SpaceNet

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

SpaceNet: Make Free Space For Continual Learning

Ghada Sokar[a,*], Decebal Constantin Mocanu[a,b] and Mykola Pechenizkiy[a]

[a]*Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands*
[b]*Faculty of Electrical Engineering, Mathematics, and Computer Science, University of Twente, The Netherlands*

ARTICLE INFO

ABSTRACT

The continual learning (CL) paradigm aims to enable neural networks to learn tasks continually in a sequential fashion. The fundamental challenge in this learning paradigm is catastrophic forgetting previously learned tasks when the model is optimized for a new task, especially when their data is not accessible. Current architectural-based methods aim at alleviating the catastrophic forgetting problem but at the expense of expanding the capacity of the model. Regularization-based methods maintain a fixed model capacity; however, previous studies showed the huge performance degradation of these methods when the task identity is not available during inference (e.g. class incremental learning scenario). In this work, we propose a novel architectural-based method referred as SpaceNet for class incremental learning scenario where we utilize the available fixed capacity of the model intelligently. SpaceNet trains sparse deep neural networks from scratch in an adaptive way that compresses the sparse connections of each task in a compact number of neurons. The adaptive training of the sparse connections results in sparse representations that reduce the interference between the tasks. Experimental results show the robustness of our proposed method against catastrophic forgetting old tasks and the efficiency of SpaceNet in utilizing the available capacity of the model, leaving space for more tasks to be learned. In particular, when SpaceNet is tested on the well-known benchmarks for CL: split MNIST, split Fashion-MNIST, and CIFAR-10/100, it outperforms regularization-based methods by a big performance gap. Moreover, it achieves better performance than architectural-based methods without model expansion and achieved comparable results with rehearsal-based methods, while offering a huge memory reduction.

1. Introduction

Deep neural networks (DNNs) have achieved outstanding performance in many computer vision and machine learning tasks [10, 42, 3, 16, 22, 9, 24]. However, this remarkable success is achieved in a static learning paradigm where the model is trained using large training data of a specific task and deployed for testing on data with similar distribution to the training data. This paradigm contradicts the real dynamic world environment which changes very rapidly. Standard retraining of the neural network model on new data leads to significant performance degradation on previously learned knowledge, a phenomenon known as catastrophic forgetting [28]. Continual learning, also called as lifelong learning, comes to address this dynamic learning paradigm. It aims at building neural network models capable of learning sequential tasks while accumulating and maintaining the knowledge from previous tasks without forgetting.

Several methods have been proposed to address the CL paradigm with a focus on alleviating the catastrophic forgetting. These methods generally follow three strategies: (1) rehearsal-based methods [37, 31] maintain the performance of previous tasks by replaying their data during learning new tasks, either the real data or generated one from generative models, (2) regularization-based methods [17, 41] aim at using a fixed model capacity and preserving the significant parameters for previous tasks by constraining their change, and (3) architectural-based methods [35, 40] dynamically expand the network capacity to reduce the interference between the new tasks and the previously learned ones.

Some other methods combine the rehearsal and regularization strategies [33, 34]. Rehearsal strategies tend to perform well but are not suitable to the situations where one can not access the data from previous tasks (e.g. due to data rights) or where there is computational or storage constraints hinder retaining the data from all tasks (e.g. resource-limited devices). Architectural strategies also achieve a good performance in the CL paradigm but at the expense of increasing the model capacity. Regularization strategies utilize a fixed capacity to learn all tasks. However, these methods suffer from significant performance degradation when applied in the class incremental learning (IL) scenario as argued by [15, 13, 6, 38]. Following the formulation from [13, 38], in the class IL scenario, the task identity is not available during inference and a unified classifier with a shared output layer (single-headed) is used for all classes. On the other hand, most of the current CL methods assume the availability of the task identity during inference and the model has a separate output layer for each task (multi-headed), a scenario named by [13, 38] as task incremental learning. Class IL scenario is more challenging; however, class incremental capabilities are crucial for many applications. For example, object recognition systems based on DNNs should be scalable to classify new classes while maintaining the performance of the old classes. Besides, it is more realistic to have all classes sharing the same single-headed output layer without the knowledge of the task identity after deployment.

In this paper, we propose a new architectural-based method for CL paradigm, which we name as SpaceNet. We address the scenario that is not largely explored: class IL in which

---

*Corresponding author
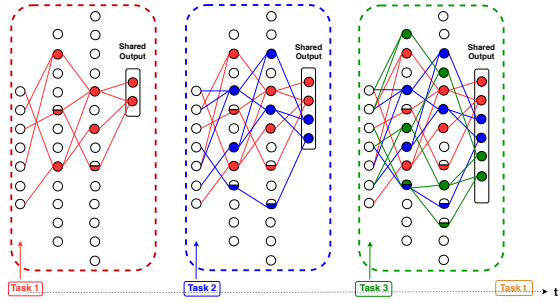
Figure 1: An overview of SpaceNet method for learning a sequence of tasks. All tasks have the same shared output layer. The fully filled circles represent the neurons that are most important and specific for task t, where partially filled ones are less important and shared. Sparse connections are learned for each task and compacted in the most important neurons, making free space for learning more tasks. After learning task t, the corresponding weights are kept fixed.

the model has a unified classifier with a shared output layer for all tasks and the task identity is not accessible during inference. We also assume that the data from previous tasks is not available during learning new tasks. Different from previous architectural-based methods, SpaceNet utilizes effectively the fixed capacity of a model instead of expanding the network. The proposed method is based on the adaptive training of sparse neural networks from scratch, a concept introduced by us in [30]. The motivation for using sparse neural networks is not only to free space in the model for future tasks but also to produce sparse representations (semi-distributed representations) throughout the adaptive sparse training which reduces the interference between the tasks. An overview of SpaceNet is illustrated in Figure 1. During learning each task, its sparse connections are evolved in a way that compresses them in a compact number of neurons and gradually produces sparse representations in the hidden layers throughout the training. After convergence, some neurons are reserved to be specific for that task while other neurons can be shared with other tasks based on their importance toward the task. This allows future tasks to use the previously learned knowledge during their learning while reducing the interference between the tasks. The adaptive sparse training is based on the readily available information during the standard training, no extra computational or memory overhead is needed to learn new tasks or remember the previous ones.

Our main contributions in this research are:

- We propose a new method named SpaceNet for continual learning, addressing the more challenging scenario, class incremental learning. SpaceNet utilizes the fixed capacity of the model by compressing the sparse connections of each task in a compact number of neurons throughout the adaptive sparse training. The adaptive training results in sparse representations that reduce the interference between the tasks.

- We address more desiderata for continual learning besides alleviating the catastrophic forgetting problem such as memory constraints, computational costs, a fixed model capacity, preserving the data rights of previous tasks, and non-availability of task identity during inference.

- We achieve a better performance, in terms of robustness to catastrophic forgetting, than the state-of-the-art regularization and architectural methods using a fixed model capacity, outperforming the regularization methods by a big margin.

2. Related Work

The interest in CL in recent years has led to a growing number of methods by the research community. The most common methods can be categorized into three main strategies: regularization strategy, rehearsal strategy, and architectural strategy.

Regularization methods aim to protect the old tasks by adding regularization terms in the loss function that constrain the change to neural network weights. Multiple approaches have been proposed such as: Elastic Weight Consolidation (EWC) [17], Synaptic Intelligence (SI) [41], and Memory Aware Synapses (MAS) [1]. Each of these methods proposed an estimation of the importance of each weight with respect to the trained task. During the training of a new task, any change to the important weights of the old tasks is penalized. Learning Without Forgetting (LWF) [21] is another regularization method that limits the change of model accuracy on the old tasks by using a distillation loss [12]. The current task data is used to compute the response of the model on old tasks. During learning new tasks, this response is used as a regularization term to keep the old tasks stable. Despite that regularization methods are suitable for the situations where one can not access the data from previous tasks, their performance degrade much in class incremental learning scenario [15, 13, 6, 38].

Rehearsal methods replay the old tasks data along with the current task data to mitigate the catastrophic forgetting of the old tasks. Deep Generative Replay (DGR) [37] trains a generative model on the data distribution instead of storing the original data from previous tasks. Similar work has been done by Mocanu et al. [31]. Other methods combine the rehearsal and regularization strategies such as iCaRL [34]. The authors use distillation loss along with an examplar set to impose output stability of old tasks. The main drawbacks of rehearsal methods are the memory overhead of storing old data or a model for generating them, the computational overhead of retraining the data from all previous tasks, and the unavailability of the previous data in some cases.

Architectural methods modify the model architecture in different ways to make space for new information while keeping the old one. PathNet [7] uses a genetic algorithm to find which parts of the network can be reused for learning new tasks. During the learning of new tasks, the weights of the old tasks are kept frozen. The approach has high com-

Table 1

Comparison between different CL methods on desired characteristics for CL.

Strategy	Method	Fixed Model Capacity	Memory Efficiency	Fast Training	Old Data Inaccessibility	Old Tasks Performance
Regularization	EWC	✓	✓	✓	✓	×
	SI	✓	✓	✓	✓	×
	LwF	✓	✓	✓	✓	×
Rehearsal	iCaRL	✓	×	×	×	✓
	DGR	✓	×	×		✓
Architectural	PNN	×	✓	✓	✓	✓
	PackNet	✓	×	×	✓	✓
	DEN	×	×	×	✓	✓
	SpaceNet (Our)	✓	✓	✓	✓	✓

putational complexity. Progressive Neural Network (PNN) [35] is a combination of network expansion and parameter freezing. Catastrophic forgetting is prevented by instantiating a new neural network for each task, while keeping previously learned networks frozen. New networks can take advantage of previous layers learning through the inter-network connections. In this method, the number of model parameters keeps increasing over time. Copy-Weights with Reinit (CWR) [25] is counterpart for PNN. The authors proposed an approach that has a fixed model size but has limited applicability and performance. They used fixed shared parameters between the tasks while the output layer is extended when the model faces a new task. Dynamic Expandable Network (DEN) [40] keeps the network sparse via weight regularization. Part of the weights of the previous tasks is jointly used with the new task weights to learn the new task. This part is chosen regardless of the importance of it to the old task. If the performance of the old tasks degrades much, they try to restore it by node duplication. PackNet [26] is another approach based on sparse neural networks. They prune unimportant weights after learning each task and retrain the network to free some connections for later tasks. A mask is saved for each task to specify the connections that will be used during the prediction time. This method assumes the availability of the task identity during the inference. All the weights of the network are removed except the ones corresponding to the task of the test input. Our method is different from this one in many aspects: (1) we address the class incremental learning scenario where the task identity is unknown during inference, (2) we aim to avoid the overhead of iterative pruning and fine-tuning the network after learning each task, and (3) we propose to introduce the sparsity in the representations on the top of the topological sparsity.

Most of these works use a certain strategy to address the catastrophic forgetting in the CL paradigm. However, there are more desired characteristics for CL as argued by [36, 6]. Table 1 summarizes a comparison between different algorithms from CL desiderata aspects. The continual learning algorithm should be constrained in terms of computational and memory overhead. The model size should kept fixed and additional unnecessary neural resources should not be allocated for new tasks. New tasks should be added without adding high computational complexity or retraining the model. The CL problem should be solved without the need for additional memory to save the old data or specific mask to each task. Lastly, the algorithm should not assume the availability of old data.

3. Problem Formulation

A continual learning problem consists of a sequence of tasks $\{t_1, t_2, ..., t_N\}$. Each task t_i has its own dataset D^t. The neural network model faces tasks one by one. The capacity of the model should be utilized to learn the sequence of the tasks without forgetting any of them. All samples from the current task are observed before switching to the next task. The data across the tasks is not assumed to be identically and independently distributed (iid). To handle the situations when one cannot access the data from previous tasks, we assume that once the training of the current task ends, its data becomes not available.

In this work, we address the class incremental learning scenario for CL. In this setting, all tasks share a single-headed output layer. The task identity is not available at deployment time. At any point in time, the network model should classify the input to one of the classes learned so far regardless of the task identity.

4. SpaceNet Approach for Continual Learning

In this section, we present our proposed method, SpaceNet, for deep neural networks to learn in the continual learning paradigm.

The main objectives of our approach are: (1) utilizing the model capacity efficiently by learning each task in a compact space in the model to leave a room for future tasks, (2) learning sparse representations to reduce the interference between the tasks, and (3) avoiding adding high computational and memory overhead for learning new tasks. In [29], we have introduced the idea of training sparse neural networks from scratch for single task unsupervised learning. Lately, this concept has started to be known as sparse training. In recent years, sparse training proved its success in achieving the same performance with dense neural networks for single task standard supervised/unsupervised learning, while having much faster training speed and much lower memory requirements [30, 2, 4, 5, 14, 32]. In these latter works, sparse neural networks are trained from scratch and the sparse network structure is dynamically changed throughout the training. Works from [5, 32] also show that the sparse training achieves better performance than iteratively pruning a pre-trained dense model and static sparse neural networks. Moreover, Liu et al. [23] demonstrated that there is a plenitude of sparse sub-networks with very different topologies that achieve the same performance.

Taking inspiration from these successes and observations, as none of the above discussed sparse training methods are suitable for direct use in continual learning, we propose an adaptive sparse training method for the continual learning paradigm. In particular, in this work, we adaptively train sparse neural networks from scratch to learn each task with a low number of parameters (sparse connections) and gradually develop sparse representations throughout the training instead of having fully distributed representations over all the hidden neurons. Figure 1 illustrates an overview of

SpaceNet. When the model faces a new task, new sparse connections are randomly allocated between a selected number of neurons in each layer. The learning of this task is then performed using our proposed adaptive sparse training. At the end of the training, the initial distribution of the connections is changed, more connections are grouped in the important neurons for that task. The most important neurons from the initially selected ones are reserved to be specific to this task, while the other neurons are shared between the tasks. The details of our proposed approach are illustrated in Algorithm 1. Learning each task in the continual learning sequence by SpaceNet can be divided into 3 main steps: (1) Connections allocation, (2) Task training, (3) Neurons reservation.

Connections allocation. Suppose that we have a neural network parameterized by $\mathbf{W} = \{W_l\}_{l=1}^{L}$, where L is the number of layers in the network. Initially, the network has no connections ($\mathbf{W} = \emptyset$). A list of free neurons $\mathbf{h}_l^{free}$ is maintained for each layer. This list contains the neurons that are not specific for a certain task and can be used by other tasks for connections allocation. When the model faces a new task t, the shared output layer $\mathbf{h}_L$ is extended with the number of classess in this task n_c^t. New sparse connections $W^t = \{W_l^t\}_{l=1}^{L}$ are allocated in each layer for that task. A selected number of neurons sel_l^t (which is hyperparameter) is picked from $\mathbf{h}_l^{free}$ in each layer for allocating the connections of task t. The selected neurons for task t in layer l is represented by $\mathbf{h}_l^{sel}$. Sparse parameters W_l^t with sparsity level ϵ are randomly allocated between $\mathbf{h}_{l-1}^{sel}$ and $\mathbf{h}_l^{sel}$. The parameters W^t of task t is added to the network parameters $\mathbf{W}$. Algorithm 2 describes the connections allocation process.

Algorithm 1 SpaceNet for Continual Learning

1: **Require:** loss function $\mathcal{L}$, training dataset for each task in the sequence $\mathcal{D}^t$
2: **Require:** sparsity level ϵ, rewiring ratio r
3: **Require:** number of selected neurons sel_l^t, number of specific neurons $spec_l^t$
4: **for each** layer l **do**
5:     $\mathbf{h}_l^{free} \leftarrow \mathbf{h}_l$ ▷ Initialize free neurons with all neurons in l
6:     $\mathbf{h}_l^{spec} \leftarrow \emptyset$
7:     $W_l \leftarrow \emptyset$
8:     $W_L^{saved} \leftarrow \emptyset$
9: **end for each**
10: **for each** available task t **do**
11:     $\mathbf{W} \leftarrow ConnectionsAllocation(\epsilon, sel_l^t, \mathbf{h}^{free})$ ▷ Perform Algorithm 2
12:     $W^t \leftarrow TaskTraining(\mathbf{W}, D^t, \mathcal{L}, r)$ ▷ Perform Algorithm 3
13:     $\mathbf{h}_l^{free} \leftarrow NeuronsReservation(spec_l^t)$ ▷ Perform Algorithm 4
14:     $W_L^{saved} \leftarrow W_L^{saved} \cup W_L^t$ ▷ Retain the connections of last layer for task t
15:     $W_L \leftarrow W_L \setminus W_L^t$
16: **end for each**

Algorithm 2 Connections allocation

1: **Require:** number of selected neurons sel_l^t, sparsity level ϵ
2: $\mathbf{h}_L \leftarrow \mathbf{h}_L \cup n_c^t$ ▷ Expand the shared single output layer with new task classes
3: **for each** layer **do**
4:     $(\mathbf{h}_{l-1}^{sel}, \mathbf{h}_l^{sel}) \leftarrow$ randomly select sel_{l-l}^t and sel_l^t neurons from $\mathbf{h}_{l-1}^{free}$ and $\mathbf{h}_l^{free}$
5:     randomly allocate parameters W_l^t with sparsity ϵ between $\mathbf{h}_{l-1}^{sel}$ and $\mathbf{h}_l^{sel}$
6:     $W_l \leftarrow W_l \cup W_l^t$
7: **end for each**

Task training. The task is trained using our proposed adaptive sparse training. The training data D^t of task t is forwarded through the network parameters $\mathbf{W}$. The parameters of the task W^t is optimized with the following objective function:

$$\min_{W^t} \mathcal{L}(W^t; D^t, W^{1:t-1}), \tag{1}$$

where $\mathcal{L}$ is the loss function and $W^{1:t-1} = \mathbf{W} \setminus W^t$ are the parameters of the previous tasks. The parameters $W^{1:t-1}$ are freezed during learning task t. During the training process, the distribution of sparse connections of task t is adaptively changed, ending up with the sparse connections compacted in a fewer number of neurons. Algorithm 3 shows the details of the adaptive sparse training algorithm. After each training epoch, a fraction r of the sparse connections W_l^t in each layer is dynamically changed based on the importance of the connections and neurons in that layer. Their importance is estimated using the information that is already calculated during the training epoch, no additional computation is needed for importance estimation as we will discuss next. The adaptive change in the connections consists of two phases: (1) Drop and (2) Grow.

Drop phase. A fraction r of the least important weights is removed from each sparse parameter W_l^t. Connection importance is estimated by its contribution to the change in the loss function. The first-order Taylor approximation is used to approximate the change in loss during one training iteration i as follows:

$$\mathcal{L}(\mathbf{W}^{i+1}) - \mathcal{L}(\mathbf{W}^i) \approx \sum_{j=0}^{m-1} \frac{\partial \mathcal{L}}{\partial W_j^i}(W_j^{i+1} - W_j^i) = \sum_{j=0}^{m-1} I_{i,j}, \tag{2}$$

where $\mathcal{L}$ is the loss function, $\mathbf{W}$ is the sparse parameters of the network, m is the total number of parameters, and $I_{i,j}$ represents the contribution of the parameter j in the loss change during the step i, i.e. how much does a small change to the parameter change the loss function [19]. The importance Ω_l^j of connection j in layer l at any step is cumulative of the magnitude of $I_{i,j}$ from the beginning of the training till this step. It is calculated as follows:

$$\Omega_l^j = \sum_{i=0}^{iter} ||I_{i,j}||, \tag{3}$$

where *iter* is the current training iteration.

Grow phase. The same fraction r of the removed connections are added in each sparse parameter W_l^t. The newly added weights are zero-initialized. The probability of growing a connection between two neurons in layer l is proportional to the importance of these two neurons G_l. The importance $a_l^{(i)}$ of the neuron i in layer l is estimated by the summation of the importance of ingoing connections of that neuron as follows:

$$a_l^{(i)} = \sum_{j=0}^{C_{in}-1} \Omega_l^j, \tag{4}$$

where C_{in} is the number of ingoing connections of a neuron i in layer l. The matrix G_l is calculated as follows:

$$G_l = \mathbf{a}_{l-1}\mathbf{a}_l^T \tag{5}$$

Assuming that the number of growing connections in layer l is k_l, the top-k_l positions which contains the highest values in G_l and zero-value in W_l are selected for growing the new connections.

Algorithm 3 Adaptive sparse training

1: **Require:** Loss function $\mathcal{L}$, Training dataset $\mathcal{D}^t$, rewiring ratio r
2: **for each** training epoch **do**
3:     perform standard forward pass through the network parameters $\mathbf{W}$
4:     update parameters W^t using Equation 1
5:     **for each** sparse parameter W_l^t **do**
6:         $\widetilde{W}_l^t \leftarrow$ sort W_l^t based on the importance Ω_l in Equation 3
7:         $(W_l^t, k_l) \leftarrow$ drop $(\widetilde{W}_l^t, r)$ ▷ Remove the weights with smallest importance
8:         compute $\mathbf{a}_{l-1}$ and $\mathbf{a}_l$ from Equation 4   ▷ Neurons importance for task t
9:         $G_l \leftarrow \mathbf{a}_{l-1}\mathbf{a}_l^T$
10:        $\widetilde{G}_l \leftarrow$ sortDescending(G_l)
11:        $Gpos \leftarrow$ select top-k_l positions in $\widetilde{G}_l$ where W_l equals zero
12:        $W_l^t \leftarrow$ grow$(W_l^t, Gpos)$ ▷ Grow k_l zero-initialized weights in $Gpos$
13:     **end for each**
14: **end for each**

For convolutional neural networks, the drop and grow phases are performed in a coarse manner to impose structure sparsity instead of irregular sparsity. In particular, in the drop phase, we consider coarse removal for the whole kernel instead of removing scalar weights. The kernel importance is calculated by the summation over the importance of its $k \times k$ elements calculated by Equation 3. Similarly, in the grow phase, the whole connections of a kernel are added instead of adding single weights. Analogous to multilayer perceptron networks, the probability of adding a kernel between two feature maps is proportional to their importance. The importance of the feature map is calculated by the summation of the importance of its connected kernels.

Neurons reservation. After learning the task, a fraction of the neurons from $\mathbf{h}_l^{sel}$ in each layer is reserved for this task and removed from the list of free neurons $\mathbf{h}_l^{free}$. The choice of these neurons is based on their importance to the current task calculated by equation 4. These neurons become specific to the current task which means that no more connections from other tasks will go in these neurons. The other neurons in $\mathbf{h}_l^{sel}$ are still exist in the free list $\mathbf{h}_l^{free}$ and could be shared by future tasks. Algorithm 4 describes the details of neurons reservation process.

Algorithm 4 Neurons reservation

1: **Require:** number of specific neurons $spec_l^t$
2: **for each** layer l **do**
3:     compute the neuron importance a_l for task t using Equation 4
4:     $\widetilde{a}_l \leftarrow$ sortDescending$(\mathbf{a}_l)$
5:     $\mathbf{h}_l^{t_{spec}} \leftarrow$ top-$spec_l^t$ from $\widetilde{a}_l$
6:     $\mathbf{h}_l^{spec} \leftarrow \mathbf{h}_l^{spec} \cup \mathbf{h}_l^{t_{spec}}$
7:     $\mathbf{h}_l^{free} \leftarrow \mathbf{h}_l^{free} \setminus \mathbf{h}_l^{t_{spec}}$
8: **end for each**

After learning each task, its sparse connections in the last layer (classifier) are removed from the network and retained aside in W_L^{saved}. Removing the classifiers ($W_L^{1:t-1}$) of the old tasks during learning the new one contributes to alleviating the catastrophic forgetting problem. If they are all kept, the weights of the new task will try to get higher values than the weights of the old tasks to be able to learn which results in a bias towards the last learned task during inference. At deployment time, the output layer connections W_L^{saved} for all learned tasks so far are returned to the network weights W_L. All tasks share the same single-headed output layer.

Link to Hebbian Learning The way we evolve the sparse neural network during the training of each task has a connection to Hebbian learning. Hebbian learning [11] is considered as a plausible theory for biological learning methods. It is an attempt to explain the adaptation of brain neurons during the learning process. The learning is performed in a local manner. The weight update is not based on the global information of the loss. The theory is usually summarized as "cells that fire together wire together". It means that if a neuron participates in the activation of another neuron, the synaptic connection between these two neurons should be strengthened. Analogous to Hebb's rule, we consider changing the structure of the sparse connections in a way that increases the number of connections between strong neurons.

5. Experiments

We compare SpaceNet with well-known approaches from different CL strategies. The goals of this experimental study are: (1) evaluating SpaceNet ability in maintaining the performance of previous tasks in the class IL scenario using two typical DNN models (i.e. multilayer perceptron and convolutional neural networks), (2) analyzing the effectiveness of

our proposed adaptive sparse training in the model performance, and (3) comparing between different CL methods in terms of performance and other requirements of CL such as model size and using extra memory. We evaluated our proposed method on three well-known benchmarks for continual learning: split MNIST [20, 41], split Fashion-MNIST [39, 6], and CIFAR-10/100 [18, 41].

5.1. Split MNIST

Split MNIST is first introduced by Zenke et al. [41]. It consists of five tasks. Each task is to distinguish between two consecutive MNIST-digits. This dataset becomes a commonly used benchmark for evaluating continual learning approaches. Most authors use this benchmark in the multi-headed form where the prediction is limited to two classes only, determined by the task identity during the inference. While for our settings, the input image has to be classified into one of the ten MNIST-digits from 0 to 9 (single-headed layer).

5.1.1. Experimental Setup

The standard training/test-split for MNIST was used resulting in 60,000 training images and 10,000 test images. For a fair comparison, our model has the same architecture used by Van et al. [38]. The architecture is a feed-forward network with 2 hidden layers. Each layer has 400 neurons with ReLU activation. We use this fixed capacity to learn all tasks. 10% of the network weights are used for all tasks (2% for each task). Each task is trained for 4 epochs. We use a batch size of 128. The network is trained using stochastic gradient descent with a learning rate 0.01. The selected number of neurons sel_l^t in each hidden layer to allocate the connections for a new task is 80. The number of neurons that are reserved to be specific for each task $spec_l^t$ is 40. The hyperparameters are selected using random search. The experiment is repeated 10 times with different random seeds.

5.1.2. Results

Table 2 shows the average accuracy of different well-known approaches. As illustrated in the table, regularization methods fail to maintain the performance of the previously learned tasks in the class IL scenario. LWF [21] tries to mitigate catastrophic forgetting but the accuracy is still far from the satisfactory level. The experiment shows that SpaceNet is capable of achieving very good performance. It manages to keep the performance of previously learned tasks, outperforming the regularization methods by a big gap around 51.6%. We compare our method also to the DEN algorithm which is the most related one to our work, both being architectural strategies. As discussed in the related work section, DEN keeps the connections sparse by sparse-regularization and restores the drift in old tasks performance using node duplication. In the DEN method, the connections are remarked with a timestamp (task identity) and in the inference, the task identity is required to test on the parameters that are trained up to this task identity only. This implicitly means that T different models are obtained using DEN, where T is the total number of tasks. To make the

Table 2

Average test accuracy on split MNIST using different approaches. Results for regularization and rehearsal methods are adopted from [38, 13] except "SpaceNet-Rehearsal".

Strategy	Method	Accuracy	Extra memory requirements	Old task data	Model expansion
Regularization	EWC	20.01± 0.06	No	No	No
	SI	19.99± 0.06			
	MAS	19.52± 0.29			
	LWF	**23.85** ± 0.44			
Rehearsal	DGR	90.79 ± 0.41	Yes	Yes	No
	iCaRL	94.57 ± 0.11			
	SpaceNet-Rehearsal	**95.08** ± 0.15			
Architectural	DEN	56.95 ± 0.02	Yes	No	Yes
	Static-SparseNN	61.25 ± 2.30	**No**	**No**	**No**
	SpaceNet	**75.53** ± 1.82			

comparison, we adapt the official code provided by the authors to work on the class IL scenario, where there is no access to the task identity during inference. After training all tasks, the test data is evaluated on the model created each timestamp t. The class with the highest probability from all models is taken as the final prediction. Besides that DEN has computational overhead for learning a new task when comparing to SpaceNet, it also increases the number of neurons in each layer by around 35 neurons, while SpaceNet still has free neurons in each layer. As shown in the table, SpaceNet obtains the best performance among the methods from its strategy (category), reaching an accuracy of about 75.53%, with 18.5% better than the DEN algorithm.

Rehearsal methods succeeded in maintaining their performance in the class IL scenario to a certain level. Replaying the data from previous tasks during learning a new task mitigates the problem of catastrophic forgetting. However, retraining old tasks data has a cost of requiring additional memory for storing the data and the generative model in case of generative replay methods. Making rehearsal methods resource-efficient is still an open research problem. The results of SpaceNet are considered very satisfactory and promising compared to rehearsal methods given that we do not use any of the old tasks data and the number of connections is much smaller i.e. SpaceNet has 28 times fewer connections than DGR.

Please note that it is easy to combine SpaceNet with rehearsal strategies. We perform an experiment in which the old tasks data are repeated during learning new tasks, while keeping the connections of the old tasks fixed. We refer to this experiment as "SpaceNet-Rehearsal". Replaying the old data helps to find weights for the new task that do not degrade the performance of the old tasks. As shown in Table 2, "SpaceNet-Rehearsal" outperforms all the state-of-the-art methods, including the rehearsal ones, while having a much smaller number of connections. However, replaying the data from the previous tasks is outside the purpose of this paper where we try besides maximizing performance to cover the scenarios when one has no access to the old data, minimize memory requirements, and reduce the computational overhead for learning new tasks or remember the previous ones.

A comparison between different methods in terms of other requirements for CL is also shown in Table 2. Regularization methods satisfy many desiderata of CL while losing the performance. SpaceNet is able to compromise between the performance and other requirements that are not even satis-
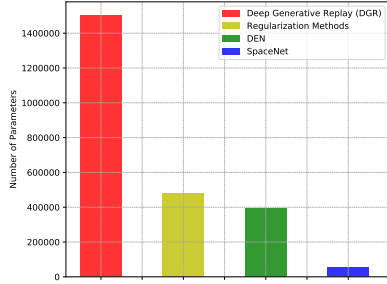
Figure 2: Comparison between SpaceNet and other CL methods on split MNIST in terms of model size.

fied by other architectural methods. Moreover, we compare the model size of our approach with the other methods. As illustrated in Figure 2, SpaceNet model with at least one order of magnitude fewer parameters than any of the other method studied.

We further analyze the effect of our proposed adaptive sparse training in performance. We compare our approach with another baseline, referred as "Static-SparseNN". In this baseline, we run our proposed approach for CL but with static sparse connections and train the model with the standard training process. As shown in Table 2, the adaptive sparse training increases the performance of the model by a good margin. The average accuracy for all tasks is increased by 14.28%.

5.2. Split Fashion-MNIST

An additional experiment for validating our approach is performed on the Fashion-MNIST dataset [39]. This dataset is more complex than MNIST. The images show individual articles of clothing. The authors argued that it is considered as a drop-in replacement for MNIST. However, it has the same sample size and structure of training and test sets as MNIST. This dataset is used by Farquhar and Gal [6] to evaluate different CL approaches. They construct split Fashion-MNIST which consists of five tasks. Each task has two consecutive classes of Fashion-MNIST.

5.2.1. Experimental Setup

The same setting and architecture used for the MNIST dataset are used in this experiment. We use the official code from [38] to test the accuracy of their implemented CL approaches on split Fashion-MNIST. We do not change the experimental settings to evaluate the performance of the methods on a more complex dataset using such small neural networks.

5.2.2. Results

We observe the same findings that regularization methods fail to remember previous tasks. The performance of rehearsal methods on this more difficult dataset starts to deteriorate. Replaying the data with the SpaceNet approach achieves the best performance. As shown in Table 3, while the accuracy of DEN degrades much, SpaceNet maintains

Table 3

Average test accuracy overall tasks of split Fashion-MNIST using different approaches.

Strategy	Method	Accuracy	Extra memory requirements	Old task data	Model expansion
Regularization	EWC	19.47 ± 0.98	No	No	No
	SI	19.93 ± 0.01			
	LWF	$\mathbf{20.76 \pm 1.65}$			
Rehearsal	DGR	73.58 ± 3.90	Yes	Yes	No
	iCaRL	80.70 ± 1.29			
	SpaceNet-Rehearsal	$\mathbf{84.18 \pm 0.24}$			
Architectural	DEN	31.51 ± 0.04	Yes	No	Yes
	Static-SparseNN	56.80 ± 2.30			
	SpaceNet	$\mathbf{64.83 \pm 0.69}$	**No**	**No**	**No**

a stable performance on the tasks. The sparse training in SpaceNet increases the performance by 8% compared to "Static-SparseNN".

5.3. CIFAR-10/100

In this experiment, we show that our proposed approach can be applied also to convolutional neural networks (CNNs). We evaluate spaceNet on complex datasets: CIFAR-10 and CIFAR-100 [18]. CIFAR-10 and CIFAR-100 are well-known benchmarks for classification tasks. They contain tiny natural images of size (32×32). CIFAR-10 consists of 10 classes and has 60000 samples (50000 training + 10000 test), with 6000 images per class. While CIFAR-100 contains 100 classes, with 600 images per class (500 train + 100 test). Zenke et al. [41] uses these two datasets to create a benchmark for CL which they referred as CIFAR-10/100. It has 6 tasks. The first task contains the full dataset of CIFAR-10, while each subsequent task contains 10 consecutive classes from CIFAR-100 dataset. Therefore, task 1 has a 10x larger number of samples per class which makes this benchmark challenging as the new tasks have limited data.

5.3.1. Experimental Setup

For a fair and direct comparison, we follow the same architecture used by Zenke et al. [41] and Maltoni and Lomonaco [27]. The architecture consists of 4 convolutional layers (32-32-64-64 feature maps). The kernel size is 3×3. Max pooling layer is added after each 2 convolutional layers. Two sparse feed-forward layers follow the convolutional layers (512-60 neurons), where 60 is the total number of classes from all tasks. In our case, no dropout is implemented and the model is optimized using stochastic gradient descent with learning rate 0.1. Each task is trained for 20 epochs. 12% of the network weights is used for each task. Since the number of feature maps in each layer in the used architecture is too small, the number of selected feature maps for each task sel_l^t equals to the number of feature maps in this layer. The number of specific feature maps in each hidden layer $spec_l^t$ is as follows: [2, 2, 5, 6, 30]. The hyperparameters are selected using random search.

5.3.2. Results

Figure 3 shows the accuracy of different popular CL methods for each task of CIFAR-10/100 after training all tasks. The results of other algorithms are extracted from the work done by Maltoni and Lomonaco [27] and re-plotted. "Naive" algorithm is referred by the authors to the simple finetuning where there is no limitation for forgetting other than early
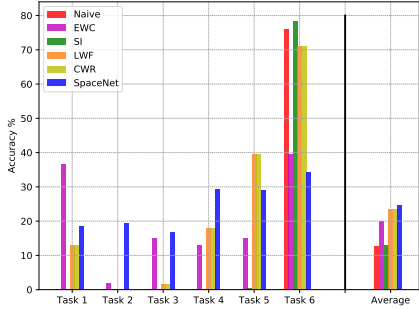
Figure 3: Accuracy on each task of CIFAR-10/100 benchmark for different CL approaches after training the last task. Results for other approaches are adopted from Maltoni and Lomonaco [27]. Task 1 is the full dataset of CIFAR10, while task 2 to task 6 are the first 5 tasks from CIFAR100. Each task contains 10 classes. The missing rectangles for some of the methods for some of the tasks means that accuracy for that particular case is 0. The "Average" x-axis label shows the average accuracies computed overall tasks for each method. SpaceNet managed to utilize the available model capacity efficiently between the tasks, unlike other methods that have high performance on the last task but completely forgetting some other previous tasks.

stopping. SI totally fails to remember all old tasks and the model is fitted just on the last learned one. Other algorithms have a good performance on some tasks, while the performance on the other tasks is very low. Despite that the architecture used in this experiment is small, SpaceNet managed to utilize the available space efficiently between the tasks. As the figure shows, SpaceNet outperforms all the other algorithms in terms of average accuracy. In addition, the standard deviation over all tasks accuracy is much (few times) smaller than the standard deviation of any other state-of-the-art method. This means that the model is not biased towards a single task and the accuracy of the learned tasks is close to each other. This clearly highlights the robustness of SpaceNet and its strong capabilities in remembering old tasks. To show that SpaceNet is far from reaching its true potential, we increase the number of feature maps in the first four convolution layers to (64-64-128-128). Using this bit larger architecture, the average accuracy for all tasks is increased by around 3%.

6. Analysis

In this section, we analyze the representations learned by SpaceNet, the distribution of the sparse connections after the adaptive sparse training, and the relation between the learned distribution of the connections and the importance of the neurons. We performed this analysis on the Split MNIST benchmark.

First, we analyze the representations learned by SpaceNet. We visualize the activations of the two hidden layers of the multilayer perception network used for Split MNIST. After learning the first task of Split MNIST, we analyze the representations of random test samples from this task. Fig-
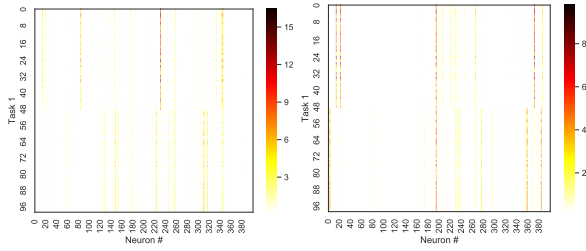
(a) First hidden layer.   (b) Second hidden layer.

Figure 4: Heatmap of the first and second hidden layers activations after forwarding the test data of task 1 of split MNIST. The y-axis represents the test samples. The first 50 samples belong to class 0 while the other 50 belong to class 1.
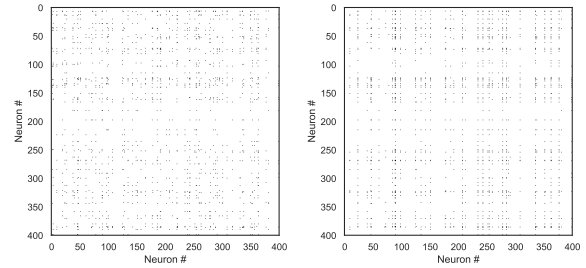

(a) Initial connections.   (b) Learned connections.

Figure 5: Connections distribution between two layers for one task of the Split MNIST benchmark. Figure (a) shows the initial random distribution of the connections on the selected neurons. Figure (b) shows the connections after the adaptive sparse training. The connections are compacted in some of the neurons.

ure 4 shows the representations of 50 random samples from the test set of class 0 and another 50 samples from the test set of class 1. The figure illustrates that the representations learned by SpaceNet are highly sparse. A small percentage of activations is used to represent an input. This reveals that the designed topological sparsity of SpaceNet not only helps to utilize the model capacity efficiently to learn more tasks but also led to sparsity in the activation of the neurons which reduces the interference between the tasks. It is worth highlighting that our findings from this research are aligned with the early work by French [8]. French argued that catastrophic forgetting is a direct consequence of the representational overlap of different tasks and semi-distributed representations could reduce the catastrophic forgetting problem.

Next, we analyze how the distribution of the connections changes as a result of the adaptive training. We visualize the sparse connections of the second task of the Split MNIST benchmark before and after its training. The initially allocated connections are randomly distributed between the selected neurons as shown in Figure 5a. Instead of having the sparse connections distributed over all the selected neurons, the evolution procedure makes the connections of a task grouped in a compact number of neurons as shown in Figure 5b, leaving space for future tasks.

We further analyze whether the connections are grouped
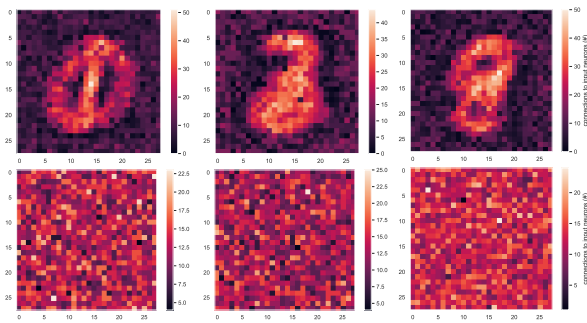
(a) Task 1 (0 or 1).   (b) Task 2 (2 or 3).   (c) Task 5 (8 or 9).

Figure 6: Visualization of the number of connected weights to each of the input neurons for three different tasks in the Split MNIST benchmark. The connections are reshaped to 28×28 to be visualized as an image. The first row represents the connections distribution results from our proposed method, SpaceNet. While the second row results from the "Static-SparseNN" baseline discuss in the experiments section.

in the right neurons (e.g. the important ones) or not. To qualitatively evaluate this point, we visualize the number of existing connections outgoing from each neuron in the input layer. The input layer consists of 784 neurons (28×28). Consider the first layer of the multilayer perception network used for the Split MNIST benchmark. The layer is parameterized by the sparse weights $W_{l=1} \in R^{784 \times 400}$. We visualize the learned connections corresponding to some of Split MNIST tasks. For each $W_{l=1}^{t}$, we sum over each row to get the number of connections linked to each of the 784 input neurons. We then reshape the output vector to 28×28. Figure 6 shows the visualization of connections distribution for three different tasks of the Split MNIST benchmark. As shown in the figure, more connections are grouped in the input neurons that define the shape of each digit. For example in Figure 6a, in the first row, most of the connections are grouped in the neurons representing class 0 and class 1. The figure also illustrates the distribution of the connections in the case of "Static-SparseNN" baseline discussed in the experiments section. As shown in the figure, in the second row, the connections are distributed over all the neurons of the input layer regardless of the importance of this neuron to the task which could lead to the interference between the tasks.

7. Conclusion

In this work, we have proposed SpaceNet, a new technique for deep neural networks to learn a sequence of tasks in the continual learning paradigm. SpaceNet learns each task in a compact space in the model with a small number of connections, leaving a space for other tasks to be learned by the network. We address the class incremental learning scenario, where the task identity is unknown during inference. The proposed method is evaluated on the well-known benchmarks for CL: split MNIST, split Fashion-MNIST, and CIFAR-10/100.

Experimental results show the effectiveness of SpaceNet in alleviating the catastrophic forgetting problem. Results on split MNIST and split Fashion-MNIST outperform the existing well-known regularization methods by a big margin: around 51% and 44% higher accuracy on the two datasets respectively, thanks to the technical novelty of the paper. SpaceNet achieved better performance than the existing architectural methods, while using a fixed model capacity without network expansion. Moreover, the accuracy of SpaceNet is comparable to the studied rehearsal methods and satisfactory given that we use 28 times lower memory footprint and do not use the old tasks data during learning new tasks. It worths mentioning that even if it was a bit outside of the scope of this paper, when we combined SpaceNet with a rehearsal strategy, the hybrid obtained method (i.e. SpaceNet-Rehearsal) outperformed all the other methods in terms of accuracy. The experiments also show how the proposed method efficiently utilizes the available space in a small CNN architecture to learn a sequence of tasks from a more complex dataset CIFAR-10/100. Unlike other methods that have a high performance on the last learned task only, SpaceNet is able to maintain good performance on previous tasks as well. Its average accuracy computed overall tasks is higher than the ones obtained by the state-of-the-art methods, while the standard deviation is much smaller. This demonstrates that SpaceNet has the best trade-off between non-catastrophic forgetting and using a fixed model capacity.

The proposed method showed its success in addressing more desiderata for CL besides alleviating the catastrophic forgetting problem such as: persevering old data rights, memory efficiency, using a fixed model size, and avoiding any extra computation for adding or retaining knowledge. We finally showed that the learned representations by SpaceNet is highly sparse and the adaptive sparse training results in redistributing the sparse connections in the important neurons for each task.

There are several potential research directions to expand this work. In the future, we would like to combine SpaceNet with a resource-efficient generative-replay method to enhance its performance in terms of accuracy, while reducing even more the memory requirements. Another interesting direction is to investigate the effect of balancing the magnitudes of the weights across all tasks to mitigate the bias towards a certain task.

References

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

[2] Bellec, G., Kappel, D., Maass, W., Legenstein, R., 2018. Deep rewiring: Training very sparse deep networks, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=BJ_wN01C-.

[3] Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L., 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence 40, 834–848.

[4] Dettmers, T., Zettlemoyer, L., 2019. Sparse networks from

scratch: Faster training without losing performance. arXiv preprint arXiv:1907.04840 .

[5] Evci, U., Gale, T., Menick, J., Castro, P.S., Elsen, E., 2019. Rigging the lottery: Making all tickets winners. arXiv preprint arXiv:1911.11134 .

[6] Farquhar, S., Gal, Y., 2019. Towards robust evaluations of continual learning, in: Privacy in Machine Learning and Artificial Intelligence workshop, ICML. URL: http://arxiv.org/abs/1805.09733.

[7] Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A.A., Pritzel, A., Wierstra, D., 2017. Pathnet: Evolution channels gradient descent in super neural networks. arXiv preprint arXiv:1701.08734 .

[8] French, R.M., 1991. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks, in: Proceedings of the 13th annual cognitive science society conference, pp. 173–178.

[9] Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., Lew, M.S., 2016. Deep learning for visual understanding: A review. Neurocomputing 187, 27–48.

[10] He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, pp. 1026–1034.

[11] Hebb, D.O., Hebb, D., 1949. The organization of behavior. volume 65. Wiley New York.

[12] Hinton, G., Vinyals, O., Dean, J., 2014. Distilling the knowledge in a neural network. nips deep learning workshop. arXiv preprint arXiv:1503.02531 .

[13] Hsu, Y.C., Liu, Y.C., Ramasamy, A., Kira, Z., 2018. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. arXiv preprint arXiv:1810.12488 .

[14] Junjie, L., Zhe, X., Runbin, S., Cheung, R.C., So, H.K., 2019. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers, in: International Conference on Learning Representations.

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

[16] Kenton, J.D.M.W.C., Toutanova, L.K., 2019. Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of NAACL-HLT, pp. 4171–4186.

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

[18] Krizhevsky, A., Hinton, G., et al., 2009. Learning multiple layers of features from tiny images. Technical Report. Citeseer.

[19] Lan, J., Liu, R., Zhou, H., Yosinski, J., 2019. Lca: Loss change allocation for neural network training, in: Advances in Neural Information Processing Systems, pp. 3619–3629.

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

[21] Li, Z., Hoiem, D., 2017. Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence 40, 2935–2947.

[22] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017. Feature pyramid networks for object detection, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2117–2125.

[23] Liu, S., Van der Lee, T., Yaman, A., Atashgahi, Z., Ferraro, D., Sokar, G., Pechenizkiy, M., Mocanu, D.C., 2020. Topological insights in sparse neural networks. arXiv preprint arXiv:2006.14085 .

[24] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E., 2017. A survey of deep neural network architectures and their applications. Neurocomputing 234, 11–26.

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

[26] Mallya, A., Lazebnik, S., 2018. 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.

[27] Maltoni, D., Lomonaco, V., 2019. Continuous learning in single-incremental-task scenarios. Neural Networks 116, 56–73.

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

[29] Mocanu, D.C., Mocanu, E., Nguyen, P.H., Gibescu, M., Liotta, A., 2016a. A topological insight into restricted boltzmann machines. Machine Learning 104, 243–270.

[30] Mocanu, D.C., Mocanu, E., Stone, P., Nguyen, P.H., Gibescu, M., Liotta, A., 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nature communications 9, 2383.

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

[32] Mostafa, H., Wang, X., 2019. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization, in: International Conference on Machine Learning, pp. 4646–4655.

[33] Pomponi, J., Scardapane, S., Lomonaco, V., Uncini, A., 2020. Efficient continual learning in neural networks with embedding regularization. Neurocomputing .

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

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

[36] Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y.W., Pascanu, R., Hadsell, R., 2018. Progress & compress: A scalable framework for continual learning, in: ICML.

[37] Shin, H., Lee, J.K., Kim, J., Kim, J., 2017. Continual learning with deep generative replay, in: Advances in Neural Information Processing Systems, pp. 2990–2999.

[38] van de Ven, G.M., Tolias, A.S., 2018. Three scenarios for continual learning, in: Continual Learning Workshop NeurIPS. URL: http://arxiv.org/abs/1904.07734.

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

[40] Yoon, J., Yang, E., Lee, J., Hwang, S.J., 2018. Lifelong learning with dynamically expandable networks, in: International Conference on Learning Representations.

[41] Zenke, F., Poole, B., Ganguli, S., 2017. Continual learning through synaptic intelligence, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org. pp. 3987–3995.

[42] Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V., 2018. Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697–8710.