Addressing the Stability-Plasticity Dilemma via Knowledge-Aware Continual Learning

Ghada Sokar \(^1\) Decebal Constantin Mocanu \(^2\) V Mykola Pechenizkiy \(^1\)

Abstract

Current methods in continual learning (CL) tend to focus on alleviating catastrophic forgetting of previous tasks. This hinders balancing other CL desiderata such as forward transfer, and memory and computation efficiency. CL desiderata become much more competing when a model operates on class-Incremental Learning (class-IL) without accessing past data since it is prone to ambiguities between old and new classes. In this paper, we present Knowledge-Aware coNtinual learner (KAN) that attempts to study the stability-plasticity dilemma to balance CL desiderata in class-IL. In particular, KAN is a new task-specific components method based on dynamic sparse training that introduces reusability and selective transfer of past knowledge in this class of methods. KAN selectively reuses relevant knowledge while addressing class ambiguity, preserves old knowledge, and utilizes the model capacity efficiently. Experiments show the effectiveness of KAN in providing models with multiple CL desirable properties, outperforming state-of-the-art methods on various challenging benchmarks.

1. Introduction

Continual learning (CL) aims to build intelligent agents based on deep neural networks that can learn tasks sequentially and use past knowledge in future learning without forgetting. The main challenge in this paradigm is the stability-plasticity dilemma (Mermillod et al., 2013). Optimizing all model weights on a new task (highest plasticity) causes catastrophic forgetting of past tasks (McCloskey & Cohen, 1989). While fixing all weights (highest stability) hinders learning new tasks. Finding the right balance between stability and plasticity is challenging. The challenge becomes more difficult when other CL requirements are considered.

CL desiderata include: (1) using fixed-capacity models, an efficient CL approach could not add a new model for each task, (2) ability to learn from scratch as we cannot always assume the availability of large datasets for pretraining (Yoon et al., 2019), (3) forward transfer, exploiting past relevant knowledge in future learning, (4) reducing forgetting without access to past data (replay-free), (5) memory and computation efficiency, and (6) not relying on task labels at inference (class-Incremental Learning (class-IL)). See (Díaz-Rodríguez et al., 2018; Hadsell et al., 2020) for full details on these requirements. Balancing these desiderata is challenging. For instance, using fixed-capacity models without replaying old data increases the difficulty of achieving good performance on both past and new tasks due to the stability-plasticity dilemma. This sharpens the community’s focus on forgetting at the expense of other desiderata (Wolczyk et al., 2021). Recently, attention to other desiderata, particularly forward transfer, has increased (Wolczyk et al., 2021; Ke et al., 2020; Veniat et al., 2021; Lee et al., 2021). Current efforts focus on task-Incremental Learning (task-IL), where each task is a separate classification problem, relying on task labels at inference (i.e., relaxing requirement (6)). However, addressing the more realistic class-IL with other desiderata has not been tackled widely yet. Using a unified classifier for all classes raises additional challenges that make CL desiderata more competitive, as we will discuss next.

In this work, we aim to balance the above-mentioned CL desiderata in class-IL. We focus on task-specific component methods (Golkar et al., 2019; Mallya & Lazebnik, 2018; Sokar et al., 2021c) since they offer more flexibility to address the stability-plasticity dilemma by using different components (connections/neurons) for each task. Typically, the components of a new task are flexible to learn, while the components of past tasks are kept fixed. Yet, the focus of current methods is to address forgetting, which leads to three main limitations that hinder satisfying the other desiderata: (a) adding redundant components, (b) criteria of component allocation do not promote forward transfer, and (c) the particular challenges of class-IL are overlooked.

(a) Adding redundant components New components are added for every new task. However, reusing relevant past components would increase memory and computation effi-
Addressing the Stability-Plasticity Dilemma via Knowledge-Aware Continual Learning

Figure 1. (a) By reusing some of Task A’s components during learning Task B, we can achieve similar performance as adding new components while having less memory and computational costs. (b) The initial topology allocated for a new task affects the performance. (c) Performing the same model-altering scheme in task-IL and class-IL leads to different performance. Details are in Appendix B.

We study the core of the stability-plasticity dilemma in class-IL. We address the following question: Which components are eligible to be added, updated, fixed, or reused when a CL model faces a new task to balance CL desiderata? With our proposed Knowledge-Aware coNtinual learner (KAN), we find that considering the semantic similarity between old and new classes is crucial in addressing this question. With awareness of existing knowledge, a CL model could identify relevant past knowledge, reuse it in future learning (forward transfer), and add the necessary components to capture new specific representation (resource and computation efficiency). It could also protect irrelevant knowledge and limit its transfer to a new task (mitigating forgetting). Our contributions are:

- We introduce reusability and selective forward transfer of past knowledge in the task-specific component strategy, which enables balancing CL desiderata.
- We address the challenges of the class-IL scenario: component-agnostic inference and class ambiguities.
- We illustrate that the initial sparse topology of each task highly affects forward and backward transfer.
- Experimental results show that KAN outperforms baseline methods on various benchmarks, including challenging cases where high similarity exists across tasks.

2. Related Work

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

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 capacity for new tasks (Rusu et al., 2016; Yoon et al., 2018) or use a fixed-capacity and each task is trained via a sparse sub-network within a model (Mallya & Lazebnik, 2018; Mallya et al., 2018; Yoon et al., 2019). These methods require task labels to specify the components of a task at inference. Few methods have been recently proposed for class-IL, using a fixed-capacity model. SupSup (Wortsman et al., 2020) learns a mask for each task over a randomly initialized fixed network. FSLL (Mazumder et al., 2021) addressed few-shot CL. It uses large data to train a dense model on the first task.
Then, a few unimportant weights are used to learn future tasks. SpaceNet (Sokar et al., 2021c) learns a sparse sub-network for each task from scratch using dynamic sparse training (Mocanu et al., 2018; Hoefler et al., 2021), where the weights and the sparse topology are jointly optimized.

(2) **Regularization-based.** A fixed-capacity model is used, and all weights are optimized for learning each task. Forgetting is addressed either by constraining changes in the important weights of past tasks (Zenke et al., 2017; Kirkpatrick et al., 2017; Aljundi et al., 2018) or via distillation loss (Li & Hoiem, 2017; Dhar et al., 2019). Previous studies (Kemker et al., 2018; Hsu et al., 2018; Farquhar & Gal, 2019; van de Ven & Tolias, 2018) showed the performance gap between class-IL and task-IL using this strategy.

**Replay-based** In this category, forgetting is addressed by replaying: (1) a subset of old samples (Rebuffi et al., 2017; Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2018a; Riemer et al., 2018; Chaudhry et al., 2018b; Bang et al., 2021), (2) pseudo-samples from a generative model (Mocanu et al., 2016; Shin et al., 2017; Sokar et al., 2021a), or (3) generative high-level features (Liu et al., 2020; van de Ven et al., 2020). These methods use a classification model and buffer to store old samples or a generative model to generate them.

Attention to task similarities and other CL properties has been recently increased. Most efforts are devoted to task-IL. In (Ramasesh et al., 2020), the analysis showed that higher layers are more prone to forgetting, and intermediate semantic similarity across tasks leads to maximal forgetting. SAM (Sokar et al., 2021b) meta-trains a self-attention mechanism for selective transfer in dense networks. CAT (Ke et al., 2020) 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 (Lee et al., 2021), an expectation-maximization method was proposed to select the shared or added layers to promote transfer in task-IL. Similarly, (Veniat et al., 2021) uses modular neural network architecture and search 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 (Chen et al., 2020), the lottery ticket hypothesis (Franke & Carbin, 2018) is studied for CL.

### 3. Network Structure Altering

A model is altered via some actions to address the stability-plasticity dilemma. Next, we will present commonly used and our proposed actions. A summary is also in Table 1.

#### 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 $W^l$, are added for each new task $t$. Each task has a sparse sub-network resulting from either pruning dense connections (Mallya & Lazebnik, 2018) (Figure 2b) or adding a sparse sub-network from scratch (Sokar et al., 2021c) (Figure 2c). The current practice is to randomly add new connections in each layer using unimportant components of past tasks; $W^l = \{W^l_j : 1 \leq l \leq L\}$ where $L$ is the number of layers. This will be addressed in Section 4 to enable selective transfer and reusability.
- **“Fix”**: Once a task has been learned, its connections $W^l$ are frozen (i.e., controls stability).
- **“Update”**: The newly added weights $W^l$ are flexible to learn (i.e., allows plasticity).

Note that regularization methods add dense and shared connections at time step $t = 0$ (Figure 2a). Each task updates all weights. Stability is controlled via regularization.
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 D). Hence, we believe that the following neuron-level actions are needed:

- **“Fix”**: After a task has been learned, its important neurons should be frozen to reduce the drift in its representation. Other neurons are “free” to be updated.
- **“Reuse”**: Reusing neurons that capture useful representation in learning future tasks. Namely, outgoing connections are allowed from relevant neurons even if they are fixed. Details are provided in the next section.

Most previous methods operate at the connection level except (Sokar et al., 2021c), in which important neurons of each task are fixed (Figure 2c). However, it does not reuse relevant neurons, limiting the utilization of past knowledge.

4. Knowledge-Aware Continual Learning

We consider the problem of learning a sequence of $T$ tasks using a fix-capacity model with a unified classifier. Each task $t$ brings new $C$ classes and has its data $D^t = \{D^t_c\}_{c=1}^C$, where $D^t_c$ is the data of class $c$ in task $t$. Once a task has been trained, its data is discarded. Our goal is to study how model components should be altered (added, updated, fixed, or reused) to address multiple desiredata jointly.

To balance CL desiderata, we propose three novel component-altering considerations: **First**, reusing some of the past relevant components during future learning. **Second**, the criteria for adding new connections consider past tasks to reduce forgetting and the new task to promote knowledge transfer and reusability. **Third**, addressing the trade-off between reusing relevant knowledge and class ambiguity. To conduct our study, we design Knowledge-Aware coNtinual learner (KAN), a new task-specific components model that incorporates these novel considerations.

### 4.1. Reusing Previous Components

Here, we ask the following question: “Is it always necessary to add new connections in each layer for each new task?” Typically, some of the previous classes are similar to the new classes, and some are dissimilar. If 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 start allocating new sparse connections 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 $W^t = \{W^t_l : l_{reuse} \leq l \leq L\}$. The topology from layer 1 up to but excluding layer $l_{reuse}$ remains unchanged.

The advantage of reusing previous components is two-fold: (1) reducing memory and computation costs and (2) utilizing 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). This could improve task performance (Section 5.2.3).

One may expect that the higher the similarity between old and new classes, the more layers could be reused (i.e., higher value for $l_{reuse}$). This could be true in task-IL, where the previous similar knowledge can be easily fully utilized in learning a new task without sacrificing the performance on old tasks (Figure 1c). However, in class-IL, the absence of task labels at inference raises a competition between forward transfer and forgetting avoidance. Reusing past knowledge up to the highest-level layer leads to ambiguity between old and new classes. To address this trade-off, we propose constraints on reusing previous components (Section 4.4).

Note that previous methods (Lomonaco & Maltoni, 2017; Hayes & Kanan, 2020; van de Ven et al., 2020; Sokar et al., 2021b) studied using a pre-trained fixed dense feature extractor that captures generic features for all tasks. This differs from the studied case here, in which each task learns a new sparse sub-network from scratch without relying on pre-trained models (Section 1). In this more challenging case, reusing past components in learning a new task requires the identification of neurons that enable the selective transfer of relevant past knowledge, as we will demonstrate next.

### 4.2. Adding New Connections

Starting from $l_{reuse}$, we add new sparse connections $W^t_l$ in each layer $l$. This raises the following question: “How to allocate the initial sparse topology for a new task?”

| METHOD                | ADD WEIGHTS | FIX OLD WEIGHTS | FIX NEURONS | REUSE PAST COMPONENTS | SELECTIVE TRANSFER |
|-----------------------|-------------|-----------------|-------------|-----------------------|-------------------|
| EWC (Kirkpatrick et al., 2017), MAS (Aljundi et al., 2018) | Dense       | ×               | ×           | ×                     | ×                 |
| PACKNET (Malaya & Lazebnik, 2018)                            | DENSE+FREE  | ×               | ×           | ×                     | ×                 |
| NOISENET (Sokar et al., 2021c)                               | MSE         | ×               | ×           | ×                     | ×                 |
| KAN (OURS)                                                    | MSE         | ×               | ×           | ×                     | ×                 |

Table 1. Actions used to alter the model components at the connection and neuron levels by different methods.
To obtain an efficient CL model, allocating a new topology should consider: (i) selective transfer of relevant knowledge to promote forward transfer and reusability and (ii) preserving old representation to avoid forgetting. To address each of these objectives, we select a set of candidate and free neurons in each layer to allocate new connections, as we will discuss next. Details are also 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 $R_l^c$ that has a high potential of being useful if they are “reused” in learning class $c$ in a new task $t$. Classes with semantic similarities are most likely to share similar representations (Appendix C). 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 $t$, $D_t^c$, to the trained model at time step $t-1$, $f^{t-1}(W^{t-1})$, and calculate the average activation $A_l^c$ in each layer $l$ as follows:

$$A_l^c(W^{t-1}) = \frac{1}{|D_t^c|} \sum_{i=1}^{|D_t^c|} a_i(x_i^{c,t})$$

(1)

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

$$R_l^c = \{ i | A_l^c \in \text{top-}\kappa(A_l^c) \}$$

(2)

where $\kappa$ denotes the desired number of candidate neurons, and $A_l^c$ is the average activation of neuron $i$ in layer $l$. Neuron activity shows its effectiveness in a separate line of research, pruning of neural networks (Hu et al., 2016; Luo et al., 2017; Dekhovich et al., 2021). It is used to estimate the importance of a neuron/connection for single task standard supervised learning. To assess our choice of this metric to identify the potential of a neuron in learning a new class, we compare against other two metrics: (1) Random where the candidate neurons are randomly chosen from all neurons in a layer and (2) Lowest where the candidate neurons are the neurons with the lowest activation (See Section 5.2).

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

**Identify Free Neurons.** To preserve previous representation while allowing reusability of neurons, we allow outgoing connections from “fixed” neurons that might be selected as candidates, but we do not allow incoming connections to these neurons. The outgoing connections from fixed neurons in layer $l$ can be connected to “free” neurons in layer $l + 1$, but cannot be connected to fixed neurons. Hence, in each layer, we select a subset of the free neurons for each task, denoted as $S_l^{Free}$. For these neurons, incoming and outgoing connections are allowed to be allocated and used to capture the specific representation for a new class.

**Allocation.** The neurons used to allocate new sparse weights $W_l^c$ between layers $l - 1$ and $l$ for a class $c$ are:

$$h_{l_{alloc}}^c = \{ R_{l-1}^c \cup S_{l-1}^{Free} \}$$

(3)

**4.3. Fixed and Updated Components**

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 proposed approach in (Sokar et al., 2021c) to train a sparse topology and identify the important neurons (Appendix E). In short, a sparse topology is optimized by a dynamic sparse training approach to produce sparse representation for each task. 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 $g_t$ through the neurons of layer $l$ is:

$$g_t = g_t \odot (1 - h_l^{fix})$$

(4)

where $h_l^{fix}$ is a binary vector where a value of one represents a fixed neuron in layer $l$. This allows us 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 topological allocation, we protect their knowledge. Thus, selective transfer enables forward transfer while preserving similar and dissimilar knowledge.

**4.4. Addressing Class Ambiguities**

One of the desirable properties of a CL agent is to exploit past knowledge in future learning. Despite that similarity
across tasks is a favor for forward transfer and reusability, it increases the challenge to balance CL desiderata when old data is inaccessible. Namely, when similarity exists, the model is more prone to ambiguities between old and new classes. This is because a unified classifier is used to distinguish similar classes that are not presented together in the same task. The balance between reusing old knowledge and maintaining the performance of old tasks is challenging.

To address this challenge, we propose three constraints for reusability. These constraints aim to (1) allow a new task to learn its specific representation and (2) increase the decision margin between classes (see Section 5.2.2 for analysis).

**Allow tasks to learn specific representation.** Learning specific representation reduces ambiguity between similar classes. To this end, we add two constraints for reusability. First, reusability of past components could be till at least one layer before the output one (i.e., \( l_{\text{reuse}} \in [0, L - 2] \)). Hence, new sparse connections are added for a new task, at least in the last two layers. Second, the output connections are allocated using the free neurons only (i.e., no candidate neurons are used: \( h_{L-1}^{\text{alloc}} = S_{L-1}^{\text{Free}} \)). By adding new connections and learning using free neurons in the highest level layer, a new task could capture its specific representation.

**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 (Li et al., 2020). To this end, once a task has been trained, we force all neurons allocated for this task in the last layer, \( h_{L-1}^{\text{alloc}} \), to be fixed and not reusable.

5. Experiments and Results

**Baselines.** Our study addresses the stability-plasticity dilemma in the replay-free setting using a fixed-capacity model. Therefore, we compare with several representative regularization methods that use dense fixed-capacity, EWC (Kirkpatrick et al., 2017), MAS (Aljundi et al., 2018), and LWF (Li & Hoiem, 2017). In addition, we compare with task-specific components methods that use sparse sub-networks within a fixed-capacity model, PackNet (Mallya & Lazebnik, 2018) and SpaceNet (Sokar et al., 2021c).

**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 (Krizhevsky et al., 2012).

**Similar Sequences** To assess replay-free methods under more challenging conditions, we design two new benchmarks with high semantic similarity across tasks. Under 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. The first benchmark, **sim-CIFAR10**, is constructed from CIFAR-10 by shuffling the order of classes to increase the similarity across tasks (Appendix A; Table 5). Same as split-CIFAR10, it consists of 5 tasks. The second benchmark, **sim-CIFAR100**, is constructed from the CIFAR-100 dataset (Krizhevsky et al., 2009). 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 to increase the similarity across tasks (Appendix A; Table 6).

**Mix datasets** The considered data sets are CIFAR10 (Krizhevsky et al., 2009), MNIST (LeCun, 1998), NotMNIST (Bulatov, 2011), and FashionMNIST (Xiao et al., 2017). 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.** We follow (Serra et al., 2018; Ke et al., 2020; Veniat et al., 2021) to use an AlexNet-like architecture (Krizhevsky et al., 2012) that is trained from scratch using stochastic gradient descent (SGD). We start reusing full layers in learning 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. For all benchmarks, we use \( l_{\text{reuse}} \) of 3. Earlier tasks before reusing add sparse connections in each layer using the free neurons.

**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) (Lopez-Paz & Ranzato, 2017), which measures the influence of learning a new task on past tasks (large negative BWT means forgetting), (3) **Floating-point operations (FLOPs)**, which measure the computation costs, and (4) **Model size (#params)**, which is the number of model parameters.

More details on the experimental setup are in Appendix A.

5.1. Results

Table 2 shows the performance on each benchmark. KAN 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 with 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, KAN 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 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 the high performance achieved in task-IL (Appendix D). Besides the additional memory and computational costs of pruning and retraining dense models, the performance is lower than other task-specific components methods. Operating at the connection and neuron levels, as in SpaceNet and KAN, enables much 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).

KAN 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. KAN exploits the similarity across tasks in learning future tasks while addressing class ambiguities. Moreover, the superior performance is accompanied by using a smaller memory and less computation cost than all other baselines.

5.2. Analysis

5.2.1. Effect of Selective Transfer

To measure the impact of selective transfer and the initially allocated topology on forward and backward transfer in KAN, we compare our selection strategy for the candidate neurons to two other potential strategies discussed in Section 4.2: Random and Lowest. To reveal the effect of select transfer on the performance of a new task, we also report the learning accuracy (LA) (Riemer et al., 2018), 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 past components in its learning onwards. Figure 3 shows the results on Mix datasets and sim-CIFAR10 benchmarks. 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 for these tasks). As illustrated in the figure, using the relevant neurons with the highest activation to allocate the initial topology for a new task leads to higher LA on new tasks and lower negative BWT on past tasks than using random neurons. KAN 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 topology in performance, although topological optimization occurs during training.

5.2.2. Class Ambiguities

We performed an ablation study to assess each of our proposed contributions in addressing class ambiguities in class-IL (Section 4.4). 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 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 KAN without using orthogonal weights in the output layer “\( \text{w/o orth } W_o \)”. We obtain this baseline by fixing part of the neurons in the last layer instead of fixing all neurons. We use the same fraction

| METHOD          | ACC (%) | SPLIT-CIFAR10 FLOPs (↓) | #PARAMS (↓) | EWC  | 15.73±0.81 | -74.50±5.10 | 23471808 | 54.63±1.93 | -31.33±2.82 | 5.60±0.15 | 23520960 |
|-----------------|---------|-------------------------|-------------|------|------------|------------|---------|-----------|-------------|----------|-----------|
| LWF             | 14.40±0.69 | -52.86±1.12             | 23471808    |      | 40.60±3.25 | -60.46±3.54 | 5.60±0.15 | 23520960 | 5.60±0.15 |
| MAS             | 13.50±0.66 | -81.70±1.02             | 23471808    |      | 56.86±1.81 | -26.83±3.44 | 5.60±0.15 | 23520960 | 5.60±0.15 |
| PackNet         | 10.12±2.55 | -20.35±3.63             | 18777440    |      | 16.61±2.35 | -18.58±1.38 | 29.40±0.15 | 18516768 | 5.60±0.15 |
| SpaceNet        | 32.86±2.73 | -36.29±2.78             | 2112512     |      | 56.25±1.69 | -30.02±1.79 | 0.30±0.215 | 1269200 | 5.60±0.15 |
| KAN (ours)      | 33.74±2.18 | -21.26±2.21             | 2086160     |      | 59.02±1.76 | -25.91±1.51 | 0.28±0.15 | 1197308 | 5.60±0.15 |

Table 2. Evaluation results on four CL benchmarks in the replay-free class-IL setting with fixed-capacity models.
used for other fully connected layers (Appendix A). Table 3 shows the results. Having a large decision margin via the orthogonal output weights increases the performance of the model. We observe that the difference between KAN 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 evaluate the effect of allocating the output weights using the free neurons only, we add another baseline that uses free $S_{L-1}^{\text{free}}$ and candidate $R_{L-1}$ neurons in allocation. We denote this baseline as “w/$R_{L-1}$”. As shown in Table 3, using free neurons only allows learning specific representation of each task which decreases the ambiguities across tasks. The performance in class-IL is improved by 2.19% and 4.93% on Mix datasets and sim-CIFAR10, 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., $l_{\text{reuse}} = L - 1$). We denoted this baseline as “w/$l_{\text{reuse}} = L - 1$”. 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.

5.2.3. Utilizing the fixed capacity model

Using fixed-capacity models constrains the sparsity level of the topology used for each task. Very high sparse models might limit the performance of complex tasks. Here, we illustrate that reusing previous components in learning similar tasks allows dissimilar ones to have a higher density. We performed this analysis on Mix datasets. To this end, for the first four dissimilar tasks in this benchmark, we increased the density level for the layers that will be reused in the second four tasks (i.e., $l < l_{\text{reuse}}$). The new density for each task in this baseline is 12%, while the one used to obtain the main results in Section 5.1 is 5% (Appendix A). Figure 4 shows the ACC and LA for each case. Using higher density for the dissimilar tasks that are not encountered before increases their performance. The achieved ACC and LA across all time steps is higher than using a density of 5% for each task. Without reusability, as in SpaceNet, a model could not fit all tasks with the higher density. This reveals the importance of utilizing the available capacity efficiently.

6. Conclusion

Addressing the stability-plasticity dilemma while balancing CL desiderata is a challenging task. We showed that the challenge increases when a CL model operates in class-IL. With our proposed task-specific component method, KAN, we find that altering the model components based on exploiting past knowledge helps in achieving multiple desirable properties of a CL model. Critically, the initial topology allocated for a new task (i.e., the distribution of connections) affects the forward and backward transfer. Selective transfer of relevant knowledge while preserving it balances between forward transfer and forgetting avoidance. 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

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In Proceedings of the European Conference on Computer Vision (ECCV), pp. 139–154, 2018.

Atashgahi, Z., Sokar, G., van der Lee, T., Mocanu, E., Mocanu, D. C., Veldhuis, R., and Pechenizkiy, M. Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. arXiv preprint arXiv:2012.00560, 2020.

Bang, J., Kim, H., Yoo, Y., Ha, J.-W., and 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.

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. Deep rewiring: Training very sparse deep networks. In International Conference on Learning Representations, 2018.

Bulatov, Y. Notmnist dataset. Technical report, 2011. URL http://yaroslavvb.blogspot.it/2011/09/notmnist-dataset.html.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In Proceedings of the European Conference on Computer Vision (ECCV), pp. 532–547, 2018a.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-gem. In International Conference on Learning Representations, 2018b.

Chen, T., Zhang, Z., Liu, S., Chang, S., and Wang, Z. Long live the lottery: The existence of winning tickets in lifelong learning. In International Conference on Learning Representations, 2020.

Dekhovich, A., Tax, D. M., Sluiter, M. H., and Bessa, M. A. Neural network relief: a pruning algorithm based on neural activity. arXiv preprint arXiv:2109.10795, 2021.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and 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.

Dettmers, T. and Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance. arXiv preprint arXiv:1907.04840, 2019.

Dhar, P., Singh, R. V., Peng, K.-C., Wu, Z., and Chellappa, R. Learning without memorizing. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5138–5146, 2019.

Díaz-Rodríguez, N., Lomonaco, V., Filliat, D., and Maltoni, D. Don’t forget, there is more than forgetting: new metrics for continual learning. In Workshop on Continual Learning, NeurIPS 2018 (Neural Information Processing Systems, 2018.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In International Conference on Machine Learning, pp. 2943–2952. PMLR, 2020.

Farquhar, S. and Gal, Y. Towards robust evaluations of continual learning. In Privacy in Machine Learning and Artificial Intelligence workshop, ICML, jun 2019.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In International Conference on Learning Representations, 2018.

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

Hadsell, R., Rao, D., Rusu, A. A., and Pascanu, R. Embracing change: Continual learning in deep neural networks. Trends in cognitive sciences, 2020.

Hayes, T. L. and Kanan, C. Lifelong machine learning with deep streaming linear discriminant analysis. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 220–221, 2020.

Hoeffer, T., Alistarh, D., Ben-Nun, T., Dryden, N., and 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. URL http://jmlr.org/papers/v22/21-0366.html.

Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., and Kira, Z. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. In NeurIPS Continual learning Workshop, 2018.

Hu, H., Peng, R., Tai, Y.-W., and Tang, C.-K. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pp. 448–456. PMLR, 2015.

Jayakumar, S., Pascanu, R., Rae, J., Osindero, S., and Elsen, E. Top-kast: Top-k always sparse training. Advances in Neural Information Processing Systems, 33:20744–20754, 2020.
Addressing the Stability-Plasticity Dilemma via Knowledge-Aware Continual Learning

Ke, Z., Liu, B., and Huang, X. Continual learning of a mixed sequence of similar and dissimilar tasks. Advances in Neural Information Processing Systems, 33, 2020.

Kemker, R., McClure, M., Abitino, A., Hayes, T. L., and Kanan, C. Measuring catastrophic forgetting in neural networks. In Thirty-second AAAI conference on artificial intelligence, 2018.

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. Technical report, Citeseer, 2009.

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

LeCun, Y. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998.

Lee, S., Behpour, S., and 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.

Li, X., Wu, C., Menta, M., Herranz, L., Raducanu, B., Bagdanov, A. D., Jui, S., and van de Weijer, J. Generative feature replay for class-incremental learning. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 915–924. IEEE Computer Society, 2020.

Lomonaco, V. and Maltoni, D. Core50: a new dataset and benchmark for continuous object recognition. In Conference on Robot Learning, pp. 17–26. PMLR, 2017.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 6470–6479, 2017.

Luo, J.-H., Wu, J., and 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.

Mallya, A. and 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.

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), pp. 67–82, 2018.

Masana, M., Liu, X., Twardowski, B., Menta, M., Bagdanov, A. D., and van de Weijer, J. Class-incremental learning: survey and performance evaluation. arXiv preprint arXiv:2010.15277, 2020.

Mazumder, P., Singh, P., and Rai, P. Few-shot lifelong learning. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pp. 2337–2345, 2021.

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

Mermillod, M., Bugaiska, A., and Bonin, P. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. Frontiers in psychology, 4:504, 2013.

Mocanu, D. C., Vega, M. T., Eaton, E., Stone, P., and Liotta, A. Online contrastive divergence with generative replay: Experience replay without storing data. arXiv preprint arXiv:1610.05555, 2016.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity.
Addressing the Stability-Plasticity Dilemma via Knowledge-Aware Continual Learning

inspired by network science. *Nature communications*, 9 (1):1–12, 2018.

Mostafa, H. and Wang, X. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655. PMLR, 2019.

Özdenizci, O. and Legenstein, R. Training adversarially robust sparse networks via bayesian connectivity sampling. In *International Conference on Machine Learning*, pp. 8314–8324. PMLR, 2021.

Raihan, M. A. and Aamodt, T. Sparse weight activation training. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 15625–15638. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/b44182379bf9fae976e6ae5996e13cd8-Paper.pdf.

Ramasesh, V. V., Dyer, E., and Raghu, M. Anatomy of catastrophic forgetting: Hidden representations and task semantics. In *International Conference on Learning Representations*, 2020.

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

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., and Tesauro, G. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2018.

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.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

Sokar, G., Mocanu, D. C., and 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)*, 2021a.

Sokar, G., Mocanu, D. C., and 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, 2021b.

Sokar, G., Mocanu, D. C., and Pechenizkiy, M. Spacenets: Make free space for continual learning. *Neurocomputing*, 439:1–11, 2021c.

van de Ven, G. M. and Tolias, A. S. Three scenarios for continual learning. In *Continual Learning Workshop NeurIPS*, 2018.

van de Ven, G. M., Siegelmann, H. T., and Tolias, A. S. Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11(1):1–14, 2020.

Veniat, T., Denoyer, L., and Ranzato, M. Efficient continual learning with modular networks and task-driven priors. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=EKV158tSfwv.

Wolczky, M., Zajkac, M., Pascanu, R., Kuci’nski, L., and Milo’s, P. Continual world: A robotic benchmark for continual reinforcement learning. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=5qsptDcsdEj.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., and Farhadi, A. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33, 2020.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

Yoon, J., Kim, S., Yang, E., and Hwang, S. J. Scalable and order-robust continual learning with additive parameter decomposition. In *International Conference on Learning Representations*, 2019.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995. PMLR, 2017.

Zhu, H. and Jin, Y. Multi-objective evolutionary federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(4):1310–1322, 2019.
A. Experimental Settings

A.1. Benchmarks

CIFAR-10 (Krizhevsky et al., 2009) is a well-known dataset for classification tasks. It contains tiny natural images of size \(32 \times 32\). It consists of 10 classes and has 6000 images per class (6000 training + 1000 test). We use this dataset to construct two benchmarks: split-CIFAR10 and sim-CIFAR10. Split-CIFAR10 is the standard benchmark for CL. It consists of 5 tasks, and each task has two classes. The benchmark has the typical class order of CIFAR10 (Table 4). To assess the performance of a model under more challenging conditions, we construct sim-CIFAR10 with higher similarity between classes across tasks. This is obtained by shuffling the class order, as shown in Table 5.

CIFAR-100 (Krizhevsky et al., 2009) is a more complex dataset with 100 classes and fewer samples for each. It has 600 images per class (500 train + 100 test). The images have the same size as those in CIFAR10. Classes within the same superclass in this dataset have semantic similarities. We exploit this property to construct a new benchmark with a high similarity level across tasks. We called this benchmark sim-CIFAR100. We choose eight superclasses and two classes from each superclass. We distribute the classes from the same superclass in different tasks, constructing 8 tasks. The class order and the superclasses used in each task are illustrated in Table 6.

MNIST dataset (LeCun, 1998) contains grayscale images of size \(28 \times 28\), representing hand-written digits from 0 to 9. It has 60,000 training images and 10,000 test images.

Fashion-MNIST dataset (Xiao et al., 2017) is more complex than MNIST. The images show individual articles of clothing. It has the same sample size and structure of training and test sets as MNIST.

NotMNIST dataset (Bulatov, 2011) is similar to MNIST. It has 10 classes, with letters A-J taken from different fonts.

Following (Serra et al., 2018; Veniat et al., 2021), we used CIFAR10, MNIST, Fashion MNIST, and NotMNIST datasets to construct a sequence of tasks from different domains, named as Mix datasets. We split each dataset into two parts of five classes each. We organize the tasks such that the model sees all dissimilar tasks before it encounters new tasks similar to the previously seen ones. Table 7 shows the details of the task order and the classes in each task.

A.2. Network architecture

We follow (Serra et al., 2018; Ke et al., 2020) to use an AlexNet-like architecture (Krizhevsky et al., 2012). The details of the network architecture are in Table 8. For sim-CIFR100, we replace the dropout layers with batch normalization (Ioffe & Szegedy, 2015) since it was more effective in reducing the overfitting to the small number of samples per class in this dataset. Moreover, for regularization-based methods, batch normalization achieves higher accuracy than dropout layers. Hence, we use batch normalization for these methods in all benchmarks except for Mix datasets since the tasks come from different domains.
A.3. Implementation Details

All input images are resized to 32 × 32. Gray-scale images are converted to 3 channels. The network is trained using stochastic gradient descent with a batch size of 64 and a learning rate of 0.1. Each task is trained for 40 epochs. The hyperparameters are selected using a random search.

We use $l_{\text{reuse}}$ of 3 for all benchmarks (i.e., the convolution layers are reused in learning future tasks). We start reusing full layers in learning future tasks after acquiring similar tasks. Therefore, for mix datasets and sim-CIFAR100 benchmarks, we start reusing previous components from task 5, while for split-CIFAR10 and sim-CIFAR10, we start from task 3. To allocate a new topology, a set of neurons is selected in each layer $h_l^{\text{alloc}}$. For all benchmarks, 70% of $h_l^{\text{alloc}}$ are free neurons $h_l^{\text{free}}$ and 30% are candidates $R_l^C$.

Table 9 shows the percentage of allocated neurons in each layer, the percentage of fixed neurons after training, and the density level of the connections allocated for all benchmarks. Note the density level is reported with respect to the allocated number of neurons.

For SpaceNet, the percentage of $h_l^{\text{alloc}}$, fixed neurons, and density levels are the same as the ones used for KAN. Connections are allocated in each layer for each task using the free neurons.

For regularization-based methods, EWC (Serra et al., 2018), MAS (Aljundi et al., 2018), and LWF (Li & Hoiem, 2017), we used the public code from (Masana et al., 2020) to produce results on the studied benchmarks. We use a regularization factor of 5000 and 1 for EWC and MAS, respectively.

For PackNet, we use the official code from (Mallya & Lazebnik, 2018). Note that PackNet was originally designed for task-IL, as discussed in Section 2. To adapt PackNet to class-IL, we use all the learned connections during inference without masks. A dense model is trained from scratch on the CL tasks. After learning each task, a percentage of unimportant weights is pruned. We select this percentage such that all tasks have the same sparsity level. Each task is trained for 40 epochs, and another 20 epochs are performed after pruning to restore the performance. The sparsity level used for each task is 10% for mix datasets and sim-CIFAR100 benchmarks. For split-CIFAR10 and sim-CIFAR10, a sparsity level of 20% is used. We tried using a higher sparsity level for packNet. However, this results in lower performance.

### A.4. Evaluation Metrics

#### Average Accuracy (ACC)

The average accuracy after a model has been trained sequentially till task $T$.

$$ ACC = \frac{1}{T} \sum_{i=1}^{T} a_{T,i}, $$

where $a_{j,i}$ is the accuracy on task $i$ after learning the $j$-th task in the sequence, and $T$ is the total number of seen tasks.

#### Backward transfer (BWT) (Lopez-Paz & Ranzato, 2017)

This metric measures the influence of learning new tasks on the performance of previous tasks. Formally BWT is calculated as follows:

$$ BWT = \frac{1}{T-1} \sum_{i=1}^{T-1} a_{T,i} - a_{i,i}. $$

Larger negative backward transfer indicates forgetting.

#### Learning Accuracy (LA) (Riemer et al., 2018)

This metric measures the average accuracy for each task directly after it is learned as follows:

$$ LA = \frac{1}{T} \sum_{i=1}^{T} a_{i,i}. $$

#### Model size (# param)

This metric estimates the memory cost consumed by a CL model. The network size is estimated by the summation of the number of connections allocated in its layers as follows:

$$ \#\text{params} = \sum_{i=1}^{L} \| \mathbf{W}_i \|_0, $$

where $\mathbf{W}_i$ is the actual weights used in layer $l$ after the model learns all tasks, $\| \cdot \|_0$ is the standard $L_0$ norm, and $L$ is the number of layers in a model. For sparse networks, $\| \mathbf{W}_i \|_0$ is controlled by the sparsity level of each task.

#### Floating-point operations (FLOPs)

This metric estimates the computational cost of a method by calculating how many FLOPs are required for training. We follow the method described in (Evci et al., 2020) to calculate the FLOPs. The FLOPs are calculated with the total number of multiplications and additions layer by layer in the network that occurs during forward and backward passes.

Let $f_D$ be the number of FLOPs required to train a dense model on the sequence of tasks. The FLOPs required to
train a model with sparse subnetworks is \( f_s \approx (1 - s) \times f_D \), where \( s \) is the sparsity level of the model after learning all tasks. The FLOPs of task \( t \) using PackNet that trains dense connections and prunes the model after convergence is \( f_D \times d_t + f_{tune}^t \), where \( d_t \) is the density of the trained connections at time \( t \) and \( f_{tune}^t \) is the number of FLOPs required to finetune each task after pruning. Note that some regularization methods, such as EWC (Kirkpatrick et al., 2017), require additional FLOPs to calculate the regularization loss. Here, we omit this cost, focusing on the computational costs resulting from training either dense or sparse networks.

B. Details of Illustrative Experiments

In this appendix, we give the details of the illustrative experiments in Section 1. The experiments are performed on two tasks, A and B, constructed from CIFAR10. Task A has two classes, \{cat, car\}, while Task B has classes of \{dog, truck\}. We use the same experimental settings of sim-CIFAR10 and network architecture described in Appendix A.

Analysis 1: Addition of new components versus reusing existing ones. In this experiment, sparse connections are allocated for Task A. Two baselines are evaluated. One baseline allocates new sparse connections in each layer for Task B (i.e., applying the SpaceNet method). The other baseline represents KAN. It reuses the previous components and starts allocating new connections from \( l_{reuse} \). The results of this analysis are illustrated in Figure 1a.

Analysis 2: The effect of the initial topology in performance. Figure 1b shows the performance using KAN (“Topology 1”), and the lowest baseline described in Section 4 (“Topology 2”) for allocating the sparse topology of Task B. Different initial typologies leads to different behavior and could either balance or be biased toward one of CL requirements.

Analysis 3: Using the same altering of the model components in task-IL and class-IL. Figure 1a shows the performance when all the learned components of Task A are reused in learning Task B (i.e., \( l_{reuse} = L - 1 \)).

C. Representation of Similar and Dissimilar Tasks

In this appendix, we visualize the representation of two sequences: one contains similar tasks while the other sequence contains dissimilar tasks. The similar sequence has two tasks from CIFAR10 where Task 1 has the two classes of \{car, cat\} while Task 2 has the two classes of \{dog, truck\}. The dissimilar sequence has the same classes with different order where Task 1 is \{car, truck\} and Task 2 is \{cat, dog\}. We used the smaller architecture from (Sokar et al., 2021c) for visualization purposes.
Addressing the Stability-Plasticity Dilemma via Knowledge-Aware Continual Learning

Table 10. Performance of task-specific component methods in task-IL and class-IL on sim-CIFAR10 and Mix datasets.

| Strategy                        | Method     | sim-CIFAR10 Task-IL | sim-CIFAR10 Class-IL | Mix datasets Task-IL | Mix datasets Class-IL |
|---------------------------------|------------|---------------------|----------------------|----------------------|-----------------------|
| Connection level                | PackNet    | 89.33±0.02          | 32.46±1.22           | 96.52±0.05           | 16.61±2.35            |
| Connection and neuron levels    | KAN        | 94.44±0.28          | 45.23±2.14           | 93.41±0.26           | 59.01±1.76            |

Figure 5 shows the representations of a subset of the neurons in the last hidden layer for each sequence. We average the activation of the model trained on Task 1 over all samples of each class in Task 1 and Task 2. As illustrated in Figure 5a, there are high representational similarities between each class in Task 1 and the corresponding class that shares some semantic similarity in Task 2. On the other hand, if the tasks are dissimilar (Figure 5b), the new task has a different representation from the previous one.

D. Connection Level versus Neuron Level Altering

Section 5.1 showed that methods that alter the model components based on the connection and neuron levels are more efficient than the ones that alter at the connection level only (i.e., PackNet). To reveal that altering at the two levels is more necessitous in class-IL, we illustrate the performance of PackNet in task-IL. Table 10 shows the results on sim-CIFAR10 and Mix datasets.

PackNet achieves a good performance in task-IL despite the low performance achieved in class-IL, especially on the Mix datasets benchmark. This illustrates the challenges that arise from operating in the class-IL setting without access to task labels at inference. Reducing the interference between representations is more effective in this setting.

KAN achieves comparable performance to PackNet in task-IL with reusing previous components and selective transfer. More interestingly, KAN does not store a mask for each task to select the specific components at inference (i.e., component-agnostic inference). This leads to more efficient CL models with less memory and computation costs.

Note that the hyper-parameters (e.g., sparsity level, fixed neuron percentage, etc.) of KAN are not tuned for task-IL. We used the same hyperparameters as for class-IL. Higher performance might be achieved using hyperparameter optimization for task-IL. Yet, it is not the focus of this paper.

E. Training Sparse Topologies

We used the training procedure proposed in (Sokar et al., 2021c) to train a sparse topology allocated by KAN. The training approach is based on dynamic sparse training (DST) (Appendix F). The goal of this approach is to learn sparse representation for each task. During training, the sparse topology is optimized by redistributing the connections to group them in the most important neurons for the current task. The redistribution is performed through “drop-and-grow” cycles. A factor of the least important connections are dropped, and the same fraction is added between the most important neurons. The importance of a connection is estimated based on the gradient of the loss function. Please see the full details in Algorithm 3 from (Sokar et al., 2021c).

Identify neuron importance. Following SpaceNet (Sokar et al., 2021c), the importance of a neuron is calculated by the summation of the importance of its connected weights. Hence, we estimate the importance of a neuron by the summation of the importance of its outgoing connections.

F. Dynamic Sparse Training

Dynamic sparse training (DST) is a line of research that aims to reduce the computation and memory overhead of training dense neural networks by leveraging the redundancy in the parameters (i.e., being over-parametrized) (Denil et al., 2013). The basic idea is to train a sparse neural network from scratch and optimize the weights and topology simultaneously. Efforts in this line of research are devoted to single standard task supervised and unsupervised learning. The first work in this direction was proposed by (Mocanu et al., 2018). They showed that dynamic training of sparse networks from scratch achieves higher performance than dense models and static sparse neural networks trained from scratch. Recently, many interesting works have addressed this training strategy by proposing different algorithms for optimizing the topology during training (Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019; Evci et al., 2020; Bellec et al., 2018; Jayakumar et al., 2020; Liu et al., 2021b; Raihan & Aamodt, 2020).

In SpaceNet (Sokar et al., 2021c), a DST approach was proposed to optimize the topology to produce sparse representation for continual learning. DST demonstrated its success in other fields as well, such as feature selection (Atashgahi et al., 2020), ensembling (Liu et al., 2021a), federated learning (Zhu & Jin, 2019), text classification and language modeling tasks (Liu et al., 2021c), adversarial training (Özdenizci & Legenstein, 2021), and deep reinforcement learning (Sokar et al., 2021d).