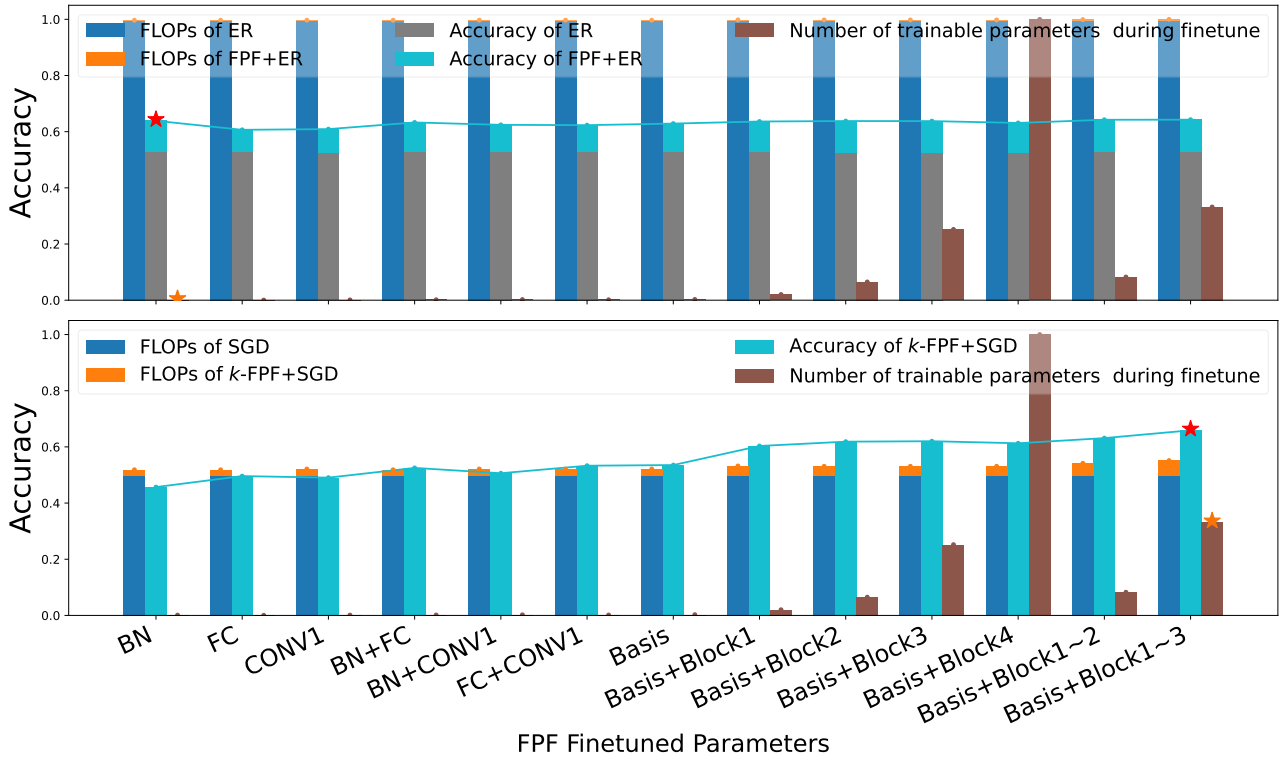

Figure 6. Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF(Top) and k-FPF(Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in the x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show both FPF and k-FPF only require to finetune a small portion of task-specific parameters. k-FPF halves FPF's FLOPs. **Different from the results of k-FPF in class-IL, in Seq-PACS, since the output classes for different tasks are always the same, the last FC layer will not have a large bias on particular classes. Only finetuning BN or CONV1 layers for k-FPF can get comparable performance with ER.** Similar to class-IL, since experience replay is not allowed during the training of CL method SGD, a little more parameters are required to be finetuned by k-FPF to get comparable performance with FPF (about 24.92% of all parameters).