Forget-free Continual Learning with Winning Subnetworks

Haeyong Kang [*1] **Rusty John Lloyd Mina** [*1] **Sultan Rizky Hikmawan Madjid** [1] **Jaehong Yoon** [1]
Mark Hasegawa-Johnson [2] **Sung Ju Hwang** [1 3] **Chang D. Yoo** [1]

Abstract

Inspired by *Lottery Ticket Hypothesis* that competitive subnetworks exist within a dense network, we propose a continual learning method referred to as *Winning SubNetworks (WSN)* which sequentially learns and selects an optimal subnetwork for each task. Specifically, WSN jointly learns the model weights and task-adaptive binary masks pertaining to subnetworks associated with each task whilst attempting to select a small set of weights to be activated (winning ticket) by reusing weights of the prior subnetworks. The proposed method is inherently immune to catastrophic forgetting as each selected subnetwork model does not infringe upon other subnetworks. Binary masks spawned per winning ticket are encoded into one N-bit binary digit mask, then compressed using Huffman coding for a sub-linear increase in network capacity with respect to the number of tasks. Code is available at `https://github.com/ihaeyong/WSN`.

1. Introduction

Continual Learning (CL), also known as Lifelong Learning (Thrun, 1995), is a paradigm for learning a series of tasks in a sequential manner. One of the major goals in continual learning is to mimic human cognition, exemplified by the ability to incremental learning new concepts over his/her lifespan. An ideal continual learner encourages positive forward/backward transfer, utilizing the learned knowledge from previous tasks when solving for new ones, and updating the previous task knowledge with the new task knowledge. Nevertheless, this is nontrivial due to the phenomenon referred to as *catastrophic forgetting* or *catastrophic interference* (McCloskey & Cohen, 1989), where the model performance on previous tasks significantly decreases upon learning on new tasks. Various approaches have been proposed to solve catastrophic forgetting during continual learning, which can be broadly categorized as follows: (1) **Regularization-based methods** (Kirkpatrick et al., 2017; Chaudhry et al., 2020; Jung et al., 2020; Titsias et al., 2020; Mirzadeh et al., 2021) aim to keep the learned information of past tasks during continual training aided by sophisticatedly designed regularization terms, (2) **Rehearsal-based methods** (Rebuffi et al., 2017; Chaudhry et al., 2019a;b; Saha et al., 2021) utilize a set of real or synthesized data from the previous tasks and revisit them, and (3) **Architecture-based methods** (Mallya et al., 2018; Serrà et al., 2018; Li et al., 2019; Wortsman et al., 2020) propose to minimize the inter-task interference via newly designed architectural components.

Despite the remarkable success of recent works on continual learning, existing methods suffer from a limitation that the overall memory usage increases as new tasks arrive. Rehearsal-based CL requires additional room to store the replay buffer or generative models, and architecture-based methods leverage additional model capacity to account for new tasks. The methods lead to an essential question: how can we build a memory-efficient CL model that does not exceed the capacity of the backbone network or even requires a much smaller capacity? Several studies have shown that deep neural networks are over-parameterized (Denil et al., 2013; Han et al., 2016; Li et al., 2016) and thus removing redundant/unnecessary weights can achieve on-par or even better performance than the original dense network. More recently, Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2019) demonstrates the existence of sparse subnetworks, named *winning tickets*, that preserve the performance of a dense network. However, searching for optimal winning tickets during continual learning with iterative pruning methods requires repetitive pruning and retraining for each arriving task, which is impractical.

To tackle the problem, we suggest a novel CL method that finds the high-performing *Winning SubNetwork* (WSN) given tasks without the need for retraining and rewinding, as shown in Figure 1 (d). Unlike previous pruning-based CL approaches (Mallya et al., 2018; Wortsman et al., 2020) (see Figure 1 (a)), which obtain task-specific subnetworks given

[*]Equal contribution [1]Korea Advanced Institute of Science and Technology (KAIST), South Korea [2]University of Illinois at Urbana-Champaign, USA [3]AITRICS, South Korea. Correspondence to: Chang D. Yoo <cd_yoo@kaist.ac.kr>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

(a) Fixed Backbone Network

(b) Biased Transfer

(c) Selective Reuse Expansion beyond Dense Network

(d) Selective Reuse Expansion within Network (Our WSN)

Figure 1. Concept Comparison: (a) Piggyback (Mallya et al., 2018), and SupSup (Wortsman et al., 2020) nd the optimal binary mask on a xed backbone network a given task (b) PackNet (Mallya & Lazebnik, 2018) and CLNP (Golkar et al., 2019) forces the model to reuse all features and weights from previous subnetworks which causes bias in the transfer of knowledge (c) APD (Yoon et al., 2020) selectively reuse and dynamically expand the dense network (d) Our WSN selectively reuse and dynamically expand subnetworks within a dense network. Green edges are reused weights

a pre-trained backbone network, we incrementally learn model weights and task-adaptive binary masks (the subnetworks) within the neural network. To allow the forward transfer when a model learns on a new task, we reuse the learned subnetwork weights for the previous tasks, however selectively, as opposed to using all the weights (Mallya & Lazebnik, 2018) (see Figure 1 (b)), that may lead to biased transfer. Further, our method eliminates the threat of catastrophic forgetting during continual learning by freezing the subnetwork weights for the previous tasks, and does not suffer from the negative transfer, unlike Yoon et al. (2018) (see Figure 1 (c)), whose subnetwork weights for the previous tasks can be updated when training on the new tasks.

The magnitudes of the weights are often used as a pruning criterion for nding the optimal subnetwork used in LTH. However, in CL, relying only on the weight magnitude may be suboptimal since the weights are shared across classes, and thus training on the new tasks will change the weights trained for previous tasks (reused weights). This will trigger an avalanche effect where weights selected to be part of the subnetworks for later tasks will always be better in the eyes of the learner, which will result in the catastrophic forgetting of the knowledge for the prior tasks. Thus, in CL, it is important for the learner to train on the new tasks without changing the reused weights. To nd the optimal subnetworks, we decouple the information of the learning parameter and the network structure into two separate learnable parameters, namely, *weights* and *weight scores*. The weight scores are binary masks that have the same shapes as the weights. Now, subnetworks are found by selecting the weights with the top-k percent weight ranking scores. More importantly, decoupling the weights and the subnetwork structure allows us to nd the optimal subnetwork online without iterative retraining, pruning, and rewinding. To this end, the proposed method is designed to jointly learn the weights and the structure of the optimal subnetworks, whose overall size is smaller than a dense network.

Our contributions can be summarized as follows:

- We propose a novel forgetting-free continual learning method inspired by Lottery Ticket Hypothesis (LTH), which learns a compact subnetwork for each task while keeping the weights selected by the previous tasks intact. The proposed method does not perform any explicit pruning for learning the subnetwork. This not only eliminates catastrophic forgetting but also enables forward transfer from the previous tasks to new ones.
- Our method obtains compact subnetworks using Huffman coding with a sub-linear increase in the network capacity, outperforming existing continual learning methods in terms of accuracy-capacity trade-off and backward transfer.

2. Related Works

Continual Learning. Continual learning (McCloskey & Cohen, 1989; Thrun, 1995; Kumar & Daume III, 2012; Li & Hoiem, 2016), or lifelong learning, is the problem of continuous learning on a sequence of tasks while utilizing and preserving previously learned knowledge of the tasks. There exist several major directions to tackle the challenges for continual learning, such as catastrophic forgetting: *Regularization-based approaches* (Kirkpatrick et al., 2017; Chaudhry et al., 2020; Jung et al., 2020; Titsias et al., 2020; Mirzadeh et al., 2021) lessen the catastrophic forgetting by imposing regularization constraints that inhibit the change of weights or nodes for past tasks. *Rehearsal-based approaches* (Rebuf et al., 2017; Chaudhry et al., 2019a;b; Saha et al., 2021; Deng et al., 2021) store small data summaries to the tasks and replay them to retain the past tasks knowledge. Some methods in this line of works (Shin et al., 2017; Aljundi et al., 2019) accommodate the generative model to construct the pseudo-rehearsals. Architecture-based approaches (Mallya et al., 2018; Serrà et al., 2018; Li et al., 2019; Wortsman et al., 2020) utilize additional capacity to expand (Xu & Zhu, 2018; Yoon et al., 2018) or isolate (Rusu et al., 2016) model parameters thereby preserving the learned knowledge without the forgetting. Both rehearsal and architecture-based methods have shown remarkable ef cacy in suppressing catastrophic forgetting but require additional capacity for the task-adaptive parameters or the replay buffers.

Pruning-based Continual Learning. When most works have dived to increase the performance of continual learners by adopting additional memory, another line of works has pursued to build memory and computational ef cient continual learners utilizing pruning-based constraints. CLNP (Golkar et al., 2019) selects the important neurons for a given task using ℓ_1 regularization to induce sparsity and freezes them to maintain the performance. After that, the model reinitializes the neurons that were not selected for future task training. Piggyback (Mallya et al., 2018) trains task-speci c binary masks on the weights given the pre-trained model. The method does not allow knowledge transfer among tasks while memorizing the learned masks for each task. Then, the performance highly depends on the quality of the backbone model. HAT (Serrà et al., 2018) proposes task-speci c learnable attention vectors to identify important weights per task, where the masks are formulated to layerwise cumulative attention vectors during continual learning. Very recently, LL-Tickets (Chen et al., 2021) shows the existence of a sparse subnetwork, called lifelong tickets, that performs well on all tasks during continual learning. When obtained tickets cannot suf ciently learn the new task while maintaining performance on past tasks, the method searches for more prominent tickets from current ones. However, LL-Tickets require external data to maximize knowledge distillation with learned models for prior tasks and the ticket expansion process is composed of another series of retraining and pruning steps. On the other hand, our WSN jointly learns the model and task-adaptive binary masks pertaining to subnetworks associated with each task whilst attempting to select a small set of weights to be activated (winning ticket) by reusing weights of the prior subnetworks.

3. Forget-Free Continual Learning with Winning SubNetworks

In this section, we propose our pruning-based continual learning method, *Winning SubNetworks* (WSN). In WSN, the neural network searches for the task-adaptive winning tickets and updates only the weights that have not been trained on the previous tasks. After training for each task, the model freezes the subnetwork parameters so that the proposed method is immune to the catastrophic forgetting by design. We emphasize that our WSN can selectively transfer the previously learned knowledge to the future task (forward transfer), which also substantially reduces the training time it takes to converge during sequential learning. The strength becomes more critical to the large-scaled continual learning problems where a continual learner trains on several tasks in a sequence.

Problem Statement. Consider a supervised learning setup where T tasks arrive to a learner in a sequential order. We denote that $\mathcal{D}_t = \{x_{i,t}, y_{i,t}\}_{i=1}^{n_t}$ is the dataset of task t, composed of n_t pairs of raw instances and corresponding labels. We assume a neural network $f(\cdot;\ )$, parameterized by the model weights and standard continual learning scenario aims to learn a sequence of tasks by solving the following optimization procedure at each step t:

$$= \text{minimize} \frac{1}{n_t}\sum_{i=1}^{n_t} \mathcal{L}(f(x_{i,t};\ ); y_{i,t}), \qquad (1)$$

where $\mathcal{L}(\cdot;\cdot)$ is a classi cation objective loss such as cross-entropy loss. $\mathcal{D}_t$ for task t is only accessible when learning task t, yet rehearsal-based continual learning methods allow memorizing a small portion of the dataset to replay. We further assume that task identity is given in both the training and the testing stage.

To allow room for learning future tasks, a continual learner often adopts over-parameterized deep neural networks. As a continual learner often adopts over-parameterized deep neural networks to allow resource freedom for future tasks, we can nd the subnetworks that obtain on par or even better performance. Given the neural network parameters , the binary attention mask m_t that describes the optimal subnetwork for task t such that $|m_t|$ is less than the model capacity c follows as:

$$m_t = \underset{m_t \in \{0,1\}^{|\ |}}{\text{minimize}} \frac{1}{n_t}\sum_{i=1}^{n_t} \mathcal{L}\left(f(x_{i,t};\ \ m_t); y_{i,t}\right) \quad C$$
$$\text{subject to } |m_t| \quad c, \qquad (2)$$

where task loss $C = \mathcal{L}\left(f(x_{i,t};\ ); y_{i,t}\right)$ and $c \quad |\ |$. In the optimization section, we describe how to obtain m_t using a single learnable weight score s that is subject to updates while minimizing task loss jointly for each task.

3.1. Winning SubNetworks

Let each weight be associated with a learnable parameter we call *weight score* s, which numerically determines the

(a) selected weights $\hat{}_{t-1}$ at prior task

(b) forward pass using reused weights

(c) backward pass only on non-used weights

(d) selected weights $\hat{}_t$ with subsets of reused weights

Figure 2. An illustration of Winning SubNetworks (WSN): (a) The top-c% weights $\hat{}_{t-1}$ at prior task are obtained, (b) In the forward pass of a new task, WSN reuses weights selected from prior tasks, (c) In the backward pass, WSN updates only non-used weights, and (d) after several iterations of (b) and (c), we acquire again the top-c% weights $\hat{}_t$ including subsets of reused weights. for the new task.

importance of weight associated with it; that is, a weight with a higher weight score is seen as more important. We find a sparse subnetwork $\hat{}_t$ of the neural network and assign it as a solver of the current task t. We use subnetworks instead of the whole original network as solvers for two reasons: (1) Lottery Ticket Hypothesis (Frankle & Carbin, 2019) shows the existence of a subnetwork that performs as well as the whole network, and (2) subnetwork requires less capacity than dense networks, and therefore it inherently reduces the size of the expansion of the solver.

Motivated by such benefits, we propose a novel Winning SubNetworks (WSN) which is the joint training method for continual learning that trains on task - while selecting an important subnetwork given the task t as shown in Fig. 2. The illustration of WSN explains how to acquire binary weights within a full network step by step. We find $\hat{}_t$ by selecting the c% weights with the highest weight scores s, where c is the target layerwise capacity ratio in %. The selection of weights is represented by a task-dependent binary weight mask m_t where a value of 1 denotes that the weight is selected during the forward pass and 0 otherwise. Formally, m_t is obtained by applying a indicator function 1_c on s where $1_c(s) = 1$ if s belongs to top-c% scores and 0 otherwise. Therefore, the subnetwork $\hat{}_t$ for task t is obtained by $\hat{}_t = \quad m_t$.

3.2. Optimization Procedure for Winning SubNetworks

To jointly learn the model weights and task-adaptive binary masks of subnetworks associated with each task, given an objective $L(\;)$, we optimize and s with:

$$\underset{;s}{\text{minimize}}\; L(\quad m_t; D_t): \qquad (3)$$

However, this vanilla optimization procedure presents two problems: (1) updating all when training for new tasks will cause interference to the weights allocated for previous tasks, and (2) the indicator function always has a gradient value of 0; therefore, updating the weight scores s with its loss gradient is not possible. To solve the first problem, we update the weights selectively by allowing updates only on the weights that have not been selected in the previous tasks. To do that, we use an accumulate binary mask $M_{t-1} = \_{i=1}^{t-1} m_i$ when learning task t, then for an optimizer with learning rate ; the is updated as follows:

$$\frac{@L}{@}\;(1 \quad M_{t-1}) \quad ; \qquad (4)$$

effectively freezing the weights of the subnetworks selected for the previous tasks. To solve the second problem, we use Straight-through Estimator (Hinton, 2012; Bengio et al., 2013; Ramanujan et al., 2020) in the backward pass since m_t is obtained by top-c% scores. Specifically, we ignore the derivatives of the indicator function and update the weight score as follows:

$$s \quad s \qquad \frac{@L}{@s} \;: \qquad (5)$$

The use of separate weight scores s as the basis for selecting subnetwork weights makes it possible to reuse some of the weights from previously chosen weights m_t in solving the current task t, which can be viewed as transfer learning. Likewise, previously chosen weights that are irrelevant to the new tasks are not selected, instead, weights from the set of not-yet-chosen weights are selected to meet the target network capacity for each task, which can be viewed as finetuning from tasks $f1;:::;t \quad 1g$ to task t. Our WSN optimizing procedure is summarized in pseudo algorithm 1.

Algorithm 1 Winning SubNetworks (WSN)

```
input $\{D_t\}_{t=1}^{T}$, model weights , score weights $s$, binary mask
    $M_0 = 0^{| |}$, layer-wise capacity $c$
1: Randomly initialize  and $s$.
2: for task $t = 1, ..., T$ do
3:     for batch $b_t$  $D_t$ do
4:         Obtain mask $m_t$ of the top-$c$% scores $s$ at each layer
5:         Compute $L($  $m_t; b_t)$
6:                 $\frac{\partial L}{\partial}$ $(1$  $M_{t}$ $_1)$      ▷ Weight update
7:         $s$  $s$  $(\frac{\partial L}{\partial s})$      ▷ Weight score update
8:     end for
9:     $M_t$  $M_{t}$ $_1$ $\vee$ $m_t$      ▷ Accumulate binary mask
10: end for
```

3.3. Binary Mask Encoding

The subnetworks need a binary mask to store the task-specific weights for each task. One point with the subnetworks is that the binary masks to save increase as the number of tasks is added to the deep-learning models. In order to alleviate the point and achieve forget-free continual learning, we use a compression algorithm for compressing all task binary masks to save. First, to compress compactly, we convert a sequence of binary masks into a single accumulated decimal mask and change each integer into ASCII code to represent a unique single symbol. Then, with the symbols, We test a lossless compression algorithm - Huffman encoding (Huffman, 1952). We empirically observed that Huffman Encoding was able to compress 7-bit binary maps and decompress ones to infer without bit loss approximately with a 78% compression rate and showed the compression rate is sub-linearly increasing with the size of binary bits in the experimental results.

4. Experiments

We now validate our method on several benchmark datasets against relevant continual learning baselines. We consider task-incremental continual learning with a multi-head configuration for all experiments in the paper. We follow the experimental setups in recent works (Saha et al., 2021; Yoon et al., 2020; Deng et al., 2021).

Datasets and architectures. We use six different popular sequential datasets for CL problems with five different neural network architectures as follows: 1) Permuted MNIST (PMNIST): A variant of MNIST (LeCun, 1998) where each task has a deterministic permutation to the input image pixels. 2) 5-Datasets: A mixture of 5 different vision datasets (Saha et al., 2021): CIFAR-10 (Krizhevsky et al., 2009), MNIST (LeCun, 1998), SVHN (Netzer et al., 2011), FashionMNIST (Xiao et al., 2017), and notMNIST (Bulatov, 2011). 3) Omniglot Rotation: An OCR images datasets, composed of 100 tasks as of each includes 12 classes. We further preprocess and the raw images by generating their rotated version in $90°, 180°$, and $270°$, followed by Yoon et al. (2020). 4) CIFAR-100 Split (Krizhevsky et al., 2009): A visual object dataset, constructed by randomly dividing 100 classes of CIFAR-100 into 10 tasks with 10 classes per task. 5) CIFAR-100 Superclass: We follow the setting from Yoon et al. (2020) that divides CIFAR-100 dataset into 20 tasks according to the 20 superclasses, and each superclass contains 5 different but semantically related classes. 6) TinyImageNet (Stanford, 2021): A variant of ImageNet (Krizhevsky et al., 2012) containing 40 of 5-way classification tasks with the image sized by $64 \times 64 \times 3$.

We use two-layered MLP with 100 neurons per layer for PMNIST, variants of LeNet (LeCun, 1998) for the experiments on Omniglot Rotation and CIFAR-100 Superclass experiments. Also, we use a modified version of AlexNet similar to Serrà et al. (2018); Saha et al. (2021) for CIFAR-100 Split dataset and a reduced ResNet-18 similar to Chaudhry et al. (2019b); Saha et al. (2021) for 5-Datasets. For TinyImageNet, we also use the same network architecture as Gupta et al. (2020); Deng et al. (2021), which consists of 4 Conv layers and 3 fully connected layers.

Baselines. We compare our WSN with strong CL baselines; regularization-based methods: HAT (Serrà et al., 2018) and EWC (Kirkpatrick et al., 2017), rehearsal-based methods: GPM (Saha et al., 2021), and a pruning-based method: PackNet (Mallya & Lazebnik, 2018) and SupSup (Wortsman et al., 2020). PackNet and SupSup are set to baseline to show the effectiveness of re-used weights. We also compare with naive sequential training strategy, referred to as FINETUNE. Multitask Learning (MTL) and Single-task Learning (STL) are not a CL method. MTL trains on multiple tasks simultaneously, and STL trains single task independently.

We summarize the architecture-based baselines as follows:

1. PackNet (Mallya & Lazebnik, 2018): iterative pruning and network re-training architecture for packing multiple tasks into a single network.
2. SupSup (Wortsman et al., 2020): finding supermasks (subnetworks) within a randomly initialized network for each task in continual learning.
3. WSN (ours): jointly training model and finding task-adaptive subnetworks of novel/prior parameters for continual learning.

Experimental settings. As we directly implement our method from the official code of Saha et al. (2021), we provide the values for HAT and GPM reported in Saha et al. (2021). For Omniglot Rotation and Split CIFAR-100 Superclass, we deploy the proposed architecture as reported in Yoon et al. (2020) in multi-head settings with hyperparameters. All our experiments run on a single-GPU setup of NVIDIA V100. We provide more details of the datasets,

Table 1. Performance comparison of the proposed method and baselines on various benchmark datasets. We report the mean and standard deviation of the average accuracy (ACC), average capacity (CAP), and average backward transfer (BWT) across ve independent runs. The best results are highlighted in bold. Values with y and denote reported performances from (Saha et al., 2021) and (Yoon et al., 2020). We consider PackNet (Mallya & Lazebnik, 2018) and SupSup (Wortsman et al., 2020) as the baselines.

Method	Permuted MNIST			5 Datasets			Omniglot Rotation		
	ACC (%)	CAP (%)	BWT	ACC (%)	CAP (%)	BWT	ACC (%)	CAP (%)	BWT
STL	97.37 (0.01)	1,000.0	-	93.44 (0.12)	500.0	-	82.13 (0.08)	10,000.0	-
FINETUNE	78.22 (0.84)	100.0	-0.21 (0.01)	80.06 (0.74)	100.0	-0.17 (0.01)	44.48 (1.68)	100.0	-0.45 (0.02)
EWC (Kirkpatrick et al., 2017)	92.01 (0.56)	100.0	-0.03 (0.00)	88.64 (0.26)^{y}	100.0^{y}	-0.04 (0.01)^{y}	68.66 (1.92)	100.0	-
HAT (Serrà et al., 2018)	-	-	-	91.32 (0.18)^{y}	100.0^{y}	-0.03 (0.00)^{y}	-	-	-
GPM (Saha et al., 2021)	94.96 (0.07)	100.0	-0.02 (0.01)	91.22 (0.20)^{y}	100.0	-0.01 (0.00)^{y}	85.24 (0.37)	100.0	-0.01 (0.00)
PackNet (Mallya & Lazebnik, 2018)	96.37 (0.04)	96.38	0.0	92.81 (0.12)	82.86	0.0	30.70 (1.50)	399.2	0.0
SupSup (Wortsman et al., 2020)	96.31 (0.09)	122.89 (0.07)	0.0	93.28 (0.21)	104.27 (0.21)	0.0	58.14 (2.42)	407.12 (0.17)	0.0
WSN, $c = 0.03$	94.84 (0.11)	19.87 (0.16)	0.0	90.57 (0.65)	12.11 (0.06)	0.0	80.68 (2.60)	75.87 (1.24)	0.0
WSN, $c = 0.05$	95.65 (0.03)	26.49 (0.16)	0.0	91.61 (0.21)	17.26 (0.25)	0.0	87.28 (0.72)	79.85 (1.19)	0.0
WSN, $c = 0.1$	96.14 (0.03)	40.41 (0.54)	0.0	92.67 (0.12)	28.01 (0.28)	0.0	83.10 (1.56)	83.08 (1.61)	0.0
WSN, $c = 0.3$	96.41 (0.07)	77.73 (0.36)	0.0	93.22 (0.32)	62.30 (0.69)	0.0	81.89 (1.15)	102.2 (0.89)	0.0
WSN, $c = 0.5$	96.24 (0.11)	98.10 (0.25)	0.0	93.41 (0.13)	86.10 (0.57)	0.0	79.80 (2.16)	121.2 (0.50)	0.0
MTL	96.70 (0.02)^{y}	100.0	-	91.54 (0.28)^{y}	100.0	-	81.23 (0.52)	100.0	-

architectures, and experimental settings, including the hyperparameter con gurations for all methods in Appendix A.

(a) 5-Dataset - ResNet-18 (b) Omniglot Rotation - LeNet

Figure 3. Accuracy over Total Capacity Usage The performances of WSN gets better than others as total model capacities increase, and then, saturated approximately at 80 %.

Performance metrics. We evaluate all methods based on the following metrics:

1. Accuracy (ACC) measures the average of the nal classi cation accuracy on all tasks: $ACC = \frac{1}{T}\sum_{i=1}^{T} A_{T,i}$, where $A_{T,i}$ is the test accuracy for task i after training on task T.

2. Capacity (CAP) measures the total of percentage of non-zero weights plus the prime masks for all tasks as follows: $\text{Capacity} = (1 - S) + \frac{(1 - \;)T}{32}$, where we assume all task weights of 32-bit precision, S is the sparsity of M_T and the average compression rate 0.78 that we acquired through 7bit Huffman encoding, which depends on the size of bit-binary maps; the compression rate also depends on compression methods typically.

3. Backward Transfer (BWT) measures the forgetting during continual learning. Negative BWT means that learning new tasks causes the forgetting on past tasks: $BWT = \frac{1}{T-1}\sum_{i=1}^{T-1} A_{T,i} - A_{i,i}$.

5. Results and Discussion

5.1. Comparisons with the Baseline

We evaluate our algorithm on three-standard benchmark datasets: Permuted MNIST, 5-Datasets, and Omniglot Rotation. We set PackNet and SupSup to baselines as non-reused weight methods and compared WSN with the baselines, including other algorithms as shown in Table 1. Our WSN outperformed all the baselines in three measurements, achieving the best average accuracy of 96.41%, 93.41%, and 87.28% while using the least capacity compared to the other existing methods, respectively. Moreover, our WSN is proved to be a forget-free model like PackNet. Compared with PackNet, however, our WSN showed the effectiveness of reused weights for continual learning and its scalability in the Omniglot Rotation experiments with the least capacities by a large margin. Figure 3 provides the accuracy over total capacity usage. Our WSN's accuracy is higher than PackNet's, approximately at 80% total capacity usage on 5-Dataset, and WSN outperformed others at 80% total capacity usage on Omniglot Rotation. The lower performances of PackNet might attribute to Omniglot Rotation dataset statistics since, regardless of random rotations, tasks could share common visual features such as circles, curves, and straight lines. Therefore, non-reused methods might be dif cult to train a new task model unless prior weights were transferred to the current model, a.k.a. forward transfer learning.

5.2. Comparisons with the SOTA

We use a multi-head setting to evaluate our WSN algorithm under the more challenging visual classi cation benchmarks. The WSN's performances are compared with others w.r.t three measurements on three major benchmark datasets as shown in Table 2. Our WSN outperformed all state-of-the-arts, achieving the best average accuracy of 76.38%, 61.79%, and 71.96%. In these experiments, WSN is also a forget-free

Table 2.Performance comparisons of the proposed method and other state-of-the-art including baselines - PackNet (Mallya & Lazebnik, 2018) and SupSup (Wortsman et al., 2020) - on various benchmark datasets. We report the mean and standard deviation of the average accuracy (ACC), average capacity (CAP), and average backward transfer (BWT) across 5 independent runs with 5 seeds under the same experimental setup (Deng et al., 2021). The best results are highlighted in bold. Also, y denotes results reported from Deng et al. (2021).

Method	CIFAR-100 Split			CIFAR-100 Superclass			TinyImageNet		
	ACC (%)	CAP (%)	BWT (%)	ACC (%)	CAP (%)	BWT (%)	ACC (%)	CAP (%)	BWT (%)
La-MaML (Gupta et al., 2020)	71.37 (0.67)y	100.0	-5.39 (0.53)y	54.44 (1.36)y	100.0	-6.65 (0.85)y	66.90 (1.65)y	100.0	-9.13 (0.90)y
GPM (Saha et al., 2021)	73.18 (0.52)y	100.0	-1.17 (0.27)y	57.33 (0.37)y	100.0	-0.37 (0.12)y	67.39 (0.47)y	100.0	1.45 (0.22)y
FS-DGPM (Deng et al., 2021)	74.33 (0.31)y	100.0	-2.71 (0.17)y	58.81 (0.34)y	100.0	-2.97 (0.35)y	70.41 (1.30)y	100.0	-2.11 (0.84)y
PackNet (Mallya & Lazebnik, 2018)	72.39 (0.37)	96.38 (0.00)	0.0	58.78 (0.52)	126.65 (0.00)	0.0	55.46 (1.22)	188.67 (0.00)	0.0
SupSup (Wortsman et al., 2020)	75.47 (0.30)	129.00 (0.03)	0.0	61.70 (0.31)	162.49 (0.00)	0.0	59.60 (1.05)	214.52 (0.89)	0.0
WSN, c = 0:03	70.65 (0.36)	18.56 (0.25)	0.0	54.99 (0.71)	22.30 (0.22)	0.0	68.72 (1.63)	37.19 (0.21)	0.0
WSN, c = 0:05	72.44 (0.27)	25.09 (0.42)	0.0	57.99 (1.34)	27.37 (0.33)	0.0	71.22 (0.94)	41.98 (0.52)	0.0
WSN, c = 0:1	74.55 (0.47)	39.87 (0.62)	0.0	60.45 (0.37)	38.55 (0.20)	0.0	71.96 (1.41)	48.65 (3.03)	0.0
WSN, c = 0:3	75.98 (0.68)	80.26 (1.53)	0.0	61.47 (0.30)	63.47 (1.33)	0.0	70.92 (1.37)	73.44 (2.35)	0.0
WSN, c = 0:5	76.38 (0.34)	99.13 (0.48)	0.0	61.79 (0.23)	80.93 (1.58)	0.0	69.06 (0.82)	92.03 (1.80)	0.0
Multitask	79.75 (0.38)y	100.0	-	61.00 (0.20)y	100.0	-	77.10 (1.06)y	100.0	-

(a) Per Task Accuracy over (b) Model Capacity

Figure 4.Performances and Compressed Capacities Sequence of TinyImageNet Dataset Experiments: (a) Setting c = 0:1 shows generalized performances over others and (b) Within 40 tasks, the 7-bits compressed model increase its capacity the least.

model (BWT = ZERO) with the least model capacity. Note that we assume the model capacities are compared based on the model size without extra memory, such as samples. We highlight that our method achieves the highest accuracy, the lowest capacity, and backward transfer on all datasets. Figure 4 shows the process of performance and compressed capacity changing with the number of tasks on the TinyImageNet datasets, where the "Average Progressive Capacity" metric is de ned as the average capacity (the proportion of the number of network weights used for any one of the tasks) after 5 run of the experiment with different seed values. Furthermore, we consistently showed the progressively improved performances of WSN than others on CIFAR-100 Split datasets as shown in Figure 6.

5.3. Forget-Free Performance and Model Capacity

We prepare results on performance and bit-map compressed capacity on the TinyImageNet dataset as shown in Figure 4 - "c=0.1" and "c=0.1+7bit-Huffman" refers respectively to using 10% of network weights and no compression on the binary mask and the latter refers to the same with 7bit-Huffman encoding. In Figure 4 (a), using initial capacity, c = 0:1 shows better performances over others. With xed c = 0:1, the bit-wise Huffman compression rate delivers positive as the number of tasks increases, as shown in Figure 4 (b). The most interesting part is that the average compression rate gets higher as the number of bits to compress increases. (see Appendix B.2), and the increasing ratio of reused weights (symbols with a high probability in the Huffman encoding) might affect the compression rate (symbols with a small probability might be rare in the Huffman tree, where the infrequent symbols tend to have long bit codes). We investigated how the compression rate is related to the total model capacity. The more bits the binary mask compression does, the less the model capacity to save is required. This shows that within 40 tasks, N-bit Huffman compressed capacities are sub-linearly increasing as binary map capacities increase linearly. The 7-bit Huffman encoding is enough to compress binary maps without exceeding the model capacity even though the compression rate depends on compression methods typically.

5.4. Catastrophic Forgetting From WSN's Viewpoint

We interpreted how reused weights affect the inference performances on the TinyImageNet dataset as shown in Figure 5. For more precise interpretability, we divide all used weights for solving sequential tasks into speci c sets. All used weights (a) within a trained dense network are separated as follows: all used represents all activated sets of weights up to task t 1. per task represents an activated set of weights at task t. new per task represents a new activated set of weights at task t. reused per task represents an intersection set of weights per task and all used weights. reused for all tasks represents an intersection set of weights reused from task 1 up to task t 1.

First, our WSN adaptively reuses weights to solve the sequential tasks. In Figure 5 (b), both initial task capacity and progressive task capacity start from c value; the proportion of reused weights per task decreases in the early task learning stage. However, it tends to be progressively saturated to c = 0:1 value since the number of the new activated set of weights decreases, and the proportion of reused weights for all prior tasks tends to be decreasing. In other words, WSN

(a) A Diagram of Capacities (b) Capacities over c (c) Accuracy of Reused Weights (d) Accuracy w/o Reused Weights

Figure 5. Layer-wise Analysis on TinyImageNet Dataset Experiments: (a) Weights reusability within a dense network, (b) Capacities except to binary maps are determined by $c = 0.1$, (c) The most significant forgetting occurs from weights without reused per task; the significance of weights reused for all tasks gets lower, and from task 7, all used weights seem to be enough to infer all tasks, and (d) In the inference step, we inspected a layer-wise forgetting caused by removing (setting masking value 1 to 0) reused weights per task on the trained model and observed performance drops significantly at Conv1 layer.

Figure 6. Comparisons on CIFAR100-Split with (Deng et al., 2021): It was difficult to determine the superiority of WSN only by comparing the performances in the early stage, however, it shows the progressively better performances of WSN than others.

uses diverse weights to solve the sequential tasks within all used weights rather than depending on the reused weights for all prior tasks as the number of tasks increases.

Second, our WSN provides a stepping stone for forget-free continual learning. Regarding the benefits of WSN, in Figure 5 (c), we interpret the importance of three types of weights through an ablation study. The additional evaluations were performed by the acquired task binary masks and trained models to investigate the importance of reused weights in each layer, where the "w/" refers to "with reused network weights" and the "w/o" refers to "without reused network weights." Model forgetting occurred from the performances without using weights reused per task severely. The most significant weights were weights reused per task, the subset of all used weights; the importance of the weights reused for all prior tasks decreases as the number of tasks increases since its capasity gets small relatively, as shown in Figure 5 (b). Moreover, in Figure 5 (d), we inspected layer-wise forgetting caused by removing weights reused per task of network layers; the performance sensitivities were quite diverse. In particular, we observed that the most performance drops at the Conv1 layer.

Finally, our WSN reuses weights if the knowledge from previous tasks is already enough to solve the task at hand and employs a few new weights otherwise. Specifically, from task 7, all used weights seem to be enough to infer all tasks since the weights reused per task catch up with the task performances. For more generalized forget-free continual learning, the model should consider the layer-wise sensitivity of weights reused per task when selecting weights reused for all prior tasks. These analyses might broadly impact other machine learning fields, such as transfer learning, semi-supervised learning, and domain adaptation.

Table 3. Performance comparisons of Re-initialized WSN and Initialized ones with $c = 0.5$ on the three benchmark datasets.

Method	CIFAR-100 Split		CIFAR-100 Superclass		TinyImageNet	
WSN	ACC (%)	CAP (%)	ACC (%)	CAP (%)	ACC (%)	CAP (%)
Re-init.	75.68(0.42)	104.47(0.00)	61.72(0.29)	113.74(0.00)	61.13(1.13)	127.49(0.00)
Init.	76.38(0.34)	99.13(0.48)	61.79(0.23)	80.93(1.58)	69.06(0.82)	92.03(1.80)

(a) re-initialized for each task (b) initialized only ones

Figure 7. Comparisons of Capacities on TinyImageNet Dataset: (a) re-initialized weight score per task and (b) initialized weight score only once before training task 1.

5.5. Re-initialized WSN v.s. Initialized WSN

Although the re-initialized weight score explores task-relevant subnetworks, our WSN would lose all task knowledge when the weight score is re-initialized. To validate it, we have performed analysis on WSN with re-initialization of weight scores, as shown in Table 3 and Figure 7. When the weight score is re-initialized for each task, WSN learns all the weights of the network only after a small number of

Table 4.Computational ef ciency of WSN compared with PackNet and SupSup. We report model training time in hours.

Method	Permuted MNIST			5 Dataset			Omniglot Rotation		
	ACC (%)	CAP (%)	Training TIME (HOUR)	ACC (%)	CAP (%)	Training TIME (HOUR)	ACC (%)	CAP (%)	Training TIME (HOUR)
PackNet	96.37 (0.04)	96.38 (0.00)	0.49 (0.03)	92.81 (0.12)	82.86 (0.00)	3.38 (0.11)	30.70 (1.50)	399.2 (0.00)	7.30 (0.01)
SupSup	96.31 (0.09)	122.89 (0.07)	0.48 (0.06)	93.28 (0.21)	104.27 (0.21)	3.20 (0.01)	58.14 (2.42)	407.12 (0.17)	6.92 (0.03)
WSN (best)	96.41 (0.07)	77.73 (0.36)	0.35 (0.02)	93.41 (0.13)	86.10 (0.57)	3.02 (0.03)	87.28 (0.72)	79.85 (1.19)	6.33 (0.04)

Method	CIFAR-100 Split			CIFAR-100 Superclass			TinyImageNet		
	ACC (%)	CAP (%)	Training TIME (HOUR)	ACC (%)	CAP (%)	Training TIME (HOUR)	ACC (%)	CAP (%)	Training TIME (HOUR)
PackNet	72.39 (0.37)	96.38 (0.00)	1.04 (0.19)	58.78 (0.52)	126.65 (0.00)	0.46 (0.01)	55.46 (1.22)	188.67 (0.00)	1.39 (0.03)
SupSup	75.47 (0.30)	129.00 (0.03)	0.79 (0.14)	61.70 (0.31)	162.49 (0.00)	0.37 (0.00)	59.60 (1.05)	214.52 (0.89)	0.92 (0.00)
WSN (best)	76.38 (0.34)	99.13 (0.48)	0.71 (0.09)	61.79 (0.23)	80.93 (1.58)	0.36 (0.00)	71.96 (1.41)	48.65 (3.03)	0.89 (0.00)

tasks, and the newly learned weights drop quickly to 0% while the reused weight shared to all tasks also drops to 0% (See Figure 7 (a)). On the other hand, WSN adaptively selects task-relevant weights. Since WSN does not initialize weight scores for every task, the network weights learned for the prior tasks will be offered to be reused with a "premium" as the weight score is updated for the new task (See Figure 7 (b)). To this end, WSN learns new tasks by appropriately selecting learned reused weights (substantially includes shared network weights) and small new learned weights. The effectiveness of the reused weights is also proven by the computational ef ciency as summarized in Table 4. WSN consistently converged faster than PackNet and SupSup on six different benchmark datasets. In Appendix B.4, the comparison study between SupSup and WSN is revealed by the long sequence of tasks.

(a) PackNet (b) WSN, $c = 0.05$

(c) WSN, $c = 0.1$ (d) WSN, $c = 0.5$

Figure 8.Task-wise Binary Map Correlations on Omniglot Rotation (Top row) and TinyImageNet (Bottom row). WSN reuses the weights while PackNet does not. WSN with $c = 0.5$ reuses task weights more than the case of $c = 0.1$ as shown in higher correlation results than others.

5.6. Sparse Binary Maps

To investigate how WSN reuses weights over sequential tasks, we prepared task-wise binary mask correlations. As shown in Figure 8 (a) and (b), WSN tends to progressively transfer weights used for prior tasks to weights for new ones compared with PackNet. Figure 8 (c) and (d) showed that the tendency of reused weights differs according to the c. This result might suggest that more sparse reused binary maps lead to generalization than others. In Appendix B.6, more results are stated.

6. Conclusion

We proposed a continual learning method inspired by Lottery Ticket Hypothesis, which continually learns a network by nding an optimal subnetwork for each task, assuming that a subnetwork works well as the dense network. Speci cally, at each arrival of a new task, we use a separate learnable weight score to obtain the subnetwork and store the binary mask for subnetwork selection, which allows obtaining a task-speci c subnetwork for any previous tasks. Then, we freeze the subnetwork weights used by the previous tasks' subnetworks using their binary masks and train the subnetwork on the new task. This process eliminated catastrophic forgetting and transferred knowledge from previous tasks to new ones. We experimentally validated our method on standard benchmark datasets, on which it obtains a compact subnetwork for all tasks, vastly outperforming existing continual learning methods while obtaining a much smaller network due to logarithmic growth in network capacity - binary masks spawned per winning ticket were encoded into one N-bit binary digit mask successfully, then compressed using Huffman coding for a sub-linear increase in network capacity for the number of tasks.

Acknowledgements. This work was partly supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (2021-0-01381) and partly supported by the IITP grant (2022-0-00184).

References

Aljundi, R., Belilovsky, E., Tuytelaars, T., Charlin, L., Caccia, M., Lin, M., and Page-Caccia, L. Online continual learning with maximal interfered retrieval. In Advances in

Neural Information Processing Systems (NeurIPS), 2019.

Bengio, Y., Léonard, N., and Courville, A. C. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, 2013.

Bulatov, Y. notmnist dataset. 2011.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Ef cient lifelong learning with a-gem. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019a.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., and Ranzato, M. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486*, 2019b.

Chaudhry, A., Khan, N., Dokania, P. K., and Torr, P. H. Continual learning in low-rank orthogonal subspaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Chen, T., Zhang, Z., Liu, S., Chang, S., and Wang, Z. Long live the lottery: The existence of winning tickets in lifelong learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Deng, D., Chen, G., Hao, J., Wang, Q., and Heng, P.-A. Flattening sharpness for dynamic gradient projection memory bene ts continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M. A., and de Freitas, N. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Golkar, S., Kagan, M., and Cho, K. Continual learning via neural pruning. *arXiv preprint arXiv:1903.04476*, 2019.

Gupta, G., Yadav, K., and Paull, L. La-maml: Look-ahead meta learning for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for ef cient neural network. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

Hinton, G. Neural networks for machine learning, 2012.

Huffman, D. A. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

Jung, S., Ahn, H., Cha, S., and Moon, T. Continual learning with node-importance based adaptive group sparse regularization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classi cation with deep convolutional neural networks. *Advances in neural information processing systems*, 25: 1097–1105, 2012.

Kumar, A. and Daume III, H. Learning task grouping and overlap in multi-task learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

LeCun, Y. The mnist database of handwritten digits. 1998.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning lters for ef cient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Li, X., Zhou, Y., Wu, T., Socher, R., and Xiong, C. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

Li, Z. and Hoiem, D. Learning without forgetting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Mallya, A., Davis, D., and Lazebnik, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Mirzadeh, S. I., Farajtabar, M., Gorur, D., Pascanu, R., and Ghasemzadeh, H. Linear mode connectivity in multitask and continual learning. In Proceedings of the International Conference on Learning Representations (ICLR), 2021.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

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

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. arXiv preprint arXiv:1606.04671, 2016.

Saha, G., Garg, I., and Roy, K. Gradient projection memory for continual learning. In Proceedings of the International Conference on Learning Representations (ICLR), 2021.

Serrà, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In Proceedings of the International Conference on Machine Learning (ICML), 2018.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In Advances in Neural Information Processing Systems (NeurIPS), 2017.

Stanford. Available online at http://cs231n.stanford.edu/tiny-imagenet-200.zip. CS 231N, 2021.

Thrun, S. A Lifelong Learning Perspective for Mobile Robot Control. Elsevier, 1995.

Titsias, M. K., Schwarz, J., Matthews, A. G. d. G., Pascanu, R., and Teh, Y. W. Functional regularisation for continual learning with gaussian processes. In Proceedings of the International Conference on Learning Representations (ICLR), 2020.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., and Farhadi, A. Supermasks in superposition. In Advances in Neural Information Processing Systems (NeurIPS), 2020.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv, 2017.

Xu, J. and Zhu, Z. Reinforced continual learning. In Advances in Neural Information Processing Systems (NeurIPS), 2018.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. In Proceedings of the International Conference on Learning Representations (ICLR), 2018.

Yoon, J., Kim, S., Yang, E., and Hwang, S. J. Scalable and order-robust continual learning with additive parameter decomposition. In Proceedings of the International Conference on Learning Representations (ICLR), 2020.

A. Experimental Details

We now validate our method on several benchmark datasets against relevant continual learning baselines. We followed similar experimental setups described in (Saha et al., 2021) for baseline comparisons and explained in (Deng et al., 2021) for SOTA comparisons, including the dataset splits, preprocessing, and training budget.

A.1. Datasets for Baseline and SOTA

The following datasets are used in baseline comparisons:

1-1) Permuted MNIST (PMNIST) is a variant of MNIST (LeCun, 1998) where each task has a deterministic permutation applied to the input image pixels. The ground truth labels remain the same. 1-2) 5-Datasets (Saha et al., 2021) is a mixture of 5 different vision datasets (CIFAR10 (Krizhevsky et al., 2009), MNIST (LeCun, 1998), SVHN (Netzer et al., 2011), FashionMNIST (Xiao et al., 2017), and notMNIST (Bulatov, 2011)). We pad 0 values to raw images of MNIST and FashionMNIST and convert them to RGB format to have a dimension of $3 \times 32 \times 32$. Afterward, we normalize the raw image data as in (Serrà et al., 2018). 1-3) Omniglot Rotation has split 1200 classes into 100 tasks with 12 classes for each task. We further preprocess the raw images by generating their rotated version in $(90^\circ, 180^\circ, 270^\circ)$ as in (Yoon et al., 2018). We use the Omniglot dataset to test the scalability of our method for a large number of tasks.

In performing SOTA comparison, we are employing the following datasets: 2-1) CIFAR-100 Split (Krizhevsky et al., 2009) is constructed by randomly dividing 100 classes of CIFAR-100 into 10 tasks with 10 classes per task. 2-2) CIFAR-100 Superclass (Yoon et al., 2018) is divided into 20 tasks according to the 20 superclasses of the CIFAR-100 dataset, and each superclass contains 5 different but semantically related classes. 2-3) TinyImageNet (Stanford, 2021) is constructed by splitting 200 classes into 40 5-way classification tasks without data augmentation.

A.2. Architecture Details

Two-layered MLP: In conducting the PMNIST experiments, we are following the exact setup as denoted by (Saha et al., 2021) fully-connected network with two hidden layers of 100 (Lopez-Paz & Ranzato, 2017).

Reduced ResNet18: In conducting the 5-Dataset experiments, we use a smaller version of ResNet18 with three times fewer feature maps across all layers as denoted by (Lopez-Paz & Ranzato, 2017).

Modified LeNet: In conducting the Omniglot Rotation and CIFAR-100 Superclass experiments, we use a large variant of LeNet as the base network with 64-128-2500-1500 neurons based on (Yoon et al., 2020).

Modified AlexNet: In conducting the split CIFAR-100 dataset, we use a modified version of AlexNet similar to Serrà et al. (2018); Saha et al. (2021).

4 Conv layers and 3 Fully connected layers: For TinyImageNet, we use the same network architecture as Gupta et al. (2020); Deng et al. (2021).

All the networks for our experiments are implemented in a multi-head setting. They also utilize ReLU in the hidden units and softmax with cross-entropy loss in the final layer.

A.3. Training Details

In training all the models within the baseline comparison, we are using the exact setup and dataset splits described in (Saha et al., 2021). In conducting experiments with the PMNIST dataset, we keep 10% of the training data from each task for validation. On the other datasets, however, we keep only 5% of training data from each task for validation. We train all models within the baseline experiments using stochastic gradient descent (SGD). For each task in PMNIST, we train the network for 5 epochs with a batch size of 10. In 5-Dataset and Omniglot Rotation experiments, we train each task for a maximum of 100 epochs with the early termination strategy based on the validation loss proposed in (Serrà et al., 2018). For experiments in both datasets, we fix the batch size to 64. We run experiments under five different seed values for all experiments.

For conducting experiments for GPM (Saha et al., 2021) in Omniglot Rotation dataset, we use the threshold hyperparameter $\epsilon_{th} = 0.98$ for all the layers and increasing the value of ϵ_{th} by 0.0002 for each incoming new tasks.

For PackNet (Mallya & Lazebnik, 2018) and WSN, sparsity constraint c is applied across every layer within these methods except the last layer, where the multi-head classifier lies. Since PackNet (Mallya & Lazebnik, 2018) includes layer-wise capacity c for parameters that will be isolated for an incoming new set of tasks, we fix $c = 1/t$ while performing PMNIST and 5-Dataset experiments. Here, c denotes the layer-wise capacity, and t indicates the number of incoming tasks. For Omniglot experiments, we are fixing $c = 2/t$ as it turns out to be the setup that produces the most optimum accuracy-capacity tradeoff. Since PackNet (Mallya & Lazebnik, 2018) involves fine-tuning toward network parameters that are isolated for a given task to recover the accuracy, we allow the network to fine-tune up to half of the number of epochs required to finish the preceding training process.

As denoted in Table 1 and Table 2, we are performing experiments on WSN using different layer-wise capacity c ranging from 0.03 to 0.5 to analyze the behavior of WSN.

WSN decides whether to use the remaining free parameters within the architecture or reuse the weights in the past tasks upon dealing with the incoming tasks.

A.4. List of Hyperparameters

Table 5. List of hyperparameters for the baselines and our WSN. Here, 'lr' and 'optim' represents (initial) learning rate and optimizer used for training. We represent PMNIST as 'perm', 5-Datasets as '5data', Omniglot Rotation as 'omniglot', CIFAR-100 Split as 'cifar100-split', and CIFAR-100 Superclass as 'cifar100-sc'.

Methods	Hyperparameters
EWC	lr : 0.03 (perm) optim : sgd regularization coef cient : 1000 (perm)
GPM	lr : 0.01(perm, omniglot), 0.1 (5data) optim : sgd n_s : 300 (perm), 100 (5data), 125 (omniglot)
PackNet	lr = 0.001 (perm, 5data, omniglot) optim : adam c : 0.1 (perm), 0.2 (5data), 0.02 (omniglot)
WSN (ours)	lr = 0.001 (perm, 5data, omniglot, cifar100-split, cifar100-sc, tinyimagenet) optim : adam

Table 5 details hyperparameter setup for the baselines and our approach. n_s in GPM denotes the number of random training examples sampled from the replay buffer to construct the representation matrix for each architecture layer.

B. Additional Results

B.1. Additional Analysis on Baseline Comparisons.

(a) PMNIST - MLP (b) Omniglot Rotation - LeNet

Figure 9. Accuracy over Total Capacity Usage. WSN achieves best performance under Accuracy-Capacity trade-off in PMNIST and Omniglot Rotation datasets - It can be seen that WSN's accuracy is higher than other competing baselines, especially PackNet, while utilizing less total capacity.

Figure 9 shows the detailed accuracy over total capacity usage for the baselines and our approach on the PMNIST and Omniglot Rotation datasets with PackNet removed due to its substantial total capacity.

Employing WSN with the initial layer-wise capacity of $c = 0.3$ within the PMNIST dataset returns the best performing model in terms of accuracy-capacity tradeoff.

B.2. Forget-Free Performances and Capacities.

We prepare performances and bit-map compressed capacities on the TinyImageNet Dataset as shown in Figure 10. The $c = 0.1$ shows better performances over others. With xed $c = 0.1$, the bit-wise Huffman compression rate shows positive as the number of tasks increases. The most interesting part is that as the number of bits to compress increases, the average compression rate gets higher as the increased amount of reused weights might affect the compression rate (nodes with small probability might be rare). We investigated how the compression rate is related to the total model capacity. The more bits the binary mask compression does, the less the model capacity to save is required. We validated that within 40 tasks, N-bit Huffman compressed capacities are sub-linearly increasing while binary map capacities increase linearly. The 7-bit Huffman encoding is enough to compress binary maps without exceeding the model capacity even though the compression rate depends on compression methods typically.

B.3. Comparisons with the SOTA

The WSN's performances are compared with others w.r.t three measurements on three major benchmark datasets as shown in Table 6.

B.4. Comparisions of SupSup and WSN

Capability of WSN on PMNIST-250: We conduct experiments of WSN $c = 0.5; 0.7; 0.8$ (LeNet 300-100, $s=100$) with a single weight score on 250 tasks PMNIST (Wortsman et al., 2020) as shown in Figure 11. WSN performance with $c = 0.7$, initially at 97.3%, gradually drops after the 90th task, and it performed better than SupSup (96.4%) until then. When we divide PMNIST-250 into 90 tasks and use three weight scores, WSN performs better than SupSup (requiring 250 weight scores) in accuracy and memory ef ciency.

B.5. Capacities of Re-used Weights.

There exist two kinds of reused weight in a total model capacity, one is per task-dependent weights, and another is all task-dependent ones. To interpret the proportion of the two kinds of reused weights, we prepare the progressive capacities of layer-wise reused weights as shown in Figure 12. In Figure 12 (a), the c value determines the model capacity of the initial task, entire tasks, and the proportions of reused weights per task, and for all tasks. From Figure 12

(a) Per Task Accuracy over c (b) N-Bit-wise-Compression Rate (c) Progressive Capacities of Models

Figure 10. Performances and Compressed Capacities - Sequence of TinyImageNet Dataset Experiments. (a) The $c = 0.1$ shows generalized performances over others, (b) With x of $c = 0.1$, we investigate the bit-wise Huffman compression rate and observe that as the number of bits to compress increases the average compression rate gets higher, and (c) We compared the model capacity with the model capacity + the compressed binary masks over varying bits. Within the 40-tasks, the 7-bits compressed capacities are the least increasing along with the $c = 0.1$ model capacity.

Table 6. Performance comparisons of the proposed method and other state-of-the-art on various benchmark datasets. We report the mean and standard-deviation of the average accuracy (ACC), average capacity (CAP), and average backward transfer (BWT) across 5 independent runs with 5 seeds under the same experimental setup (Deng et al., 2021). The best results are highlighted in bold. Also, y and denotes results reported from (Deng et al., 2021) and (Gupta et al., 2020) respectively.

Method	CIFAR-100 Split			CIFAR-100 Superclass			TinyImageNet		
	ACC (%)	CAP (%)	BWT (%)	ACC (%)	CAP (%)	BWT (%)	ACC (%)	CAP (%)	BWT (%)
EWC (Kirkpatrick et al., 2017)	72.77 (0.45)^{y}	100.0	-3.59 (0.55)^{y}	50.26 (1.48)^{y}	100.0	-7.87 (1.63)^{y}	-	-	-
GEM (Lopez-Paz & Ranzato, 2017)	70.15 (0.34)^{y}	100.0	-8.61 (0.42)^{y}	50.35 (0.80)^{y}	100.0	-9.50 (0.85)^{y}	50.57 (0.61)	100.0	-20.50 (0.10)
ICARL (Rebuf et al., 2017)	53.50 (0.81)^{y}	100.0	-20.44 (0.82)^{y}	49.05 (0.51)^{y}	100.0	-11.24 (0.27)^{y}	54.77 (0.32)	100.0	-3.93 (0.55)
ER (Chaudhry et al., 2019b)	70.07 (0.35)^{y}	100.0	-7.70 (0.59)^{y}	51.64 (1.09)^{y}	100.0	-7.86 (0.89)^{y}	48.32 (1.51)	100.0	-19.86 (0.70)
La-MaML (Gupta et al., 2020)	71.37 (0.67)^{y}	100.0	-5.39 (0.53)^{y}	54.44 (1.36)^{y}	100.0	-6.65 (0.85)^{y}	66.90 (1.65)^{y}	100.0	-9.13 (0.90)^{y}
GPM (Saha et al., 2021)	73.18 (0.52)^{y}	100.0	-1.17 (0.27)^{y}	57.33 (0.37)^{y}	100.0	-0.37 (0.12)^{y}	67.39 (0.47)^{y}	100.0	1.45 (0.22)^{y}
FS-DGPM (Deng et al., 2021)	74.33 (0.31)^{y}	100.0	-2.71 (0.17)^{y}	58.81 (0.34)^{y}	100.0	-2.97 (0.35)^{y}	70.41 (1.30)^{y}	100.0	-2.11 (0.84)^{y}
PackNet (Mallya & Lazebnik, 2018)	72.39 (0.37)	96.38 (0.00)	0.0	58.78 (0.52)	126.65 (0.00)	0.0	55.46 (1.22)	188.67 (0.00)	0.0
SupSup (Wortsman et al., 2020)	75.47 (0.30)	129.00 (0.03)	0.0	61.70 (0.31)	162.49 (0.00)	0.0	59.60 (1.05)	214.52 (0.89)	0.0
WSN, $c = 0.03$	70.65 (0.36)	18.56 (0.25)	0.0	54.99 (0.71)	22.30 (0.22)	0.0	68.72 (1.63)	37.19 (0.21)	0.0
WSN, $c = 0.05$	72.44 (0.27)	25.09 (0.42)	0.0	57.99 (1.34)	27.37 (0.33)	0.0	71.22 (0.94)	41.98 (0.52)	0.0
WSN, $c = 0.1$	74.55 (0.47)	39.87 (0.62)	0.0	60.45 (0.37)	38.55 (0.20)	0.0	71.96 (1.41)	48.65 (3.03)	0.0
WSN, $c = 0.3$	75.98 (0.68)	80.26 (1.53)	0.0	61.47 (0.30)	63.47 (1.33)	0.0	70.92 (1.37)	73.44 (2.35)	0.0
WSN, $c = 0.5$	76.38 (0.34)	99.13 (0.48)	0.0	61.79 (0.23)	80.93 (1.58)	0.0	69.06 (0.82)	92.03 (1.80)	0.0
Multitask	79.75 (0.38)^{y}	100.0	-	61.00 (0.20)^{y}	100.0	-	77.10 (1.06)^{y}	100.0	-

Figure 11. Comparisons of SupSup and WSN on PMNIST-250.

(b), the capacity of Conv4 is greater than that of Conv1, while the Conv4 proportion of reused weights for all tasks is smaller than Conv1. From the results, we conclude that higher progressive capacity varieties result in less amount of reused weights for all tasks. In Figure 12 (c), the Conv4 has the properties such as the most dynamic change of progressive capacity and the least progressive capacity of reused weights for all tasks. From Figure 12 (d), within Linear1 and Linear2, except for the early task learning stage, the progressive capacities show the same tendency, and the WSN did not show performance drops as the progressive capacity of reused weights for all tasks decreases as shown in Figure 10 (a).

B.6. Sparse Binary Maps

We prepared binary map correlation results on several benchmark datasets. Figure 14 showed the progressive reused weights tendency, however, it tends to reuse weights least since PMNIST has an independent task. Also, Figure 15 resulted in the same tendency as in PMNIST results. Figure 16 shows the most discrete binary map since all tasks consist of 5 individual datasets. Figure 17, Figure 18, and Figure 19 showed a similar sparse binary map correlation

(a) Capacities over c (b) Conv1 vs. Conv4 Layer (c) Conv4 vs. Linear2 Layer (d) Linear1 vs. Linear2 Layer

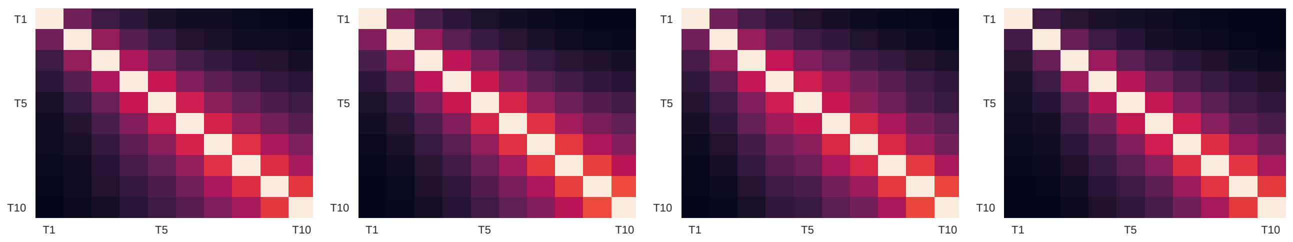
Figure 12. The 4 Conv & 3 FC Layer-wise Average Capacities on Sequence of TinyImageNet Dataset Experiments. (a) The proportion of reused weights per task depends on c value, and the proportion of reused weights for all tasks tends to be decreasing, (b) The capacity of Conv4 with high variance is greater than Conv1 with low variance, and the Conv4 proportion of reused weights for all tasks are smaller than the Conv1, (c) The capacity of Conv4 with high variance is greater than Liner2 with low variance and the Conv4 proportion of reused weights for all tasks are smaller than the Linear2, and (d) The capacity of Linear1 is greater than Linear2 in the early task learning stage, and the Linear1 proportion of reused weights for all tasks are smaller than the Linear2 in the early task learning stage. From (b), (c), and (d), the proportion of reused weights per task tends to be progressively saturated to $c = 0.1$ value.

according to the c value. From these observations, we could conclude that the sparsity of reused binary maps leads to better performances than others.

B.7. Additional Analysis

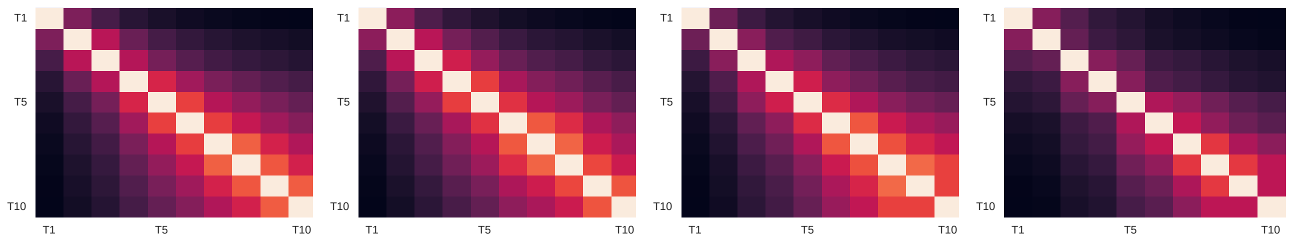
We reported the progressive ratio of weights used so far per layer as the number of tasks increases in Figure 12. Here, we interpreted how reused weights affect the performances on CIFAR-100 Split dataset as shown in Figure 13. The $c = 0.5$ represents the average accuracy of reused weights per task; the most signi cant weights were the subset of all prior weights since model forgetting occurs from the performances without being reused per task. The weights being reused for all tasks doesn't seem to be always positive effect on forget-free in that the importance of the weights decreases as the number of tasks increases. Our WSN reuse weights if the knowledge from previous tasks is already enough to solve the task at hand, and employs a few new weights otherwise. To be speci c, in task 5, all used weights seem to be enough to infer all CIFAR-100 Split tasks since the weights reused per task catch up with the task performance.

Figure 13. Reused Weight Performances on CIFAR-100 Split: (left) A diagram to explain a dense network of reused weights, (right) An average accuracy of reused weights; the most signi cant forgetting occurs from weights without reused per task; the signi - cance of weights reused for all tasks gets lower as the number of tasks increases; In task 5, all used weights seem to be enough to infer all CIFAR-100 Split tasks since the weights reused per task catch up with the task performance.



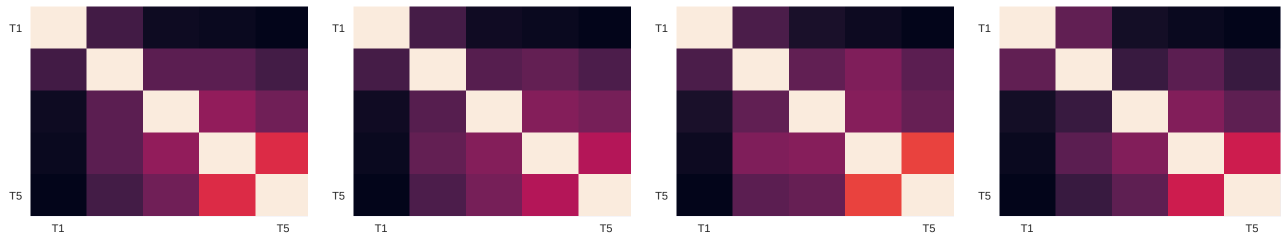
(a) c = 0:05 (b) c = 0:1 (c) c = 0:3 (d) c = 0:5

Figure 14. **Average Binary Map Correlation** on Sequence of PMNIST Experiments: the binary maps get overlapped with prior ones as the number of tasks increases..



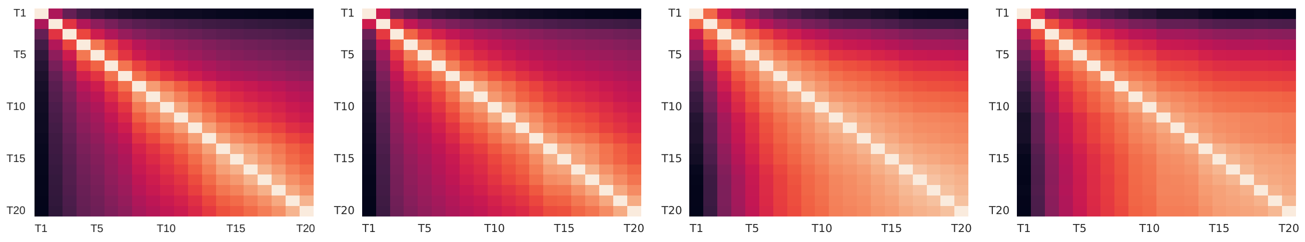
(a) c = 0:05 (b) c = 0:1 (c) c = 0:3 (d) c = 0:5

Figure 15. **Average Binary Map Correlation** on Sequence of CIFAR-100 Split Experiments: the binary maps get overlapped with prior ones as the number of tasks increases..



(a) c = 0:05 (b) c = 0:1 (c) c = 0:3 (d) c = 0:5

Figure 16. **Average Binary Map Correlation** on Sequence of 5-Dataset Experiments: the binary maps get overlapped with prior ones as the number of tasks increases..



(a) c = 0:05 (b) c = 0:1 (c) c = 0:3 (d) c = 0:5

Figure 17. **Average Binary Map Correlation** on Sequence of CIFAR-100 Superclass Experiments: the binary maps get overlapped with prior ones as the number of tasks increases..