Avoiding Forgetting and Allowing Forward Transfer in Continual Learning via Sparse Networks

Ghada Sokar[1]✉, Decebal Constantin Mocanu[2,1], and Mykola Pechenizkiy[1]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
{g.a.z.n.sokar, m.pechenizkiy}@tue.nl
[2] University of Twente, Enschede, The Netherlands
d.c.mocanu@utwente.nl

Abstract. Using task-specific components within a neural network in continual learning (CL) is a compelling strategy to address the *stability-plasticity* dilemma in fixed-capacity models without access to past data. Current methods focus only on selecting a sub-network for a new task that reduces forgetting of past tasks. However, this selection could limit the forward transfer of *relevant* past knowledge that helps in future learning. Our study reveals that satisfying both objectives jointly is more challenging when a unified classifier is used for all classes of seen tasks–class-Incremental Learning (class-IL)–as it is prone to ambiguities between classes across tasks. Moreover, the challenge increases when the semantic similarity of classes across tasks increases. To address this challenge, we propose a new CL method, named AFAF[3], that aims to Avoid Forgetting and Allow Forward transfer in class-IL using fix-capacity models. AFAF allocates a sub-network that enables *selective* transfer of relevant knowledge to a new task while preserving past knowledge, *reusing* some of the previously allocated components to utilize the fixed-capacity, and addressing class-ambiguities when similarities exist. The experiments show the effectiveness of AFAF in providing models with multiple CL desirable properties, while outperforming state-of-the-art methods on various challenging benchmarks with different semantic similarities.

Keywords: Continual learning · Class-incremental learning · Stability plasticity dilemma · Sparse training.

1 Introduction

Continual learning (CL) aims to build intelligent agents based on deep neural networks that can learn a sequence of tasks. The main challenge in this paradigm is the stability-plasticity dilemma [34]. Optimizing all model weights on a new task (*highest plasticity*) causes forgetting of past tasks [33]. While fixing all weights (*highest stability*) hinders learning new tasks. Finding the balance between stability and plasticity is challenging. The challenge becomes more difficult

[3] Code is available at: https://github.com/GhadaSokar/AFAF.

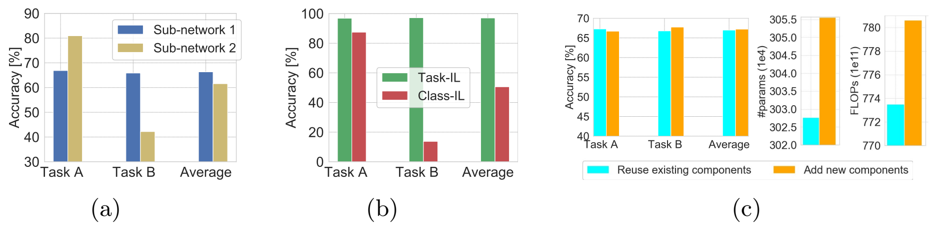


Fig. 1: (a) Two different sub-networks are evaluated for Task B in class-IL. One sub-network balances forgetting and forward transfer, while the other maintains the performance of Task A at the expense of Task B. (b) Performing the same model-altering scheme in task-IL and class-IL leads to different performance. (c) By reusing some of Task A's components during learning Task B, we can achieve similar performance in class-IL as adding new components while reducing the memory and computational costs, represented by model parameters (#params) and floating-point operations (FLOPs), respectively. Details are in Appendix B.

when other CL requirements are considered, such as using fixed-capacity models without access to past data and limiting memory and computational costs [13].

Task-specific components strategy [12,30,48] offers some flexibility to address this dilemma by using different components (connections/neurons), i.e., sub-network within a model, for each task. The components of a new task are flexible to learn, while the components of past tasks are fixed. There are some challenges that need to be tackled in this strategy to balance multiple CL desiderata:

(a) Selection of a new sub-network. Current methods focus solely on forgetting and choose a sub-network for a new task that would maintain the performance of past tasks regardless of its effectiveness for learning the new task. This hinders the forward transfer of relevant past knowledge in future learning.

(b) Managing the fixed-capacity and training efficiency. Typically, new components are added for every new task in each layer, which may unnecessarily consume the available capacity and increase the computational costs. Using fixed-capacity models for CL requires utilizing the capacity efficiently.

(c) Operating in the class-Incremental Learning setting (class-IL). Unlike task-Incremental Learning (task-IL), where each task has a separate classifier under the assumption of the availability of task labels at inference, in class-IL, a *unified* classifier is used for all classes in seen tasks so far. The latter is more realistic, yet it brings additional challenges due to agnosticity to task labels: **(1) class ambiguities**, using past knowledge in learning new classes causes ambiguities between old and new classes. **(2) component-agnostic inference**, all model components are used at inference since the task label is not available to select the corresponding components. This increases interference between tasks and the performance dependency on the sub-network of each task.

In response to these challenges, we study the following question: *How to alter the model structure when a new task arrives to balance CL desiderata in class-IL? Specifically, which components should be added, updated, fixed, or reused?*

We summarize our findings below with illustrations shown in Figure 1:

- The chosen sub-network of each task has a crucial role in the performance, affecting both forgetting and forward transfer (Figure 1a).
- The optimal altering of a model differs in task-IL and class-IL. For example, all components from past similar tasks could be *reused* in learning a new task with minimal memory and computational costs in task-IL. However, this altering limits learning in class-IL due to class ambiguities (Figure 1b).
- Reusability of past *relevant* components is applicable in class-IL under careful considerations for class ambiguity. It enhances memory and computational efficiency while maintaining performance (Figure 1c).
- Neuron-level altering is crucial in class-IL (e.g., *fixing* some neurons for a task), while connection-level altering could be sufficient in task-IL since only one sub-network is selected at inference using the task label (Appendix C).
- The challenge in balancing CL desiderata increases in class-IL when similarity across tasks increases (Section 5.1).

Motivated by these findings, we propose a new CL method, named AFAF, based on sparse sub-networks within a fix-capacity model to address the abovementioned challenges. Without access to past data, AFAF aims to Avoid Forgetting and Allow Forward transfer in class-IL. In particular, when a new task arrives, we identify the relevant knowledge from past tasks and allocate a new sub-network that enables selective forward transfer of this knowledge while maintaining past knowledge. Moreover, we reuse some of the allocated components from past tasks. To enable selective forward transfer and component re-usability jointly with forgetting avoidance in class-IL, AFAF considers the extra challenges of class-IL (class-ambiguities and agnostic component inference) in model altering. We propose two variants of standard CL benchmarks to study the challenging case where high similarity exists across tasks in class-IL. Experimental results show that AFAF outperforms baseline methods on various benchmarks while reducing memory and computation costs. In addition, our analyses reveal the challenges of class-IL over task-IL that necessitate different model altering.

2 Related Work

We divide CL methods into two categories: replay-free and replay-based.

Replay-free In this category, past data is inaccessible during future learning. It includes two strategies: **(1)** ***Task-specific components.*** *Specific* components are assigned to each task. Current methods either *extend* the initially allocated capacity of a model for new tasks [43,55] or use a *fixed-capacity* model and add a *sparse* sub-network for each task [30,29,54,32,52,48]. Typically, connections of past tasks are kept *fixed*, and the newly added ones are *updated* to learn the current task. The criteria used for *adding* new connections often focus solely on avoiding forgetting, limiting the selective transfer of relevant knowledge to learn future tasks. Moreover, most methods rely on task labels to pick the connections corresponding to the task at inference. SpaceNet [48] addressed component-agnostic inference by learning sparse representation during training and *fixing* a fraction of the most important neurons of each task to reduce interference. Sparse representations are learned by training each sparse sub-network using dynamic sparse training [36,14], where weights and the sparse topology are jointly optimized. Yet, selective transfer and class ambiguity are not addressed. **(2)** ***Regularization-based.*** A *fixed-capacity* model is used, and *all* weights are updated for each task. Forgetting is addressed by constraining changes in the important weights of past tasks [56,19,1] or via distillation loss [25,9].

Replay-based In this category, forgetting is addressed by *replaying*: (1) a subset of old samples [41,27,42,5,3], (2) pseudo-samples from a generative model [37,45,46], or (3) generative high-level features [50]. A buffer is used to store old samples or a generative model to generate them.

Attention to task similarities and forward transfer has recently increased. Most efforts are devoted to task-IL. In [40], the analysis showed that higher layers are more prone to forgetting, and intermediate semantic similarity across tasks leads to maximal forgetting. SAM [47] meta-trains a self-attention mechanism for selective transfer in dense networks. CAT [18] addresses the relation between task similarities and forward/backward transfer in task-IL, where task labels are used to find similar knowledge in dense models. In [23], an expectation-maximization method was proposed to select the shared or added *layers* to promote transfer in task-IL. Similarly, [51] uses a modular neural network architecture and searches for the optimal path for a new task by the composition of neural modules. A task-driven method is used to reduce the exponential search space. In [6], the lottery ticket hypothesis [11] is studied for CL.

3 Network Structure Altering

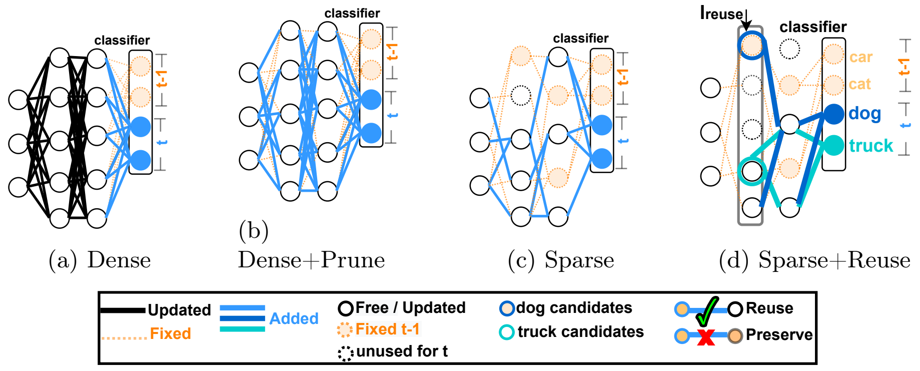
When a model faces a new task, task-specific components methods alter its structure via some actions to address the stability-plasticity dilemma. Next, we will present commonly used and our proposed actions.

3.1 Connection-Level Actions

Most state-of-the-art methods alter fixed-capacity models at the ***connection level***. The connection-level actions include:

- "**Add**": New connections, parameterized by $\mathbf{W}^t$, are added for each new task t. Each task has a *sparse* sub-network resulting from either pruning dense connections [30] (Figure 2b) or adding sparse ones from scratch [48] (Figure 2c). The current practice is to ***randomly*** add connections in *each* layer using unimportant components of past tasks focusing solely on forgetting [48,30]; $\mathbf{W}^t = \{\mathbf{W}^t_l : 1 \leq l \leq L-1\}$ where L is the number of layers. This will be addressed in Section 4 to enable ***selective transfer*** and ***reusability***.
- "**Update**": $\mathbf{W}^t$ are updated during task t training (i.e., allows plasticity).
- "**Fix**": Once a task has been learned, $\mathbf{W}^t$ are frozen (i.e., controls stability).

Note that regularization methods *add* **dense** connections at step $t = 0$ (Figure 2a). Each task *updates* all weights. Stability is controlled via regularization.



(a) Dense (b) Dense+Prune (c) Sparse (d) Sparse+Reuse

Fig. 2: Overview of a model when it faces a new task t using different methods. (a) Regularization methods use and update all connections with regularization for learning task t [19,1]. (b) Dense connections are added, and unimportant connections are pruned after training [30]. Connections of previous tasks are fixed. (c) Sparse connections are added from scratch [48]. Connections and important neurons of past tasks are fixed (dashed lines and filled circles). (d) Our proposed method, AFAF. Starting from layer l_{reuse}, new connections are added for each class using the candidate neurons for the class to enable selective transfer and free neurons to capture residual knowledge. New outgoing connections are allowed from fixed neurons at layer l but could only connect free neurons (unfilled circles) in layer $l+1$ to preserve the old knowledge.

3.2 Neuron-Level Actions

Component-agnostic inference in class-IL makes reducing task interference at the connection level only more challenging since connections from different tasks share the same neurons (Figure 2b). Reducing interference at the *representational* level is more efficient (Appendix C). Hence, we believe that the following neuron-level actions are needed:

- "**Fix**": After learning a task, its *important* neurons should be frozen to reduce the drift in its representation [48]. Other neurons are "free" to be updated.
- "**Reuse**": Reusing neurons that capture useful representation in learning future tasks. Fixed neurons could be reused by allowing ***outgoing*** connections only from these neurons. Details are provided in the next section.

Most previous methods operate at the connection level except the recent work, SpaceNet [48], which shows its favorable performance over replay-free methods using a fixed-capacity model via *fixing* the important neurons of each task and learning sparse representations (Figure 2c). However, besides adding new components for every task in each layer, it does not *selectively* transfer relevant knowledge. A summary is provided in Table 1. Our goal is to address selective transfer and reusability of previous relevant neurons while maintaining them stable.

Table 1: Actions used to alter the model components at the connection and neuron levels by different methods.

Method	Add weights	Fix old weights	Fix neurons	Reuse past components	Selective transfer
EWC [19], MAS [1]	Dense	$\times$	$\times$	$\times$	$\times$
PackNet [30]	Dense+Prune	$\surd$	$\times$	$\times$	$\times$
SpaceNet [48]	Sparse	$\surd$	$\surd$	$\times$	$\times$
AFAF (ours)	Sparse	$\surd$	$\surd$	$\surd$	$\surd$

4 Proposed Method

We consider the problem of learning T tasks sequentially using a *fix-capacity* model with a unified classifier. Each task t brings new C classes. The output layer is extended with new C neurons. Let $\mathcal{D}^t = \{D^{t_c}\}|_{c=1}^{C}$, where D^{t_c} is the data of class c in task t. Once a task has been trained, its data is *discarded*.

AFAF is a new task-specific component method that dynamically trains a sparse sub-network for each task from scratch. Our goal is to alter the model components to (i) reuse past components and (ii) selectively transfer relevant knowledge while (iii) reducing class ambiguity in class-IL. This relies on the selection of a new sub-network and its training. When a new task arrives, AFAF adds a sub-network such that past knowledge is transferred from relevant neurons while retaining it, and residual knowledge is learned. Section 4.1 introduces the selection mechanism of a sub-network and illustrates the added and reused components. Section 4.2 discusses the considerations for class ambiguity. In Section 4.3, we present task training and identify the fixed and updated components.

4.1 Selection of a New Sub-network

Notation Let $l \in \{1, .., L\}$ represents a layer in a neural network model. A sparse connections W_l^t, with a sparsity ϵ_l, are allocated for task t in layer l between a subset of the neurons denoted by $\mathbf{h}_l^{alloc}$ and $\mathbf{h}_{l+1}^{alloc}$ in layers l and $l+1$, respectively. We use the term *neuron* to denote a node in multilayer perceptron networks or a feature map in convolution neural networks (CNNs). The selected neurons in each layer determine the initial sub-network for a new task.

Typically, when there is a semantic similarity between old and new classes, the existing learned components likely capture some useful knowledge for the current task. Hence, we propose to reuse some of these components. Namely, instead of adding new connections in each layer, we allocate new sparse connections starting from layer l_{reuse} (Figure 2d). l_{reuse} is a hyper-parameter that controls the trade-off between adding new connections and reusing old components based on existing knowledge and the available capacity. Hence, the connections of a new task t are $\mathbf{W}^t = \{\mathbf{W}_l^t : l_{reuse} \leq l \leq L-1\}$. Layers from $l = 1$ up to but excluding layer l_{reuse} remains unchanged. We show that reusing past components reduces memory and computational costs (Section 5.1), balances forward and backward transfer (Section 5.2), and utilizes the available fixed-capacity efficiently by allowing new dissimilar tasks to acquire more resources that are saved by reusability, i.e., higher density for sparse sub-networks (Appendix D).

Starting from l_{reuse}, we add new sparse connections $\mathbf{W}_l^t$ in each layer l. The selected neurons for allocation should allow: (i) selective transfer of relevant knowledge to promote forward transfer and (ii) preserving old representation to avoid forgetting. To this end, we select a set of *candidate* and *free* neurons in each layer, as we will discuss next. Details are provided in Algorithm 1.

Identify Candidate Neurons. To *selectively transfer* the relevant knowledge, for each layer $l \geq l_{reuse}$, we identify a set of candidate neurons $\mathcal{R}_l^c$ that has a high potential of being useful when "reused" in learning *class* c in a new task t. Classes with semantic similarities are most likely to share similar representations (Appendix F). Hence, we consider the average activation of a neuron as a metric to identify its potential for reusability. In particular, we feed the data of each class c in a new task t, $\mathcal{D}^{t_c}$, to the trained model at time step $t-1$, $f^{t-1}(\mathbf{W}^{t-1})$, and calculate the average activation $\mathcal{A}_l^c$ in each layer l as follows:

$$\mathcal{A}_l^c(\mathbf{W}^{t-1}) = \frac{1}{|\mathcal{D}^{t_c}|} \sum_{i=1}^{|\mathcal{D}^{t_c}|} \mathbf{a}_l(x_i^{t_c}), \tag{1}$$

where $\mathbf{a}_l$ is the vector of neurons activation at layer l when a sample $x_i^{t_c} \in \mathcal{D}^{t_c}$ is fed to the model $f^{t-1}(\mathbf{W}^{t-1})$, and $|\mathcal{D}^{t_c}|$ is the number of samples of class c. Once the activation is computed, neurons with the top-κ activation are selected as potential candidates $\mathcal{R}_l^c$ as follows:

$$\mathcal{R}_l^c = \{i | \mathcal{A}_{l_i}^c \in \text{top-}\kappa(\mathcal{A}_l^c)\}, \tag{2}$$

where κ denotes the number of candidate neurons, and $\mathcal{A}_{l_i}^c$ is the average activation of neuron i in layer l. Neuron activity shows its effectiveness as an estimate of a neuron/connection importance in pruning neural networks for single task learning [15,28,7]. To assess our choice of this metric to identify the candidate neurons, we compare against two metrics: **(1) Random**, where the candidate neurons are randomly chosen from all neurons in a layer, and **(2) Lowest**, where the candidates are the neurons with the lowest activation (See Section 5.2).

Note that by exploiting the representation of candidate neurons $\mathcal{R}_{l_{reuse}}^c$ in layer l_{reuse}, we selectively *reuse* past components connected to these neurons in preceding layers ($l < l_{l_{reuse}}$). Hence, these components are stable but reusable.

Identify Free Neurons. To preserve past representation while allowing reusability of neurons, "fixed" neurons that might be selected as candidates should be stable. Hence, we allow *outgoing* connections from these neurons but not incoming connections. The outgoing connections from fixed neurons in layer l can be connected to "free" neurons in layer $l+1$. To this end, in each layer, we select a subset of the free neurons for *each task*, denoted as $\mathcal{S}_l^{Free}$. For these neurons, incoming and outgoing connections are allowed to be allocated and used to capture the specific representation of a new task.

Algorithm 1 AFAF Sub-network Allocation

```
1: Require: $l_{reuse}$, sparsity level $\epsilon_l$, $|\mathbf{h}_l^{alloc}|$, $\kappa$, $\mathbf{h}_l^{Fix}$
2: for each class $c$ in task $t$ do                                   ▷ Get candidate neurons
3:     $\mathcal{A}_l^c \leftarrow$ calculate average activation of $\mathcal{D}^{t_c}$          ▷ Eq 1
4:     $\mathcal{R}_l^c \leftarrow$ get candidates for class $c$ $\forall l \geq l_{reuse}$       ▷ Eq 2
5: end for each
6: $\mathcal{S}_l^{Free} \leftarrow$ randomly select subset of free neurons $\forall l \geq l_{reuse}$
7: $\mathcal{R}_{L-1}^c, \mathcal{R}_L^c \leftarrow \emptyset$                ▷ No candidate neurons used in last layer
8: $\mathcal{S}_L^{Free} \leftarrow \{c\}$   $\forall c \in t$              ▷ Output neurons for new classes
9: for each class $c$ in task $t$ do
10:     for $l \leftarrow l_{reuse}$ to $L-1$ do
11:         $\mathbf{h}_l^{alloc} \leftarrow \{\mathcal{R}_l^c \cup \mathcal{S}_l^{Free}\}$                ▷ Neurons for allocation
12:         $\mathbf{h}_{l+1}^{alloc} \leftarrow \{(\mathcal{R}_{l+1}^c \setminus \mathbf{h}_{l+1}^{Fix}) \cup \mathcal{S}_{l+1}^{Free}\}$    ▷ Neurons for allocation
13:         Allocate $\mathbf{W}_l^c$ with sparsity $\epsilon_l$ between $\mathbf{h}_l^{alloc}$ & $\mathbf{h}_{l+1}^{alloc}$
14:         $\mathbf{W}_l \leftarrow \mathbf{W}_l \cup \mathbf{W}_l^c$
15:     end for each
16: end for each
```

Allocation. The neurons used to allocate new sparse weights $\mathbf{W}_l^c$ between layers l and $l+1$ for a class c are as follows:

$$\begin{aligned} \mathbf{h}_l^{alloc} &= \{\mathcal{R}_l^c \cup \mathcal{S}_l^{Free}\}, \\ \mathbf{h}_{l+1}^{alloc} &= \{(\mathcal{R}_{l+1}^c \setminus \mathbf{h}_{l+1}^{Fix}) \cup \mathcal{S}_{l+1}^{Free}\}, \end{aligned} \tag{3}$$

where $\mathbf{h}_{l+1}^{Fix}$ is the set of fixed neurons at layer $l+1$.

4.2 Addressing Class Ambiguities

Unlike task-IL, reusing past relevant knowledge in future learning may result in class ambiguity in class-IL (Section 1). To this end, we propose three constraints for reusability that aim to (1) allow a new task to learn its specific representation and (2) increase the decision margin between classes (see Section 5.2 for analysis).

Learn specific representation. Learning specific representation reduces ambiguity between similar classes. Hence, we add two constraints. First, new connections should be added in at least one layer before the classification layer to capture the specific representation of the task (i.e., $l_{reuse} \in [2, L-2]$). Second, the output connections are allocated using free neurons only (i.e., no candidate neurons are used; $\mathbf{h}_{L-1}^{alloc} = \mathcal{S}_{L-1}^{Free}$) since candidate neurons in the highest-level layer are highly likely to capture the specific representation of past classes.

Increase the decision margin between classes. To learn more discriminative features and increase the decision margin between classes, we use orthogonal weights in the output layer [24]. To this end, once a task has been trained, we force all its neurons in the last layer, $\mathbf{h}_{L-1}^{alloc}$, to be fixed and not reusable.

4.3 Training

The new connections are trained with stochastic gradient descent. During training, the weights and important neurons of past tasks are kept fixed to protect past representation. We follow the approach in [48] to train a sparse topology (connections distribution) and identify the portion of the important neurons from $\mathbf{h}_l^{alloc}$ in each layer that will be fixed after training (Appendix G). In short, a sparse topology is optimized by a dynamic sparse training approach to produce sparse representation. To reuse important neurons of past tasks while protecting the representation, we block the gradient flow through all-important neurons of past tasks, even if they are reused as candidates for the current task. The gradient $\mathbf{g}_l$ through the neurons of layer l is:

$$\mathbf{g}_l = \mathbf{g}_l \otimes (1 - \mathbb{1}_{\mathbf{h}_l^{Fix}}), \tag{4}$$

where $\mathbb{1}$ is the indicator function. This *allows* not to forget past knowledge while reusing it for selective transfer. Since the important neurons of dissimilar classes are less likely to be involved in the sub-network selection, they are protected.

5 Experiments and Results

Baselines. Our study focuses on the replay-free setting using a fixed-capacity model. Therefore, we compare with several representative regularization methods that use *dense* fixed-capacity, **EWC** [19], **MAS** [1], and **LWF** [25]. In addition, we compare with task-specific components methods that use *sparse* sub-networks within a fixed-capacity model, **PackNet** [30] and **SpaceNet** [48].

Benchmarks. We performed our experiments on three sets of benchmarks: (1) standard split-CIFAR10, (2) sequences with high semantic similarity at the class level across tasks, and (3) sequence of mixed datasets where tasks come from different domains to study the stability-plasticity dilemma for sequences with concept drift and interfering tasks.

Standard Evaluation. We evaluate the standard **split-CIFAR10** benchmark with 5 tasks. Each task consists of 2 consecutive classes of CIFAR10 [21].

Similar Sequences. To assess replay-free methods under more challenging conditions, we design two new benchmarks with high semantic similarity across tasks. In the absence of past data, we test the unified classifier's ability to distinguish between similar classes when they are not presented together within the same task. **(1) sim-CIFAR10** is constructed from CIFAR-10 by shuffling the order of classes to increase the similarity across tasks (Appendix A; Table 5). It consists of 5 tasks. **(2) sim-CIFAR100** is constructed from CIFAR-100 [20]. Classes within the same superclass in CIFAR-100 have high semantic similarity. Hence, we construct a sequence of 8 tasks with two classes each and distribute the classes from the same superclass in different tasks (Appendix A; Table 6).

Mix datasets. The considered datasets are CIFAR10 [20], MNIST [22], NotMNIST [4], and FashionMNIST [53]. We construct a sequence of 8 tasks with 5 classes each. The first four tasks are dissimilar, while the second four are similar to the first ones (Appendix A; Table 7).

Implementation Details. Motivated by the recent study on architectures for CL [35], we follow [44,18,51] to use an AlexNet-like architecture [21] that is trained from scratch using stochastic gradient descent. We start reusing relevant knowledge in future tasks after learning similar ones. Therefore, for Mix and sim-CIFAR100, we start reusing past components from task 5, while for split-CIFAR10 and sim-CIFAR10, we start from task 3. Earlier tasks add connections in each layer using the free neurons. For all benchmarks, we use l_{reuse} of 4.

Evaluation Metrics. To evaluate different CL requirements, we assess various metrics: (1) **Average accuracy (ACC)**, which measures the performance at the end of the learning experience, (2) **Backward Transfer (BWT)** [27], which measures the influence of learning a new task on previous tasks (large negative BWT means forgetting), (3) **Floating-point operations (FLOPs)**, which measure the required computational cost, and (4) **Model size (#params)**, which is the number of model parameters. More details are in Appendix A.

5.1 Results

Table 2 shows the performance on each benchmark. AFAF consistently outperforms regularization-based methods and other task-specific components methods on all benchmarks. The difference in performance between split-CIFAR10 and sim-CIFAR10, which have the same classes in a different order, reveals the challenge caused by having similar classes across tasks in class-IL. All studied methods have lower ACC and BWT on sim-CIFAR10 than split-CIFAR10. Yet, AFAF is the most robust method towards this challenge. When the similarity across tasks increases more, as in sim-CIFAR100, regularization methods and PackNet fail to achieve a good performance. We also observe that LWF outperforms other regularization-based methods in most cases except on the Mix datasets benchmark, where there is a big distribution shift across tasks from different domains. Most task-specific components methods outperform the regularization-based ones with much lower forgetting.

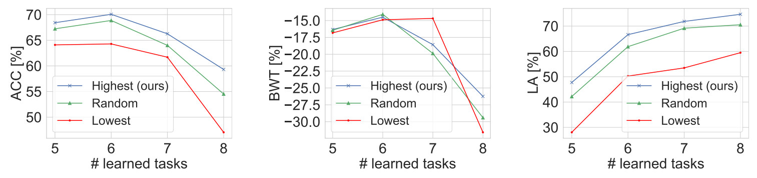
Analyzing task-specific components methods, we observe that altering the model at the connection level only by PackNet is not efficient in class-IL despite its high performance in task-IL (Appendix C). Besides the additional memory and computational overhead of pruning and retraining dense models, the performance is lower than other task-specific components methods. Altering at the connection and neuron levels, as in SpaceNet and AFAF, enables higher performance. The gap between these methods increases when the sequence has a larger number of tasks with high similarities (i.e., sim-CIFAR100 and Mix datasets).

AFAF consistently achieves higher ACC and BWT than SpaceNet on all benchmarks with various difficulty levels. Interestingly, the achieved performance is obtained by *reusing* relevant knowledge via selective transfer. AFAF exploits the similarity across tasks in learning future tasks while addressing class ambiguities. Moreover, the performance gain is accompanied by using a smaller memory and less computational cost than all the baselines.

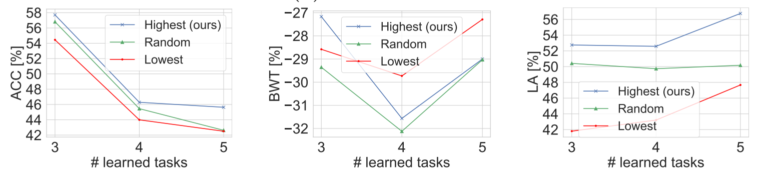
Table 2: Evaluation results on four CL benchmarks in the replay-free class-IL setting with fixed-capacity models.

	Split-CIFAR10				Sim-CIFAR10			
Method	ACC (↑)	BWT (↑)	FLOPs (↓)	#params (↓)	ACC (↑)	BWT (↑)	FLOPs (↓)	#params (↓)
Dense Model	-	-	1×(14.97e14)	1×(23459520)	-	-	1×(14.97e14)	1×(23459520)
EWC [19]	38.30±0.81	-59.30±2.03	1×	1×	28.90±3.11	-66.10±5.49	1×	1×
LWF [25]	48.10±2.28	-42.33±1.15	1×	1×	40.43±1.22	-50.36±1.98	1 ×	1×
MAS [1]	38.30±1.06	-56.56±3.30	1×	1×	28.93±4.05	-61.86±5.66	1×	1×
PackNet [30]	44.33±0.85	-50.40±1.45	3.081×	1×	32.63±1.22	-61.96±1.52	3.081×	1×
SpaceNet [48]	51.63±1.28	-36.50±1.53	0.154×	0.154×	42.86±4.57	-30.69±4.63	0.325×	0.325×
AFAF (ours)	**52.35±2.35**	**-32.93±3.19**	**0.148**×	**0.148**×	**45.23±2.14**	**-29.35±3.54**	**0.322**×	**0.322**×

	Sim-CIFAR100				Mix datasets			
Method	ACC (↑)	BWT (↑)	FLOPs (↓)	#params (↓)	ACC (↑)	BWT (↑)	FLOPs (↓)	#params (↓)
Dense Model	-	-	1×(23.96e13)	1×(23471808)	-	-	1×(5.600e15)	1×(23520960)
EWC [19]	15.73±1.89	-74.50±5.10	1×	1×	54.63±1.93	-31.33±2.82	1×	1×
LWF [25]	14.40±2.69	-52.86±1.12	1×	1×	40.60±3.25	-60.46±3.54	1×	1×
MAS [1]	13.50±0.66	-81.70±1.02	1×	1×	56.86±1.81	-26.83±3.44	1×	1×
PackNet [30]	10.12±2.55	-20.35±3.83	5.241×	0.8×	16.61±2.35	**-18.58±1.38**	5.250×	0.8×
SpaceNet [48]	32.86±2.73	-36.29±2.78	0.089×	0.089×	56.25±1.69	-30.02±1.79	0.053×	0.053×
AFAF (ours)	**33.74±2.18**	**-21.26±2.21**	**0.088**×	**0.088**×	**59.02±1.76**	-25.91±1.51	**0.050**×	**0.050**×



(a) Mix datasets



(b) sim-CIFAR10

Fig. 3: Performance of AFAF backbone with different strategies for selecting the candidate neurons used to add new connections.

5.2 Analysis

Effect of Selective Transfer. To measure the impact of selective transfer and the initially allocated sub-network on forward and backward transfer in AFAF, we compare our selection strategy for the candidate neurons to two other potential strategies discussed in Section 4.1: *Random* and *Lowest*. To reveal the role of selective transfer on performance, we also report the Learning Accuracy (LA) [42], which is the average accuracy for each task directly after it is learned (Appendix A). We calculate this metric starting from the first task that reuses components in its learning onwards. Figure 3 shows the results on Mix datasets and sim-CIFAR10. We present the performance of the tasks that reuse past components (i.e., $t \geq 5$ and $t \geq 3$ for Mix and sim-CIFAR10, respectively) since the performance of other earlier tasks is the same for all baselines (i.e., the allocation is based on the free neurons). As shown in the figure, using the relevant neurons with the highest activation to allocate a new sub-network leads to higher LA on new tasks and lower negative BWT on past tasks than using random neurons. AFAF also has higher ACC than the Random baseline by 4.79% and 3.01% on Mix and sim-CIFAR10, respectively. On the other hand, the Lowest baseline limits learning new tasks. It has much lower ACC and LA than the other two baselines. Note that the high BWT of this Lowest baseline is a factor of its low LA. This analysis shows the effect of the *initial* sub-network on performance, although topological optimization occurs during training.

Class Ambiguities. We performed an ablation study to assess each of our proposed contributions in addressing class ambiguities in class-IL (Section 4.2). We performed this analysis on Mix datasets and sim-CIFAR10 benchmarks. To show that class ambiguity causes more challenges in class-IL, we also report the performance in task-IL. To compare only the effect of class ambiguities in both

Table 3: Effect of each contribution in addressing class ambiguities: orthogonal output weights, using free neurons only for allocating output weights, and constraining reusing all past components. ACC is reported in class-IL and task-IL.

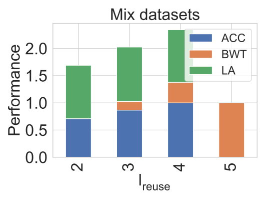
	sim-CIFAR10		Mix datasets	
Method	class-IL	task-IL	class-IL	task-IL
AFAF (ours)	**45.23±2.14**	**94.57±0.05**	**59.02±1.76**	**93.41±0.26**
w/o orth $\mathbf{W}_L$	41.74±2.05	93.20±0.26	56.38±3.34	93.39±0.30
w/ $\mathcal{R}_{L-1}$	40.30±2.14	93.27±0.17	56.83±1.19	92.85±0.28
w/ $l_{reuse} = L - 1$	22.27±1.61	83.54±1.64	40.98±2.75	85.05±1.01

scenarios, we assume components-agnostic inference for task-IL. Yet, task labels are used to select the output neurons that belong to the task at hand.

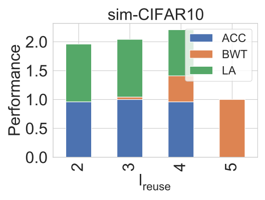
Analysis 1: Effect of orthogonal output weights. We evaluate a baseline that denotes AFAF without using orthogonal weights in the output layer "w/o orth W_L". We obtain this baseline by fixing part of the neurons in the last layer instead of fixing all neurons. We use the same fraction used for other fully connected layers (Appendix A). Table 3 shows the results. Having a large decision margin via orthogonal output weights increases the performance. We observe that the difference between AFAF and this baseline is larger in class-IL than task-IL, which indicates that task-IL is less affected by class ambiguities.

Analysis 2: Effect of using free neurons only in the last layer. To analyze this effect, we add another baseline that uses free $\mathcal{S}_{L-1}^{free}$ and candidate $\mathcal{R}_{L-1}$ neurons in allocating the output weights. We denote this baseline as "w/$\mathcal{R}_{L-1}$". As shown in Table 3, using free neurons only allows learning specific representation that decreases the ambiguities across tasks. The performance in class-IL is improved by 4.93% and 2.19% on sim-CIFAR10 and Mix datasets, respectively.

Analysis 3: Effect of constraining reusing all past components. We analyze the performance obtained by reusing all past components in learning similar tasks (i.e., $l_{reuse} = L-1$). We denoted this baseline as "w/$l_{reuse} = L-1$". As shown in Table 3, the performance has decreased dramatically. Despite that the degradation also occurs in task-IL, it has less effect. This shows the challenge of balancing performance, memory, and computational costs in class-IL.



(a) Mix datasets



(b) Sim-CIFAR10

Fig. 4: Normalized performance of AFAF at different values of l_{reuse}.

Reusable layers. We analyze the effect of reusing full layers on performance. We evaluate the performance at the possible values of $l_{reuse} \in [2, L-2]$ (Section 4.2). A min-max scaling is used to normalize the ACC, BWT, and LA; exact values are in Appendix E. Figure 4 shows the performance of Mix datasets and sim-CIFAR10. Adding new components in lower-level layers ($l_{reuse} \in \{2, 3\}$) enables new tasks to achieve high LA but increases forgetting (negative BWT), leading to lower ACC. Reusing the components in lower-level layers while adding new components to learn specific representations in the higher-level ones ($l_{reuse} = 4$) achieves a balance between ACC, BWT, and LA. While using a higher value for l_{reuse} limits the performance of new tasks due to class ambiguities, leading to the lowest LA and ACC.

We performed another study to illustrate the effect of reusing past components in utilizing the fixed-capacity. We show that reusability allows for using higher density for tasks that are dissimilar to the previously learned knowledge, which increases the performance. Details are provided in Appendix D.

6 Conclusion

Addressing the stability-plasticity dilemma while balancing CL desiderata is a challenging task. We showed that the challenge increases in the class-IL setting, especially when similar classes are not presented together within the same task. With our proposed task-specific components method, AFAF, we show that altering the model components based on exploiting past knowledge helps in achieving multiple desirable CL properties. Critically, the choice of the sub-network of a new task affects the forward and backward transfer. Hence, we proposed a selection mechanism that selectively transfers relevant knowledge while preserving it. Moreover, we showed that complete layers could be reused in learning similar tasks. Finally, we addressed the class ambiguity that arises in class-IL when similarities increase across tasks and showed that model altering at the connection and neuron levels is more efficient for component-agnostic inference.

References

1. Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., Tuytelaars, T.: Memory aware synapses: Learning what (not) to forget. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 139–154 (2018)
2. Atashgahi, Z., Sokar, G., van der Lee, T., Mocanu, E., Mocanu, D.C., Veldhuis, R., Pechenizkiy, M.: Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. Machine Learning pp. 1–38 (2021)
3. Bang, J., Kim, H., Yoo, Y., Ha, J.W., Choi, J.: Rainbow memory: Continual learning with a memory of diverse samples. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8218–8227 (2021)
4. Bulatov, Y.: Notmnist dataset. Technical report (2011), `http://yaroslavvb.blogspot.it/2011/09/notmnist-dataset.html`
5. Chaudhry, A., Ranzato, M., Rohrbach, M., Elhoseiny, M.: Efficient lifelong learning with a-gem. In: International Conference on Learning Representations (2018)

6. Chen, T., Zhang, Z., Liu, S., Chang, S., Wang, Z.: Long live the lottery: The existence of winning tickets in lifelong learning. In: International Conference on Learning Representations (2020)
7. Dekhovich, A., Tax, D.M., Sluiter, M.H., Bessa, M.A.: Neural network relief: a pruning algorithm based on neural activity. arXiv preprint arXiv:2109.10795 (2021)
8. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., de Freitas, N.: Predicting parameters in deep learning. In: Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2. pp. 2148–2156 (2013)
9. Dhar, P., Singh, R.V., Peng, K.C., Wu, Z., Chellappa, R.: Learning without memorizing. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5138–5146 (2019)
10. Evci, U., Gale, T., Menick, J., Castro, P.S., Elsen, E.: Rigging the lottery: Making all tickets winners. In: International Conference on Machine Learning. pp. 2943–2952. PMLR (2020)
11. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2018)
12. Golkar, S., Kagan, M., Cho, K.: Continual learning via neural pruning. arXiv preprint arXiv:1903.04476 (2019)
13. Hadsell, R., Rao, D., Rusu, A.A., Pascanu, R.: Embracing change: Continual learning in deep neural networks. Trends in cognitive sciences (2020)
14. Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., Peste, A.: Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. Journal of Machine Learning Research **22**(241), 1–124 (2021)
15. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016)
16. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. PMLR (2015)
17. Jayakumar, S., Pascanu, R., Rae, J., Osindero, S., Elsen, E.: Top-kast: Top-k always sparse training. Advances in Neural Information Processing Systems **33**, 20744–20754 (2020)
18. Ke, Z., Liu, B., Huang, X.: Continual learning of a mixed sequence of similar and dissimilar tasks. Advances in Neural Information Processing Systems **33** (2020)
19. 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)
20. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
21. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25**, 1097–1105 (2012)
22. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
23. Lee, S., Behpour, S., Eaton, E.: Sharing less is more: Lifelong learning in deep networks with selective layer transfer. In: International Conference on Machine Learning. pp. 6065–6075. PMLR (2021)
24. Li, X., Chang, D., Ma, Z., Tan, Z.H., Xue, J.H., Cao, J., Yu, J., Guo, J.: Oslnet: Deep small-sample classification with an orthogonal softmax layer. IEEE Transactions on Image Processing **29**, 6482–6495 (2020)

25. Li, Z., Hoiem, D.: Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence **40**(12), 2935–2947 (2017)
26. Liu, S., Chen, T., Atashgahi, Z., Chen, X., Sokar, G., Mocanu, E., Pechenizkiy, M., Wang, Z., Mocanu, D.C.: Deep ensembling with no overhead for either training or testing: The all-round blessings of dynamic sparsity. arXiv preprint arXiv:2106.14568 (2021)
27. Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continual learning. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 6470–6479 (2017)
28. Luo, J.H., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. In: Proceedings of the IEEE international conference on computer vision. pp. 5058–5066 (2017)
29. Mallya, A., Davis, D., 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). pp. 67–82 (2018)
30. Mallya, A., Lazebnik, S.: Packnet: Adding multiple tasks to a single network by iterative pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7765–7773 (2018)
31. Masana, M., Liu, X., Twardowski, B., Menta, M., Bagdanov, A.D., van de Weijer, J.: Class-incremental learning: survey and performance evaluation. arXiv preprint arXiv:2010.15277 (2020)
32. Mazumder, P., Singh, P., Rai, P.: Few-shot lifelong learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 2337–2345 (2021)
33. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. Psychology of learning and motivation **24**, 109–165 (1989)
34. Mermillod, M., Bugaiska, A., Bonin, P.: The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. Frontiers in psychology **4**, 504 (2013)
35. Mirzadeh, S.I., Chaudhry, A., Yin, D., Nguyen, T., Pascanu, R., Gorur, D., Farajtabar, M.: Architecture matters in continual learning. arXiv preprint arXiv:2202.00275 (2022)
36. Mocanu, D.C., Mocanu, E., Stone, P., Nguyen, P.H., Gibescu, M., Liotta, A.: Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nature communications **9**(1), 1–12 (2018)
37. Mocanu, D.C., Vega, M.T., Eaton, E., Stone, P., Liotta, A.: Online contrastive divergence with generative replay: Experience replay without storing data. arXiv preprint arXiv:1610.05555 (2016)
38. Özdenizci, O., Legenstein, R.: Training adversarially robust sparse networks via bayesian connectivity sampling. In: International Conference on Machine Learning. pp. 8314–8324. PMLR (2021)
39. Raihan, M.A., Aamodt, T.: Sparse weight activation training. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 15625–15638. Curran Associates, Inc. (2020)
40. Ramasesh, V.V., Dyer, E., Raghu, M.: Anatomy of catastrophic forgetting: Hidden representations and task semantics. In: International Conference on Learning Representations (2020)
41. Rebuffi, S.A., Kolesnikov, A., Sperl, G., 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)

42. Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., Tesauro, G.: Learning to learn without forgetting by maximizing transfer and minimizing interference. In: International Conference on Learning Representations (2018)
43. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. arXiv preprint arXiv:1606.04671 (2016)
44. Serra, J., Suris, D., Miron, M., Karatzoglou, A.: Overcoming catastrophic forgetting with hard attention to the task. In: International Conference on Machine Learning. pp. 4548–4557. PMLR (2018)
45. Shin, H., Lee, J.K., Kim, J., Kim, J.: Continual learning with deep generative replay. In: Advances in Neural Information Processing Systems. pp. 2990–2999 (2017)
46. Sokar, G., Mocanu, D.C., Pechenizkiy, M.: Learning invariant representation for continual learning. In: Meta-Learning for Computer Vision Workshop at the 35th AAAI Conference on Artificial Intelligence (AAAI-21) (2021)
47. Sokar, G., Mocanu, D.C., Pechenizkiy, M.: Self-attention meta-learner for continual learning. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1658–1660 (2021)
48. Sokar, G., Mocanu, D.C., Pechenizkiy, M.: Spacenet: Make free space for continual learning. Neurocomputing **439**, 1–11 (2021)
49. Sokar, G., Mocanu, E., Mocanu, D.C., Pechenizkiy, M., Stone, P.: Dynamic sparse training for deep reinforcement learning. In: International Joint Conference on Artificial Intelligence (2022)
50. van de Ven, G.M., Siegelmann, H.T., Tolias, A.S.: Brain-inspired replay for continual learning with artificial neural networks. Nature communications **11**(1), 1–14 (2020)
51. Veniat, T., Denoyer, L., Ranzato, M.: Efficient continual learning with modular networks and task-driven priors. In: International Conference on Learning Representations (2021)
52. Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., Farhadi, A.: Supermasks in superposition. Advances in Neural Information Processing Systems **33** (2020)
53. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)
54. Yoon, J., Kim, S., Yang, E., Hwang, S.J.: Scalable and order-robust continual learning with additive parameter decomposition. In: International Conference on Learning Representations (2019)
55. Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. In: International Conference on Learning Representations (2018)
56. Zenke, F., Poole, B., Ganguli, S.: Continual learning through synaptic intelligence. In: International Conference on Machine Learning. pp. 3987–3995. PMLR (2017)
57. Zhu, H., Jin, Y.: Multi-objective evolutionary federated learning. IEEE Transactions on Neural Networks and Learning Systems **31**(4), 1310–1322 (2019)