Efficient Expansion and Gradient Based Task Inference for Replay Free Incremental Learning: Supplementary

Soumya Roy
Amazon
meetsoumyaroy@gmail.com

Vinay Verma
Amazon/Duke University
vinayugc@gmail.com

Deepak Gupta
Amazon
deepakgupta.cbs@gmail.com

In this supplementary material, we present additional ablation results and document our experimental settings.

1. Additional Ablations

1.1. Influence of seeds

We use 5 different seeds to understand how the results of our proposed method change with different seed values. The seed value impacts our method in four different ways - network initialization, sample order, random augmentation and class order. The question of seed influence is more important in adaptive parameter growth (APG) than static parameter growth (SPG) as APG uses task complexity to grow the model. As is clear from Table 1, our reported results are below the seed mean and reasonably stable for different seed values on all three splits of CIFAR-100. In Table 1, we also provide the parameter growth of a method averaged over all seeds and task sequences. It is interesting to note that the average parameter growth of the APG model is remarkably stable for different seeds. Thus, we can conclude that our methods are robust under different experimental conditions.

1.2. Task Prediction

For the sake of completeness of this paper, we present the average task prediction accuracy on the CIFAR100/5 split in Table. 2.

1.3. Task-wise accuracy

We present the task-wise accuracy of the SPG model on different splits of CIFAR-100 in Fig. 1. Since this is the CIL scenario, the accuracy of a task i refers to the average incremental accuracy *till* task i. To avoid clutter, we only present results of important baselines. It should be noted that different methods have different first task accuracies as they have different optimization hyperparameters and expansion/regularization strategies. For example, EFT [8] expands from the first task while IL2A [12] works better with the Adam optimizer [4]. Similarly, the results for task-wise accuracy of the SPG model on different splits of ImageNet-100 is shown in Fig. 2.

2. Experimental Settings

In this section, we provide details about our hyperparameter settings and baselines.

2.1. CIFAR-100

2.1.1 Training hyperparameters

Since EFT [8] is our best performing baseline, we borrow the class order and hyperparameter settings (including seed) from their publicly available code. We train our model for 250 epochs with batch size of 128, initial learning rate of 0.01, learning rate drop of 0.1 at 100, 150 and 200 epochs, SGD optimizer with momentum of 0.9 and weight decay of $5e-3$.

We use ResNet-18 [2] architecture for CIFAR datasets to evaluate our method. It should be noted that we train batch norm and linear layers from scratch for each task.

2.1.2 Expansion hyperparameters

We follow the same expansion hyperparameters for every task sequence. Let α_1, α_2, α_3 and α_4 be the number of filters used for creating the four residual blocks (using the $\_make\_layer()$ function in standard PyTorch implementation of ResNet-18 [1]). For each task, we increase the filters as follows:

$$\alpha_1 = \alpha_1 + 1,\ \alpha_2 = \alpha_2 + 5,\ \alpha_3 = \alpha_3 + 10,\ \alpha_4 = \alpha_4 + 10 \quad (1)$$

For the first task, α_1 = 64, α_2 = 128, α_3 = 256 and α_4 = 512 which is the standard ResNet-18 filter distribution. The criterion for selecting this hyperparameter is that we wanted to have an average parameter growth of around 4% like EFT [8].

[1] github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

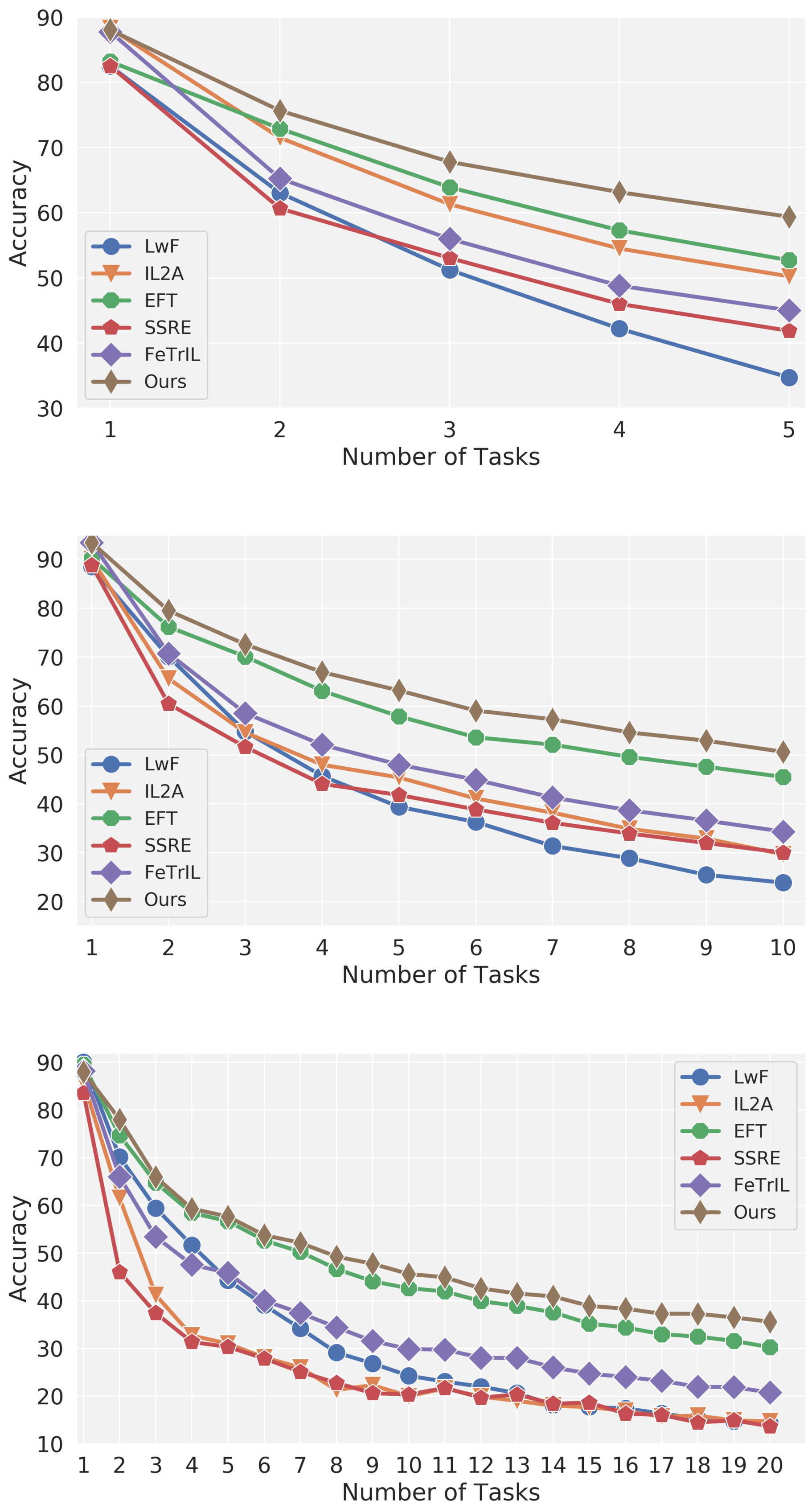

Figure 1. Task-wise CIL results of SPG model on 5, 10 and 20 splits of CIFAR-100.

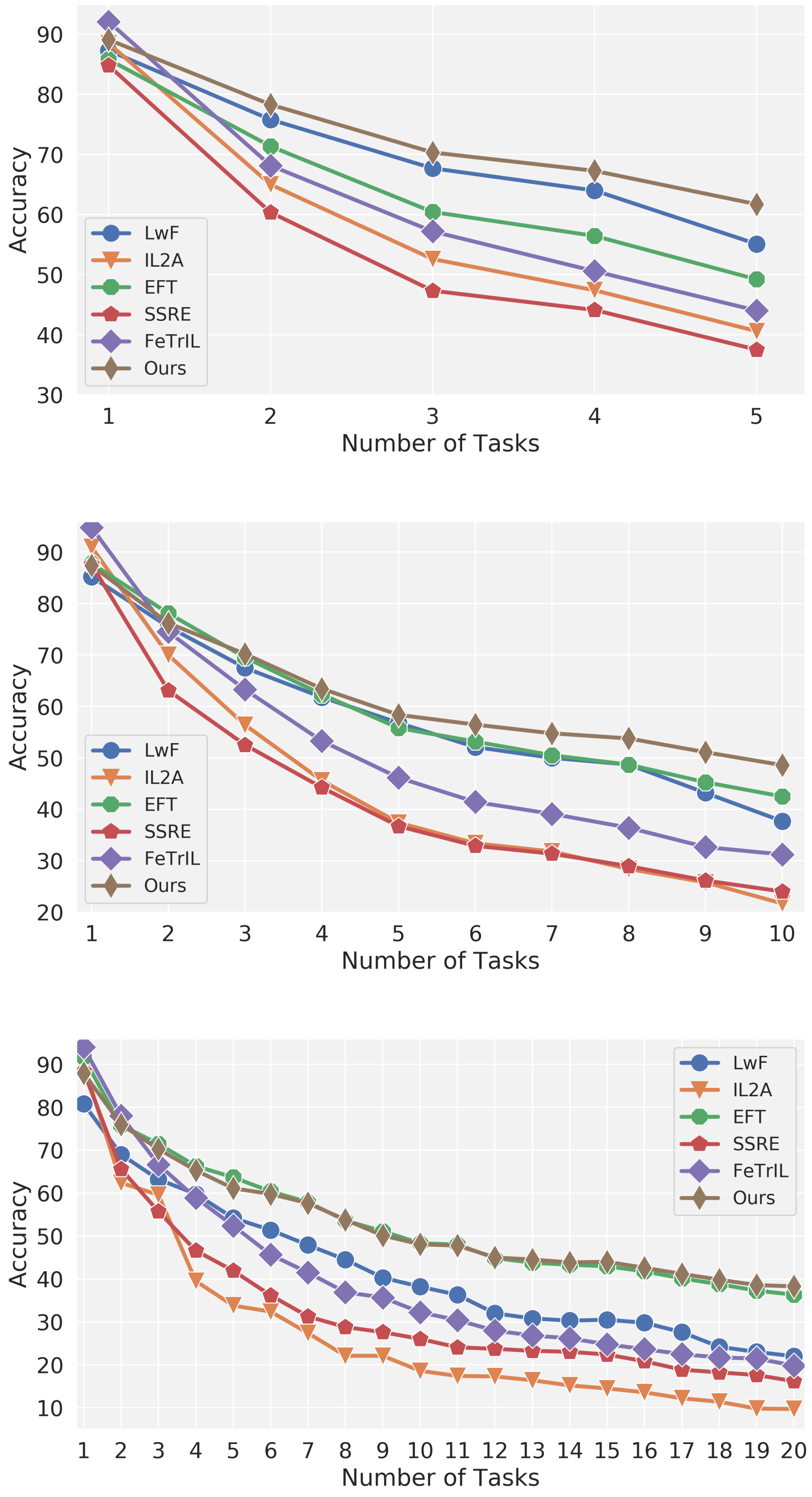

Figure 2. Task-wise CIL results of SPG model on 5, 10 and 20 splits of ImageNet-100.

Method	5	10	20
5 seeds (SPG) [4.1%]	59.8 ± 0.5	50.8 ± 0.7	36.9 ± 0.7
Reported (SPG) [4.1%]	59.4	50.6	35.6
5 seeds (APG) [3.6%]	59.3 ± 0.5	50.8 ± 0.5	37.4 ± 0.7
Reported (APG) [3.6%]	59.2	50.5	36.4

Table 1. Mean and standard deviation for different splits of CIFAR-100. [X%] shows the average parameter growth of the model over all task sequences.

Methods	Accuracy
Ensemble Class Prediction [3]	39.8
Entropy [8]	57.7
cross-entropy	57.2
∇(cross-entropy) + mean filters	58.8
∇(cross-entropy) + aug + mean filters	61.3
∇(cross-entropy) + entropy aug	61.2
∇(cross-entropy) + entropy aug + mean filters	**61.9**

Table 2. Average task prediction accuracy till last task for different task prediction methods on CIFAR100/5 split.

For adaptive parameter growth, we define the minimum filter growth as:

$$\alpha_1 = \alpha_1 + 1, \ \alpha_2 = \alpha_2 + 1, \ \alpha_3 = \alpha_3 + 1, \ \alpha_4 = \alpha_4 + 1 \quad (2)$$

The maximum filter growth is defined using Eq. 1.

2.1.3 Augmentations

For class incremental learning (CIL), we apply 10 instances of a random data augmentation scheme, along with the standard unaugmented test sample, to create the batch $\mathcal{X}_k$ (i.e., A = 11). It should be noted that we use the same random data augmentation scheme for task prediction that we use for training the network. Our data augmentation scheme is same as EFT [8], i.e., from PyTorch library, we use:

1. RandomCrop(32, padding=4)
2. RandomHorizontalFlip()
3. RandomRotation(10)

2.1.4 Baselines

We borrow most of the baseline results from the EFT paper. We also run the publicly available code of IL2A [12] using our class order, split (5/10/20) and seed setting. Results for SSRE [13] and FeTrIL [6] are obtained by running the PyCIL [11] framework using our class order, split and seed settings. It should be noted that in the main paper, we define average incremental accuracy as the average accuracy for all seen classes.

2.2. Tiny ImageNet

2.2.1 Training hyperparameters

Like CIFAR-100, we borrow the class order, hyperparameter settings (including seed) and baselines from the EFT [8] paper. We train our model for 140 epochs with batch size of 128, initial learning rate of 0.01, learning rate drop of 0.1 at 70, 100 and 120 epochs, SGD optimizer with momentum of 0.9 and weight decay of $5e-4$. To evaluate our method, we use the VGG-16 [7] architecture with batch norm for the Tiny ImageNet dataset.

2.2.2 Expansion hyperparameters

If $\alpha_{1,j}$ is the original number of filters for layer j in VGG-16 and $\alpha_{i,j}$ are their values before task $i+1$, then for task $i+1$, we increase the filters as follows:

$$\begin{aligned} \alpha_{i+1,j} &= \alpha_{i,j} + 1 \ if \ \alpha_{1,j} \ = \ 64 \ or \ 128 \\ \alpha_{i+1,j} &= \alpha_{i,j} + 8 \ if \ \alpha_{1,j} \ = \ 256 \ or \ 512 \end{aligned} \quad (3)$$

For adaptive parameter growth, we define the minimum filter growth as:

$$\alpha_{i+1,j} = \alpha_{i,j} + 1$$

We define the maximum filter growth using Eq. 3.

2.3. ImageNet-100

2.3.1 Training hyperparameters

We use the same class subset, class order and hyperparameter settings as DER [9]. We train our model for 120 epochs (unlike DER, we do not warm up) with batch size of 256, initial learning rate of 0.1, learning rate drop of 0.1 at 30, 60, 80 and 90 epochs, SGD optimizer with momentum of 0.9 and weight decay of $5e-4$.

We use ResNet-18 [2] architecture for ImageNet dataset to evaluate our method. It should be noted that we train batch norm and linear layers from scratch for each task.

2.3.2 Expansion hyperparameters

We follow the same expansion hyperparameters as CIFAR-100, except for the ImageNet-100/20 split. If α_1, α_2, α_3

and α_4 are the number of filters used for creating the four residual blocks (using the $\_make\_layer$ function in standard PyTorch implementation of ResNet-18), then for each task in ImageNet-100/20 split, we increase the filters as follows:

$$\alpha_1 = \alpha_1 + 2,\ \alpha_2 = \alpha_2 + 10,\ \alpha_3 = \alpha_3 + 10,\ \alpha_4 = \alpha_4 + 10 \tag{4}$$

For the first task, α_1 = 64, α_2 = 128, α_3 = 256 and α_4 = 512 which is the standard ResNet-18 filter distribution. This is because the ImageNet-100/20 split is harder than the corresponding CIFAR-100/20 split. For adaptive parameter growth and ImageNet-100/20 split, we define the minimum and maximum filter growths using Eq. 2 and Eq. 4 respectively.

2.3.3 Augmentations

For class incremental learning (CIL), we apply 20 instances of a random data augmentation scheme, along with the standard unaugmented test sample, to create the batch $\mathcal{X}_k$ (i.e., A = 21). It should be noted that we use the same random data augmentation scheme for task prediction that we use for training the network. Our data augmentation scheme is same as [1], i.e., from PyTorch library, we use:

1. RandomResizedCrop(224)
2. RandomHorizontalFlip()
3. ColorJitter(brightness=63 / 255)

2.3.4 Baselines

We run the baselines LwF [5], EFT [8] and IL2A [12] using their publicly available code. Results for SSRE [13] and FeTrIL [6] are obtained by running the PyCIL [11] framework. We use the same class subset, class order and seed for all our baseline experiments. It should be noted that in the main paper, we define average incremental accuracy as the average accuracy for all seen classes.

2.4. Generative (GAN) Continual Learning

We choose the StackGAN-v2 [10] architecture for the incremental GAN experiment. StackGAN-v2 contains four blocks in the generator and discriminator networks. In the generator network, there are $1024, 512, 256, 128$ filters from first to the fourth block and the final image construction layer contains 64 filters. We extend the last layer by 4 filters; hence the respective increase in filters are $64, 32, 16, 8$ from first to the fourth block. During training of the i^{th} task, all the previous task parameters are frozen; the parameter grows over the previous task parameters and not just over the global parameter. In our approach, we only grow the generator parameters and the discriminator is fixed for all the tasks; without any constraint, the discriminator parameter learns the current task. For the above discussed filter growth, the generator achieves a growth rate of 11.5%. We also observe that further filter growth shows better results. Our selected task sequences (cats, birds and churches) are highly diverse. The cat images are generally indoor or outdoor animal images; however, the next task (birds) are in a highly complex background and with fine-grained information; so the adaptation of birds from cats is difficult. Our model shows significant gains on the birds dataset using only 11.5% extra parameters. The adaptation of churches from the birds dataset (birds to buildings) is also very difficult. Our proposed model adapts to this dataset and shows state-of-the-art results compared to the recent strong baselines.

2.5. Heterogeneous Task Sequence

We borrow the baselines and hyperparameter settings (including seed) from the EFT [8] paper. To evaluate our method, we use the VGG-16 [7] architecture with batch norm.

SVHN→CIFAR10→CIFAR100: If $\alpha_{i,j}$ is the number of filters for layer j in VGG-16 before task $i+1$, then we increase the filters as follows:

$$\begin{aligned} \alpha_{2,j} &= \alpha_{1,j} + 10 \\ \alpha_{3,j} &= \alpha_{2,j} + 10 \ if \ \alpha_{1,j} &= 64 \ or \ 128 \\ \alpha_{3,j} &= \alpha_{2,j} + 20 \ if \ \alpha_{1,j} &= 256 \ or \ 512 \end{aligned}$$

CIFAR100→CIFAR10→SVHN: If $\alpha_{i,j}$ is the number of filters for layer j in VGG-16 before task $i+1$, then we increase the filters as follows:

$$\begin{aligned} \alpha_{2,j} &= \alpha_{1,j} + 10 \ if \ \alpha_{1,j} &= 64 \ or \ 128 \\ \alpha_{2,j} &= \alpha_{1,j} + 20 \ if \ \alpha_{1,j} &= 256 \ or \ 512 \\ \alpha_{3,j} &= \alpha_{2,j} + 10 \end{aligned}$$

2.6. Softwares

Experiments are run on a single V100 gpu using Linux, Python 3.6 and PyTorch 1.7.1 softwares.

2.7. Input Processing

The data transformation scheme used in our method is borrowed from EFT [8] for CIFAR-100 and Tiny ImageNet datasets, while for ImageNet-100, we use the data transformation scheme used in [1]. The codes for both these methods are publicly available.

References

[1] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *ECCV*, 2020. 5

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Computer Vision and Pattern Recognition*, 2016. 1, 4

[3] Gyuhak Kim, Changnan Xiao, Tatsuya Konishi, Zixuan Ke, and Bing Liu. A theoretical study on solving continual learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 4

[4] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 1

[5] Zhizhong Li and Derek Hoiem. Learning without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 5

[6] G. Petit, A. Popescu, H. Schindler, D. Picard, and B. Delezoide. Fetril: Feature translation for exemplar-free class-incremental learning. *WACV*, 2023. 4, 5

[7] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4, 5

[8] Vinay Kumar Verma, Kevin J Liang, Nikhil Mehta, Piyush Rai, and Lawrence Carin. Efficient feature transformations for discriminative and generative continual learning. In *CVPR*, pages 13865–13875, 2021. 1, 4, 5

[9] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *CVPR*, pages 3014–3023, 2021. 4

[10] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. *Transactions on Pattern Analysis and Machine Intelligence*, 2018. 5

[11] Da-Wei Zhou, Fu-Yun Wang, Han-Jia Ye, and De-Chuan Zhan. Pycil: A python toolbox for class-incremental learning, 2021. 4, 5

[12] Fei Zhu, Zhen Cheng, Xu-Yao Zhang, and Cheng-lin Liu. Class-incremental learning via dual augmentation. *NeurIPS*, 34:14306–14318, 2021. 1, 4, 5

[13] Kai Zhu, Wei Zhai, Yang Cao, Jiebo Luo, and Zheng-Jun Zha. Self-sustaining representation expansion for non-exemplar class-incremental learning. In *CVPR*, pages 9286–9295, 2022. 4, 5