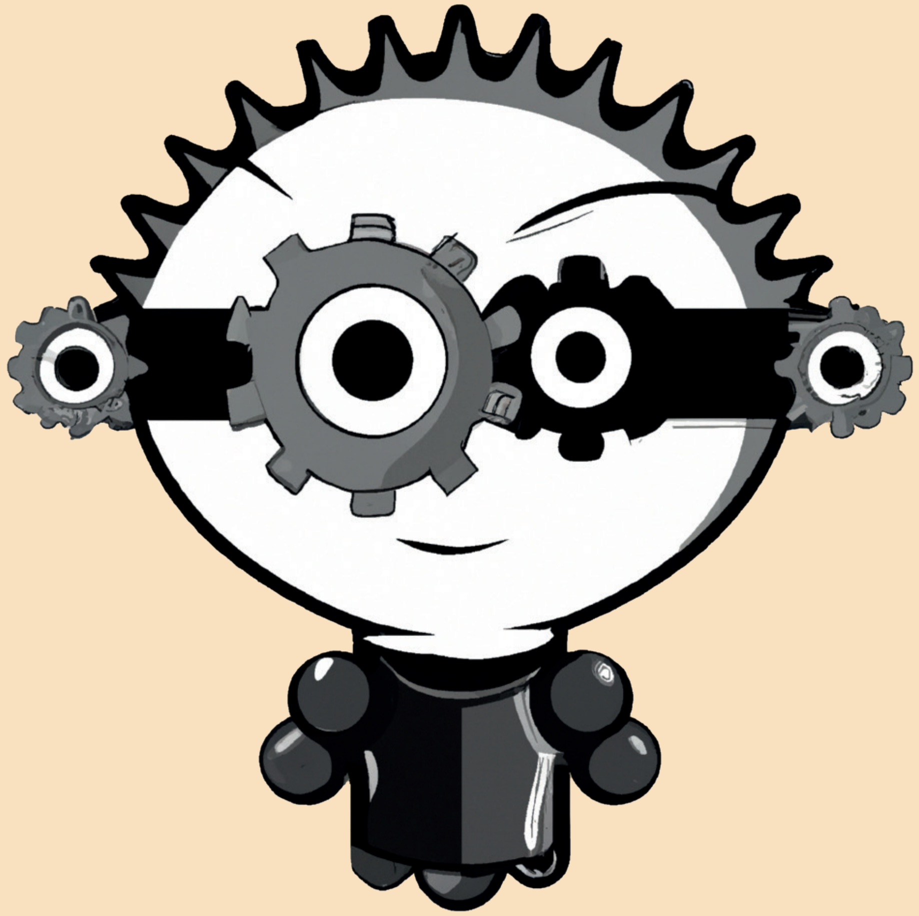
Learning Continually Under Changing Data Distributions

Learning Continually Under Changing Data Distributions

Ghada Sokar

Learning Continually Under Changing Data Distributions



Ghada Sokar

Learning Continually Under Changing Data Distributions

Ghada Sokar

Learning Continually Under Changing Data Distributions by Ghada Sokar.
Eindhoven: Technische Universiteit Eindhoven, 2023. Proefschrift.

A catalogue record is available from the Eindhoven University of Technology Library

ISBN 978-90-386-5853-7

SIKS Dissertation Series No. 2023-27
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Keywords:	continual learning, reinforcement learning, sparse neural network training, dynamic sparse training, sparse representation, changing data distributions, non-stationarity
Cover image:	DALL.E 2
Cover design:	ipskampprinting.nl
Printed by:	ipskampprinting.nl

Learning Continually Under Changing Data Distributions

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr. S.K. Lenaerts, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op dinsdag 31 oktober 2023 om 13:30 uur

door

Ghada Sokar

geboren te Ar Rass, Saoedi-Arabië

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. M.T. de Berg
promotor:	prof.dr. M. Pechenizkiy
copromotor:	dr. D.C. Mocanu
leden:	prof.dr. C. Dovrolis (Georgia Institute of Technology, The Cyprus Institute)
	prof.dr. D. Ernst (University of Liège)
	prof.dr. T. Karkkainen (University of Jyvaskyla)
	dr. C.P. de Campos
adviseur(s):	dr. P.S. Castro (Google DeepMind)

Summary

Classical deep learning has demonstrated remarkable accomplishments in various domains. However, these achievements have primarily been in isolated single-task learning scenarios, assuming a stationary data distribution and necessitating a large number of training examples upfront. The rapid change in the real world, accompanied by the generation of new data across various domains, highlights the need for deep neural networks to possess the ability to continually learn and adapt over time. This capability enables various applications, including autonomous vehicles that can adjust to new road conditions and traffic patterns, as well as chatbots that can stay updated with the emergence of new vocabulary and evolving user behavior. However, such a dynamic learning regime has many challenges. For instance, networks tend to forget previously learned knowledge when they are adapted to new data. Over time, networks also lose their ability to effectively adapt to new information. Additionally, in order to learn continuously, it is crucial to utilize a network's capacity efficiently to accommodate all the data encountered, while also ensuring that the training process remains memory and computationally efficient.

In this thesis, we analyze the underlying factors contributing to the challenges of continuous learning through the lens of learned representations, training regimes, and network capacity utilization. Using these insights, we propose novel methods for training deep neural networks on continuous data. We study two learning paradigms. First, Continual Learning (CL), where a model sequentially learns new tasks over time. Second, Deep Reinforcement Learning (DRL), where a model is trained on a single environment (task); however, it continually

learns new samples that are collected during training while interacting with the environment. The main contributions of this thesis are summarized as follows:

- SpaceNet: We propose the first dynamic sparse training method for continual learning. SpaceNet trains a sparse sub-network from scratch for each task and optimizes the sparse topology during training to learn sparse representations. Our results demonstrate that dense representations learned by conventional training contribute to forgetting past tasks. In contrast, sparse representations effectively reduce interference among tasks and forgetting. Furthermore, we show that the capacity of over-parameterized models can be utilized efficiently to learn multiple tasks sequentially.

- SAM: We investigate the characteristics of useful representations that help to learn new tasks. Our analyses show that having prior generic knowledge before learning the continual sequence helps to improve performance. In addition, selecting the relevant representation from the past while learning new tasks promotes forward transfer and reduces forgetting via selective update of networks' parameters. We demonstrate these findings using our proposed Self-Attention Meta-learner (SAM).

- DST for DRL: We introduce for the first time Dynamic Sparse Training (DST) to the reinforcement learning paradigm and study the adaptability of networks to changing distributions. A sparse neural network is trained from scratch, and its topology is dynamically adapted to the changing data distribution during training. We show that networks trained using dynamic sparse training have faster adaptability and learning speed. Moreover, they outperform dense networks while reducing the floating-point operations (FLOPs) required for training.

- ReDo: We investigate the underlying reasons behind the loss of adaptability of dense networks to new samples over time. To this end, we analyze changes in network utilization in deep reinforcement learning by monitoring neuron activity. Our study reveals the existence of the dormant neuron phenomenon, where an agent's network suffers from an increasing number of inactive neurons, thereby affecting network expressivity. We found that target non-stationarity is the main cause of this phenomenon. To tackle this, we propose a simple method (*ReDo*) that *Re*cycles *Do*rmant neurons during training. Our experiments demonstrate significant performance improvements by reducing dormant neurons via recycling.

We conclude this thesis by discussing our findings regarding effective representations, training regimes, and network utilization for learning paradigms with changing data distributions. We highlight the potential of sparse training in addressing different challenges in such paradigms. We further discuss the limitations of our proposed methods and present a range of promising directions for future work. These directions aim to contribute to developing more accurate and efficient AI systems capable of adapting and improving over time when encountering new information.

List of Publications

Ghada Sokar has the following publications:

Journal Publications

1. Zahra Atashgahi, **Ghada Sokar**, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Raymond Veldhuis, Mykola Pechenizkiy. *Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders*. Machine Learning, 1-38, 2022.

2. **Ghada Sokar**, Decebal Constantin Mocanu, Mykola Pechenizkiy. *SpaceNet: make free space For continual learning*. Neurocomputing, 1–11, 2021.

Conference Publications

3. **Ghada Sokar**, Rishabh Agarwal, Pablo Samuel Castro, Utku Evci. *The dormant neuron phenomenon in deep reinforcement learning*. International Conference on Machine Learning (ICML), PMLR, **Oral**, 2023.

4. Bram Grooten, **Ghada Sokar**, Shibhansh Dohare, Elena Mocanu, Matthew E. Taylor, Mykola Pechenizkiy, Decebal Constantin Mocanu. *Automatic noise filtering with dynamic sparse training in deep reinforcement learning*. International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2023.

5. **Ghada Sokar**, Zahra Atashgahi, Mykola Pechenizkiy, Decebal Constantin Mocanu. *Where to pay attention in sparse training for feature selection?* Advances in Neural Information Processing Systems (NeurIPS), 2022.

6. **Ghada Sokar**, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, Peter Stone. *Dynamic sparse training for deep reinforcement learning*. International Joint Conference on Artificial Intelligence (IJCAI), 2022. Adaptive and Learning Agents Workshop at the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), **Best Paper Award**, 2022.

7. **Ghada Sokar**, Decebal Constantin Mocanu, Mykola Pechenizkiy. *Avoiding forgetting and allowing forward transfer in continual learning via sparse networks*. The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD), 2022.

8. Shiwei Liu, Tianlong Chen, Zahra Atashgahi, Xiaohan Chen, **Ghada Sokar**, Elena Mocanu, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. *Deep ensembling with no overhead for either training or testing: the all-round blessings of dynamic sparsity*. International Conference on Learning Representations (ICLR), 2022.

9. **Ghada Sokar**, Decebal Constantin Mocanu, Mykola Pechenizkiy. *Self-attention meta-learner for continual learning*. International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2021.

10. **Ghada Sokar**. *Continual lifelong learning for intelligent agents*. International Joint Conference on Artificial Intelligence (IJCAI), Doctoral Consortium, 2021.

11. **Ghada Sokar**, Decebal Constantin Mocanu, Mykola Pechenizkiy. *Learning invariant representation for continual learning*. Meta-Learning for Computer Vision Workshop at the AAAI Conference on Artificial Intelligence (AAAI), 2021.

12. Shiwei Liu, Tim Van der Lee, Anil Yaman, Zahra Atashgahi, Davide Ferraro, **Ghada Sokar**, Mykola Pechenizkiy, and Decebal Constantin Mocanu. *Topological insights into sparse neural networks*. The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD), 2020.

13. **Ghada Sokar**, Yassien Zakaria, Asmaa Rabie, Kareem Madkour, Ira Leventhal, Jochen Rivoir, Xinli Gu, and Haralampos-G. Stratigopoulos. "IP Session on Machine Learning Applications in IC Test-Related Tasks." In 2019 IEEE 37th VLSI Test Symposium (VTS), pp. 1-1. IEEE, 2019.

14. **Ghada Sokar**, Elsayed E. Hemayed, and Mohamed Rehan. "A generic OCR using deep siamese convolution neural networks." In 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), pp. 1238-1244. IEEE, 2018.

Preprint (under review / in preparation)

15. Murat Onur Yildirim, Elif Ceren Gok, **Ghada Sokar**, Decebal Constantin Mocanu, Joaquin Vanschoren. *Continual learning with dynamic sparse training: exploring algorithms for effective model updates*. 2023.

16. **Ghada Sokar**, Decebal Constantin Mocanu, Mykola Pechenizkiy. *The role of data in continual learning*. 2023.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning has achieved many recent breakthroughs in machine learning [HZRS16, TL19, LMW$^+$22, KNH$^+$22, DBK$^+$20]. Despite the remarkable performance of deep neural network models, the learning paradigm of such models has some limitations. These models are typically trained on *individual isolated* tasks, assuming a static world, and rely on having a *large* number of training examples available before training. The data distribution remains constant during the training process. After training, the model's applicability is limited to making inferences on data drawn from the same training distribution and is inflexible to learning new data. Such limitations hinder the applicability of these models to real-life applications in which data evolve over time. The rapid change in the real world, accompanied by the generation of new data across various domains, highlights the need for deep neural networks to possess the ability to learn and adapt continually. This capability would push the advance

This chapter is partly based on:
[1] Ghada Sokar. *Continual lifelong learning for intelligent agents*. International Joint Conference on Artificial Intelligence (IJCAI), Doctoral Consortium, 2021.
[2] Ghada Sokar, Zahra Atashgahi, Decebal Constantin Mocanu, and Mykola Pechenizkiy. *Where to pay attention in sparse training for feature selection?* Advances in Neural Information Processing Systems (NeurIPS), 2022.
[3] Ghada Sokar, Decebal Constantin Mocanu, Mykola Pechenizkiy. *Avoiding forgetting and allowing forward transfer in continual learning via sparse networks*. The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD), 2022.
[4] Bram Grooten, **Ghada Sokar**, Shibhansh Dohare, Elena Mocanu, Matthew E. Taylor, Mykola Pechenizkiy, Decebal Constantin Mocanu. *Automatic noise filtering with dynamic sparse training in deep reinforcement learning*. International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2023.

in machine learning to the next level. It would enable the building of up-to-date Artificial Intelligent (AI) systems, which are crucial for many applications. Fraud detection systems provide an example of this concept, as they can improve security by identifying newly emerging fraud patterns while simultaneously retaining their capability to detect older patterns. Another example that highlights the necessity of continuous learning models is evident in autonomous vehicles. These systems must constantly learn new situations on the road, evolving traffic patterns, and new environments. As a system operates and gathers more experience, it can incorporate new knowledge to make more informed decisions, enhancing overall performance.

A native solution to enable current deep neural networks to learn continually is *retraining from scratch* on old and new data. Although this approach is straightforward, it comes with significant drawbacks. Recent successes in deep learning have been achieved by large models with millions or even trillions of parameters [HZRS16, BMR+20]. Retraining such models on massive data each time new samples arrive requires prohibitive memory and computational costs, as well as substantial energy consumption. For instance, it is estimated that GPT-3 [BMR+20] costs 552 tons of CO_2 equivalent emissions to train [PGL+21]. This hinders the building of greener AI systems. Moreover, the training of these large models is extremely expensive. Recognizing these limitations has driven the emergence of a novel learning regime for deep neural networks. In this regime, a network continually learns new data over time while retaining previously acquired knowledge and leveraging it to facilitate future learning. Unlike the classical learning paradigm, which assumes the availability of all data prior to training, this new regime enables networks to learn incrementally without the need for all data upfront. However, this introduces the challenge of training networks on changing data distributions, necessitating novel techniques to address this dynamic nature of the learning process.

In this thesis, we study two learning paradigms that fall under this learning regime, which we dub *"learning on changing data distributions"*: continual learning and deep reinforcement learning. Before introducing our research questions, we provide the problem formulation and main challenges for each learning paradigm in the next two sections.

1.1 Continual Learning

In Continual Learning (CL), a neural network model learns a series of tasks sequentially $\{1, 2, ..., t, ..., T\}$; where T is the total number of tasks. Each task

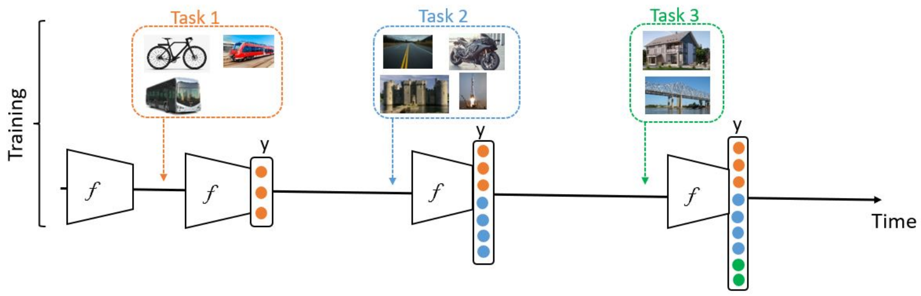


Figure 1.1: An illustration of the class-incremental learning scenario in the continual learning paradigm. A model learns new tasks sequentially. Each task brings a new set of classes. Task labels are not available at the test time. The model can make inferences on all previously seen classes at any point.

t has its dataset D^t. A model faces tasks one by one. All samples from the current task are observed before switching to the next task. The goal is to learn a function f parameterized by θ that minimizes the loss ℓ on the current task without affecting previous tasks. Formally, at task t, the objective is as follows:

$$\underset{\theta}{\arg\min}\, \ell(f_\theta, D^t) \qquad s.t \quad \ell(f_\theta, D^i) \leq \ell(f_\theta^{(t-1)}, D^i) \qquad \forall i < t, \tag{1.1}$$

where $f_\theta^{(t-1)}$ is the learned function till task $t-1$. In Section 1.1.2, we will discuss current methods that address this objective along with other continual learning desiderata. Before diving into that, we discuss different existing scenarios for continual learning.

1.1.1 Scenarios for Continual Learning

Recent works have divided the continual learning paradigm into different scenarios based on incoming data [vdVT18, WZSZ23]. It includes the following scenarios:

- Instance-Incremental Learning (IIL): Training samples are coming from the same task and arrive in batches.
- Domain-Incremental Learning (DIL): Tasks have the same label space, but the input distribution is different. Thus, new tasks contain the same classes, but the data is drawn from different domains.

- Task-Incremental Learning (TIL): Tasks have disjoint label space. New tasks bring new classes. The task identity/label is available during training and testing. Given a task label, the model should predict a label from the set of classes within one task.

- Class-Incremental Learning (CIL): Similar to TIL, tasks have disjoint label space. However, the task identity is available only during training. At inference, the model should classify the input to one of all seen classes so far.

In this thesis, we consider the supervised image classification problem and focus on the most challenging scenario (class-incremental learning (CIL)) [vdVT18, HLRK18]. An overview of this scenario is illustrated in Figure 1.1. Furthermore, we address the even more challenging situation where a model operates with *no* access to previous training samples, and its *fixed-capacity* is utilized without expansion. These constraints address multiple CL desiderata, as we will discuss next.

In Section 1.2, we will discuss the reinforcement learning paradigm, wherein the scenario of instance incremental learning (IIL) finds its manifestation.

1.1.2 Desiderata of Continual Learning

The goal of continual learning is to optimize the objective function presented in Equation 1.1 while taking into consideration the following desiderata [HRRP20, DRLFM18]:

- **Backward transfer**. Learning new tasks should not negatively affect performance on old tasks and may even enhance their performance.

- **Forward transfer**. The model should use the previously learned knowledge to help in future learning.

- **Minimal access to old task data**. The model should not rely on the availability of past data while learning new tasks to account for possible privacy concerns and computational and memory efficiency.

- **Minimal expansion of model capacity**. The model should not get bigger with every new task, and the capacity of the model should be utilized efficiently.

- **Task agnostic**. The model should not rely on the availability of task identity at inference.
- **Computational and memory efficiency**. To achieve scalability and efficient learning of a sequence of tasks, the computational and memory requirements of the training approach should be considered.

Addressing multiple desiderata simultaneously is challenging. For instance, forward and backward transfers may be competing. Changing the model parameters to adapt to a new task causes forgetting the previously learned knowledge (negative backward transfer). On the other hand, limiting the change of the model parameters would hinder learning new tasks. This challenge is known as the stability-plasticity dilemma [MBB13]. The competition increases when we consider other CL requirements, such as restrictions in expanding the model capacity or having no or limited access to old data. Moreover, recent studies have demonstrated that deep neural networks lose their plasticity over time, making continual learning more challenging [DMS21, AZM+23].

Research in continual learning can be categorized into three main directions: rehearsal-based, architectural-based, and regularization-based strategies [WZSZ23, LAM+19]. The rehearsal-based strategy involves replaying old data while using a fixed-capacity model. Although it has been successful in achieving better performance compared to other strategies, it has limitations related to accessing old data and managing memory and computational costs. On the other hand, the regularization-based strategy relies on constraining the changes of network weights through regularization techniques and employs fixed-capacity models without access to previous data. However, many of these methods encounter issues such as the stability-plasticity dilemma, leading to suboptimal performance. The architectural-based strategy focuses on balancing forward and backward transfer by modifying the model architecture, typically by expanding the model capacity when learning a new task. Despite these efforts, achieving a balance that fulfills all CL requirements remains an ongoing and challenging problem.

1.2 Deep Reinforcement Learning

Deep reinforcement learning (DRL) is another form of learning on changing data distributions. In this paradigm, there are no training samples upfront. An agent (model) collects the samples incrementally during training while interacting with the environment. This represents an example of instance-incremental learning discussed above, where all samples belong to one task (environment) and arrive in batches of size 1. There is no labeled data, and the agent learns by trial and error. These introduce changes in the data distribution throughout training, as we will discuss next.

Formally, we consider a Markov decision process [Put14], $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$, defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a transition probability distribution $\mathcal{P}(s'|s,a)$ indicating the probability of transitioning to state s' after taking action a from state s, and a discounting factor $\gamma \in [0,1)$. An agent's behaviour is formalized as a policy $\pi : \mathcal{S} \to Dist(\mathcal{A})$; given any state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, the value of choosing a from s and following π afterwards is given by $Q^{\pi}(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)]$. The goal in RL is to find a policy π^* that maximizes this value: for any π, $Q^{\pi^*} := Q^* \geq Q^{\pi}$.

In deep reinforcement learning, the Q-function is represented using a neural network Q_θ with parameters θ. During training, an agent interacts with the environment and collects trajectories of the form $(s,a,r,s') \in \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S}$. These samples are typically stored in a *replay buffer* [Lin92], from which batches are sampled to update the parameters of Q_θ using gradient descent. The optimization performed aims to minimize the temporal difference loss [Sut88]: $\mathcal{L} = Q_\theta(s,a) - Q_\theta^{\mathcal{T}}(s,a)$; here, $Q_\theta^{\mathcal{T}}(s,a)$ is the bootstrap target $[\mathcal{R}(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_{\tilde{\theta}}(s',a')]$ and $Q_{\tilde{\theta}}$ is a delayed version of Q_θ that is known as the *target* network.

The number of gradient updates performed per environment step is known as the *replay ratio*. This is a key design choice that has a substantial impact on performance [VHHA19, FRA+20, KAM+21, NSD+22]. Increasing the replay ratio can increase the sample-efficiency of RL agents as more parameter updates per sampled trajectory are performed. However, prior works have shown that training agents with a high replay ratio can cause training instabilities, ultimately resulting in decreased agent performance [NSD+22].

One important aspect of reinforcement learning, when contrasted with classical supervised learning on a single task, is that RL agents train on highly non-stationary data, where the non-stationarity comes in a few forms [IFL+20]:

- **Input data non-stationarity:** The data the agent trains on is collected in an online manner by interacting with the environment using its current policy π; this data is then used to update the policy, which affects the distribution of future samples.

- **Target non-stationarity:** The learning target used by RL agents is based on its own estimate $Q_{\tilde{\theta}}$, which is changing as learning progresses.

The learning nature of DRL agents makes the training challenging. Most research focuses on providing algorithmic tricks to address training instability, such as the use of target networks, prioritized experience replay, multi-step targets, among others [HMVH+18]. Although these methods have shown success in enhancing performance, gaining a deeper understanding of the behavior of deep neural networks in the context of RL learning dynamics is valuable for addressing the underlying causes of the challenges. For instance, recent works demonstrate that deep neural networks suffer from implicit underparameterization during RL training, affecting their expressivity and ability to adapt and fit new targets [LRD21, KAGL21]. Thus, similar to the observations made in continual learning, RL agents also experience a loss of plasticity over time [LZN+23]. Another work shows deep networks' tendency to overfit early samples affecting the rest of the learning process. These observations may also contribute to the fact that deep networks require a very long training time to achieve good performance [VBC+19], which results in significant memory and computational costs.

1.3 Research Questions

Enabling deep neural networks to adapt to changing data distributions is vital for numerous applications; however, achieving this objective poses several challenging problems, as previously discussed. In this thesis, we aim to improve the ability of deep networks to learn continually while enhancing memory and computational efficiency. To this end, we address the main challenges in this learning regime, including catastrophic forgetting, forward transfer, adaptability, and the loss of expressivity and plasticity. We investigate these challenges through the lens of learned representations, training regimes, and the utilization of the model capacity. More specifically, we address the following research questions:

(Q1) **How to learn new tasks representations without affecting old ones?**

One of the main obstacles in continual learning is that optimizing the network weights for a new task overwrites the previously learned one, especially when a fixed-capacity model is used, and there is no access to past data. Learning effective representation without forgetting is crucial for such a learning paradigm.

(Q2) **How to make use of previously acquired knowledge to facilitate learning a new task?**

While most research focuses on mitigating forgetting, it is equally crucial to consider the promotion of forward transfer, enabling faster learning of new tasks. By analyzing the characteristics of representations that facilitate learning new tasks, we can develop training approaches that effectively address both backward and forward transfer.

(Q3) **How to improve the adaptability to new samples and training efficiency to reduce training time?**

Adapting to changing data distributions may prolong training time, especially when a model learns by trial and error (i.e., DRL training). Exploring an alternative technique to train these models in memory and a computationally efficient manner while improving their adaptability has great value for developing models with improved capabilities and efficiency. Moreover, such advancements pave the way for training these models on edge devices, further extending their applicability.

(Q4) **What is the underlying reason behind the loss of adaptability over time during training?**

It has been observed that dense networks lose their expressivity and ability to fit new targets over time, despite being over-parameterized. By gaining insights into the behavior of deep neural networks in RL learning dynamics, we can better understand this phenomenon and develop improved DRL agents capable of learning effectively in non-stationary conditions.

1.4 Thesis Contributions and Outline

We conduct several studies to address the above-mentioned research questions. In Chapter 2, we present a novel approach that learns sparse representations for continual learning, effectively addressing forgetting without needing access to past data. Chapter 3 centers around studying the characteristics of representations that promote forward transfer in continual learning. To improve the memory and computational efficiency of training RL agents while achieving high performance, we introduce dynamic sparse training in DRL in Chapter 4. In Chapter 5, we analyze potential causes for the loss of expressivity in RL training and propose an effective method to mitigate this issue. We provide an outline of the thesis and a summary of the main contributions below:

1.4.1 Chapter 2

In Chapter 2, we introduce *SpaceNet* [SMP21b], the first dynamic sparse training approach that is designed to enable deep neural network models to learn continually while satisfying multiple continual learning desiderata. Instead of training a dense model on the continual sequence, we train a sparse subnetwork *from scratch* for each task. During training, the weights and the topology (distribution of the sparse connections) are optimized to learn sparse representations for the task. We show that sparse representations are very promising for this learning paradigm as they significantly reduce interference between tasks and forgetting. Moreover, the over-parameterized capacity of the model can be utilized efficiently to learn multiple tasks by using a sparse subnetwork for each one. Interestingly, we show that SpaceNet with a fixed capacity model outperforms methods that expand the model size over time. Furthermore, we demonstrate that SpaceNet outperforms other methods that learn dense representations by a big margin.

1.4.2 Chapter 3

The insights from SpaceNet show the effectiveness of sparse representations in reducing forgetting in the continual learning paradigm. Yet, there is a lack of understanding of the characteristics of representations useful for promoting forward transfer to future tasks. Is all the previously learned representation useful for learning new tasks? To study this question, in Chapter 3, we propose *SAM* [SMP21a], a Self-Attention Meta-learner for continual learning. SAM learns to calibrate the input features in each layer according to the input data. We show that representation useful for learning a new task is sparse. Not all previously learned knowledge might be useful for new tasks, and using dense representations reduces performance. Interestingly, we demonstrate that selecting the relevant knowledge while learning a new task is also beneficial for reducing forgetting, as the weights are also selectively updated based on their relevance. Finally, we show that having generic prior knowledge before learning a sequence of tasks improves performance.

Based on our findings from this chapter, we further propose another method based on sparse training that learns sparse representations to reduce forgetting while also selecting the potential features from the past to increase forward transfer [SMP22]. We demonstrate that the learned topology of each task plays an important role in addressing both objectives. Moreover, new tasks can reuse some learned components instead of allocating new sparse connections in each layer in the network. Additionally, we propose another effective sparse training method that can quickly select informative features from the input [SAPM22]. We optimize the sparse topology to give more attention to important features, estimated by their impact on the loss and the magnitude of their connected weights. We show that our proposed method can identify important features in a few training iterations, even for high-dimensional feature space.

1.4.3 Chapter 4

Training deep neural networks to adapt to changing data distributions is challenging. The challenge increases when a model learns a task with no true labels. Deep reinforcement learning (DRL) exemplifies this scenario. Achieving good performance with dense neural networks under these challenges requires significant training time, resulting in the consumption of prohibitive computational and memory resources [VBC+19]. To address these limitations, in Chapter 4, we introduce for the first time *dynamic sparse training* in this learning paradigm [SMM+22]. A sparse neural network is trained from scratch, and its topology is optimized during training to adapt to the changing data distribution. Our study conveys that dynamic sparse agents have a faster learning speed than dense agents. They can reach the performance of dense agents with a 40 - 50% reduction in the training steps. Comparing our method to training sparse agents with *fixed* topology reveals that the dynamic adaptability of the topology throughout training contributes to improved learning. Besides the improved performance achieved by dynamic sparse agents, they significantly reduce memory and computational costs during training and inference. Our contributions open the door for DRL agents to be trained and deployed on low-resource devices (e.g., mobile phones, tablets, and wireless sensor nodes) where the memory and computation power are strictly constrained.

In [GSD+23], we further investigate the effectiveness of dynamic sparse agents in more challenging scenarios where environments are extremely noisy. Our analyses demonstrate that dense agents fail to maintain their performance. In contrast, dynamic sparse agents are able to attend to task-relevant features, outperforming their dense counterparts by a large margin. Moreover, we show that sparse agents have faster adaptability when they face new tasks over time.

1.4.4 Chapter 5

The underlying reasons for the loss of the ability of dense models to adapt to new samples are still not fully understood. To tackle this question, in Chapter 5, we analyze the utilization of network capacity throughout the training process by tracking neuron activity [SACE23]. Our analyses reveal the existence of the *dormant neuron phenomenon* in DRL; an agent's network suffers from an increasing number of inactive neurons, thereby affecting network expressivity. Examining different components of RL training, we find that target non-stationarity is the main source of this phenomenon. We find that networks with a higher number of dormant neurons have less ability to fit targets, and increasing the model size does not help to solve this issue.

Leveraging our findings, we propose a simple and effective method (*ReDo*) that *Re*cycle *Do*rmant neurons periodically during training to maintain the utilization of network capacity. We demonstrate that *ReDo* significantly improves the agents' performance across different algorithms, network architectures, and network sizes. *ReDo* has higher effects when the network has a larger inactive capacity. In contrast to the recent approach that periodically *resets* some or all network layers [NSD+22], causing "forgetting" of previously learned knowledge, *ReDo* maintains this knowledge and effectively utilizes the network capacity. It achieves higher performance than *reset* while eliminating the required training time to recover performance after reset.

1.4.5 Chapter 6

In Chapter 6, we summarize the research questions investigated in this thesis, our proposed methods to tackle these questions, and the key findings. We further discuss the limitations of these methods. Finally, we conclude by outlining several promising avenues for future research that can build upon the insights and outcomes of this study.

1.5 Reading This Thesis

We have tried to make each chapter of this thesis self-contained to the greatest extent possible. As a result, it is not mandatory to read the chapters in a strictly sequential manner, although doing so might offer a more gradual introduction to the presented concepts.

Chapter 2

SpaceNet: Make Free Space for Continual Learning

In this chapter, we aim to explore the feasibility of simultaneously addressing multiple continual learning desiderata. We propose SpaceNet, a replay-free method that uses a fixed-capacity model to learn a sequence of tasks. Under these desirable restrictions, a model's capacity has to be utilized efficiently to fit multiple tasks without interference and forgetting. To address this objective, we introduce the first dynamic sparse training method for continual learning. SpaceNet trains a sparse subnetwork from scratch for each task and optimizes the sparse topology during training to learn sparse representations. We demonstrate that sparse representations effectively reduce forgetting for this paradigm, especially when past data is not accessible. Additionally, the utilization of sparse networks leads to substantial reductions in computational costs. We evaluate SpaceNet on a variety of well-known benchmarks and various network architectures, demonstrating its superiority over regularization-based methods that learn dense representations and architectural-based methods that expand the model gradually. Our source code is available at `https://github.com/GhadaSokar/SpaceNet`.

This chapter is integrally based on **Ghada Sokar**, Decebal Constantin Mocanu, and Mykola Pechenizkiy. *SpaceNet: make free space for continual learning.* Neurocomputing, 1–11, 2021.

2.1 Introduction

Deep neural networks (DNNs) have achieved outstanding performance across various computer vision and machine learning tasks [HZRS15, ZVSL18, CPK$^+$17, KT19, LDG$^+$17, GLO$^+$16, LWL$^+$17]. However, this remarkable success is achieved within a static learning paradigm, where models are trained on a large dataset specific to a single task and tested on data with similar distributions. This paradigm contradicts the dynamic nature of the real world, which evolves rapidly. Standard training of a neural network model on new data leads to significant performance degradation on previously learned knowledge, a phenomenon known as catastrophic forgetting [MC89]. Continual learning (CL) has emerged as a solution 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, focusing on alleviating catastrophic forgetting. These methods generally follow three strategies: rehearsal-based [SLKK17, MVE$^+$16], regularization-based [KPR$^+$17, ZPG17], and architectural-based strategies [RRD$^+$16, YYLH18]. Rehearsal-based methods are effective in preserving the performance of previously learned tasks by replaying their data during the learning of new tasks. However, these methods face limitations when access to old data is unavailable (e.g., due to privacy concerns) or when computational and memory constraints hinder retraining the data from all tasks. Regularization-based methods address forgetting by constraining the change in the important parameters of past tasks. These methods do not store past data and typically use fixed-capacity models. However, in challenging continual learning scenarios such as class incremental learning, they suffer from sub-optimal performance [KMA$^+$18, HLRK18, FG19, vdVT18]. Architectural-based methods dynamically expand the network capacity to reduce interference between tasks. They achieve a good performance at the expense of increasing the model capacity.

In this work, we propose a novel architectural-based method for the CL paradigm named SpaceNet. We address the challenging scenario class incremental learning, in which a model has a single-headed output layer, and the task identity is not accessible during inference. We also assume that the data from previous tasks is unavailable when learning new tasks. Unlike previous architectural-based methods, SpaceNet effectively utilizes the fixed capacity of a model instead of expanding the network.

SpaceNet is a new dynamic sparse training approach that continually learns tasks sequentially in an efficient manner. It trains a sparse neural network

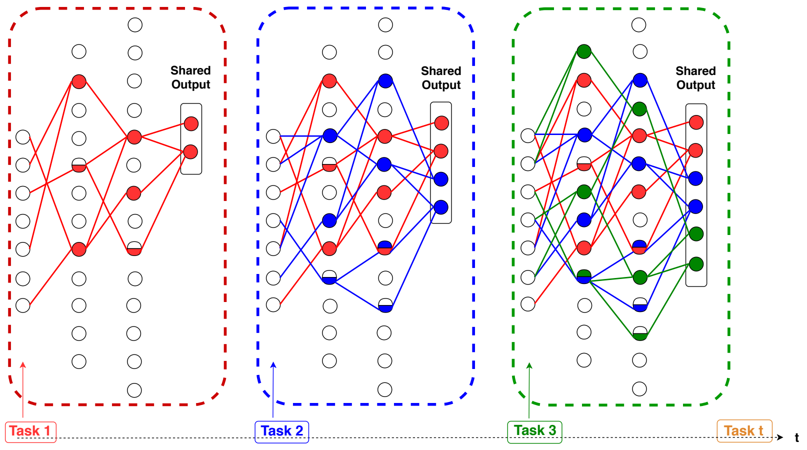


Figure 2.1: An overview of SpaceNet for learning a sequence of tasks. All tasks have the same shared output layer. The figure demonstrates the network states *after* learning each of the first three tasks in the sequence. When the model faces a new task t, sparse connections are allocated and compacted throughout the dynamic sparse training in the most important neurons for this task, making free space for learning more tasks. The fully filled circles indicate the neurons that are highly important and specific to task t, while the partially filled circles represent neurons that are less important and potentially shared with other tasks. Multiple colored circles represent the neurons that are used by multiple tasks. After learning task t, the corresponding weights are kept fixed.

from scratch for each task and optimizes the sparse topology during training to learn sparse representations. After learning the current task, the weights of its sub-network are kept fixed. In addition, we reserve some neurons, based on their importance, to be specific for that task, while others can be shared with other tasks. This allows future tasks to use the previously learned knowledge during their learning while reducing the interference between tasks. Figure 2.1 illustrates an overview of the proposed method. The usage of sparse networks in the continual paradigm has multiple advantages. The learned sparse representations reduce interference between tasks and forgetting. Utilizing a sparse subnetwork for a task leaves more space for future tasks. Additionally, training this sparse subnetwork significantly reduces computational costs. We note that the proposed dynamic sparse training approach uses readily available information during the standard training; no extra computational or memory overhead is needed to learn new tasks or remember the previous ones. We evaluate SpaceNet on the well-known benchmarks for CL: Split MNIST, Split Fashion-MNIST, CIFAR-10/100, and iCIFAR100. Our empirical results demonstrate that SpaceNet outperforms regularization-based methods by a big performance gap. Moreover, it achieves better performance than architectural-based methods without model expansion. Our main contributions are summarized as follows:

- We propose the first dynamic sparse training approach for continual learning, named SpaceNet. SpaceNet utilizes the fixed capacity of a model efficiently to learn a sequence of tasks. A sparse subnetwork is optimized for each task to learn sparse representations.
- We demonstrate the effectiveness of sparse representations in reducing the interference between tasks and forgetting, revealing its potential for the continual learning paradigm.
- We address the challenging scenario, class incremental learning, while considering more desiderata for continual learning, such as inaccessibility of previous tasks data, reducing memory and computational costs, and using fixed-capacity models.
- We demonstrate the effectiveness of the proposed approach on various benchmarks and network architectures, showing its superior performance to other regularization and architectural methods.

2.2 Background

2.2.1 Continual Learning

Problem Formulation. A continual learning problem consists of a sequence of tasks $\{1, 2, ..., t, ..., T\}$; where T is the total number of tasks. Each task t has its dataset D^t. A neural network model faces tasks one by one. All samples from the current task are observed before switching to the next task. Once the training for the current task is completed, its data becomes inaccessible. The goal is to learn the sequence of the tasks without forgetting any of them.

In this work, we address the class incremental learning scenario in CL. The task identity is not available at the test time. At any point, the model should classify the input to one of the classes learned so far, regardless of the task identity.

2.2.2 Dynamic Sparse Training (DST)

DST is a line of research that aims to train a sparse neural network from scratch. The idea was introduced in [MMN+16] for single-task unsupervised learning. In recent years, DST proved its success in achieving the same performance as dense neural networks in single-task standard supervised/unsupervised learning while having much faster training speed and much lower memory requirements [MMS+18, BKML18, DZ19, EGM+20, JZR+19, MW19]. In this training regime, the training starts with a random sparse neural network, and during training, the topology of the network dynamically evolves. The topology evolves through drop-and-grow cycles in which a fraction of sparse connections are dropped, and the same fraction is grown among other neurons. The criteria used for dropping and growing determine the learned topology.

Works from [EGM+20, MW19] also showed that sparse training performs better than iteratively pruning a pre-trained dense model and static sparse neural networks. Moreover, [LVdLY+21] demonstrated a plenitude of sparse subnetworks with very different topologies that achieve the same performance.

2.3 Related Work

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

Regularization methods preserve the knowledge of old tasks by adding regularization terms in the loss function to constrain the change in important weights of past tasks. Multiple approaches have been proposed such as: Elastic Weight Consolidation (EWC) [KPR+17], Synaptic Intelligence (SI) [ZPG17], and Memory Aware Synapses (MAS) [ABE+18]. Each of these methods proposes an estimation of weight importance relative to the learned task. During the training of a new task, changes to the important weights associated with previous tasks are penalized. Learning Without Forgetting (LWF) [LH17] is another regularization method that limits the change of old tasks performance by using a distillation loss [HVD15a]. 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 old tasks stable. Despite that regularization methods are suitable for situations where one can not access previous data, their performance degrades much in the class incremental learning scenario [KMA$^+$18, HLRK18, FG19, vdVT18].

Rehearsal methods replay old tasks data along with the current task data to mitigate the catastrophic forgetting of old tasks. Deep Generative Replay (DGR) [SLKK17] 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. [MVE$^+$16]. Other methods combine the rehearsal and regularization strategies, such as iCaRL [RKSL17]. iCaRL uses a distillation loss along with an exemplar set to impose output stability on old tasks. The main drawbacks of rehearsal methods are (1) the memory overhead of storing old data or a model to generate them, (2) the computational overhead of retraining the data from all previous tasks, and (3) the unavailability of the previous data in some cases.

Architectural methods modify the model architecture in different ways to make space for new knowledge while maintaining the old one. PathNet [FBB$^+$17] uses a genetic algorithm to find which parts of a network can be reused for learning new tasks. While learning new tasks, the weights of old tasks are kept frozen. This approach has high computational complexity. CLNP [GKC19] uses a simpler way to find the parts that can be reused in a network by calculating the average activity of each neuron. The least active neurons are reassigned for learning new tasks. Progressive Neural Network (PNN) [RRD$^+$16] is a combination of network expansion and parameter freezing. Catastrophic forgetting is prevented by instantiating a new neural network model for each task while keeping previously learned networks frozen. New networks can take advantage of previous ones through inter-network connections. In this method, the number of model parameters keeps increasing over time. Copy-Weights with Reinit (CWR) [LM17] uses a fixed-capacity model but has limited applicability and performance. Fixed shared parameters are used for all tasks, while the output layer is extended and trained when the model faces a new task. Dynamic Expandable Network (DEN) [YYLH18] keeps the network sparse via weight regularization. A subset of previously learned weights is jointly retrained with the new task weights. This subset is chosen regardless of its importance to old tasks. If the performance of old tasks degrades much, they try to restore it by node duplication.

Recent methods have been proposed based on sparse neural networks [MDL18, ML18]. PackNet [ML18] prunes the unimportant weights after learning each task and retrains the network to free some connections for later tasks. A mask is saved for each task to select the connections that will be used during the test

Table 2.1: Comparison between different CL methods in terms of their fulfillment of different continual learning desiderata.

Strategy	Method	Fixed Model Capacity	Memory Efficiency	Fast Training	Old Data Inaccessibility	Old Tasks Performance
Regularization	EWC [KPR+17]	√	√	√	√	×
	SI [ZPG17]	√	√	√	√	×
	LWF [LH17]	√	√	√	√	×
Rehearsal	iCaRL [RKSL17]	√	×	√	×	√
	DGR [SLKK17]	√	×	×	√	√
Architectural	PNN [RRD+16]	×	×	√	√	√
	PackNet [ML18]	√	×	×	√	√
	DEN [YYLH18]	×	×	×	√	√
	SpaceNet (Ours)	√	√	√	√	√

time. Instead of learning the network weights, Piggyback [MDL18] learns a mask for each task to select some weights from a pre-trained dense network. These methods require the task identity during inference to activate the corresponding mask to a test input. Our method is different from these ones 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 computational overhead of iterative pruning and fine-tuning the network after learning each task, and (3) we aim to learn sparse representations on 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 desiderata for CL as argued by [SCL+18, FG19]. Table 2.1 provides a comparison of various algorithms in terms of their fulfillment of different continual learning desiderata. The CL 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 old data or a specific mask for each task. Lastly, the algorithm should not assume the availability of old data.

2.4 SpaceNet

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

The main objectives of our approach are: (1) utilizing the fixed capacity of a model efficiently by learning each task in a compact space, leaving room for future tasks, (2) learning effective representations to reduce interference between

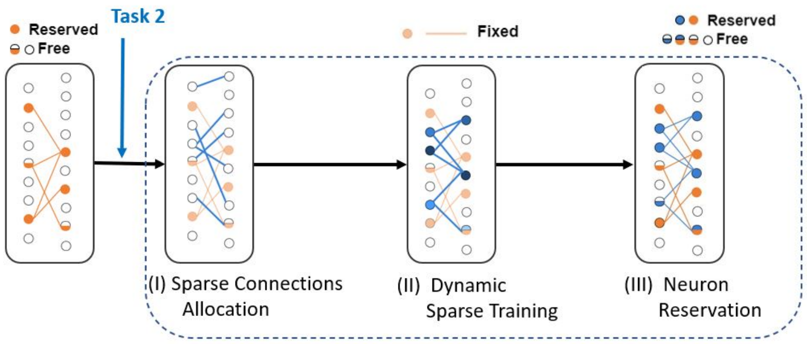

Reserved
Free
Task 2
Fixed
Reserved
Free
(I) Sparse Connections Allocation
(II) Dynamic Sparse Training
(III) Neuron Reservation

Figure 2.2: An overview of the main three steps of SpaceNet illustrated on one layer. (I) When a new task arrives, new sparse connections are allocated among non-reserved (free) neurons. (II) During our proposed dynamic sparse training, the topology is optimized by redistributing the connections among the important neurons for the current task (higher importance is represented via darker colors. (III) After training, a subset of the most important neurons is reserved, and the weights of the subnetwork are fixed.

tasks and forgetting without replaying old data, and (3) avoiding adding high computational and memory overhead for learning new tasks.

To address these objectives, we propose a new training strategy for continual learning based on dynamic sparse training. Unlike previous methods in which the weights of a dense model are optimized on each task, we dynamically train a sparse subnetwork from scratch for each task. The goal of this dynamic sparse training is to learn sparse representations for each task. We hypothesize that these sparse representations are more effective in reducing the interference between tasks than the typical dense representations. SpaceNet can be divided into three main steps: (1) Connections allocation, (2) Task training, and (3) Neurons reservation, as illustrated in Figure 2.2. When a model faces a new task, new sparse connections are randomly allocated between a selected number of non-reserved neurons in each layer. The learning of this task is then performed using our proposed dynamic sparse training method.

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 $\epsilon$, rewiring fraction $r$
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
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
```

During training, the initial distribution of the connections is evolved, and more connections are grouped in the important neurons for that task. After training, the most important neurons from the initially selected ones are reserved to be specific to this task, while the other neurons are shared between tasks. The details of our proposed approach are illustrated in Algorithm 1. Next, we will provide the details of each of the main steps of SpaceNet.

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 a 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 $\mathbf{h}_{l-1}^{sel}$

Algorithm 2 Connections allocation

```
1: Require: number of selected neurons $sel_l^t$, sparsity level $\epsilon$
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 $\epsilon$ between $\mathbf{h}_{l-1}^{sel}$ and $\mathbf{h}_l^{sel}$
6:     $W_l \leftarrow W_l \cup W_l^t$
7: end for
```

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.

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

$$\min_{W^t} \mathcal{L}(W^t; D^t, W^{1:t-1}), \tag{2.1}$$

where $\mathcal{L}$ is the loss function and $W^{1:t-1} = \mathbf{W} \setminus W^t$ are the parameters of previous tasks. The parameters $W^{1:t-1}$ are kept fixed during learning task t. During the training process, the distribution of sparse connections of task t is adaptively changed, ending up with sparse connections compacted in fewer neurons. Algorithm 3 shows the details of our proposed dynamic 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 dynamic 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:

$$\mathcal{L}(\mathbf{W}^{i+1}) - \mathcal{L}(\mathbf{W}^i) \approx \sum_{j=0}^{m-1} \frac{\partial \mathcal{L}}{\partial W_j^i}(W_j^{i+1} - W_j^i) = \sum_{j=0}^{m-1} I_{i,j}, \tag{2.2}$$

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 [LLZY19]. 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:

$$\Omega_l^j = \sum_{i=0}^{iter} |I_{i,j}|, \tag{2.3}$$

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. The importance $a_l^{(i)}$ of the neuron i in layer l is estimated by the summation of the importance of incoming connections of that neuron as follows:

$$a_l^{(i)} = \sum_{j=0}^{C_{in}-1} \Omega_l^j, \tag{2.4}$$

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

$$G_l = \mathbf{a}_{l-1}\mathbf{a}_l^T. \tag{2.5}$$

Let the number of growing connections in layer l be k_l, the top-k_l positions which contain the highest values in G_l and zero-value in W_l are selected for growing the new connections.

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 2.3. 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.

Algorithm 3 Dynamic sparse training

```
1: Require: loss function $\mathcal{L}$ , training dataset $\mathcal{D}^t$, rewiring fraction $r$
2: for each training epoch do
3:     perform standard forward pass through the network parameters $\mathbf{W}$
4:     update parameters $W^t$ according to Equation 2.1
5:     for each sparse parameter $W_l^t$ do
6:         $\widetilde{W_l^t}$ ← sort $W_l^t$ based on the importance $\Omega_l$ in Equation 2.3
7:         $(W_l^t, k_l)$ ← drop $(\widetilde{W_l^t}, r)$ // Remove the weights with smallest importance
8:         compute $\mathbf{a}_{l-1}$ and $\mathbf{a}_l$ from Equation 2.4 // Neurons importance for task t
9:         $G_l \leftarrow \mathbf{a}_{l-1}\mathbf{a}_l^T$
10:        $\widetilde{G_l}$ ← sortDescending($G_l$)
11:        $Gpos$ ← select top-$k_l$ positions in $\widetilde{G_l}$ where $W_l$ equals zero
12:        $W_l^t$ ← grow($W_l^t$,$Gpos$) // Grow $k_l$ zero-initialized weights in $Gpos$
13:    end for
14: end for
```

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 2.4
4:     $\widetilde{\mathbf{a}_l}$ ← sortDescending($\mathbf{a}_l$)
5:     $\mathbf{h}_l^{t_{spec}}$ ← 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
```

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 2.4. These neurons become specific to the current task, meaning no more connections from other tasks will go into these neurons. The other neurons in $\mathbf{h}_l^{sel}$ still exist in the free list $\mathbf{h}_l^{free}$ and could be shared by future tasks. Algorithm 4 describes the details of the neurons reservation process.

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 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 [HH49] 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 and 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 redistributes the connections between strong neurons.

2.5 Experiments

In this section, we evaluate our proposed method on the well-known benchmarks for continual learning and compare it against methods from different continual learning strategies.

2.5.1 Datasets

- Split MNIST [ZPG17]. It consists of five tasks. Each task contains two consecutive classes from the MNIST dataset.

- Split Fashion-MNIST [XRV17, FG19]. The images show individual articles of clothing. Same as Split MNIST, it consists of five tasks. Each task has two consecutive classes of Fashion-MNIST.

- CIFAR-10/100 [ZPG17]. The benchmark is constructed from CIFAR-10 and CIFAR-100 datasets [KH+09]. It has six tasks. The first task contains the full dataset of CIFAR-10, while each subsequent task contains 10 consecutive classes from CIFAR-100. Note that the number of samples across the first task and the rest is not equal, with the first task having a $10\times$ larger number of samples per class.

2.5.2 Architectures

To train on Split MNIST and Split Fashion MNIST, we use a multilayer perceptron network with two hidden layers. Each layer has 400 neurons with ReLU activation.

For CIFAR-10/100, we follow the architecture used by [ZPG17, ML19] for a fair and direct comparison. It consists of 4 convolutional layers (32-32-64-64 feature maps). The kernel size is 3×3. Max pooling layer is added after every two 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. We replace the dropout layers with batch normalization [IS15].

2.5.3 Experimental Details

Split MNIST and Split Fashion-MNIST. 10% of the network weights are used for all tasks (2% for each task). The rewiring fraction r equals 0.2. Each task in Split MNIST and Split Fashion-MNIST is trained for 4 epochs and 20 epochs, respectively. We use a batch size of 128. The network is trained using stochastic gradient descent with a learning rate of 0.01 and cross-entropy loss. 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 a random search. The experiment is repeated 10 times with different random seeds.

CIFAR-10/100. The model is optimized using stochastic gradient descent with a learning rate of 0.1 and cross-entropy loss. 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 the number of feature maps in this layer, excluding the specific neurons in that 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 a random search.

2.5.4 Evaluation Metrics

To evaluate different CL requirements, we assess two metrics:

1. Average Accuracy (ACC). ACC is the average classification accuracy across all tasks, calculated at the end of learning the whole sequence.
2. Backward transfer (BWT) [LPR17]. BWT measures the influence of learning new tasks on the performance of previous tasks. Large *negative* BWT indicates catastrophic forgetting.

Formally, the ACC and BWT are calculated as follows:

$$\begin{aligned} ACC &= \frac{1}{T}\sum_{i=1}^{T} R_{T,i}, \\ BWT &= \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}, \end{aligned} \tag{2.6}$$

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

2.5.5 Experimental Results

Split MNIST and Fashion MNIST

Table 2.2 shows the average accuracy (ACC) and the backward transfer (BWT) of different well-known approaches. The experiments reveal the following findings:

SpaceNet outperforms regularization-based methods by a big gap. As illustrated in the table, regularization methods fail to maintain the performance of previously learned tasks in the class incremental learning scenario. They have the lowest BWT performance. In contrast, SpaceNet manages to keep the performance of previously learned tasks, causing a much lower negative backward transfer. In terms of ACC, it outperforms regularization-based methods by 55.52% and 44.87% on Split MNIST and Split Fashion MNIST, respectively.

SpaceNet consumes less model capacity than DEN and achieves higher performance. We compare our method to the DEN algorithm, which is the most related to our work, both being architectural strategies. As discussed in the related work section, DEN keeps the connections sparse by regularization and restores the drift in old tasks' performance using node duplication. The connections are remarked with a timestamp (task identity). At inference, the task

Table 2.2: ACC and BWT on Split MNIST and Split Fashion MNIST using different approaches. Results for regularization and rehearsal methods on Split MNIST are adopted from [vdVT18, HLRK18].

Strategy	Method	Split MNIST		Split Fashion MNIST	
		ACC (%)	BWT (%)	ACC (%)	BWT (%)
Regularization	EWC	**20.01** ± 0.06	**-99.64** ± 0.01	19.47 ± 0.98	-99.13 ± 0.39
	SI	19.99 ± 0.06	-99.62 ± 0.11	19.93 ± 0.01	-99.08 ± 0.51
	MAS	19.52 ± 0.29	-99.73 ± 0.06	**19.96** ± 0.01	**-98.82** ± 0.10
Rehearsal	DGR	90.79 ± 0.41	-9.89 ± 1.02	73.58 ± 3.90	-32.56 ± 3.74
	iCaRL	**94.57** ± 0.11	**-3.27** ± 0.14	**80.70** ± 1.29	**-10.39** ± 1.97
Architectural	DEN	56.95 ± 0.02	-21.71 ± 1.29	31.51 ± 0.04	-47.94 ± 1.69
	SpaceNet	**75.53** ± 1.82	**-15.99** ± 1.83	**64.83** ± 0.69	**-23.98** ± 1.89

identity is required to select only the parameters trained up to this task identity. This implicitly means that T different models are obtained using DEN, where T is the total number of tasks. To make the comparison, we adapt the official code provided by the authors to work on the class incremental learning 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 for each timestamp t. The class with the highest probability from all models is the final prediction. As shown in the table, SpaceNet obtains better performance and lower forgetting. It achieves an accuracy of 75.53% on Split MNIST, outperforming DEN by 18.5%. While the accuracy of DEN degrades much on the more complex benchmark Split Fashion MNIST, SpaceNet maintains a good performance, reaching ACC of 64.83% and BWT of -23.98%. Furthermore, in our investigation, we observed that DEN introduces an increase of approximately 35 neurons per layer in Split MNIST. In contrast, SpaceNet retains unused neurons within the initially allocated capacity. Specifically, SpaceNet has 92 and 91 unused neurons in the first and second hidden layers, respectively. Similarly, in Split Fashion MNIST, DEN expands each hidden layer by 37 neurons, while SpaceNet maintains 90 and 93 unused neurons in the first and second hidden layers, respectively.

Replaying old tasks' data maintains their performance. Replaying the data from previous tasks while learning a new task mitigates catastrophic forgetting, achieving the highest BWT. However, retraining old tasks data has the cost of requiring additional memory for storing the data or the generative model. Making rehearsal methods resource-efficient is still an open research problem. The results of SpaceNet in terms of both ACC and BWT are 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

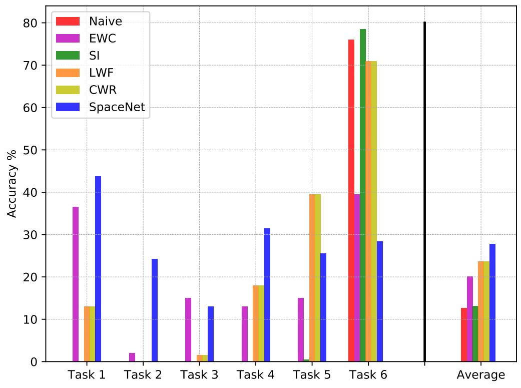


Figure 2.3: Accuracy on each task of CIFAR-10/100 benchmark for different CL approaches after training the last task. Results for other approaches are adopted from [ML19]. Task 1 is the full dataset of CIFAR10, while task 2 to task 6 are the first 5 tasks from CIFAR100. Each task contains 10 classes. The absence of accuracy for certain tasks in some methods indicates zero accuracy value. 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 tasks, unlike other methods that have high performance on the last task but completely forget some other previous tasks.

connections than DGR.

CIFAR-10/100

Figure 2.3 shows the accuracy of each task of CIFAR-10/100 after training all tasks using different popular CL methods. The results of other algorithms are extracted from [ML19] and re-plotted. The "Naive" algorithm is defined by the authors as simple finetuning where there is no mechanism for addressing forgetting other than early stopping. As shown in the figure, SI totally fails to remember all old tasks, and the model has a good performance only 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 all tasks. It outperforms all the other algorithms in terms of average accuracy. In addition, the standard deviation over all tasks accuracy is much (a few times) smaller than the standard deviation of any other state-of-the-art method. This means that the model is not biased towards the final learned task, and the accuracy of all learned tasks is close to each other. This highlights the robustness of SpaceNet and its strong capabilities to remember old tasks.

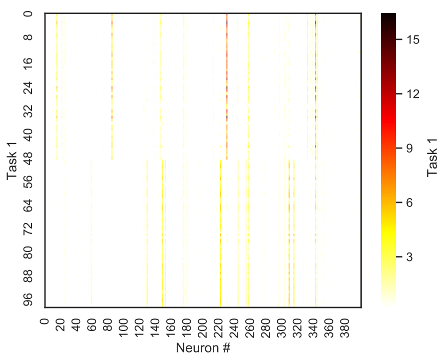
This experiment shows that SpaceNet utilizes the small available capacity. Yet, the model capacity could reach its limit after learning a certain number of tasks. In this case, we can allocate more resources (units) to the network to fit more tasks since we have fully utilized the existing ones. To show this case on the same architecture and settings, we increase the number of sparse connections allocated for each task in the second layer to 16.5% of the layer weights. This leads the second layer to approximately reach its maximum capacity after learning the first five tasks. When the model faces the last task of the CIFAR-10/100 benchmark, we allocate 8 new feature maps in the second and third convolutional layers. SpaceNet continues to learn this task. The average accuracy achieved in this experiment equals 27.89 ± 0.84.

2.6 SpaceNet Analysis

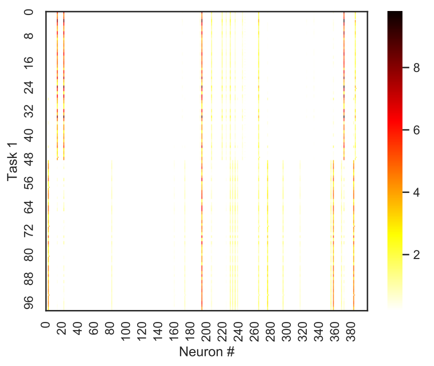
2.6.1 Learned Representation

We analyze the representations learned by SpaceNet via visualizing the activations of the two hidden layers of the network. We conduct this analysis on the Split MNIST benchmark. After learning the first task, we examine the representations of a randomly selected subset of the test set from this task.

Figure 2.4 shows the representations of 50 random samples from class 0 and another 50 samples from class 1. As illustrated in the figure, the representations learned by SpaceNet are highly sparse. A small percentage of activations is used to represent an input. These observations demonstrate that the proposed sparse training algorithm not only optimizes the network's capacity effectively by employing a sparse network for each task but also generates sparse representations. These sparse representations play a crucial role in reducing interference between tasks. It is worth highlighting that our findings are aligned with the earlier work conducted by [Fre91]. French argued that catastrophic forgetting arises as a result of the overlap in representations between different tasks and that employing semi-distributed representations can mitigate the issue of catastrophic forgetting.



(a) First hidden layer.



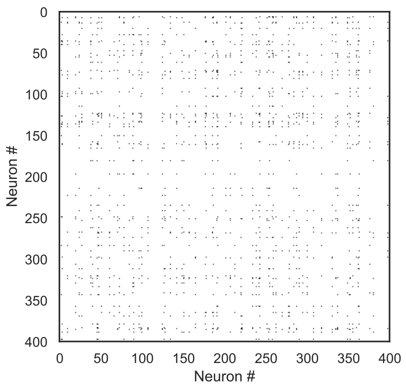
(b) Second hidden layer.

Figure 2.4: Heatmap of the first and second hidden layers activations after forwarding a subset of 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.

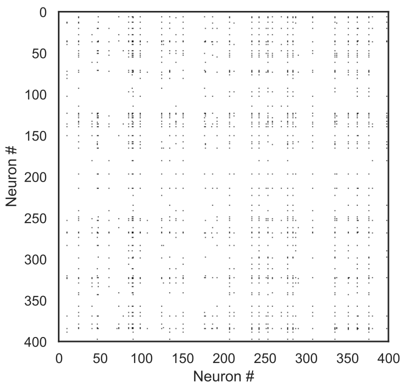
Table 2.3: Effect of learning sparse representation in the continual learning paradigm.

Method	Split MNIST		Split Fashion MNIST	
	ACC (%)	BWT (%)	ACC (%)	BWT (%)
Static-SparseNN	61.25 ± 2.30	-29.32 ± 2.80	56.80 ± 2.30	-29.79 ± 2.22
SpaceNet	**75.53** ± 1.82	**-15.99** ± 1.83	**64.83** ± 0.69	**-23.98** ± 1.89

To further investigate the role of learning sparse representation in performance, we compare SpaceNet with a baseline we dubbed as “Static Sparse-NN”. This baseline allocates a sparse subnetwork for each task using the same strategy as in SpaceNet. Yet, the initially allocated topology is kept fixed throughout training. We conducted this analysis on Split MNIST and Split Fashion MNIST. As shown in Table 2.3, redistributing the connections among the important neurons and learning sparse representations via our proposed dynamic sparse training method increases performance by a good margin. The average accuracy is increased by 14.28% and 8% on Split MNIST and Split Fashion MNIST, respectively. While the backward transfer is increased by 13.3% and 6%.



(a) Initial connections.



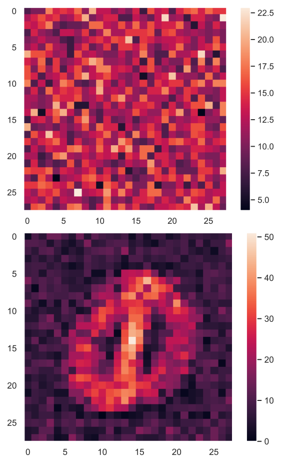
(b) Learned connections.

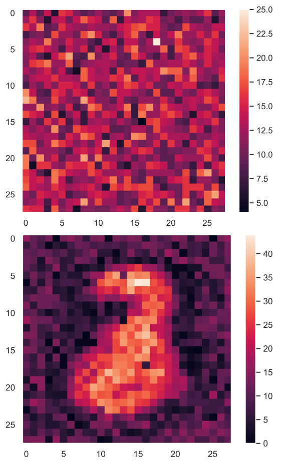
Figure 2.5: Connections distribution between the two hidden layers for one task of the Split MNIST benchmark. The initial random distribution of the connections on the selected neurons (a) and the learned distribution after training (b). The connections are compacted in some of the neurons.

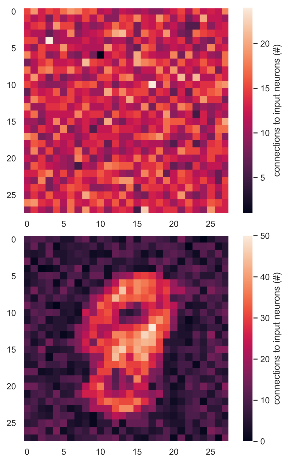
2.6.2 Learned Topology

We next investigate the effect of drop-and-growing cycles of our proposed dynamic sparse training method on the randomly initialized sparse topology of a task. To this end, we analyze the distribution of the connections between the first and the second hidden layers before and after training a task. We conducted this analysis on the second task of the Split MNIST benchmark. As shown in Figure 2.5a, the sparse connections are initially randomly distributed among a set of chosen neurons. However, as training progresses, the optimization of the topology leads to a redistribution of connections, grouping them within a smaller number of neurons which are important for this task, as illustrated in Figure 2.5b. This creates space to allocate new subnetworks for future tasks.

We further analyze whether the new distribution of the connections is among the important neurons for a task. To qualitatively evaluate this, we examine the number of outgoing connections after training from each neuron in the input layer. Ideally, neurons with a high number of connections should correspond to the important pixels in the image. We compare learned distribution by SpaceNet with the Static-SparseNN baseline. Figure 2.6 shows the distribution of each task topology over the 784 input neurons, reshaped to 28×28 as the original


(a) Task 1 (0 or 1).


(b) Task 2 (2 or 3).



(c) Task 5 (8 or 9).

Figure 2.6: Visualization of the number of outgoing connections from each input neuron for three different tasks in Split MNIST in the case of the Static-SparseNN baseline (top) and SpaceNet (bottom). SpaceNet redistributes the connections in the important neurons that identified the digits in each task.

input image. As we can see, SpaceNet reallocates the connections towards the areas that correspond to digit identification, while there are only a limited number of connections in the background.

2.6.3 Memory Efficiency Improvement

Constraining the memory usage is necessary when learning a large number of tasks continuously. We compare the memory consumed by neural network models used by each method by measuring the total number of model parameters. As illustrated in Figure 2.7, among the methods studied, the rehearsal-based approach, DGR, has the highest memory requirement. This is due to the need for both a classification model and a generative model to generate replayed data for previous tasks. On the other hand, regularization-based methods employ dense neural network models, resulting in a higher number of parameters compared to the architectural-based method, DEN, which employs L1 regularization to learn sparse connections. In contrast, SpaceNet leverages a highly sparse network for each task, with at least one order of magnitude fewer parameters than

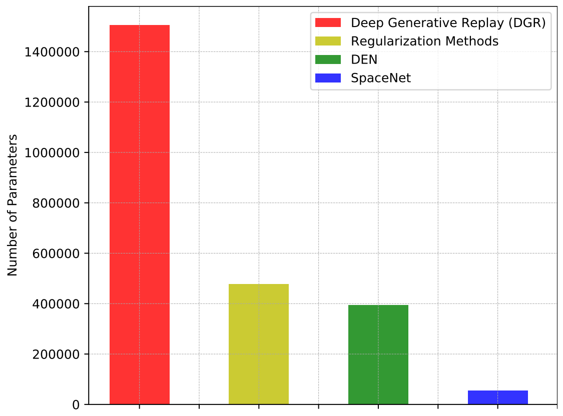


Figure 2.7: Comparison between SpaceNet and other CL methods on split MNIST in terms of model size.

Table 2.4: Comparison between different strategies across different continual learning desiderata.

Strategy	Method	Old task data	Extra memory	Model expansion
Regularization	EWC SI MAS	No	No	No
Rehearsal	DGR iCaRL	Yes	Yes	No
Architectural	DEN	No	Yes	Yes
	Static-SparseNN SpaceNet	**No**	**No**	**No**

any other studied method examined in the study.

Table 2.4 shows a comparison between different methods in terms of other requirements for CL. Regularization methods satisfy many desiderata of CL while suffering from catastrophic forgetting as illustrated earlier. Rehearsal methods require extra memory to store the old data or the generative model. SpaceNet is able to compromise between performance and other requirements that are not even satisfied by other architectural methods.

Table 2.5: ACC and BWT on the iCIFAR100 benchmark using two different architectures.

		CNN		WRN	
Strategy	Method	ACC (%)	BWT (%)	ACC (%)	BWT (%)
Regularization	EWC	13.65 ± 0.15	-64.38 ± 0.71	16.09 ± 0.29	-73.10 ± 1.11
	SI	14.45 ± 0.11	-66.47 ± 0.49	16.75 ± 0.30	-77.75 ± 0.90
	MAS	14.51 ± 0.22	-66.61 ± 0.31	16.51 ± 0.20	-76.82 ± 0.19
Architectural	SpaceNet	**28.11** ± 0.74	**-45.62** ± 0.88	**34.10** ± 0.92	**-34.18** ± 0.91

2.6.4 Utilizing Model Capacity

Can increasing the model size reduce forgetting? To study this question, we evaluate the performance of SpaceNet and the regularization methods using two models: the conventional CNN network detailed in Section 2.5.2 and a more modern architecture Wide Residual Networks (WRN) [ZK16]. We use WRN-28-10, with a depth of 28 and a widen factor of 10. We perform this analysis on a more challenging benchmark iCIFAR-100 [RKSL17], which consists of 5 tasks, each has 20 classes from CIFAR-100.

Table 2.5 shows the average accuracy and the backward transfer using the two architectures. As illustrated in the table, SpaceNet manages to utilize the available capacity of the conventional CNN architecture achieving higher accuracy than the regularization methods by 13.5%. SpaceNet is also more robust to forgetting, BWT is higher than the regularization strategy by 19%. Our results also show that using a larger network, WRN, does not help the regularization strategy to alleviate the catastrophic forgetting problem. A small increase is gained in the average accuracy, probably due to achieving higher performance in the last task using the larger model, while the forgetting (negative backward transfer) is increased by around 11%. On the other hand, SpaceNet takes advantage of the larger capacity. The ACC of SpaceNet is increased by 6%, and the forgetting is decreased by 11.5%.

2.7 Conclusion and Future Work

In this chapter, we proposed SpaceNet, a new technique for deep neural networks to learn a sequence of tasks in the continual learning paradigm. SpaceNet learns each task using a sparse subnetwork, leaving a space for other tasks to be learned by the model. We proposed a dynamic sparse training algorithm that optimizes the sparse topology of each task throughout training to learn sparse representation. We addressed the challenging 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, CIFAR-10/100, and iCIFAR100.

We demonstrated that SpaceNet addresses multiple CL desiderata: (1) it reduces catastrophic forgetting, outperforming the regularization and architectural-based methods by a big margin, (2) the achieved performance is obtained using a fixed capacity model without network expansion, (3) it requires much less memory and computational costs than rehearsal based methods, (4) and it does not rely on old tasks data.

Our analyses illustrated the effectiveness of our proposed dynamic sparse training method in learning a sparse topology that allocates the connections in the important neurons for a task. Moreover, we showed the importance of learning sparse representations in reducing interference between tasks. Unlike other methods that have a high performance on the last learned task only, SpaceNet utilizes the capacity of the model efficiently among all seen tasks.

Despite the success of SpaceNet in reducing forgetting, there is no explicit mechanism to promote forward transfer from past tasks. Addressing forward transfer would further allow utilizing the capacity more efficiently by making use of the learned connections of past tasks instead of allocating new sparse connections for each new task. Moreover, combining SpaceNet with a memory-efficient replay method could improve the performance even further while being efficient.

Chapter 3

Self-Attention Meta-Learner for Continual Learning

Although catastrophic forgetting is one of the main challenges in continual learning, enabling forward transfer from past tasks to the current one is another crucial component in this learning paradigm. In this chapter, we focus on investigating the characteristics of the representations that facilitate learning new tasks. Many approaches in continual learning begin by training a model from scratch, optimizing it at each step to learn task-specific representations without considering their generalization to future tasks. Additionally, each task utilizes all knowledge acquired from previous tasks, even though certain parts of this knowledge may not be directly relevant to the current task. This leads to task interference, particularly when the data from previous tasks is inaccessible. We aim to investigate these two limitations. To this end, we propose a new method, named Self-Attention Meta-Learner (SAM), which learns prior knowledge for continual learning that permits learning a sequence of tasks. SAM incorporates an attention mechanism that learns to select the relevant representations for each future task. Each task builds a specific representation branch on top of the selected knowledge, avoiding interference between tasks. Our analysis demonstrates that useful representations for future tasks are sparse. Moreover, by learning specific representations on top of the selected knowledge from the past, we can perform better

This chapter is integrally based on **Ghada Sokar**, Decebal Constantin Mocanu, and Mykola Pechenizkiy. *Self-attention meta-learner for continual learning*. Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), 2021.

than several state-of-the-art methods. Finally, we demonstrate that popular existing continual learning methods gain a performance boost when incorporated with SAM. Our source code is available at `https://github.com/GhadaSokar/Self-Attention-Meta-Learner-for-Continual-Learning`.

3.1 Introduction

Continual learning aims to build machines that mimic human learning. The main characteristics of human learning are (1) humans never learn in isolation, (2) they build on top of the learned knowledge in the past instead of learning from scratch, (3) and acquiring new knowledge does not lead to forgetting the past knowledge. These capabilities are crucial for autonomous agents interacting in the real world [PKP+19, LLS+20]. For instance, systems like chatbots, recommendation systems, and autonomous driving interact with a dynamic and open environment and operate on non-stationary data. These systems should quickly adapt to new situations with the help of previous knowledge, acquire new experiences, and retain previously learned experiences.

Deep neural networks (DNNs) have achieved outstanding performance in different areas such as visual recognition, natural language processing, and speech recognition [ZVSL18, CPK+17, KT19, LDG+17, GLO+16, LWL+17]. However, DNNs are effective in learning single tasks (static environments). Meanwhile, the performance degrades when a network is trained on non-stationary data. Continual learning (CL) is a research area that addresses this problem and aims to provide neural networks with continual learning capability.

The continual learning paradigm has many desiderata, including but not limited to avoiding forgetting past tasks, allowing forward transfer to future tasks, having a bounded system size, minimal or no access to past data, and not relying on task labels during inference (see [LLS+20, SCL+18, DRLFM18] for the complete list). Since these desiderata are competing with each other, most of the previous methods target subsets of them, and the main focus is on addressing forgetting. In this work, we shift the focus to some desiderata which are not widely addressed to the best of our knowledge and are discussed in [CL18]. The first one is the necessity of having a good quantity of prior knowledge to facilitate learning new tasks. However, in most previous approaches, the model starts from randomly initialized parameters, and then the parameters are optimized to achieve the highest performance on the first task. The knowledge gained from this task may contain only a bit or even no useful knowledge for future tasks [CL18]. Second, selecting the useful and relevant parts only from previ-

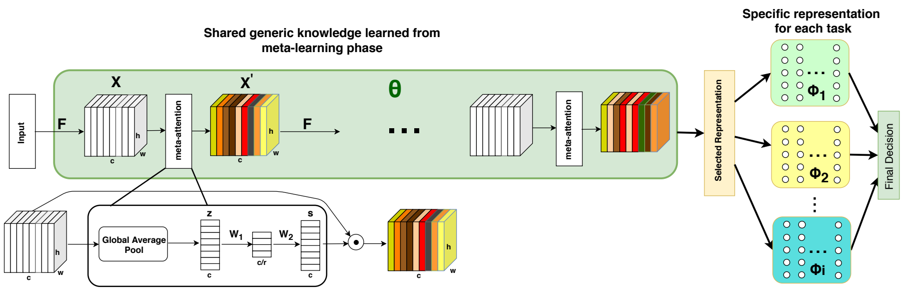


Figure 3.1: An overview of our proposed method, SAM. The network consists of two sub-networks. The first sub-network, parameterized by θ, is trained using an optimization-based meta-learning algorithm to learn the prior generic knowledge. This learned representation is shared between all tasks. A self-attention module is added after each layer to select the relevant representation for each task. The second sub-network contains a specific representation parameterized by ϕ_i for each task t_i that is learned on top of the selected representation whenever the model faces this task.

ous knowledge to learn each future task instead of using the whole knowledge. We draw inspiration from *human learning*. For instance, a computer science student should have a mathematical background to learn other advanced courses such as artificial intelligence, computer graphics, database management, simulation modeling, etc. This prerequisite knowledge facilitates learning each of these courses quickly. However, in each course, one picks only the relevant information from their mathematical background depending on the context of each course instead of using the whole knowledge.

To address these desiderata, we propose to learn a prior representation for continual learning that permits and is more proficient at learning future tasks. We address the more realistic scenario where models might be deployed in an environment different from the ones they were pre-trained on. Therefore, we propose to learn this representation via meta-learning to permit generalization to out-of-distribution tasks. Moreover, we take advantage of meta-learning to allow the network to learn to pick the *relevant* representation from the currently existing one, depending on the incoming data. To this purpose, we incorporate a self-attention mechanism with the meta-learner. During the continual learning time, we train each task in the sequence by building on top of the selected representation from prior knowledge. These tasks are sampled from new distributions different from the one used in constructing the prior knowledge. Our empirical evaluation shows the importance of the proposed desiderata in the continual learning setting and their effectiveness in promoting the learning of each task and reducing interference. We also demonstrate that building on top of the relevant knowledge helps identify the correct target in the more challenging scenario where task identity is not available during inference.

Our contributions in this work can be summarized as follows:

- We demonstrate the importance of having prior generic knowledge and selective transfer in facilitating learning new tasks. We propose SAM, a Self-Attention Meta learner that includes these two aspects.
- We address the more challenging and realistic scenario where the task identity is not available during inference and assume that data from past tasks are not accessible.
- Our empirical results demonstrate that SAM performs better than state-of-the-art methods.
- We show the improved performance of popular existing continual learning strategies when they are integrated with SAM.

3.2 Related Work

3.2.1 Continual Learning Strategies

Many works have been proposed to address the catastrophic forgetting issue [MMO95, MC89] in deep neural networks. Regularization approaches add a regularization term to the learning objective to constrain the changes in important weights of past tasks [ABE+18, KPR+17, ZPG17]. The metric used to estimate the parameter importance differs between these approaches. In Elastic Weight Consolidation (EWC) [KPR+17], weight importance is calculated using an approximation of the diagonal of Fisher information matrix. In Synaptic Intelligence (SI) [ZPG17], weight importance is computed online during training. The importance is estimated by the amount of change in the loss by a weight summed over its trajectory. On the other hand, in Memory Aware Synapses (MAS) [ABE+18], the weights are estimated using the sensitivity of the learned function rather than the loss. Learning Without Forgetting (LWF) [LH17] is another regularization method that constrains the change of model predictions on old tasks, rather than the weights, by using a distillation loss [HVD15b].

Other existing methods modify the model architecture to adapt to new tasks. Progressive Neural Network (PNN) [RRD+16] instantiates a new network for each task and keeps previously learned networks frozen. CopyWeights with Reinit (CWR) [LM17] uses a fixed number of shared parameters for all tasks. The shared knowledge comes from freezing the shared weights after training the first batch. They initialize the shared weights using random weights or from a pre-trained model (ImageNet). Dynamic expandable network (DEN) [YYLH18] expands the model when the performance of old tasks degrades.

3.2.2 Continual Learning and Meta Learning

Recently, a new direction has emerged by combining meta-learning methods with the CL paradigm. In [FRKL19], a modification was proposed for the MAML algorithm to adapt to the online setting. They focus on maximizing the forward transfer and sidestep the problem of catastrophic forgetting by maintaining a buffer of all observed data. MER [RCA+18] combines experience replay with the Reptile [NS18] meta-learning algorithm in the online setting. Instead of storing all the observed data, they keep a fixed-size memory for all tasks and update the buffer with reservoir sampling. Another approach by [JW19] uses the meta-learning paradigm to learn to continually learn. They pre-train a network and continually learn tasks sampled from the same distribution. A more realistic scenario was presented in [CRO+20], where the continual tasks may come from a new distribution that is not encountered during pre-training. However, they relax the assumptions by allowing for task revisiting and optimizing for fast adapting. The setting of the CL problem differs in each of these methods. Yet, meta-learning seems a promising direction for addressing CL in all these different settings.

3.2.3 Attention

Attention has emerged as an improvement in machine translation systems in natural language processing [BCB15]. Recently, attention mechanisms have been addressed in many computer vision tasks [CZX+17, HWCW19, WJQ+17, FLT+19, ZDS+18, HSS18, PRV+19]. Few works have used the attention mechanisms in the CL paradigm. In [DSP+19], attention distillation loss is combined with the distillation loss from [LH17] to constrain the changes in old tasks. In [SSMK18], a hard attention mechanism was proposed to determine the important neurons for each task. These neurons are masked during learning future tasks.

3.3 Self-Attention Meta-Learner (SAM)

The ultimate goal of continual learning is to mimic human learning. The starting point to address this goal is to collect and learn prior knowledge that can help in continuously learning a sequence of tasks. This prior knowledge should be characterized by the good generality that enables out-of-domain tasks to learn on top of it. Moreover, each task in the continual sequence should pick the relevant knowledge from the previously learned knowledge. In this section, we describe the details of our proposed method, SAM, that addresses these two goals.

Figure 3.1 shows an overview of SAM. A neural network consists of two parts. The first part represents the prior knowledge parameterized by the shared learned meta-parameters θ. An attention module follows each layer in this shared sub-network which learns to pick the relevant features from that layer corresponding to the input. The second part learns specific representation to each task t_i parameterized by ϕ_i. Each task uses a few layers to capture the class-specific discriminative features. The input to this part is the selected relevant knowledge from prior knowledge. At deployment time, the input x is passed through the neural network $f(x;\theta,\phi_1,\phi_2,..,\phi_i,..)$ to predict the corresponding class from all learned classes so far.

We can divide our approach into two main phases: prior knowledge construction and the continuous learning of tasks. The training procedure for these two phases is shown in Algorithm 5. The details of each phase are discussed in the following paragraphs.

Prior knowledge construction. As discussed earlier, prior knowledge should generalize well to out-of-domain tasks. To this end, we train the shared parameters θ using the optimization-based meta-learning algorithm MAML [FAL17], which proves its ability to generalize to out-of-distribution tasks [FL18]. MAML learns parameters θ that can be quickly adapted to a new task using a small number of fine-tuning steps. In particular, the MAML algorithm consists of an "inner loop" (Algorithm 5, Lines 4-7) and an "outer loop" (Algorithm 5, Lines 2-8). In the inner loop, the parameters θ are adapted to multiple tasks using one or a few steps of gradient descent to obtain the parameters θ'_i which are specific for task instance $\mathcal{T}_i$. In the outer loop, the initialization θ is updated by differentiating through the inner loop to obtain a new initialization that improves inner-loop learning.

We train our meta-learner using tasks $\mathcal{T}_i \sim p(\mathcal{T}_{meta})$ from a certain domain

Algorithm 5 SAM

```
Require: $p$ ($\mathcal{T}_{meta}$): distribution over meta tasks
Require: $\alpha, \beta$: meta step size hyperparameters
Require: $\mathcal{N}_{meta}$: number of training steps for meta-learning
Require: $\eta$: step size hyperparameter for continual learning
Require: $\mathcal{N}_{CL}$: number of training steps for continual learning
// Learning prior knowledge with self-attention meta-learner
1  Randomly initialize $\theta$
2  for $n$=1,...,$\mathcal{N}_{meta}$ steps do
3      Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T}_{meta})$
4      for all $\mathcal{T}_i$ do
5          Sample minibatches $\mathscr{D}_i^{tr}$, $\mathscr{D}_i^{v}$ uniformly from $\mathscr{D}_i^{train}$, $\mathscr{D}_i^{val}$
6          Forward pass the minibatch through the meta-learner including the attention
             modules using eq. 3.3, 3.4, and 3.2
           // One or few steps of gradient descent
7          $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}(\mathscr{D}_i^{tr}, \theta)$
8      $\theta = \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i} \mathcal{L}(\mathscr{D}_i^{v}, \theta_i')$
// Learning continuously a sequence of tasks $(t_1, t_2, t_3, ...)$
9  Keep the meta-learned parameters $\theta$ fixed and shared for all tasks
10 foreach $t_i$ in tasks $(t_1, t_2, ...)$ do
11     Randomly initialize $\phi_i$ that represents the specific parameters to task $t_i$
12     for $j$=1,...,$\mathcal{N}_{CL}$ steps do
13         Sample minibatch $D_{t_i}^{tr}$ from $D_{t_i}$
14         $\phi_i = \phi_i - \eta \nabla_{\phi_i} \mathcal{L}(D_{t_i}^{tr}, \phi_i; \theta)$
```

and optimize the parameters such that when the model is faced with a new task, the model can adapt quickly. The objective of the MAML algorithm is:

$$\min_{\theta} \sum_{i} \mathcal{L}(\theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}_i^{train}), \mathcal{D}_i^{val}), \tag{3.1}$$

where $\mathcal{D}_i^{train}$ corresponds to the training set for task $\mathcal{T}_i$ which is used in the inner optimization and $\mathcal{D}_i^{val}$ is the validation data that is used for evaluating the outer loss $\mathcal{L}$. The inner optimization is performed via one or a few steps of gradient descent with a step size α. Further details for the meta-training procedure are included in Algorithm 5. The $\mathcal{T}_{meta}$ tasks are used only for constructing the prior knowledge and are different from the sequence of CL tasks that may come from another domain.

Selection of relevant knowledge. The learned prior knowledge is shared between all tasks. When the model faces a new task, it builds on top of this knowledge. Instead of using all the learned knowledge, the task picks the appropriate knowledge to use, which helps in its learning. To address this point, we incorporate a self-attention mechanism [HSS18] in our meta-learner. Rather than the standard training of the self-attention mechanism as in [HSS18], we make use of meta-learning to allow the network to learn to recalibrate the useful knowledge based on the input data. In particular, an attention block is added after each layer in the meta-learner (shared sub-network). This block's role is to adaptively recalibrate the convolutional channels (or the hidden neurons) in each layer. It learns to boost the informative features corresponding to the input and suppress the less useful ones. The input of the attention block is the feature maps $\mathbf{X} = \{X_1, X_2, ..., X_c\}$ resulted from applying the convolutional operator F on the output of the previous layer, where $\mathbf{X} \in \mathbb{R}^{h \times w \times c}$ and h, w, and c are the height, width, and depth of the feature maps. The output of each attention block is a vector $\mathbf{s}$ of size c that contains the rescaling value for each channel, where c is the number of channels (depth). The recalibrated feature map $X_i^{'} \in \mathbb{R}^{h \times w}$ is obtained as follows:

$$X_i^{'} = X_i \circ s_i, \tag{3.2}$$

where s_i is the scalar value in the vector $\mathbf{s}$ corresponding to the channel i and the operation $\circ$ represents a channel-wise multiplication between the feature map X_i and the scalar s_i. The structure of the attention block is shown in Figure 3.1. The attention and gating mechanism consists of two steps. The first step is to generate channel-wise statistics by compressing the global spatial information using global average pooling, resulting in a vector $\mathbf{z}$ of c channels. The i-th element of $\mathbf{z}$ is calculated by:

$$z_i = \frac{1}{h \times w} \sum_{k=1}^{h} \sum_{j=1}^{w} X_i(k, j). \tag{3.3}$$

The second step is to model the interdependencies between channels. These dependencies are captured using two feedforward fully connected layers. The first layer is a bottleneck layer that reduces the dimension of the c channels with a reduction ratio r (which is a hyperparameter), followed by a second layer that increases the dimensionality back to c channels. A sigmoid activation is applied to these c channels as a simple gating mechanism, producing the output vector $\mathbf{s}$, which can be calculated as follows:

$$\mathbf{s} = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})), \tag{3.4}$$

where the operation δ is the ReLU function, σ is the sigmoid function, and $\mathbf{W}_1 \in \mathbb{R}^{\frac{c}{r} \times c}$ and $\mathbf{W}_2 \in \mathbb{R}^{c \times \frac{c}{r}}$ are the parameters of the two feedforward fully connected layers respectively. The training of the attention mechanism is part of the meta-learning phase (e.g., $\mathbf{W}_1$ and $\mathbf{W}_2$ are part of the shared parameters θ), therefore we called it “meta-attention”.

Previously, we illustrated the attention mechanism in convolutional neural networks. It is very easy to adapt it to multilayer perceptron networks by ignoring the global average pooling step. The recalibrated hidden neurons are generated by performing element-wise multiplication between the output vector of the attention block $\mathbf{s}$ and the hidden neurons $\mathbf{x}$.

Learning a sequence of tasks. When the model encounters a new task t_i, a few specific layers to this task parameterized by ϕ_i are added to the model. These layers can be any differential layers (e.g., convolutional or feedforward fully connected layers). The input to these layers is the selected representation from prior knowledge. The parameters ϕ_i are trained in an end-to-end manner with a specific output layer for this task with the following objective:

$$\min_{\phi_i} \mathcal{L}_i(\phi_i, D_{t_i}; \theta), \tag{3.5}$$

where $\mathcal{L}_i$ is the loss for task t_i and D_{t_i} is the training set of task t_i. The shared parameters θ are kept fixed during learning each task.

Final decision module. This module predicts the final output $\hat{y}$ corresponding to the test input x in the class incremental learning scenario. First, we aggregate the output layers $(\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, ..., \hat{\mathbf{y}}_N)$ of all tasks, where N is the number of encountered tasks so far. Then, the final output is predicted by getting the index of the highest value from the concatenated output vector. This index corresponds to the predicted class by the learned model and is calculated as follows:

$$\arg\max_{idx}(\hat{\mathbf{y}}_1 \oplus \hat{\mathbf{y}}_2 \oplus ... \oplus \hat{\mathbf{y}}_N), \tag{3.6}$$

where $\hat{\mathbf{y}}_i$ is the output layer of task t_i, $\oplus$ represents the concatenation operation over the output vectors, and idx is the index of the element with the highest value.

3.4 Experiments

In this section, we evaluate our method on the commonly used benchmarks for CL. We compare our method with state-of-the-art approaches in regularization and architectural strategies. We also consider another baseline, “*Scratch*”, where an independent network is trained for each task individually from scratch. Each independent network has the same architecture as the network trained across all CL tasks. To get the predicted class using this baseline in the class incremental learning scenario, we use the final decision module of SAM. We name this baseline Scratch (Task Agnostic) or “*Scratch(TA)*” for short.

3.4.1 Split CIFAR-10/100

The Split CIFAR-10/100 benchmark [ZPG17] consists of a combination of CIFAR-10 and CIFAR-100 datasets [KH+09]. It contains 6 tasks. The first task contains the full dataset of CIFAR-10, while each subsequent task contains 10 consecutive classes from the CIFAR-100 dataset.

Experimental Setup

We follow the architecture used in [ZPG17, ML19]. The shared sub-network consists of 2 blocks. Each block contains a 3×3 convolutional layer followed by batch normalization [IS15], a ReLU activation, and an attention module with a reduction ratio (r) of 8. The convolutional layer in each block has 32 feature maps. A max-pooling layer follows these two blocks. The shared sub-network is initialized by the self-attention meta-learner. The specific sub-network for each task consists of two 3×3 convolutional layers with 64 feature maps, followed by a max-pooling layer, one fully connected layer of 512 neurons, and an output layer. The specific layers for each task are randomly initialized. We use cross-entropy loss for training. Each experiment is repeated 5 times with different random seeds. Additional training details are included in Appendix A.1.

Self-attention meta-learner: We trained a convolutional neural network on MiniImagenet [RL16]. The MiniImagenet dataset is a common benchmark used for few-shot learning. It contains 64 training classes, 12 validation classes, and 24 test classes. The model consists of 4 blocks with a max-pooling layer following each block. The number of feature maps in the convolutional layers in the first and second two blocks is 32 and 64, respectively. The learned weights of the first two blocks are copied to the shared sub-network as prior knowledge.

Since there are differences between the structure of our architecture and the regular network used in the baselines, we analyze another split for the shared and specific sub-networks in our results and analysis, where the shared sub-network contains all layers except the fully connected one. In particular, the shared sub-network consists of all 4 blocks, initialized by the self-attention meta-learner, with a max-pooling layer following every two blocks.

Results

Figure 3.2 shows the accuracy of each task after training all tasks, along with the average accuracy computed over all tasks. The regularization methods (EWC [KPR+17], LWF [LH17], and SI [ZPG17]) suffer from forgetting old tasks while having a good performance on the last trained task. On the other hand, the performance achieved by SAM on each task is close to each other. SAM outperforms the regularization methods by a big margin. We also compare our method to the counterpart architectural baseline, CWR [LM17], which uses a fixed shared pre-trained knowledge for all tasks and trains an output layer for each task. As shown in the figure, the learned representation by SAM generalizes better than the CWR method. The average accuracy of our proposed method is higher than CWR by around 24.5%. It is worth highlighting that SAM achieves better performance than optimizing a separate network for each task from scratch (*Scratch(TA)*).

We perform another experiment where the shared sub-network consists of 4 blocks initialized by the self-attention meta-learner. Increasing the depth of the shared sub-network while keeping it fixed decreases the performance of SAM to 30.40%. However, the prior knowledge learned by SAM is still more proficient;

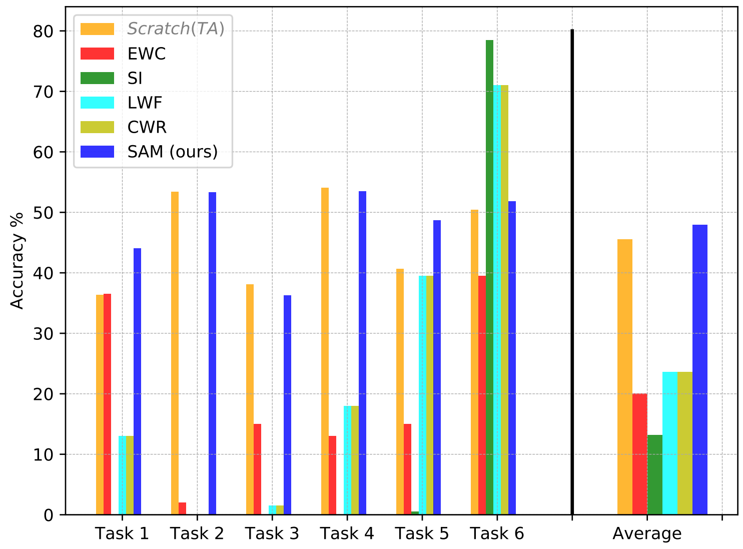


Figure 3.2: The accuracy of each task of the Split CIFAR-10/100 benchmark in the class incremental learning scenario after training all tasks. The "Average" x-axis label shows the average accuracy computed over all tasks for each method. Results for other methods except "Scratch(TA)" are reported from [ML19].

it performs better than the second-best performer method by around 7%.

3.4.2 Split MNIST

Split MNIST [ZPG17] is the most commonly used benchmark for CL. It consists of 5 tasks. Each task has two consecutive classes from MNIST. We study the performance of SAM and the baselines in the case of class-incremental learning and task-incremental learning.

Experimental Setup

We use multilayer perceptron networks. Our model follows the same architecture used in [vdVT18]. The shared sub-network consists of 2 blocks. Each block consists of one hidden layer with 400 neurons followed by a ReLU activation and an attention module with a reduction ratio (r) of 10. The shared sub-network is initialized by the self-attention meta-learner. A specific output layer for each task follows the shared sub-network. The weights of the output layer are randomly initialized. Since all layers in the used architecture are shared except the output, we ensure a fair comparison with the baselines. We use cross-entropy loss for training. Each experiment is repeated 5 times with different random seeds. Additional training details are included in Appendix A.1.

Self-attention meta-learner: We train a multilayer perceptron network on the Omniglot dataset [LSGT11]. The Omniglot dataset consists of 1623 characters from 50 different alphabets. Each character has 20 instances. The images are resized to 28×28. The model has the same architecture as our shared sub-network, with a batch normalization that follows each hidden layer in each block. The learned weights of the model are used as prior knowledge.

Results

Table 3.1 shows the average accuracy over all tasks after training the CL sequence. We report the average accuracy in class-incremental learning (CIL) and task-incremental learning (TIL). As shown in the table, regularization methods achieve a very good performance in TIL; however, they suffer from a huge drop in accuracy in CIL. SAM outperforms the regularization methods by more than 38% in the latter case while achieving a comparable performance in the case of TIL. We also compare our approach to one of the well-known architectural approaches, DEN [YYLH18]. DEN restores the drift in old tasks' performance using node duplication. DEN is originally evaluated in TIL as the connections in this method 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. We adapt the official code provided by the authors to evaluate the method in CIL and name it DEN(Task Agnostic) or DEN(TA) for short. After training all tasks, we evaluate the test data on the model created each timestamp. Then we use our final decision module to get the final prediction. Similar to other baselines, SAM achieves a comparable performance to DEN in TIL, while the performance of SAM is better than DEN(TA) by 6% in CIL. Moreover, DEN expands each hidden layer by around 35 neurons, while SAM has a fixed number of parameters. The detailed behavior of SAM for CL in CIL is also included in Appendix A.2.

Table 3.1: Average accuracy on the Split MNIST benchmark in TIL and CIL.

Method	TIL	CIL
Scratch	*99.68* ± *0.06*	-
Scratch(TA)	-	*67.8* ± *0.88*
EWC	98.64 ± 0.22	20.01± 0.06
SI	99.09 ± 0.15	19.99± 0.06
LWF	99.57 ± 0.02	23.85 ± 0.44
DEN	99.26 ± 0.001	-
DEN(TA)	-	56.95 ± 0.02
SAM (ours)	97.95 ± 0.07	**62.63** ± 0.61

3.5 Analysis

In this section, we analyze: (1) the role of the meta-attention mechanism in performance, (2) the selected representation by SAM for different classes, (3) the importance of having prior knowledge in CL, and (4) the usefulness of the representation learned by SAM for learning tasks from different domains.

3.5.1 Effect of the Meta-attention Mechanism on Performance

We conduct an ablation study to analyze the effect of the meta-attention mechanism on performance in CIL. We study two scenarios: (1) removing the attention modules from all layers in the shared sub-network, and (2) keeping the attention module only in the last block. We perform this experiment on the two studied benchmarks. For the Split CIFAR-10/100 benchmark, we analyze the performance of the two splitting of the shared and specific sub-networks discussed in Section 3.4.1. The results are summarized in Table 3.2.

When the meta-attention mechanism is excluded, and the entire knowledge is used to learn each task, the lowest accuracy is consistently observed across all cases. Utilizing an attention module solely at the last block of the shared sub-network results in slightly improved accuracy. However, incorporating a meta-attention module into each block of the shared sub-network provides significant benefits. The effect of the attention mechanism is larger when the shared sub-network is deeper (Shared: 4 blocks) as the features become more discriminative at higher layers. Restricting the input to each specific classifier to the relevant representation obtained from prior knowledge helps to select the correct classifier and improve the classification accuracy.

Table 3.2: Ablation study of SAM on the Split MNIST and Split CIFAR-10/100 benchmarks in CIL.

	Split MNIST	Split CIFAR-10/100	
Method		Shared: 2 blocks	Shared: 4 blocks
SAM without attention	51.33 ± 2.30	46.52 ± 0.33	24.21 ± 0.59
SAM with attention at the last block only	54.67 ± 0.80	46.58 ± 0.29	25.04 ± 0.14
SAM	**62.63** ± 0.61	**48.24** ± 0.30	**30.40** ± 0.31

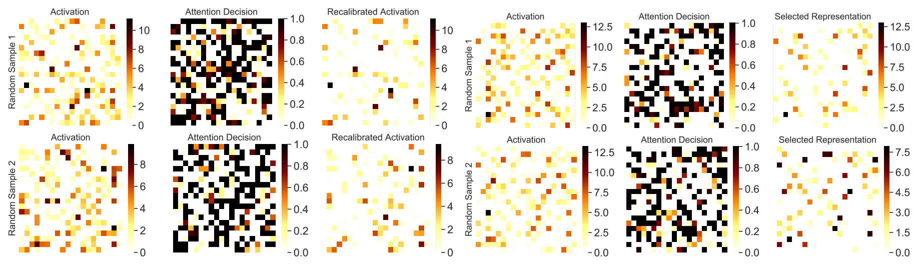
3.5.2 Learned Representation

We further investigate the effect of the meta-attention modules on the learned representation. To this end, we visualize the learned representation of each block in the shared sub-network before applying the attention module, the decision (scores) of the attention module, and the recalibrated activation. We draw these representations for random samples from different classes in different tasks from the Split MNIST benchmark.

As shown in Figure 3.3, the relevant representation for each sample is sparse and varies across different classes. Furthermore, more features are selected by the attention module in the first block, indicating that various classes may share numerous lower-level features with prior knowledge. On the other hand, many features are suppressed in the second block, and only the relevant features for the input are emphasized.

3.5.3 Role of Prior Knowledge

We hypothesize that having generic prior knowledge before learning the CL sequence and selecting the relevant representation from the past is more crucial when the sequence has dissimilar tasks. To evaluate this hypothesis, we generate three tasks based on the CIFAR10 benchmark: two similar tasks and one dissimilar task. Each task consists of two classes. The first task involves the airplane and automobile classes, the second task involves the bird and cat classes, and the third task involves the ship and truck classes. Thus, the first and the second tasks are dissimilar, while the first and third tasks are similar. We compare SAM with the setting used in most of the previous continual learning methods, where the model starts from randomly initialized weights and sequentially learns the tasks. We call this setting "Standard". We perform this analysis on the same architecture discussed in Section 3.4.1 with the two splittings of the shared



(a) The first block (b) The second block

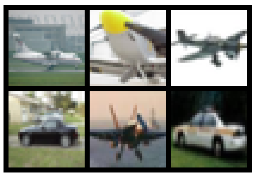
Figure 3.3: The visualization of the activation in shared sub-network for Split MNIST. The representation for each layer is reshaped to 20×20. Each row represents a random sample of a class in a certain task. We illustrate the activation before being calibrated, the output of the attention module, and the recalibrated activation in the first (a) and second (b) blocks. The last column represents the selected representation passed to the specific sub-networks.

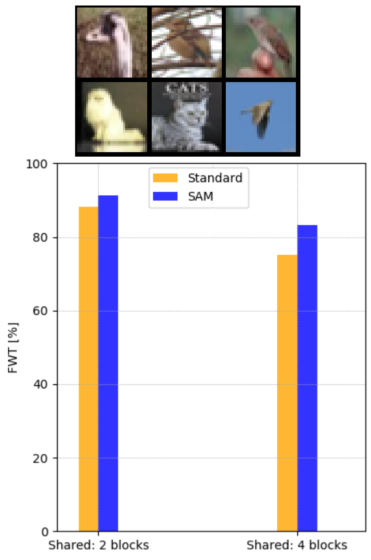
and specific sub-networks. In the standard setting, the shared sub-network is initialized with the weights learned from training the model on the first task and then frozen. We evaluate the forward transfer (FWT) in this setting and SAM on the second and third tasks. FWT is a metric proposed by [LPR17, DRLFM18] to assess the ability of the CL model to transfer knowledge for future tasks. It is estimated by the accuracy obtained on each task using the fixed shared sub-network while allowing the training of the specific sub-network.

As shown in Figure 3.4c, the forward transfer is comparable between the standard setting and SAM on the third task, as the previously learned knowledge in the standard setting from the first task is useful for the third task (they have some similarities). While in the case of dissimilar tasks, the FWT of SAM is higher than the standard setting, as shown for the second task (Figure 3.4b). Moreover, the performance gap increases when the shared sub-network gets deeper. The FWT of SAM is higher than the standard setting by around 8%. The analysis reveals the importance of having prior knowledge in the CL paradigm to promote learning future tasks.

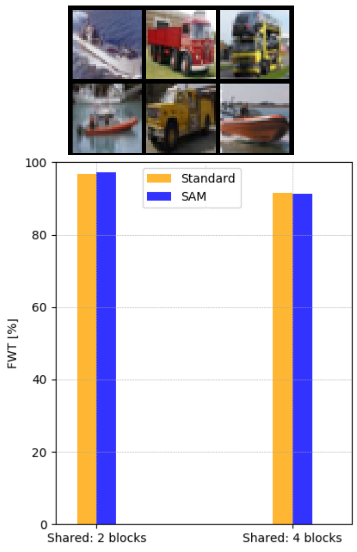
3.5.4 Knowledge Reusability

Is the representation learned by the self-attention meta-learner on a certain domain useful for learning a sequence of tasks from another domain? To answer this question, we compare SAM to the Extreme Learning Machine (ELM) [HZS04]. The authors proposed the "generalized" networks that provide the best generalization performance at a fast learning speed. The network parameters are randomly initialized and are not updated, while the parameters of the output layer


(a) Task 1 (airplane, automobile)

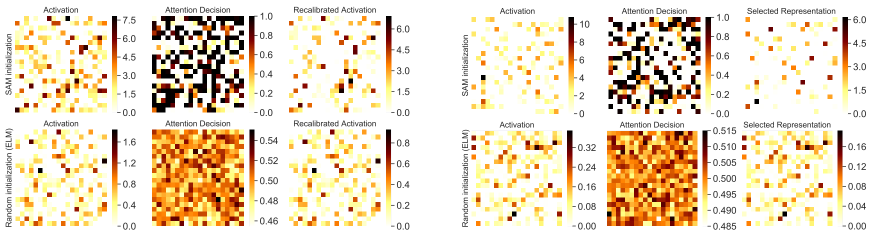


(b) Task 2 (bird, cat)



(c) Task 3 (ship, truck)

Figure 3.4: FWT comparison between SAM and the "Standard" setting used in most of the previous CL methods. In the Standard setting, the shared sub-network is initialized by the learned knowledge from Task 1 (airplane, automobile) from the CIFAR10 benchmark (a). FWT is evaluated on a dissimilar task (bird, cat) (b) and another similar task (ship, truck) (c). SAM promotes forward transfer in case of dissimilar tasks. While in the Standard setting, the knowledge learned by Task 1 contains less information useful for Task 2.



(a) The first block (b) The second block

Figure 3.5: Activations in the shared sub-network for the Split MNIST benchmark. The representation is reshaped to 20×20. The first and second rows show the representations when the shared sub-network is initialized by SAM and a random initialization (ELM), respectively.

Table 3.3: A comparison between the random initialization (ELM) and the initialization by the generic prior knowledge (SAM) for the shared sub-network.

Method	Split CIFAR-10/100	Split MNIST
ELM	37.95 ± 0.64	58.42 ± 0.91
SAM (ours)	**48.24** ± 0.30	**62.63** ± 0.61

are learned. This research is then extended by many works [BRF13, RBW+13, FMR16, BKH15, HBKV15]. We compare SAM with ELM by initializing the shared sub-network randomly and keeping its weights fixed. We visualize the representation of each hidden layer in the shared sub-network on the Split MNIST benchmark in SAM and ELM.

As shown in Figure 3.5, the self-attention meta-learner boosts and selects some features, while the random initialization gives almost equal importance to each feature. Table 3.3 compares the two methods in terms of performance on the two studied benchmarks. As shown in the table, prior knowledge learned by SAM generalizes better for the CL tasks despite the fact that the domain of the prior knowledge is different from the CL sequence.

3.6 Improvements of CL Approaches with SAM

The experimental evaluation demonstrates the effectiveness of having prior knowledge and selective transfer for the continual learning paradigm. In this

Table 3.4: Enhancing existing continual learning strategies by SAM. “Standard” represents the original form of the methods. The accuracy is reported on the Split MNIST and Split CIFAR-10/100 benchmarks in CIL.

	Split MNIST		Split CIFAR-10/100	
Method	Standard	+ SAM	Standard	+ SAM
Fine-tuning	19.86 ± 0.04	**53.87** ± 1.73	12.24 ± 0.05	**25.45** ± 1.76
SI	19.99 ± 0.06	**67.32** ± 0.43	13.39 ± 0.04	**42.92** ± 1.01
MER	32.66 ± 2.33	**50.04** ± 1.85	-	-

section, we analyze the performance of popular CL methods when they are integrated with SAM.

We evaluate the performance of the regularization method SI [ZPG17] as well as the optimization-based meta-learning method MER [RCA+18] when they are combined with SAM as well as their original form. In particular, instead of freezing the shared sub-network after learning the prior knowledge, we allow for accumulating the knowledge from each CL task. The shared sub-network is updated using a CL baseline (SI or MER). Accumulating the knowledge from each task in the shared sub-network causes catastrophic forgetting of the previously learned ones. The SI method addresses this problem by adding a regularization term in the loss function to constrain the change in the important weights of previous tasks. In MER, a small memory for experience replay is used, and the parameters are trained using the Reptile meta-learning algorithm [NS18]. We also add the simple fine-tuning method as another baseline, where new tasks are trained continuously without any mechanism to avoid forgetting in the shared sub-network. We perform this experiment on the Split MNIST and Split CIFAR-10/100 benchmarks. We use the same architectures and training details described in Section 3.4. To ensure a fair comparison with the original form of the methods, for the Split CIFAR-10/100 benchmark, we use the network split where the shared sub-network consists of 4 blocks. For the MER algorithm, we adapt the official code provided by the authors to evaluate it on the Split MNIST benchmark. Following the notation of their paper, we use a batch size $(k-1)$ of 10, the number of batches per example of 5, γ of 1.0, and β of 0.01. We use a memory buffer of size 200 to learn 1000 sampled examples across each task. The results are shown in Table 3.4.

Interestingly, when SAM is combined with the other methods, it always improves their performance. It is interesting to observe that the combination of SAM with the fine-tuning baseline increases its performance despite that there is no forgetting avoidance strategy. MER achieves good results despite using only 1000 samples from each task. Although the regularization methods suffer from a huge performance drop when applied in CIL, as shown before, combining SAM with SI leads to a significant improvement: around 47% and 29% on the Split MNIST and Split CIFAR-10/100 benchmarks, respectively. SAM reduces forgetting by allowing an adaptive update for the weights. The update of the weights becomes a function of the recalibrated activations by SAM. Therefore, the knowledge accumulated by new tasks affects a subset of the previously learned representation. Accumulating the knowledge in SAM while using the SI method to constrain the change in the important weights of old tasks outperforms SAM alone by 5% and 13% on the Split MNIST and Split CIFAR-10/100 benchmarks, respectively. This analysis confirms the importance of our proposed desiderata for continual learning. We believe that it would open up many directions for CIL by adopting SAM as the starting point.

3.7 Conclusion and Future Work

In this chapter, we demonstrated the importance of two desiderata of continual learning, which have not received much attention in current state-of-the-art approaches. First, the necessity of having a good quantity of prior knowledge to promote future learning. Second, selecting the relevant knowledge to the current task from previous knowledge instead of using the whole knowledge.

To demonstrate their effectiveness on performance, we proposed SAM, a self-attention meta-learner for continual learning. SAM learns prior knowledge that can generalize to new distributions and learns to boost the features relevant to input data. During the continual learning phase, we introduce out-of-domain tasks. Our empirical evaluation and analysis demonstrated the role of our proposed desiderata in improving performance. The experimental results showed that the proposed method outperforms state-of-the-art methods from different continual learning strategies in CIL. Remarkably, SAM achieved performance on par with the Scratch(TA) baseline despite that in this baseline an independent model is optimized for each task separately. Finally, we demonstrated that combining SAM with the existing continual learning methods boosts their performance, showing the importance of selective transfer in improving both backward and forward transfer.

SAM addresses forgetting by design to enable assessing the studied desiderata. Further research on addressing forgetting and forward transfer jointly is crucial for improving performance. Moreover, more analyses of the relationship between forward transfer and the speed of adaptability of a network to new data would help build improved models.

Chapter 4

Dynamic Sparse Training for Deep Reinforcement Learning

Training deep neural networks to adapt to changing data distributions presents significant challenges, as demonstrated earlier. These challenges are further increased when a model learns a task without access to true labels. Such difficulties are particularly evident in the context of deep reinforcement learning (DRL), where agents are trained through trial-and-error interactions with an environment, and new training samples are collected over time. In this case, achieving satisfactory performance with dense neural networks often requires substantial training time due to the inherent training difficulties involved. Hence, prohibitive computation and memory resources are consumed. In this chapter, we aim to improve the training time of DRL agents by exploring a new training strategy for this paradigm. More specifically, we introduce for the first time a dynamic sparse training approach for deep reinforcement learning to accelerate the training process. The proposed approach trains a sparse neural network from scratch and dynamically adapts its topology to the changing data distribution during training. Experiments on continuous control tasks show that our dynamic sparse agents achieve higher performance than the equivalent dense ones, reduce the parameter count

This chapter is integrally based on **Ghada Sokar**, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. *Dynamic sparse training for deep reinforcement learning*. International Joint Conference on Artificial Intelligence (IJCAI), 2022. The Adaptive and Learning Agents Workshop at the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), **Best Paper Award**, 2022.

and floating-point operations (FLOPs) by 50%, and have a faster learning speed that enables reaching the performance of dense agents with $40-50\%$ *reduction in the training steps. We release our code at* `https://github.com/GhadaSokar/Dynamic-Sparse-Training-for-Deep-Reinforcement-Learning`.

4.1 Introduction

Deep reinforcement learning (DRL) has achieved remarkable success in many applications. The power of deep neural networks as function approximators allows RL agents to scale to environments with high-dimensional state and action spaces. This enables high-speed growth in the field and the rise of many methods that improve the performance and stability of DRL agents [WLZ+20]. While the achieved performance is impressive, a long training time is required to obtain this performance. For instance, it took more than 44 days to train a Starcraft II agent using 32 third-generation tensor processing units (TPUs) [VBC+19]. The very long training time leads to high energy consumption and prohibitive memory and computation costs. In this work, we ask the following question: *Can we provide efficient DRL agents with less computation cost and energy consumption while maintaining superior performance?*

Few recent works attempt to accelerate the *inference* time of DRL agents via pruning [LC20] or training a compact network under the guidance of a larger network (knowledge distillation) [ZHL19]. Despite the computational improvement achieved at inference, extensive computations throughout the training of *dense* networks are still consumed. Our goal is to accelerate the training process as well as the inference time of DRL agents.

The long training time of a DRL agent is due to two main factors: **(1)** the extensive computational cost of training deep neural networks caused by the high number of network parameters [JYP+17] and **(2)** the learning nature of a DRL agent in which its policy is improving through many trial-and-error cycles while interacting with the environment and collecting a large amount of data. In this chapter, we introduce dynamic sparse training (DST) [MMP+21, HABN+21] in the DRL paradigm for the first time to address these two factors. Namely, we propose an efficient training approach that can be integrated with existing DRL methods. Our approach is based on training *sparse* neural networks from scratch with a fixed parameter count throughout training **(1)**. During training, the sparse topology is optimized via adaptation cycles to *quickly adapt* to the online changing distribution of data **(2)**. Our training approach enables reducing memory and computation costs substantially. Moreover, the quick adaptation to the new samples from the improving policy during training leads to a faster learning speed.

In fact, the need for neural networks that can adapt, e.g., change their control policy dynamically as environmental conditions change, was broadly acknowledged by the RL community [Sta03]. Although prior works related to the automatic selection of function approximation based on neuroevolution exist [HMI09], perhaps the most connected in the spirit to our proposed method is a combination of NeuroEvolution of Augmenting Topologies (NEAT) [SM02] and temporal difference (TD) learning (i.e., NEAT+Q [WS06]). Still, the challenge remains, and cutting-edge DRL algorithms do not account for the benefits of dynamic neural network training yet.

Our contributions in this work are as follows:

- The principles of dynamic sparse training are introduced in the deep reinforcement learning field for the first time.
- Efficient improved versions of two state-of-the-art algorithms, TD3 [FHM18] and SAC [HZAL18], are obtained by integrating our proposed DST approach with the original algorithms.
- Experimental results show that our training approach reduces the memory and computation costs of training DRL agents by 50% while achieving superior performance. Moreover, it achieves a faster learning speed, reducing the required training steps.
- Analysis insights demonstrate the promise of dynamic sparse training in advancing the field and allowing for DRL agents to be trained and deployed on low-resource devices (e.g., mobile phones, tablets, and wireless sensor nodes) where the memory and computation power are strictly constrained.

4.2 Related Work

4.2.1 Sparsity in DRL

To the best of our knowledge, the current advance in deep reinforcement learning is achieved using *dense* neural networks. Few recent studies have introduced sparsity in DRL via pruning. PoPS [LC20] first trains a dense teacher neural network to learn the policy. This dense teacher policy network guides the iterative pruning and retraining of a student policy network via knowledge distillation.

In [ZHL19], the objective is to accelerate the behavior policy network and reduce the time for sampling. A smaller network is used for the behavior policy and trained simultaneously with a large dense target network via knowledge distillation. GST [LKK$^+$21] was proposed as an algorithm for weight compression in DRL training by simultaneously utilizing weight grouping and pruning. Some other works [YETM19, VLS21] studied the existence of the lottery ticket hypothesis [FC18] in RL, which shows the presence of sparse subnetworks that can outperform dense networks when they are trained from scratch. This finding motivates our study. However, pruning dense networks increases the computational cost of the training process as it requires iterative cycles of pruning and retraining. In this work, we introduce the first efficient training algorithm for DRL agents that trains a sparse neural network directly from scratch and adapts its topology to the changing data distribution.

4.2.2 Dynamic Sparse Training (DST)

DST is the class of algorithms that train sparse neural networks *from scratch* and jointly optimize the weights and the sparse topology during training. This direction 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) [DSD$^+$13]. Efforts in this line of research are devoted to supervised and unsupervised learning. The first work in this direction was proposed by [MMS$^+$18]. They proposed a Sparse Evolutionary Training algorithm (SET) that dynamically changes the sparse connectivity during training based on the values of the connections (weights). SET achieves higher performance than dense models and static sparse networks. The success of the SET algorithm opens the path to many interesting DST methods that bring higher performance gain. These algorithms differ from each other in the way the sparse topology is adapted during training [MW19, EGM$^+$20, JPR$^+$20, LMPP21, SMP21b].

In this work, we adopt the topological adaptation from the SET algorithm in our proposed approach. The motivation is multifold. First, SET is simple yet effective; it achieves the same or even higher accuracy than dense models with high sparsity levels across different architectures (e.g., multi-layer perceptrons, convolutional neural networks, restricted Boltzmann machines). Second, unlike other DST methods that use the values of non-existing (masked) weights in the adaptation process, SET uses only the values of existing sparse connections. This makes SET truly sparse and memory-efficient [LMM$^+$20]. Finally, it does not need high computational resources for the adaptation process. It uses readily available information during the standard training. These factors are favorable for our goal to train efficient DRL agents suitable for real-world applications. We leave evaluating other topological adaptation strategies for future work.

4.3 Proposed Method

In this section, we describe our proposed method, which introduces dynamic sparse training for DRL. Here, we focus on integrating our training approach with one of the state-of-the-art DRL methods; Twin Delayed Deep Deterministic policy gradient (TD3) [FHM18]. We named our new approach Dynamic Sparse TD3 or "DS-TD3" for short. TD3 is an efficient DRL method that offers good performance in many tasks [SLXC20, WWPR20]. Yet, our approach can be merged into other DRL methods as well. Appendix B.5 shows the integration with soft actor-critic (SAC) [HZAL18].

TD3 is an actor-critic method that addresses the overestimation bias in previous actor-critic approaches. In actor-critic methods, a policy π is known as the *actor*, and a state-value function Q is known as the *critic*. Target networks are used to maintain fixed objectives for the actor and critic networks over multiple updates. In short, TD3 limits the overestimation bias using a pair of critics. It takes the smallest value of the two critic networks to estimate the Q value to provide a more stable approximation. To increase the stability, TD3 proposed a delayed update of the actor and target networks. In addition, the weights of the target networks are slowly updated by the current networks by some proportion τ. *In this work*, we aim to dynamically train the critics and actor networks along with their corresponding target networks from scratch with sparse neural networks to provide efficient DRL agents. In the rest of this section, we will explain our proposed DST approach for TD3. The full details are also provided in Algorithm 6.

Our proposed DS-TD3 consists of four main phases: sparse topology initialization, adaptation schedule, topological adaptation, and maintaining sparsity levels.

Sparse Topology Initialization (Algorithm 6 L1-L4). TD3 uses two critic networks ($Q_{\theta_1}, Q_{\theta_2}$) and one actor network (π_{ϕ}) parameterized by $\boldsymbol{\theta_1} = \{\boldsymbol{\theta}_\mathbf{1}^l\}|_{l=1}^{L}$, $\boldsymbol{\theta_2} = \{\boldsymbol{\theta}_\mathbf{2}^l\}|_{l=1}^{L}$, and $\phi = \{\phi^l\}|_{l=1}^{L}$ respectively; where L is the number of layers in a network. We initialize each of the actor and critic networks with a sparse topology. Sparse connections are allocated in each layer between the hidden neurons at layer $l-1$ and layer l. We represent the locations of the sparse connections by a binary mask $\boldsymbol{M} = \{\boldsymbol{M}^l\}|_{l=1}^{L}$. Following [MMS+18], we use Erdős–Rényi random graph [ER+60] to initialize a sparse topology in each layer l. Namely, the probability of a connection j in layer l is given by:

$$p(\boldsymbol{M}^j) = \lambda^l \frac{n^l + n^{l-1}}{n^l \times n^{l-1}}, \tag{4.1}$$

where λ^l is a hyperparameter to control the sparsity level in layer l, and n^{l-1} and n^l are the neurons count in layers $l-1$ and l, respectively. $\boldsymbol{M}^j \in \{0,1\}$; a value of 1 means the existence of a weight in location j. We omit the index l from the mask and weight matrices for readability. A sparse topology is created in each layer for the actor and critic networks:

$$\begin{aligned} &\boldsymbol{\phi} = \boldsymbol{\phi} \odot \boldsymbol{M_\phi}, \\ &\boldsymbol{\theta_i} = \boldsymbol{\theta_i} \odot \boldsymbol{M_{\theta_i}}, \qquad \forall i \in \{1,2\}, \end{aligned} \tag{4.2}$$

where $\odot$ is an element-wise multiplication operator and $\boldsymbol{M_\phi}$, $\boldsymbol{M_{\theta_1}}$, and $\boldsymbol{M_{\theta_2}}$ are binary masks to represent the sparse weights in the actor and two critic networks, respectively. The initial sparsity level is kept fixed during training.

The target policy and target critic networks are parameterized by $\boldsymbol{\phi'}$, $\boldsymbol{\theta'_1}$, and $\boldsymbol{\theta'_2}$, respectively. Initially, the target networks have the same sparse topology and the same weights as the current networks: $\boldsymbol{\phi'} \leftarrow \boldsymbol{\phi}$, $\boldsymbol{\theta'_1} \leftarrow \boldsymbol{\theta_1}$, $\boldsymbol{\theta'_2} \leftarrow \boldsymbol{\theta_2}$.

After the topological and weight initialization, the agent collects enough data before training using a purely exploratory policy. During training, for each time step, TD3 updates the pair of critics towards the minimum target value of actions selected by the target policy $\pi_{\phi'}$:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi'}(s') + \epsilon), \tag{4.3}$$

where γ is the discounting factor, r is the current reward, s' is the next state, and $\epsilon \sim clip(\mathcal{N}(0,\tilde{\sigma}), -c, c)$ is the proposed clipped noise by TD3, defined by $\tilde{\sigma}$, to increase the stability; where c is the clipped value. As discussed, TD3 proposed to delay the update of the policy network to first minimize the error in the value network before introducing a policy update. Therefore, the actor network is updated every d steps with respect to Q_{θ_1} as shown in Algorithm 6 **L17-L19**.

During the weight optimization of the actor and critic networks, only the values of the existing sparse connections are updated (i.e., the sparsity level is kept fixed). The sparse topologies of the networks are also optimized during training according to our proposed adaptation schedule.

Algorithm 6 DS-TD3 (λ^l, η, e, N, τ, d)

```
1: Initialize critic networks $Q_{\boldsymbol{\theta_1}}$, $Q_{\boldsymbol{\theta_2}}$ and actor network $\pi_{\phi}$ with sparse parameters $\boldsymbol{\theta_1}$, $\boldsymbol{\theta_2}$, $\phi$ with a sparsity level defined by $\lambda^l$:
2: Create $\boldsymbol{M_{\phi}}$, $\boldsymbol{M_{\theta_1}}$, and $\boldsymbol{M_{\theta_2}}$ with Erdős–Rényi graph
3: $\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} \odot \boldsymbol{M_{\theta_1}}$, $\boldsymbol{\theta_2} \leftarrow \boldsymbol{\theta_2} \odot \boldsymbol{M_{\theta_2}}$, $\phi \leftarrow \phi \odot \boldsymbol{M_{\phi}}$
4: Initialize target networks $\boldsymbol{\theta'_1} \leftarrow \boldsymbol{\theta_1}$, $\boldsymbol{\theta'_2} \leftarrow \boldsymbol{\theta_2}$, $\boldsymbol{\phi'} \leftarrow \phi$
5: Initialize replay buffer $\mathcal{B}$
6: for $t = 1$ to $T$ do
7:     Select action with exploration noise $a \sim \pi_{\phi}(s) + \epsilon$,
8:     $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
9:     Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$
10:    Sample mini-batch of $N$ transitions from $\mathcal{B}$
11:    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
12:    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\boldsymbol{\theta'_i}}(s', \tilde{a})$
13:    $\boldsymbol{\theta_i} \leftarrow argmin_{\boldsymbol{\theta_i}} \frac{1}{N} \sum (y - Q_{\boldsymbol{\theta_i}}(s, a))^2$
14:    if $t$ mod $e$ then
15:        $\boldsymbol{\theta_i} \leftarrow$ TopologicalAdaptation($\boldsymbol{\theta_i}, \boldsymbol{M_{\theta_i}}, \eta$) (Algo. 7)
16:    end if
17:    if $t$ mod $d$ then
18:        Update $\phi$ by the deterministic policy gradient:
19:        $\nabla_{\phi} J(\phi) \leftarrow \frac{1}{N} \sum \nabla_a Q_{\boldsymbol{\theta_1}}(s, a)|_{a=\pi_{\phi}(s)} \nabla_{\phi} \pi_{\phi}(s)$
20:        if $t$ mod $e$ then
21:            $\phi \leftarrow$ TopologicalAdaptation($\phi, \boldsymbol{M_{\phi}}, \eta$) (Algo. 7)
22:        end if
23:        Update target networks:
24:        $\boldsymbol{\theta'_i} \leftarrow \tau \boldsymbol{\theta_i} + (1 - \tau) \boldsymbol{\theta'_i}$
25:        $\boldsymbol{\phi'} \leftarrow \tau \phi + (1 - \tau) \boldsymbol{\phi'}$
26:        $\boldsymbol{\theta'_i} \leftarrow$ MaintainSparsity($\boldsymbol{\theta'_i}, \|\boldsymbol{\theta_i}\|_0$) (Algo. 8)
27:        $\boldsymbol{\phi'} \leftarrow$ MaintainSparsity($\boldsymbol{\phi'}, \|\phi\|_0$) (Algo. 8)
28:    end if
29: end for
```

Adaptation Schedule. The typical practice in DST methods applied to the supervised setting is to perform the dynamic adaptation of the sparse topology after each training epoch. However, this would not fit the RL setting directly due to its dynamic learning nature. In particular, an RL agent faces instability during

Algorithm 7 Topological Adaptation ($\boldsymbol{X}$, $\boldsymbol{M}$, η)

```
1: $c \leftarrow \eta \|\boldsymbol{X}\|_0$
2: $c_p \leftarrow c/2$ ;    $c_n \leftarrow c/2$
3: $\tilde{\boldsymbol{X}}^p \leftarrow$ get_$c_p$-$th$_smallest_positive($\boldsymbol{X}$)
4: $\tilde{\boldsymbol{X}}^n \leftarrow$ get_$c_n$-$th$_largest_negative($\boldsymbol{X}$)
5: $\boldsymbol{M}^j \leftarrow \boldsymbol{M}^j - \mathbb{1}[(0 < \boldsymbol{X}^j < \tilde{\boldsymbol{X}}^p) \vee (0 > \boldsymbol{X}^j > \tilde{\boldsymbol{X}}^n)]$
6: Generate $c$ random integers $\{x\}|_1^c$
7: $\boldsymbol{M}^j \leftarrow \boldsymbol{M}^j + \mathbb{1}[(j == x) \wedge (\boldsymbol{X}^j == 0)]$
8: $\boldsymbol{X} \leftarrow \boldsymbol{X} \odot \boldsymbol{M}$
```

Algorithm 8 Maintain Sparsity ($\boldsymbol{X}$, k)

```
1: $\tilde{\boldsymbol{X}} \leftarrow$ Sort_Descending($|X|$)
2: $\boldsymbol{M}^j = \mathbb{1}[|\boldsymbol{X}^j| - \tilde{\boldsymbol{X}}^k \geq 0], \forall j \in \{1, \ldots \|\boldsymbol{X}\|_0\}$
3: $\boldsymbol{X} = \boldsymbol{X} \odot \boldsymbol{M}$
```

training due to the lack of a true target objective. The agent learns through trial and error cycles, collecting the data online while interacting with the environment. Adapting the topology very frequently in this learning paradigm would limit the exploration of effective topologies for the data distribution and give a biased estimate of the current one. To address this point, we propose to delay the adaptation process and perform it every e time steps, where e is a hyper-parameter. This would allow the newly added connections from the previous adaptation process to grow. Hence, it would also give better estimates of the connections with the least influence on performance and an opportunity to explore other effective ones. Analysis of the effect of the adaptation schedule in the success of applying dynamic sparse training in the RL setting is provided in Section 5.2.

Topological Adaptation (Algorithm 7). We adopt the adaptation strategy of the SET method [MMS+18] in our approach. The sparse topologies are optimized according to our adaptation schedule. Every e steps, we update the sparse topology of the actor and critic networks. Here, we explain the adaptation process on the actor network as an example. The same strategy is applied to the critic networks.

The adaptation process is performed through a “drop-and-grow” cycle which consists of two steps. **The first step** is to *drop* a fraction η of the least important connections from each layer. This fraction is a subset (c_p) of the smallest positive weights and a subset (c_n) of the largest negative weights. Thus, the removed weights are the ones closest to zero. Let $\tilde{\phi}^p$ and $\tilde{\phi}^n$ be the c_p-th smallest positive and the c_n-th largest negative weights, respectively. The mask $\boldsymbol{M_\phi}$ is updated to represent the dropped connections as follows:

$$\boldsymbol{M}_{\boldsymbol{\phi}}^{\boldsymbol{j}} = \boldsymbol{M}_{\boldsymbol{\phi}}^{\boldsymbol{j}} - \mathbb{1}[(0 < \phi^j < \tilde{\phi}^p) \vee (0 > \phi^j > \tilde{\phi}^n)], \quad \forall j \in \{1, ..., \|\phi\|_0\}, \tag{4.4}$$

where $\boldsymbol{M}_{\boldsymbol{\phi}}^{\boldsymbol{j}}$ is the element j in $\boldsymbol{M_\phi}$, $\mathbb{1}$ is the indicator function, $\vee$ is the logical OR operator, and $\|.\|_0$ is the standard L_0 norm. **The second step** is to *grow* the same fraction η of removed weights in random locations from the non-existing weights in each layer. $\boldsymbol{M_\phi}$ is updated as follows:

$$\boldsymbol{M}_{\boldsymbol{\phi}}^{\boldsymbol{j}} = \boldsymbol{M}_{\boldsymbol{\phi}}^{\boldsymbol{j}} + \mathbb{1}[(j == x) \wedge (\phi^j == 0)], \quad \forall j \in \{1, .., \|\phi\|_0\}, \tag{4.5}$$

where x is a random integer generated from the discrete uniform distribution in the interval $[1, n^{(l-1)} \times (n^l)]$ and $\wedge$ is the logical AND operator. The weights of the newly added connections are zero-initialized.

Maintain Sparsity Level in Target Networks (Algorithm 8). TD3 delays the update of target networks to be performed every d steps. In addition, the target networks are slowly updated by some proportion τ instead of making the target networks exactly match the current ones (Algorithm 6 **L23-L25**). These two points lead to a slow deviation of the sparse topologies of target networks from current networks. Consequently, the slow update of the target networks by τ would slowly increase the number of non-zero connections over time. *To address this,* after each update of target networks, we prune the extra connections that make the total number of connections exceed the initially defined one. We prune the extra weights based on their smallest magnitude. Assume we have to retain k connections. The target masks of the actor ($\boldsymbol{M}'_{\phi'}$) and critics ($\boldsymbol{M}'_{\theta'_1}$, $\boldsymbol{M}'_{\theta'_2}$) are calculated as follows:

$$\begin{aligned} \boldsymbol{M}'^{j}_{\phi'} &= \mathbb{1}[|\phi'^j| - \tilde{\phi}'^k \geq 0], \quad \forall j \in \{1, ..., \|\phi'\|_0\}, \\ \boldsymbol{M}'^{j}_{\theta'_i} &= \mathbb{1}[|\theta'^j_i| - \tilde{\theta}'^k_i \geq 0], \quad \forall j \in \{1, ..., \|\theta'_i\|_0\}, \quad \forall i \in \{1, 2\}, \end{aligned} \tag{4.6}$$

where $\tilde{\phi}'^k$ and $\tilde{\theta}'^k_i$ is the k-th largest magnitude in the actor and critics respectively, and $|(.)^j|$ is the magnitude of element j in the matrix. The target networks are updated as follows:

$$
\begin{aligned}
&\phi' = \phi' \odot M'_{\phi'}, \\
&\theta'_i = \theta'_i \odot M'_{\theta'_i} \quad \forall i \in \{1,2\}.
\end{aligned}
\tag{4.7}
$$

4.4 Experiments and Results

In this section, we assess the efficiency of our proposed dynamic sparse training approach for the DRL paradigm and compare it to state-of-the-art algorithms. Experimental settings are provided in Appendix B.1.

4.4.1 Baselines

We compare DS-TD3 with the following baselines: (1) *TD3* [FHM18], the original TD3 where dense networks are used for actor and critic models, (2) *Static-TD3*, a variant of TD3 where the actor and critic models are initialized with sparse neural networks which are kept fixed during training (i.e., there is no topological optimization), and (3) *SAC* [HZH+18], a popular off-policy algorithm in which the policy is trained to maximize the trade-off between expected return and entropy, resulting in policies that explore better.

4.4.2 Benchmarks

We perform our experiments on MuJoCo continuous control tasks interfaced through OpenAI Gym. We evaluate our approach on five challenging environments (HalfCheetah-v3, Hopper-v3, Walker2d-v3, Ant-v3, and Humanoid-v3).

4.4.3 Metrics

We use multiple metrics to assess the efficiency of the studied DRL methods:

- **Return**: The standard metric used in DRL to measure the *performance* of an agent.
- **Learning curve area (LCA)**: An estimate of the *learning speed* of a model (i.e., how quickly a model learns) [CRRE18] by measuring the area under the training curve of a method.

Table 4.1: Learning curve area (LCA) (× 5000) of different methods.

Method	HalfCheetah-v3	Walker2d-v3	Hopper-v3	Ant-v3	Humanoid-v3
TD3	1.7686	0.5264	0.4788	0.5524	0.3635
Static-TD3	1.7666	0.5167	0.4984	0.5807	0.5182
DS-TD3 (ours)	**1.9560**	**0.6956**	0.5435	0.6623	**0.6089**
SAC	1.7297	0.6128	**0.5572**	**0.7969**	0.5639

Table 4.2: Average return (R) over the last 10 evaluations of 1 million time steps.

Method	HalfCheetah-v3	Walker2d-v3	Hopper-v3	Ant-v3	Humanoid-v3
TD3	11153.48±473.29	4042.36±576.57	2184.78±1224.14	4287.69±1080.88	3809.15±1053.40
Static-TD3	10583.84±307.03	3951.01±443.78	3570.88±43.71	4148.61±801.34	4989.47±546.32
DS-TD3 (ours)	**11459.88±482.55**	**4870.57±525.33**	**3587.17±70.62**	5011.56±596.95	5238.16±121.71
SAC	11415.23±357.22	4566.18±448.25	3387.36±148.73	**5848.64±385.85**	**5518.61±97.03**

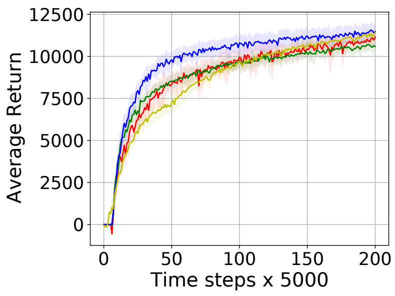
- **Network size (#params)**: An estimate of the *memory cost* measured by the number of network parameters.
- **Floating-point operations (FLOPs)**: An estimate of the *computational cost* required for training. It is the typical metric in the literature to compare a DST method against its dense counterpart.

More details can be found in Appendix B.2.

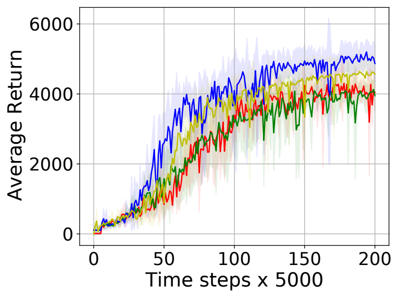
4.4.4 Results

Learning Behavior and Speed. Figure 4.1 shows the learning curve of studied methods. DS-TD3 has a much faster learning speed than the baselines, especially at the beginning of the training. After 40-50% of the steps, DS-TD3 can achieve the final performance of TD3. Static-TD3 does not have this favorable property, revealing the importance of optimizing the sparse topology during training to adapt to incoming data. Table 4.1 shows the learning curve area (LCA) of each method. DS-TD3 has higher LCA than TD3 and static-TD3 in all environments. It is also higher than SAC in three environments out of five. This metric is important to differentiate between two agents with similar final performance but very different LCA.

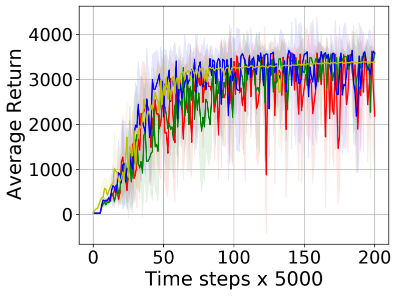
Performance. Table 4.2 shows the average return (R) over the last 10 evaluations. DS-TD3 outperforms TD3 in all environments. Interestingly, it improves TD3 performance by 2.75%, 20.48%, 64.18%, 16.88%, and 37.51% on



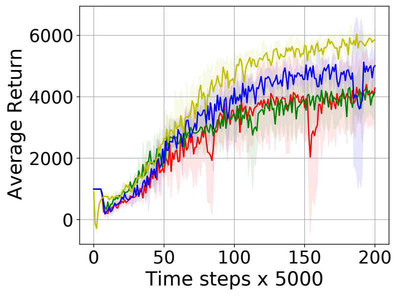
(a) HalfCheetah-v3.



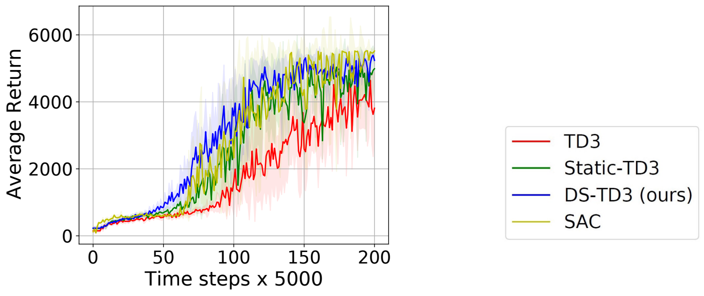
(b) Walker2d-v3.



(c) Hopper-v3.



(d) Ant-v3.



(e) Humanoid-v3.

Figure 4.1: Learning curves of the studied algorithms on different continuous control tasks. The shaded region represents the standard deviation of the average evaluation over 5 runs.

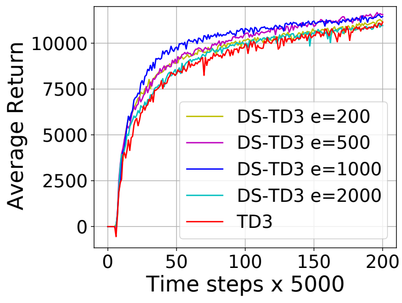


Figure 4.2: The learning curves of DS-TD3 on HalfCheetah-v3 using different adaptation schedules.

HalfCheetah-v3, Walker2d-v3, Hopper-v3, Ant-v3, and Humanoid-v3, respectively. Static-TD3 performs closely to TD3 in most cases except for Humanoid-v3, where Static-TD3 outperforms TD3 by 30.98%. DS-TD3 has a better final performance than SAC in three environments.

4.5 Analysis

4.5.1 Memory and Computation Costs

We analyze the costs needed for the training process by calculating the FLOPS and #params for the actor and critics. We perform this analysis on Half-Cheetah-v3. #params for dense TD3 is 214784, which requires $1\times(1.07e14)$ FLOPs to train. With our DS-TD3, we can find a much smaller topology that can effectively learn the policy and the function value, achieving higher performance than TD3 with a sparsity level of 51%. This consequently reduces the number of required FLOPs to $0.49\times(1.07e14)$.

4.5.2 Adaptation Schedule

We analyze the effect of the adaptation schedule on performance. In particular, we ask how frequently the sparse topology should be adapted. We perform this analysis on HalfCheetah-v3.

Figure 4.2 shows the learning curves of DS-TD3 using different adaptation schedules controlled by the hyperparameter e (Section 4.3). Adapting the topol-

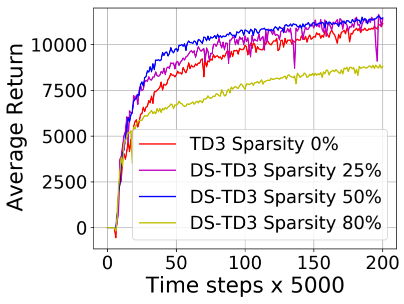
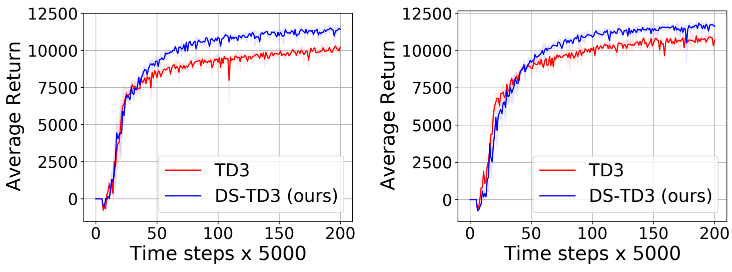


Figure 4.3: The learning curves of DS-TD3 on HalfCheetah-v3 using different sparsity levels.

ogy very frequently ($e \in \{200, 500\}$) would not allow the connections to grow and learn in the dynamic changing nature of RL. The current adaptation process could remove some promising newly added connections from the past adaptation process. This would be caused by a biased estimate of a connection's importance as it becomes a factor of the length of its lifetime. Hence, the very frequent adaptation would increase the chance of replacing some promising topologies. With less frequent adaptation cycles, $e = 1000$, DS-TD3 can learn faster and eventually achieves higher performance than other baselines. Giving the connections a chance to learn helps in having better estimates of the importance of the connections. Hence, it enables finding more effective topologies by replacing the least effective connections with ones that better fit the data. However, increasing the gap between every two consecutive adaptation processes to 2000 steps decreases the exploration speed of different topologies. As illustrated in the figure, DS-TD3 with $e = 2000$ has a close learning behavior and final performance to TD3. Yet, it still offers a substantial reduction in memory and computation costs. This analysis reveals the importance of the adaptation schedule in the success of introducing DST to the DRL field.

4.5.3 Sparsity Level

We analyze the performance of our proposed method using different sparsity levels. Figure 4.3 shows the learning curves of dense TD3 and DS-TD3. By removing 25% of the connections and training the sparse topology dynamically using DS-TD3, we can achieve a faster learning speed and a performance in-



(a) (b)

Figure 4.4: Learning curves of agents that start training with samples drawn from policies trained for 5×10^5 (a) and 7×10^5 steps (b).

crease of 2.11%. More interestingly, with a higher reduction in the size of the networks by 50%, we achieve a much faster learning speed. However, when the network has a very high sparsity level (i.e., 80%), it fails to learn effective representations for the reinforcement learning setting. Learning DRL agents using very highly sparse networks is still an open-challenging task.

4.5.4 Learning Behavior and Speed

DRL agents learn through trial-and-error due to the lack of true labels. An agent starts training with samples generated from a purely exploratory policy, and new samples are drawn from the learning policy over time. Our results show that dynamic sparse agents have faster adaptability to the newly improved samples, thanks to the generalization ability of sparse neural networks [HABN+21]. This leads to higher learning speed, especially at the beginning of the training. We hypothesize that dense neural networks, being over-parameterized, are more prone to memorize and overfit the inaccurate samples. A longer time is required to adapt to the newly added samples by the improved policy and forget the old ones.

To validate this hypothesis, we analyze the behavior of a dense TD3 agent when it starts training with samples generated from a learned policy instead of a purely exploratory one. We use two learned policies trained for 5×10^5 and 7×10^5 steps to draw the initial samples. We perform this experiment on HalfCheetah-v3. As illustrated in Figure 4.4, the learning speed of DS-TD3 and TD3 becomes close to each other at the beginning. Afterward, DS-TD3 performs better than TD3 since the new samples are generated from the current learning policies. With initial samples drawn from a more improved policy (Figure 4.4b), dense TD3 learns faster. It performs better than the dense baseline that starts learning with samples drawn from the policy trained for 5×10^5 steps (Figure 4.4a). On the other hand, DS-TD3 is more robust to over-fitting, less affected by the initial samples, and quickly adapts to the improved ones over time.

4.6 Conclusion and Future Work

Introducing dynamic sparse training principles to the deep reinforcement learning field provides an efficient training process for DRL agents. Our dynamic sparse agents achieve higher performance than the state-of-the-art methods while reducing the memory and computation costs by 50%. Optimizing the sparse topology during training to adapt to the incoming data increases the learning speed. Our findings show the potential of dynamic sparse training in advancing the DRL field. This would open the path to efficient DRL agents that could be trained and deployed on low-resource devices where memory and computation are strictly constrained.

Further research on learning with higher sparse networks can lead to more efficient agents. Exploring other dynamic sparse training approaches would also improve the adaptability to the changing distribution. A deeper investigation of the reasons behind the less ability of dense networks to new samples would help address this challenge.

Chapter 5

The Dormant Neuron Phenomenon in Deep Reinforcement Learning

The slower learning speed and less adaptability to new samples exhibited by dense neural networks contribute to prolonged training times, as discussed in the previous chapter. To effectively tackle these challenges, it is crucial to understand the underlying reasons behind these limitations. In this chapter, we investigate the behavior of dense neural networks under the learning dynamics of RL. Our study reveals the existence of the dormant neuron phenomenon in deep reinforcement learning, where an agent's network suffers from an increasing number of inactive neurons, thereby affecting network expressivity and ability to learn. We demonstrate the presence of this phenomenon across a variety of algorithms and environments, and highlight its effect on learning. To address this issue, we propose a simple and effective method (ReDo) that Recycles Dormant neurons throughout training. Our experiments demonstrate that ReDo maintains the expressive power of networks by reducing the number of dormant neurons and results in improved performance. Our code is available at `https://github.com/google/dopamine/tree/master/dopamine/labs/redo`.

This chapter is integrally based on **Ghada Sokar**, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. *The dormant neuron phenomenon in deep reinforcement learning,* International Conference on Machine Learning (ICML), **Oral**, 2023.

5.1 Introduction

The use of deep neural networks as function approximators for value-based reinforcement learning (RL) has been one of the core elements that has enabled scaling RL to complex decision-making problems [MKS$^+$15, SHM$^+$16, BCC$^+$20]. However, their use can lead to training difficulties that are not present in traditional RL settings. Numerous improvements have been integrated with RL methods to address training instability, such as the use of target networks, prioritized experience replay, multi-step targets, among others [HMVH$^+$18]. In parallel, there have been recent efforts devoted to better understanding the behavior of deep neural networks under the learning dynamics of RL [vHDS$^+$18, FKSL19, KAGL21, BPP20, LRD21, ACC21].

Recent work in so-called “scaling laws” for supervised learning problems suggest that, in these settings, there is a positive correlation between performance and the number of parameters [HNA$^+$17, KMH$^+$20, ZKHB22]. In RL, however, there is evidence that the networks *lose* their expressivity and ability to fit new targets over time, despite being over-parameterized [KAGL21, LRD21]; this issue has been partly mitigated by perturbing the learned parameters. In [IFL$^+$20, NSD$^+$22], some, or all, of the layers of an agent’s neural networks are periodically *reset*, leading to improved performance. These approaches, however, are somewhat drastic: reinitializing the weights can cause the network to “forget” previously learned knowledge and require many gradient updates to recover.

In this work, we seek to understand the underlying reasons behind the loss of expressivity during the training of RL agents. The observed decrease in the learning ability over time raises the following question: *Do RL agents use neural network parameters to their full potential?* To answer this, we analyze neuron activity throughout training and track *dormant* neurons: neurons that have become practically inactive through low activations. Our analyses reveal that the number of dormant neurons increases as training progresses, an effect we coin the “*dormant neuron phenomenon*”. Specifically, we find that while agents start the training with a small number of dormant neurons, this number increases as training progresses. The effect is exacerbated by the number of gradient updates taken per data collection step. This is in contrast with supervised learning, where the number of dormant neurons remains low throughout training.

We demonstrate the presence of the dormant neuron phenomenon across different algorithms and domains: in two value-based algorithms on the Arcade Learning Environment [BNVB13] (DQN [MKS$^+$15] and DrQ(ϵ) [YKF21, ASC$^+$21]), and with an actor-critic method (SAC [HZAL18]) evaluated on the

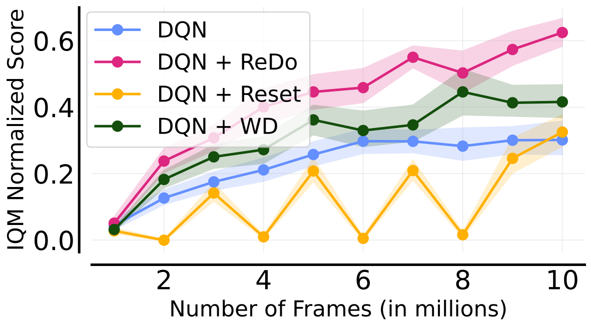


Figure 5.1: **Sample efficiency curves** for DQN, with a replay ratio of 1, when using network resets [NSD+22], weight decay (WD), and our proposed *ReDo*. Shaded regions show 95% CIs. The figure shows interquartile mean (IQM) human-normalized scores over the course of training, aggregated across 17 Atari games and 5 runs per game. Among all algorithms, *DQN+ReDo* performs the best.

MuJoCo suite [TET12]. To address this issue, we propose *Re*cycling *Do*rmant neurons (*ReDo*), a simple and effective method to avoid network under-utilization during training without sacrificing previously learned knowledge: we explicitly limit the spread of dormant neurons by "recycling" them to an active state. *ReDo* consistently maintains the capacity of the network throughout training and improves the agent's performance (see Figure 5.1). Our contributions in this work can be summarized as follows:

- We demonstrate the existence of the dormant neuron phenomenon in deep RL.

- We investigate the underlying causes of this phenomenon and show its negative effect on the learning ability of deep RL agents.

- We propose *Re*cycling *Do*rmant neurons (*ReDo*), a simple method to reduce the number of dormant neurons and maintain network expressivity during training.

- We demonstrate the effectiveness of *ReDo* in maximizing network utilization and improving performance.

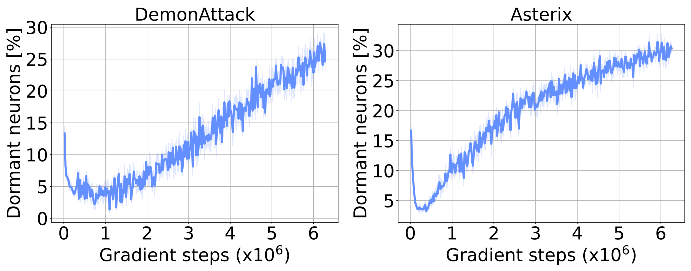


Figure 5.2: The percentage of dormant neurons increases throughout training for DQN agents.

5.2 The Dormant Neuron Phenomenon

Prior work has highlighted the fact that networks used in online RL tend to *lose* their expressive ability; in this section we demonstrate that *dormant neurons* play an important role in this finding.

Definition 5.1. Given an input distribution D, let $h_i^\ell(x)$ denote the activation of neuron i (a neuron's output) in layer ℓ under input $x \in D$ and H^ℓ be the number of neurons in layer ℓ. We define the score of a neuron i (in layer ℓ) via the normalized average of its activation as follows:

$$s_i^\ell = \frac{\mathbb{E}_{x\in D}|h_i^\ell(x)|}{\frac{1}{H^\ell}\sum_{k\in h}\mathbb{E}_{x\in D}|h_k^\ell(x)|} \tag{5.1}$$

We say a neuron i in layer ℓ is **τ-dormant** if $s_i^\ell \le \tau$.

The threshold τ allows us to detect neurons with low activations. Even though these low activation neurons could, in theory, impact the learned functions when recycled, their impact is expected to be less than the neurons with high activations.

Definition 5.2. An algorithm exhibits the **dormant neuron phenomenon** if the number of τ-dormant neurons in its neural network increases steadily throughout training.

An algorithm exhibiting the dormant neuron phenomenon is not using its network's capacity to its full potential, and this under-utilization worsens over time.

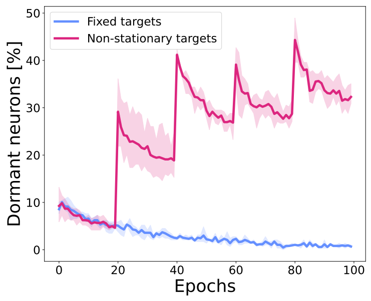


Figure 5.3: Percentage of dormant neurons when training on CIFAR-10 with fixed and non-stationary targets. Averaged over 3 independent seeds with shaded areas reporting 95% confidence intervals. The percentage of dormant neurons increases with non-stationary targets.

The remainder of this section focuses first on demonstrating that RL agents suffer from the dormant neuron phenomenon, and then on understanding the underlying causes for it. Specifically, we analyze DQN [MKS$^+$15], a foundational agent on which most modern value-based agents are based. To do so, we run our evaluations on the Arcade Learning Environment [BNVB13] using 5 independent seeds for each experiment, and reporting 95% confidence intervals. For clarity, we focus our analyses on two representative games (DemonAttack and Asterix), but include others in Appendix C.2. In these initial analyses, we focus solely on $\tau = 0$ dormancy, but loosen this threshold when benchmarking our algorithm in sections 5.3 and 5.4. Additionally, we present analyses on an actor-critic method (SAC [HZAL18]) and a modern sample-efficient agent (DrQ(ϵ) [YKF21]) in Appendix C.2.

The dormant neuron phenomenon is present in deep RL agents. We begin our analyses by tracking the number of dormant neurons during DQN training. In Figure 5.2, we observe that the percentage of dormant neurons steadily increases throughout training. This observation is consistent across different algorithms and environments, as can be seen in Appendix C.2

Target non-stationarity exacerbates dormant neurons. We hypothesize that the non-stationarity of training deep RL agents is one of the causes for the

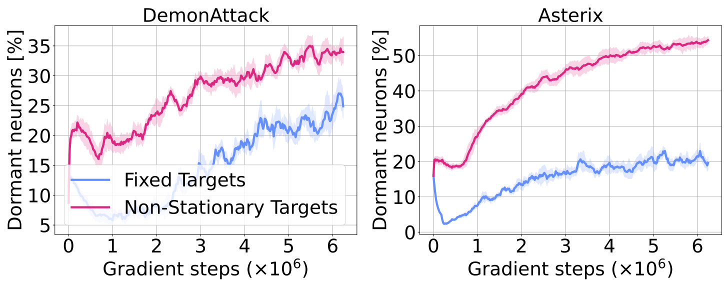


Figure 5.4: *Offline RL.* Dormant neurons throughout training with standard moving targets and fixed (random) targets. The phenomenon is still present in offline RL, where the training data is fixed.

dormant neuron phenomenon. To evaluate this hypothesis, we consider two supervised learning scenarios using the standard CIFAR-10 dataset [KH+09]: (1) training a network with *fixed targets*, and (2) training a network with *non-stationary targets*, where the labels are shuffled throughout training. Experimental details can be found in Appendix C.1. As Figure 5.3 shows, the number of dormant neurons *decreases* over time with fixed targets, but *increases* over time with non-stationary targets. Indeed, the sharp increases in the figure correspond to the points in training when the labels are shuffled. These findings suggest that the continuously changing targets in deep RL are a significant factor for the presence of the phenomenon.

Input non-stationarity does not appear to be a major factor. To investigate whether the non-stationarity due to online data collection plays a role in exacerbating the phenomenon, we measure the number of dormant neurons in the *offline* RL setting, where an agent is trained on a fixed dataset (we used the dataset provided by [ASN20]). In Figure 5.4 we can see that the phenomenon remains in this setting, suggesting that input non-stationary is not one of the primary contributing factors. To further analyze the source of dormant neurons in this setting, we train RL agents with *fixed* random targets (ablating the non-stationarity in inputs and targets). The decrease in the number of dormant neurons observed in this case (Figure 5.4) supports our hypothesis that target non-stationarity in RL training is the primary source of the dormant neuron phenomenon.

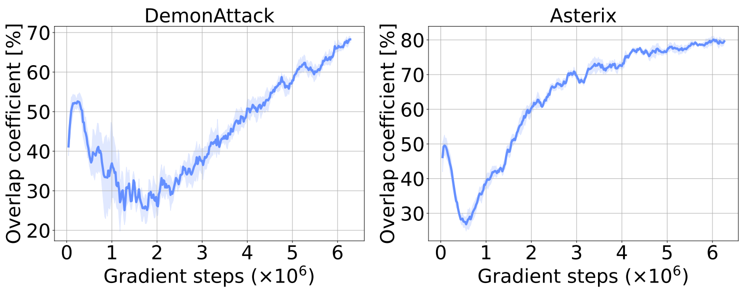


Figure 5.5: The overlap coefficient of dormant neurons throughout training. There is an increase in the number of dormant neurons that remain dormant.

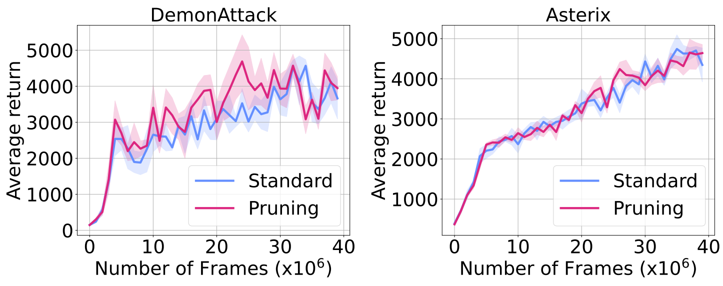


Figure 5.6: Pruning dormant neurons during training does not affect the performance of an agent.

Dormant neurons remain dormant. To investigate whether dormant neurons "reactivate" as training progresses, we track the overlap in the set of dormant neurons. Figure 5.5 plots the overlap coefficient between the set of dormant neurons in the penultimate layer at the current iteration, and the historical set of dormant neurons. The increase shown in the figure suggests that once a neuron becomes dormant, it remains that way for the rest of the training. To further investigate this, we explicitly *prune* any neuron found dormant throughout training, to check whether their removal affects the agent's overall performance. As Figure 5.6 shows, their removal does *not* affect the agent's

The overlap coefficient between two sets X and Y is defined as $overlap(X, Y) = \frac{|X \cap Y|}{\min(|X|,|Y|)}$.

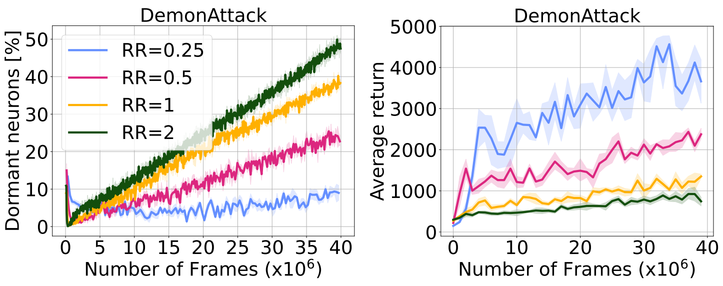


Figure 5.7: The rate of increase in dormant neurons with varying replay ratio (RR) (left). As the replay ratio increases, the number of dormant neurons also increases. The higher percentage of dormant neurons correlates with the performance drop that occurs when the replay ratio is increased (right).

performance, further confirming that dormant neurons remain dormant.

More gradient updates lead to more dormant neurons. Although an increase in replay ratio can seem appealing from a data-efficiency point of view (as more gradient updates per environment step are taken), it has been shown to cause overfitting and performance collapse [KAGL21, NSD$^+$22]. In Figure 5.7, we measure neuron dormancy while varying the replay ratio, and observe a strong correlation between replay ratio and the fraction of neurons turning dormant. Although difficult to assert conclusively, this finding could account for the difficulty in training RL agents with higher replay ratios; indeed, we will demonstrate in Section 5.4 that recycling dormant neurons and activating them can mitigate this instability, leading to better results.

Dormant neurons make learning new tasks more difficult. We directly examine the effect of dormant neurons on an RL network's ability to learn new tasks. To do so, we train a DQN agent with a replay ratio of 1 (this agent exhibits a high level of dormant neurons as observed in Figure 5.7). Next we fine-tune this network by distilling it towards a well performing DQN agent's network, using a traditional regression loss and compare this with a randomly initialized agent trained using the same loss. More details can be found in Appendix C.1. In Figure 5.8 we see that the pre-trained network, which starts with a high level of dormant neurons, shows degrading performance throughout training; in con-

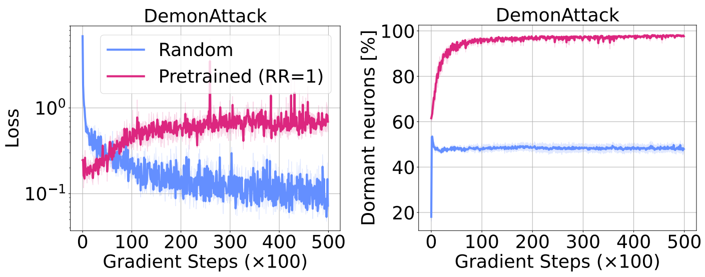


Figure 5.8: A pretrained network that exhibits dormant neurons has less ability than a randomly initialized network to fit a fixed target. Results are averaged over 5 seeds.

trast, the randomly initialized baseline is able to continuously improve. Further, while the baseline network maintains a stable level of dormant neurons, the number of dormant neurons in the pre-trained network continues to increase throughout training.

5.3 *Re*cycling *Do*rmant Neurons (*ReDo*)

Our analyses in Section 5.2, which demonstrates the existence of the dormant neuron phenomenon in online RL, suggests these dormant neurons may have a role to play in the diminished expressivity highlighted by [KAGL21] and [LRD21]. To account for this, we propose to **re**cycle **do**rmant neurons periodically during training (*ReDo*).

The main idea of *ReDo*, outlined in Algorithm 9, is rather simple: during regular training, periodically check in all layers whether any neurons are τ-dormant; for these, reinitialize their incoming weights and zero out the outgoing weights. The incoming weights are initialized using the original weight distribution. Note that if τ is 0, we are effectively leaving the network's output unchanged; if τ is small, the output of the network is only slightly changed.

Figure 5.9 showcases the effectiveness of *ReDo* in dramatically reducing the number of dormant neurons, which also results in improved agent performance. Before diving into a deeper empirical evaluation of our method in Section 5.4, we discuss some algorithmic alternatives we considered when designing *ReDo*.

Algorithm 9 *ReDo*

```
**Input:** Network parameters $\theta$, threshold $\tau$, training steps $T$, frequency $F$
for $t = 1$ to to $T$ do
    Update $\theta$ with regular RL loss
    if $t \mod F == 0$ then
        for each neuron $i$ do
            if $s_i^\ell \leq \tau$ then
                Reinitialize input weights of neuron $i$
                Set outgoing weights of neuron $i$ to 0
            end if
        end for
    end if
end for
```

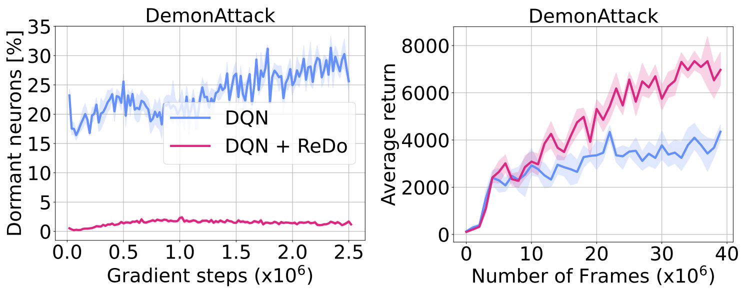


Figure 5.9: Evaluation of *ReDo*'s effectiveness (with $\tau = 0.025$) in reducing dormant neurons (left) and improving performance (right) on DQN (with $RR = 0.25$).

Alternate recycling strategies. We considered other recycling strategies, such as scaling the incoming connections using the mean of the norm of non-dormant neurons. However, this strategy performed similarly to using initial weight distribution. Similarly, alternative initialization strategies like initializing outgoing connections randomly resulted in similar or worse returns. Results of these investigations are shared in Appendix C.3.2.

Are ReLUs to blame? RL networks typically use ReLU activations, which can saturate at zero outputs, and hence zero gradients. To investigate whether the issue is specific to the use of ReLUs, in Appendix C.3.1, we measured the number of dormant neurons and resulting performance when using a different activation function. We observed that there is a mild decrease in the number of dormant neurons, but the phenomenon is still present.

5.4 Empirical Evaluations

Agents, architectures, and environments. We evaluate DQN on 17 games from the Arcade Learning Environment [BNVB13] (as used in [KAGL21,KAM+21] to study the loss of network expressivity). We study two different architectures: the default CNN used by [MKS+15], and the ResNet architecture used by the IMPALA agent [ESM+18].

Additionally, we evaluate DrQ(ϵ) [YKF21, ASC+21] on the 26 games used in the Atari 100K benchmark [KBM+19], and SAC [HZAL18] on four MuJoCo environments [TET12].

Implementation details. All our experiments and implementations were conducted using the Dopamine framework [CMG+18]. For agents trained with *ReDo*, we use a threshold of $\tau = 0.1$, unless otherwise noted, as we found this gave a better performance than using a threshold of 0 or 0.025. When aggregating results across multiple games, we report the Interquantile Mean (IQM), recommended by [ASC+21] as a more statistically reliable alternative to median or mean, using 5 independent seeds for each DQN experiment, 10 for the DrQ and SAC experiments, and reporting 95% stratified bootstrap confidence intervals.

5.4.1 Consequences for Sample Efficiency

Motivated by our finding that higher replay ratios exacerbate dormant neurons and lead to poor performance (Figure 5.7), we investigate whether *ReDo* can help mitigate these. To do so, we report the IQM for four replay ratio values: 0.25 (default for DQN), 0.5, 1, and 2 when training with and without *ReDo*. Since increasing the replay ratio increases the training time and cost, we train DQN for 10M frames, as opposed to the regular 200M. As the top-left plot in Figure 5.10 demonstrates, *ReDo* is able to avoid the performance collapse when increasing replay ratios, and even to benefit from the higher replay ratios when trained with *ReDo*.

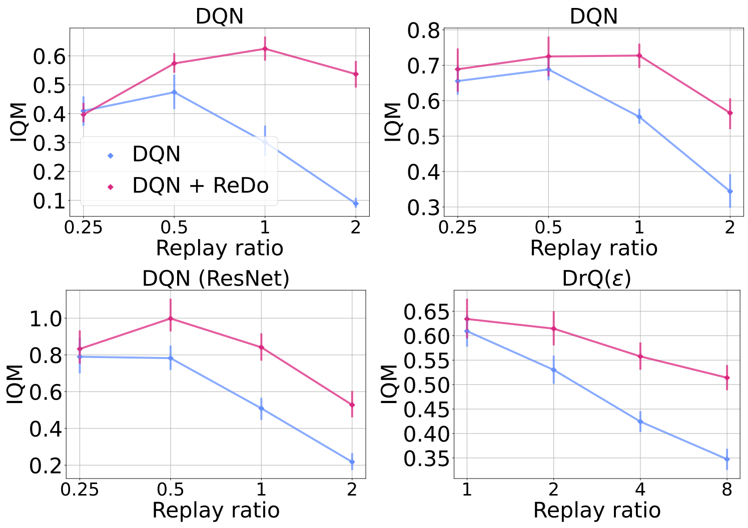


Figure 5.10: Evaluating the effect of increased replay ratio with and without *ReDo*. From left to right (top to bottom): DQN with default settings, DQN with n-step of 3, *DQN* with the ResNet architecture, and DrQ(ϵ). We report results using 5 seeds, while DrQ(ϵ) use 10 seeds; error bars report 95% confidence intervals.

Impact on multi-step learning. In the top-right plot of Figure 5.10 we added n-step returns with a value of $n = 3$ [SB18]. While this change results in a general improvement in DQN's performance, it still suffers from performance collapse with higher replay ratios; *ReDo* mitigates this and improves performance across all values.

Varying architectures. To evaluate *ReDo*'s impact on different network architectures, in the bottom-left plot of Figure 5.10 we replace the default CNN architecture used by DQN with the ResNet architecture used by the IMPALA agent [ESM+18]. We see a similar trend: *ReDo* enables the agent to make better use of higher replay ratios, resulting in improved performance.

Varying agents. We evaluate on a sample-efficient value-based agent DrQ(ϵ) [YKF21, ASC+21]) on the Atari 100K benchmark in the bottom-right plot of

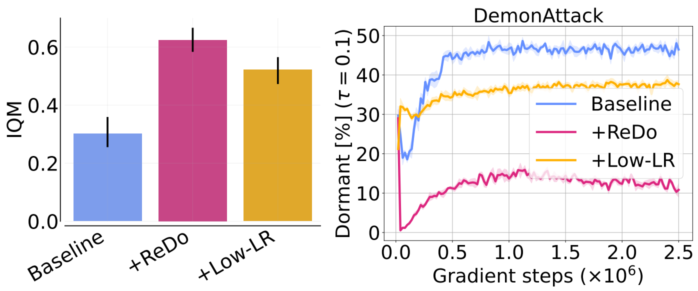


Figure 5.11: Effect of reduced learning rate in high replay ratio setting. Scaling learning rate helps, but does not solve the dormant neuron problem. Aggregated results across 17 games (left) and the percentage of dormant neurons during training on DemonAttack (right).

Figure 5.10. In this setting, we train for 400K steps, where we can see the effect of dormant neurons on performance, and study the following replay ratio values: 1 (default), 2, 4, 8. Once again, we observe *ReDo*'s effectiveness in improving performance at higher replay ratios.

In the rest of this section, we do further analyses to understand the improved performance of *ReDo* and how it fares against related methods. We perform this study on a DQN agent trained with a replay ratio of 1 using the default CNN architecture.

5.4.2 Learning Rate Scaling

An important point to consider is that the default learning rate may not be optimal for higher replay ratios. Intuitively, performing more gradient updates would suggest a *reduced* learning rate would be more beneficial. To evaluate this, we decrease the learning rate by a factor of four when using a replay ratio of 1 (four times the default value). Figure 5.11 confirms that a lower learning rate reduces the number of dormant neurons and improves performance. However, the percentage of dormant neurons is still high, and using *ReDo* with a high replay ratio and the default learning rate obtains the best performance.

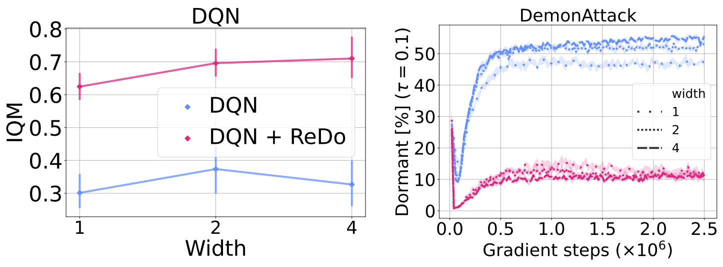


Figure 5.12: Performance of DQN trained with $RR = 1$ using different network width. Increasing the width of the network slightly improves the performance. Yet, the performance gain does not reach the gain obtained by *ReDo*. *ReDo* improves the performance across different network sizes.

5.4.3 Is Over-parameterization Enough?

In [LRD21, FKSL19], it is suggested that sufficiently over-parameterized networks can fit new targets over time; this raises the question of whether over-parameterization can help address the dormant neuron phenomenon. To investigate this, we increase the size of the DQN network by doubling and quadrupling the width of its layers (both the convolutional and fully connected). The left plot in Figure 5.12 shows that larger networks have at most a mild positive effect on the performance of DQN, and the resulting performance is still far inferior to that obtained when using *ReDo* with the default width. Furthermore, training with *ReDo* seems to improve as the network size increases, suggesting that the agent is able to better exploit network parameters, compared to when training without *ReDo*.

An interesting finding in the right plot in Figure 5.12 is that the percentage of dormant neurons is similar across the varying widths. As expected, the use of *ReDo* dramatically reduces this number for all values. This finding is somewhat at odds with that from [SDX+20]. They demonstrated that, in supervised learning settings, increasing the width decreases the gradient confusion and leads to faster training. If this observation would also hold in RL, we would expect to see the percentage of dormant neurons decrease in larger models.

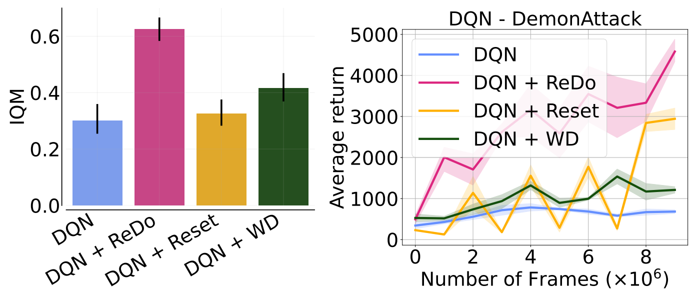


Figure 5.13: Comparison of the performance for *ReDo* and two different regularization methods (Reset [NSD+22] and weight decay (WD)) when integrated with training DQN agents. Aggregated results across 17 games (left) and the learning curve on DemonAttack (right).

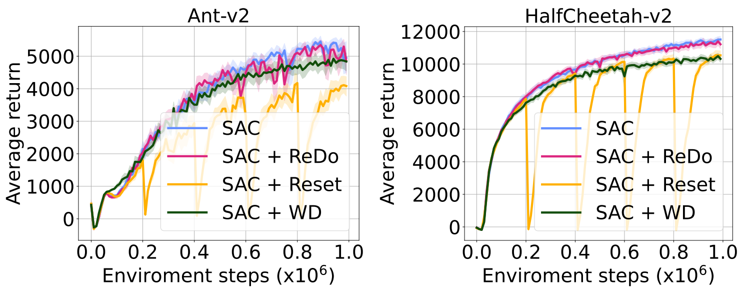


Figure 5.14: Comparison of the performance of SAC agents with *ReDo* and two different regularization methods (Reset [NSD+22] and weight decay (WD)). See Figure C.5 for other environments.

5.4.4 Comparison with Related Methods

In [NSD+22], it was also observed performance collapse when increasing the replay ratio, but attributed this to overfitting to early samples (an effect they refer to as the "primacy bias"). To mitigate this, they proposed periodically resetting the network, which can be seen as a form of regularization. We compare the performance of *ReDo* against theirs, which periodically resets only the penultimate layer for Atari environments. Additionally, we compare to adding weight decay, as this is a simpler, but related, form of regularization. It is worth

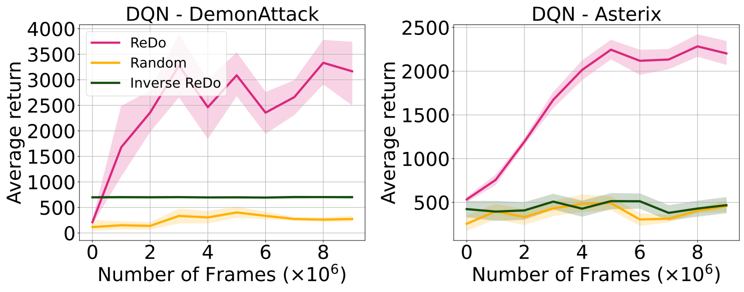


Figure 5.15: Comparison of different strategies for selecting the neurons that will be recycled. Recycling neurons with the highest score (Inverse *ReDo*) or random neurons causes performance collapse.

highlighting that [NSD+22] also found high values of replay ratio to be more amenable to their method. As Figure 5.13 illustrates, weight decay is comparable to periodic resets, but *ReDo* is superior to both.

We continue our comparison with resets and weight decay on two MuJoCo environments with the SAC agent [HZAL18]. As Figure 5.14 shows, *ReDo* is the only method that does not suffer performance degradation. The results on other environments can be seen in Appendix C.2.

5.4.5 Neuron Selection Strategies

Finally, we compare our strategy for selecting the neurons that will be recycled (Section 5.2) against two alternatives: (1) **Random:** neurons are selected randomly, and (2) **Inverse *ReDo*:** neurons with the *highest* scores according to Equation 5.1 are selected. To ensure a fair comparison, the number of recycled neurons is a fixed percentage for all methods, occurring every 1000 steps. The percentage of neurons to recycle follows a cosine schedule starting at 0.1 and ending at 0. As Figure 5.15 shows, recycling active or random neurons hinders learning and causes performance collapse.

5.5 Related Work

Function approximators in RL. The use of over-parameterized neural networks as function approximators was instrumental to some of the successes in RL, such as achieving superhuman performance on Atari 2600 games [MKS+15] and continuous control [LHP+16]. Recent works observe a change in the network's capacity over the course of training, which affects the agent's performance. In [KAGL21, KAM+21], it is shown that the expressivity of the network decreases gradually due to bootstrapping. Gulcehre et al. [GSS+22] investigate the sources of expressivity loss in offline RL and observe that underparamterization emerges with prolonged training. Lyle et al. [LRD21] demonstrate that RL agents lose their ability to fit new target functions over time, due to the non-stationary in the targets. Similar observations have been found, referred to as plasticity loss, in the continual learning setting where the data distribution is changing over time [BCD+21,DMS21]. These observations call for better understanding how RL learning dynamics affect the capacity of their neural networks.

There is a recent line of work investigating network topologies by using sparse neural networks in online [GEEC22, SMM+22, THP+22] and offline RL [AOPP21]. They show up to 90% of the network's weights can be removed with minimal loss in performance. This suggests that RL agents are not using the capacity of the network to its full potential.

Generalization in RL. RL agents are prone to overfitting, whether it is to training environments, reducing their ability to generalize to unseen environments [KZGR21], or to early training samples, which degrades later training performance [FKSL19, NSD+22]. Techniques such as regularization [HIH+21, WKSF20], ensembles [CWZR20], or data augmentation [FWH+21,JFZL19,HSW21] have been adopted to account for overfitting.

Another line of work addresses generalization via *re-initializing* a subset or all of the weights of a neural network during training. This technique is mainly explored in supervised learning [TSD21, ZVLC21, AMK21, ZBK+23], transfer learning [LXA+20], and online learning [AA20]. A few recent works have explored this for RL: Igl et al. [IFL+20] periodically reset an agent's full network and then perform distillation from the pre-reset network. Nikishin et al. [NSD+22] (already discussed in Figure 5.13) periodically resets the last layers of an agent's network. Despite its performance gains, fully resetting some or all layers can lead to the agent "forgetting" prior learned knowledge. The authors account for this by using a sufficiently large replay buffer, so as to never discard any observed experience; this, however, makes it difficult to scale to environments with more environment interactions. Further, recovering performance after each reset requires many gradient updates. Similar to our approach, Dohare et al. [DMS21] adapt the stochastic gradient descent by resetting the smallest utility features for *continual learning*. We compare their utility metric to the one used by *ReDo* in Appendix C.3.4 and observe similar or worse performance.

Neural network growing. A related research direction is to *prune* and *grow* the architecture of a neural network. On the growing front, Evci et al. [EvMU+21] and Dai et al. [DYJ19] proposed gradient-based strategies to grow new neurons in dense and sparse networks, respectively. Yoon et al. [YYLH18] and Wu et al. [WWL19] proposed methods to split existing neurons. Zhou et al. [ZSL12] add new neurons and merge similar features for online learning.

5.6 Conclusion and Future Work

In this chapter, we identified the *dormant neuron phenomenon* whereby, during training, an RL agent's neural network exhibits an increase in the number of neurons with little-or-no activation. We demonstrated that this phenomenon is present across a variety of algorithms and domains, and provided evidence that it does result in reduced expressivity and inability to adapt to new tasks.

Interestingly, studies in neuroscience have found similar types of dormant neurons (precursors) in the adult brain of several mammalian species, including humans [BCD22], albeit with different dynamics. Certain brain neurons *start off* as dormant during embryonic development, and progressively awaken with age, eventually becoming mature and functionally integrated as excitatory neurons [RBB+18, BDK+20, BCD22]. Contrastingly, the dormant neurons we investigate here emerge over time and exacerbate with more gradient updates.

To overcome this issue, we proposed a simple method (*ReDo*) to maintain network utilization throughout training by periodic recycling of dormant neurons. The simplicity of *ReDo* allows for easy integration with existing RL algorithms. Our experiments suggest that this can lead to improved performance. Indeed, the results in Figure 5.10 and 5.12 suggest that *ReDo* can be an important component in being able to successfully scale RL networks in a sample-efficient manner.

Limitations and future work. Although the simple approach of recycling neurons we introduced yielded good results, it is possible that better approaches exist. For example, *ReDo* reduces dormant neurons significantly but it doesn't completely eliminate them. Further research on initialization and optimization of the recycled capacity can address this and lead to improved performance. Additionally, the dormancy threshold is a hyperparameter that requires tuning; having an adaptive threshold over the course of training could improve performance even further. Finally, further investigation into the relationship between the task's complexity, network capacity, and the dormant neuron phenomenon would provide a more comprehensive understanding.

Similarly to the findings of [GEEC22], this work suggests there are important gains to be had by investigating the network architectures and topologies used for deep reinforcement learning. Research presented in [KH23] has demonstrated that integrating a linear approximation operator into neural networks improves their approximation capabilities. Investigating this approach in the DRL paradigm could improve learning ability and the expressivity of networks. Finally, the observed network's behavior during training (i.e., the change in the network capacity utilization), which differs from supervised learning, indicates a need to explore optimization techniques specific to reinforcement learning due to its unique learning dynamics.

Chapter 6

Conclusions and Future Work

In this chapter, we summarize the main research questions addressed in this thesis and our key findings. We further discuss the limitations of our proposed approaches. Finally, we conclude by outlining several promising avenues for future research.

6.1 Conclusions

Deep neural networks have achieved remarkable success in many tasks, surpassing human-level performance in various domains [HZRS15]. However, they lack the innate human ability to learn and adapt to new and changing data continually. This ability is crucial for many applications in which data evolve over time, such as autonomous driving, chatbots, and healthcare systems. Recently, more efforts have been devoted to enabling deep neural networks to learn on changing data distributions and adapt over time to new samples while maintaining previously learned knowledge.

In this thesis, we explored two learning paradigms within this learning regime: continual learning and reinforcement learning. In the continual learning paradigm, a model learns a series of tasks sequentially, aiming to maintain performance on all previously learned tasks. Specifically, we focus on addressing the class-incremental learning scenario (CIL) within continual learning, where each task introduces a new set of classes. In contrast, the reinforcement learning paradigm involves receiving new samples over time during training, but these samples originate from a single environment (task). Within reinforcement learning, various sources of non-stationarity emerge due to the absence of true targets and the distinctive learning nature of RL.

Our main goal is to investigate the behavior of deep neural networks in this learning regime and provide new approaches that have improved performance while being memory and computationally efficient. To this end, we addressed the following research questions in this thesis.

(Q1) Learning effective representation for continual learning. In Chapter 2, we addressed the main challenge in the continual learning paradigm, catastrophic forgetting; a model forgets the previously learned representation when it learns a new one. We demonstrated that this challenge increases when other continual learning desiderata are considered, such as using a fixed-capacity model and inaccessibility of old data. To address this challenge, we proposed the first dynamic sparse training algorithm for continual learning (SpaceNet). SpaceNet dynamically trains a sparse neural network from scratch for each task and optimizes its topology during training to produce sparse representations. We demonstrated the effectiveness of sparse representations in reducing interference between tasks and forgetting. SpaceNet outperforms regularization-based methods that learn dense representations for each task by a large margin. In addition, it has higher performance than architectural-based methods that require expanding the model capacity. Using sparse subnetworks allows learning multiple tasks within a fixed capacity model. Furthermore, it improves memory and computational efficiency significantly. Our findings have paved the way for further investigations, demonstrating the effectiveness of sparse networks in the context of continual learning [GD22, WZG+22, KMM+22, GMD23].

(Q2) Promoting forward transfer in continual learning. Another important aspect of the continual learning paradigm is using previously learned knowledge to help learn new tasks. In Chapter 3, we highlighted two aspects that promote forward transfer. First, having generic knowledge prior to learning the continual sequence, especially when tasks are dissimilar. Second, selectively transfer the relevant knowledge from the past instead of using all knowledge. To illustrate these two factors, we proposed SAM, a Self-Attention Meta-learner for continual learning. We showed that SAM outperforms other CL approaches from various strategies. Moreover, integrating SAM with state-of-the-art methods improves their performance. Our findings in this study suggest that focusing on addressing forward transfer could also satisfy other continual learning desiderata. For instance, the selective transfer of knowledge results in selective weight updates, which promote backward transfer and reduce forgetting. Learning based on relevant knowledge would increase the learning speed of a task and could reduce the number of training samples required. This, consequently, will reduce the memory and computational costs of training.

(Q3) Reducing the long training time of DRL agents. The challenges of learning on changing data distributions and the unavailability of true targets in the reinforcement learning paradigm result in a prolonged training time. In Chapter 4, we addressed this challenge. More specifically, we aim to enable deep neural networks to adapt faster to new samples and reduce the computational costs of training dense models. To this end, we introduced for the first time dynamic sparse training for deep reinforcement learning. We train a sparse network from scratch and dynamically adapt its topology over time. We showed that dynamic sparse networks have faster learning speeds than dense ones. In addition, they achieve higher performance while reducing memory and computational costs by 50%. Finally, we showed that dense models tend to overfit initial samples, which affects the rest of the learning process. Our findings revealed the potential of dynamic sparse networks in adapting to changing data distributions and opened the path to efficient DRL agents that could be trained and deployed on low-resource devices [GEEC22, THP+22].

(Q4) Understanding the reasons behind the loss of adaptability of dense models in DRL. Gaining insights into the root causes of dense networks gradually losing their capability to fit new samples over time would facilitate the development of improved agents. We addressed this goal in Chapter 5. Our study revealed the existence of the dormant neuron phenomenon, where an agent's network suffers from an increasing number of inactive neurons, thereby affecting network expressivity and ability to learn. The observation is consistent across different algorithms, domains, and architectures. Investigating different sources of non-stationarity in DRL, we found that target non-stationarity is the main source of this phenomenon. We demonstrated that removing dormant neurons from the network does not affect performance. Moreover, networks with more dormant neurons have less ability to fit new targets. We also showed that larger networks do not compensate for dormant neurons. To address this phenomenon, we proposed an effective method (*ReDo*) that periodically Recycles Dormant neurons throughout training. We showed the effectiveness of *ReDo* in improving performance and utilizing the capacity of models efficiently.

6.2 Limitations

In this thesis, we studied forgetting and forward transfer in continual learning separately, highlighting the effective representation types for each aspect. Further work is needed to integrate and address both objectives simultaneously. The achieved performance of our proposed replay-free methods is very promising. Yet, they do not reach the performance of rehearsal-based methods, which rely on retraining old data. Achieving high performance without relying on old data remains an ongoing and challenging problem.

Our investigations within the reinforcement learning paradigm have uncovered distinct behaviors in the network during training, highlighting differences compared to supervised learning. Although we provide simple and effective mechanisms to mitigate the consequences of these behaviors, the development of novel optimization techniques tailored specifically to reinforcement learning would address the root causes of the challenges. Moreover, a deeper understanding of the underlying factors that contribute to phenomena observed in this paradigm, such as overfitting old samples, would contribute to the development of improved agents.

While our proposed methods, which leverage sparsity, enhance performance, and tackle numerous challenges in the studied paradigms, this proof-of-concept does not fully utilize the memory and computational benefits offered by sparse neural networks, similar to many existing sparse training methods in the literature. This is due to the lack of hardware support for sparsity and the higher focus of the community on the algorithmic side [HABN+21]. Nevertheless, there is recent growing attention to hardware and software support for sparsity [HSRN+19, ZMZ+20, WJH+18, LMM+20, CMP21, ABB+19, CYES19]. NVIDIA released NVIDIA A100, which supports a 50% fixed sparsity level [ZMZ+20], and many other efforts on the hardware side are proposed [WJH+18, ABB+19, CYES19, LBV+18]. On the software side, libraries that support truly sparse implementations have been started for supervised learning [LMM+20, CMP21]. With a joint community effort in the algorithmic, software, and hardware directions, we would be able to actually provide faster, memory-efficient, and energy-efficient deep neural networks. Further discussion can be found in [Hoo21, MMP+21].

We demonstrated the potential of leveraging sparsity empirically. Offering theoretical analyses of these findings would support the results and pave the way for innovative approaches to address challenges in this learning regime. Recent works have made some attempts at theoretical analysis [Pog22, GXGP23], suggesting that sparsity is a key property for generalization in modern architectures like transformers. It is essential to devote efforts toward establishing a theoretical foundation for sparsity in networks that learn continually.

6.3 Future Work

In this section, we discuss potential future research directions.

Scalability of continual learning models. While current approaches in continual learning are typically evaluated on benchmarks consisting of 10-20 tasks, the question remains: can we scale these methods to handle hundreds of tasks? Achieving such a goal necessitates exploring various research directions. The feasibility of learning all tasks using a fixed capacity model is an area that requires thorough investigation. Further research on the reusability of learned components and the expansion of specific parts of the model when necessary would help in efficiently utilizing the model capacity to accommodate a large number of tasks. Additionally, determining the similarity between new tasks and previously learned knowledge would aid in making informed decisions regarding model expansion or reuse.

Data centric. Current research in the continual learning paradigm mainly focuses on developing new training algorithms or strategies for expanding network architectures. However, little attention has been given to the role of data in this context. With an abundance of new data available, training on every single sample causes significant memory and computational costs. A model needs to leverage its previously acquired knowledge to learn from only a few samples, thereby saving the labor-intensive process of labeling all samples, reducing training time, and addressing the imbalance between old and new samples. This motivates the exploration of new techniques for selecting important samples from new data and investigating novel meta-learning methods that facilitate fast adaptability using a limited number of samples. Focusing on these aspects can enhance the efficiency and effectiveness of continual learning approaches, making them more practical and applicable in real-world scenarios.

Selective knowledge transfer. We shed light on the importance of forward transfer in the continual learning paradigm. Further work on selecting the relevant knowledge from the past is needed to improve forward transfer. Attention mechanisms play an important role in this research. Exploring modern architectures such as the Vision Transformer (ViT) [DBK$^+$20] within this paradigm is a potential avenue for investigation. In addition, research on new training regimes like meta-learning would help to learn knowledge from each task that can be easily transferred to future tasks.

Studying different types of tasks and learning paradigms. In this thesis, we focused on supervised image classification tasks as the basis for our analysis in continual learning. However, considering the cost involved in labeling each new sample, it suggests exploring unsupervised or semi-supervised techniques within this paradigm. Additionally, there is a need for increased attention to be given to other types of tasks beyond image classification. Various natural language processing applications, including chatbots, language translation, and spam detection, heavily rely on continual learning abilities. Further research in this direction would also open the path to studying the applicability of continual learning to large language models. Furthermore, addressing the continual reinforcement learning scenario is another interesting research direction. Devoting more efforts to tackling non-stationarity within and across tasks in this paradigm would significantly contribute to advancing the field.

Appendices

Appendix A

Additional Experimental Details and Analyses on Chapter 3

A.1 Additional Experiment Details

A.1.1 Continual Learning Training Details

On the Split MNIST benchmark, each task is trained for 5 epochs. We use a batch size of 64. The model is trained using stochastic gradient descent with a Nesterov momentum of 0.9, a learning rate of 0.01, and cross-entropy loss. We search in the space {2,4,8,10,20} for the reduction ratio (r). The standard training/test-split for the MNIST dataset was used, resulting in 60,000 training images and 10,000 test images.

On the Split CIFAR-10/100 benchmark, each task is trained for 30 epochs. We use a batch size of 64. The model is trained using Adam optimizer ($\eta = .001, \beta 1 = 0.9, \beta 2 = 0.999$) and cross-entropy loss. We search in the space {2,4,8,16} for the reduction ratio (r). The CIFAR-10 dataset consists of 10 classes and has 60000 samples (50000 training + 10000 test), with 6000 images per class. In contrast, the CIFAR-100 dataset consists of 100 classes and has 600 images per class (500 train + 100 test).

Each experiment is repeated 5 times with different random seeds.

A.1.2 Meta-training Details

On the MiniImagenet dataset, the self-attention meta-learner is trained using 5-way classification, 5 shot training, and a meta batch-size of 4 tasks. The model is trained using 5 gradient steps with step size $\alpha = 0.01$. The meta step size for the outer loop $\beta = 0.001$. The model is trained for 60000 iterations.

On the Omniglot dataset, the self-attention meta-learner model is trained using 5-way classification, 1 shot training, and a meta batch-size of 32 tasks. The model is trained using 5 inner gradient steps with step size $\alpha = 0.4$. The meta step size for the outer loop $\beta = 0.001$. The model is trained for 1000 iterations.

We adapt the Pytorch code for the MAML algorithm from [Lon18] to incorporate the attention module in the network architecture and to work for multilayer perceptron networks.

A.2 Gradual Learning Behavior of SAM

We analyze the performance of our proposed method in CIL gradually. We perform this analysis by computing the average accuracy on all past tasks after training SAM on each task. We also analyze the performance of each task as a function of encountered tasks. With this experiment, we would like to understand how the performance of the proposed model degrades over time when new tasks are encountered in the training process. We compare the gradual performance of SAM to the Scratch(TA) baseline. Figure A.1 and A.2 show the gradual learning behavior on the Split CIFAR-10/100 and Split MNIST benchmarks, respectively. This experiment allows us to dive more into the learning process of SAM. As expected, the performance degrades when new tasks are encountered. It is interesting to see that SAM performs on par with the Scratch models, while in the Scratch baseline, a new model is trained for each task from scratch. The degradation in performance for Scratch(TA) comes from using the decision module to identify the final output in the CIL. Instead of evaluating the CL methods using the final average accuracy over all classes, this experiment would be useful for future work to determine the factors that lead to performance degradation. For instance, as shown in the figures, some tasks cause significant degradation in performance. Identifying the correct task to which the test input image belongs would help increase the model performance. This will open new research ideas for CIL.

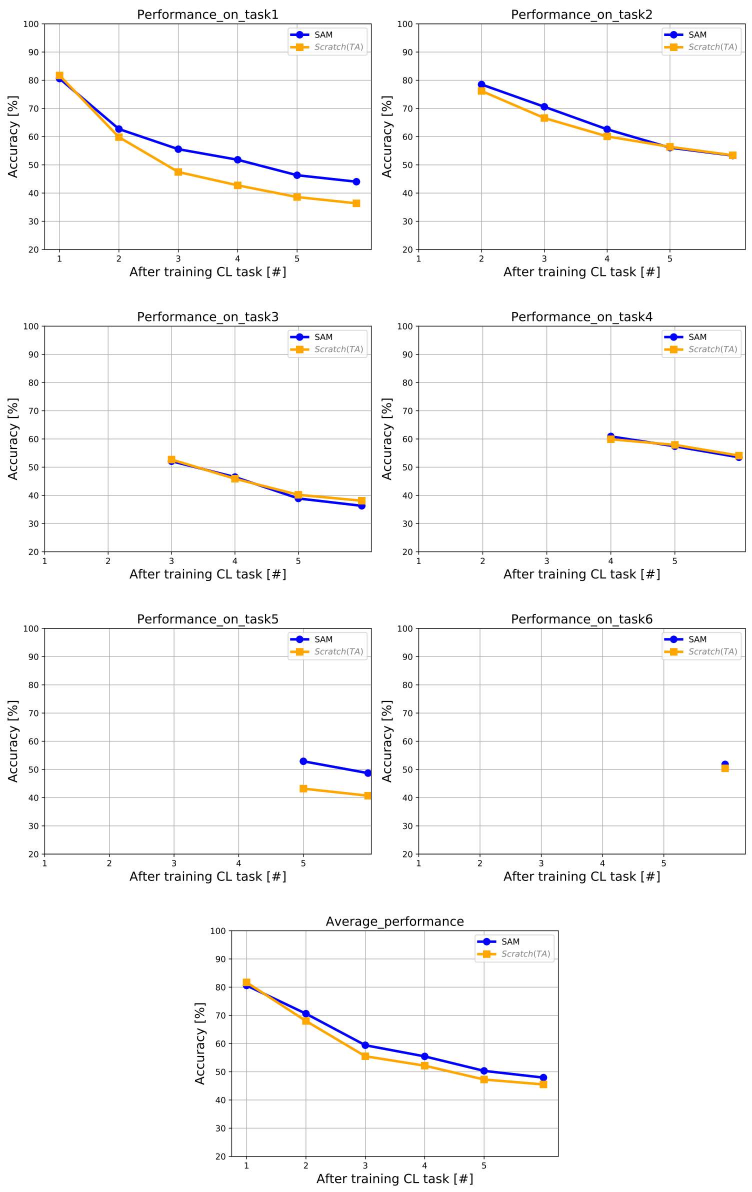


Figure A.1: The first three rows show the accuracy for each task in the Split CIFAR-10/100 benchmark as a function of the number of trained tasks so far. Each panel shows the accuracy of each task starting from the point where the model faces that task and after facing each consecutive task. The last row shows the average accuracy over all the tasks learned so far.

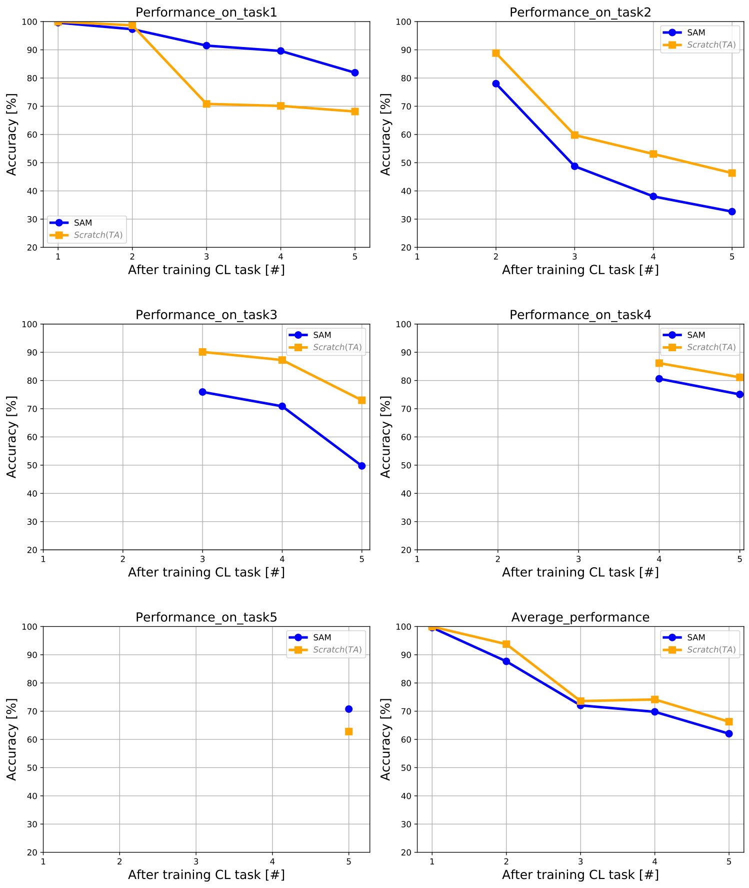


Figure A.2: The first five panels show the accuracy for each task in the Split MNIST benchmark as a function of the number of trained tasks so far. Each panel shows the accuracy of each task starting from the point where the model faces that task and after facing each consecutive task. The last panel shows the average accuracy over all the tasks learned so far.

Appendix B

Additional Experimental Details and Analyses on Chapter 4

B.1 Experimental Details

For a direct comparison with the TD3 algorithm, we follow the same setting as in [FHM18]. We use multi-layer perceptrons for the actor and critics networks with two hidden layers of 256 neurons and a ReLU activation function. A Tanh activation is applied to the output layer of the actor network. Sparse connections are allocated in the first two layers for all networks, while the output layer is dense. We use λ^2 of 64 for all environments. In contrast, λ^1 varies across environments based on each environment's state and action dimensions. We use λ^1 of 7 for HalfCheetah-v3, Hopper-v3, and Walker2d-v3. For Ant-v3 and Humanoid-v3, we use λ^1 of 40 and 61, respectively. The same sparsity levels are used for Static-TD3.

We adapt the sparse connections every e time steps, with $e = 1000$. A fraction of the sparse connections η is adapted with $\eta = 0.05$. The networks are trained using Adam optimizer with a learning rate of 0.001 and a weight decay of 0.0002. The networks are trained with mini-batches (N) of 100, sampled uniformly from a replay buffer containing the entire history of the agent.

Following the TD3 algorithm [FHM18], we add noise of $\epsilon \sim \mathcal{N}(0, 0.2)$ to the actions chosen by the target actor network and clipped to $(-0.5, 0.5)$. The actor and target networks are updated every 2 steps ($d = 2$). The τ used for updating the target networks equals 0.005. A purely exploratory policy is used for the first

25000 time steps, then an off-policy exploration strategy is used with Gaussian noise of $\mathcal{N}(0, 0.1)$ added to each action.

The hyperparameters for the dynamic sparse training $(\lambda^1, \lambda^2, \eta, e)$ are selected using random search. Each environment is run for 1 million time steps with evaluations every 5000 time steps, where each evaluation reports the average return over 10 episodes with no exploration noise. LCA is calculated using the average return computed every 5000 time steps. Our results are reported over 5 seeds.

All models are implemented with PyTorch and trained on Nvidia GPUs. We use the official code from the authors of TD3 [FHM18], which has an MIT license, to reproduce the results of TD3 with the above settings.

For SAC, we use the Pytorch implementation from [Tan18]. We follow the settings from the original paper [HZAL18] with the same architecture used for TD3. The networks are trained using Adam optimizer with a learning rate of 0.0003 and mini-batches of 256. We use τ of 0.005 and a target update interval of 1. The models are trained for 1M steps.

B.2 Evaluation Metrics

In this appendix, we explain the details of the metrics used to assess the performance of our proposed method.

Return (R). The average return is the standard metric used in the RL research to measure the *performance* of an agent. The return is the sum of rewards (r) obtained in one episode of T steps. R is calculated as follows:

$$R = \sum_{t=1}^{T} r_t. \tag{B.1}$$

Learning curve area (LCA). This metric estimates the learning speed of a model. LCA measures the area under the training curve of a method. Intuitively, the higher learning curve, the faster the learner is. We adapt this metric from [CRRE18] to fit the reinforcement learning paradigm. LCA is calculated as follows:

$$LCA = \frac{1}{\Delta} \int_0^{\Delta} R(t) dt = \frac{1}{\Delta} \sum_{t=0}^{\Delta} R(t), \tag{B.2}$$

where Δ is the number of training steps and R is the average return.

Network size (#params). This metric estimates the *memory cost* consumed by an agent. The network size is estimated by the summation of the number of connections allocated in its layers as follows:

$$\#params = \sum_{l=1}^{L} \left\| \boldsymbol{W}^{l} \right\|_0, \tag{B.3}$$

where $\boldsymbol{W}^l$ is the actual weights used in layer l, $\|.\|_0$ is the standard L_0 norm, and L is the number of layers in the model. For sparse neural networks, $\|\boldsymbol{W}^l\|_0$ is controlled by its defined sparsity level for the model.

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 [EGM+20] to calculate the FLOPs. The FLOPs are calculated with the total number of multiplications and additions layer by layer in the network.

FLOPs is the typically used metric in the literature to compare a DST method against its dense counterpart. The motivation is twofold. First, it gives an unbiased estimate of the actual required number of operations since the running time would differ from one implementation to another. Second, more importantly, existing dynamic sparse training methods in the literature are currently prototyped using masks over dense weights to simulate sparsity [HABN+21]. This is because most deep learning specialized hardware is optimized for dense matrix operations. Therefore, the running time using these prototypes would not reflect the actual gain in memory and speed using a truly sparse network. Therefore, the FLOPs and network parameters are the current commonly used metrics to estimate the computation and memory costs, respectively, for sparse neural networks [HABN+21].

B.3 Hardware and Software Support

As a joint community effort, research on sparsity is going in three parallel directions: First, hardware that supports sparsity. NVIDIA released NVIDIA A100, which supports a 50% fixed sparsity level [ZMZ+20]. Second, software libraries that support truly sparse implementations. Efforts have been started to be devoted to supervised learning [LMM+20]. Third, algorithmic methods, our focus, that aim to provide approaches that achieve the same performance of dense models using sparse networks [HABN+21]. With the parallel efforts in the three directions, we would be able to actually provide faster, memory-efficient, and energy-efficient deep neural networks. This is further discussed in [Hoo21, MMP+21].

B.4 Learning Behavior Analysis

To analyze the effect of the absence of true labels in the behavior of dense and dynamic sparse agents, we perform experiments in which an agent starts learning from samples drawn from a learned policy (Section 5.2). We test two learned policies with different performances to study how the quality of the initial samples affects the learning behavior. To this end, we train two dense policies using TD3 for 5×10^5 and 7×10^5 steps on Half-Cheetah-v3. Similarly, we train two sparse policies for the same steps using DS-TD3. Instead of using a purely exploratory policy, we draw samples from the learned policies to fill the initial buffers for dense and dynamic sparse agents that learn from scratch.

As discussed in Section 5.2, the performance of dense DRL agents is more affected by the initial samples. The better the samples are, the higher performance is. On the other hand, dynamic sparse agents adapt quickly to the improving samples over time and are less affected by the quality of the initial samples.

B.5 DS-SAC

In this appendix, we demonstrate that our proposed dynamic sparse training approach can be integrated with other state-of-the-art DRL methods. We use the soft actor-critic (SAC) method [HZH+18] and name our improved version of it as Dynamic Sparse SAC or DS-SAC.

SAC is an off-policy algorithm that optimizes a stochastic policy. A key feature of this method is entropy regularization. The policy is trained to maximize the trade-off between expected return and entropy (a measure of randomness). Thus, the agent addresses the exploration-exploitation trade-off, which results in policies that explore better. Algorithm 10 shows our proposed DS-SAC. We integrated the four components of our approach (sparse topology initialization, adaptation schedule, topological adaptation, and maintain sparsity levels) into the original algorithm.

Algorithm 10 DS-SAC

```
1: Require: $\lambda^l$, $\eta$, $e$
2: Create $\boldsymbol{M_\phi}$, $\boldsymbol{M_{\theta_1}}$, and $\boldsymbol{M_{\theta_2}}$ with Erdős–Rényi random graph with sparsity level $\lambda^l$
3: $\theta_1 \leftarrow \theta_1 \odot \boldsymbol{M_{\theta_1}}$, $\theta_2 \leftarrow \theta_2 \odot \boldsymbol{M_{\theta_2}}$, $\phi \leftarrow \phi \odot \boldsymbol{M_\phi}$
4: Initialize target networks $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$
5: $\mathcal{D} \leftarrow \emptyset$ // Initialize an empty replay pool
6: for each iteration do
7:     for each environment step do
8:         $a_t \sim \pi_\phi(a_t|s_t)$ // Sample action from the policy
9:         $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ // Sample transition from the environment
10:        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$ // Store the transition in the replay pool
11:    end for
12:    for each gradient step do
13:        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ // Update Q-functions
14:        if $t$ mod $e$ then
15:            $\theta_i \leftarrow$ TopologicalAdaptation($\theta_i, \boldsymbol{M_{\theta_i}}, \eta$) (Algo. 7)
16:        end if
17:        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ // Update policy weights
18:        if $t$ mod $e$ then
19:            $\phi \leftarrow$ TopologicalAdaptation($\phi, M_\phi, \eta$) (Algo. 7)
20:        end if
21:        $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ // Update target network
22:        $\bar{\theta}_i \leftarrow$ MaintainSparsity($\bar{\theta}_i, \|\theta_i\|_0$) (Algo. 8)
23:    end for
24: end for
```

Table B.1: The value used for λ^1 and λ^2 in each environment for the DS-SAC algorithm.

Environment	λ^1	λ^2
HalfCheetah-v3	12	80
Walker2d-v3	12	80
Hopper-v3	7	20
Ant-v3	30	64
Humanoid-v3	61	64

Tasks. We compare our proposed DS-SAC with SAC. We perform our experiments on five MuJoCo control tasks. Namely, we test the following environments: HalfCheetah-v3, Hopper-v3, Walker2d-v3, Ant-v3, and Humanoid-v3.

Experimental settings. We follow the setting from the SAC method [HZAL18]. All networks are multilayer perceptrons with two hidden layers of 256 neurons and a ReLU activation function. The networks are trained with Adam optimizer and a learning rate of 0.0003. We use mini-batches of 256. Each environment is run for 1 million steps. As DRL algorithms and their variants behave differently in various settings/environments, to cover a wider range of possible scenarios, we study here the case of hard target update where $\tau = 1$ [HZAL18]. Following the original paper, we use a target update interval of 1000 and a temperature α of 0.2. Table B.1 shows the value used for λ^1 and λ^2 to determine the sparsity levels for DS-SAC. We use e of 1000 and η of 0.1. The hyperparameters are selected using a random search. Our results are reported over 5 seeds.

Metrics. We use the same metrics discussed in Section 5.4 to assess the performance of our proposed method.

Results. Figure B.1 shows the learning behavior of DS-SAC and SAC. Consistent with our previous observations, DS-SAC learns faster, especially at the beginning of the training. The LCA of DS-SAC is higher than SAC for all environments, as shown in Table B.2. DS-SAC outperforms the final performance of SAC for all environments except one where it achieves a very close performance to it, as illustrated in Table B.3. Please note that the results of SAC are slightly different from the ones obtained in Section 4.4.4 as we study here the hard target update case of SAC [HZH$^+$18].

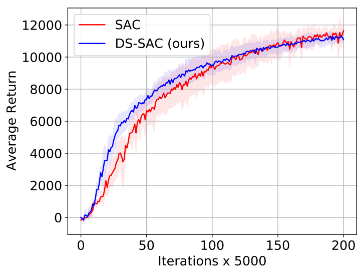
These experiments reveal that we can improve a DRL agent's learning speed and performance while reducing its required memory and computation costs for training.

Table B.2: Learning curve area (LCA) (× 5000) of SAC and DS-SAC.

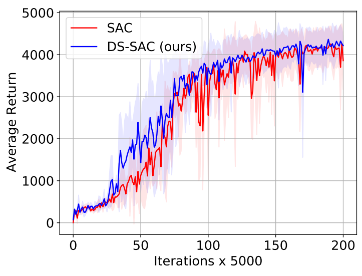
Environment	SAC	DS-SAC (ours)
HalfCheetah-v3	1.6229	**1.7081**
Walker2d-v3	0.5368	**0.5906**
Hopper-v3	0.4441	**0.4875**
Ant-v3	0.7504	**0.8229**
Humanoid-v3	0.3776	**0.6777**

Table B.3: Average return over the last 10 evaluations of 1 million time steps using SAC and DS-SAC.

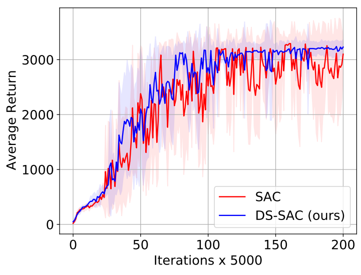
Environment	SAC	DS-SAC (ours)
HalfCheetah-v3	**11645.12** ± **425.585**	11084.39 ± 445.15
Walker2d-v3	3858.20 ± 689.913	**4216.77** ± **236.23**
Hopper-v3	3100.39 ± 374.45	**3229.39** ± **135.82**
Ant-v3	5899.30 ± 197.15	**5943.54** ± **169.95**
Humanoid-v3	5425.56 ± 196.33	**5584.64** ± **109.40**



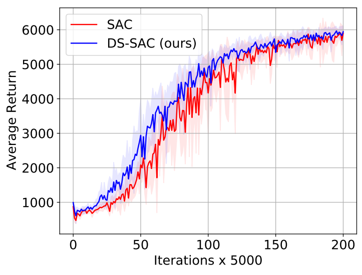
(a) HalfCheetah-v3.



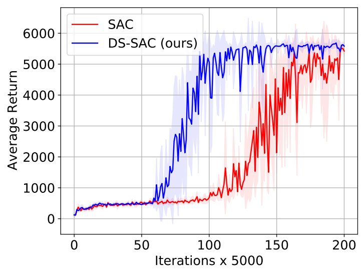
(b) Walker2d-v3.



(c) Hopper-v3.



(d) Ant-v3.



(e) Humanoid-v3.

Figure B.1: Learning curves of SAC and DS-SAC on different continuous control tasks. The shaded region represents the standard deviation of the average evaluation over 5 runs.

Appendix C

Additional Experimental Details and Analyses on Chapter 5

C.1 Experimental Details

Discrete control tasks. We evaluate DQN [MKS+15] on 17 games from the Arcade Learning Environment [BNVB13]: Asterix, Demon Attack, Seaquest, Wizard of Wor, Bream Reader, Road Runner, James Bond, Qbert, Breakout, Enduro, Space Invaders, Pong, Zaxxon, Yars' Revenge, Ms. Pacman, Double Dunk, Ice Hockey. This set is used by previous works [KAGL21, KAM+21] to study the *implicit under-parameterization* phenomenon in offline RL. For hyper-parameter tuning, we use five games (Asterix, Demon Attack, Seaquest, Breakout, Beam Rider). We evaluate DrQ(ϵ) on the 26 games of Atari 100K [KBM+19]. We used the best hyper-parameters found for DQN in training DrQ(ϵ).

Continuous control tasks. We evaluate SAC [HZAL18] on four environments from MuJoCo suite [TET12]: HalfCheetah-v2, Hopper-v2, Walker2d-v2, Ant-v2.

Code. For discrete control tasks, we build on the implementation of DQN and DrQ provided in Dopamine [CMG+18], including the architectures used for agents. The hyper-parameters are provided in Tables C.1, C.2, and C.3. For continuous control, we build on the SAC implementation in TF-Agents [GKR+18]

Table C.1: Common Hyper-parameters for DQN and DrQ(ϵ).

Parameter	Value
Optimizer	Adam [KB15]
Optimizer: ϵ	1.5×10^{-4}
Training ϵ	0.01
Evaluation ϵ	0.001
Discount factor	0.99
Replay buffer size	10^6
Minibatch size	32
Q network: channels	32, 64, 64
Q-network: filter size	8×8, 4×4, 3×3
Q-network: stride	4, 2, 1
Q-network: hidden units	512
Recycling period	1000
τ-Dormant	0.025 for default setting, 0.1 otherwise
Minibatch size for estimating neurons score	64

and the codebase of [GEEC22]. The hyper-parameters are provided in Table C.4.

Evaluation. We follow the recommendation from [ASC+21] to report reliable aggregated results across games using the interquartile mean (IQM). IQM is the calculated mean after discarding the bottom and top 25% of normalized scores aggregated from multiple runs and games.

Baselines. For weight decay, we searched over the grid $[10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}]$. The best found value is 10^{-5}. For reset [NSD+22], we consider re-initializing the last layer for Atari games (same as the original paper). They use a reset period of 2×10^4 in for Atari 100k [KBM+19], which corresponds to having 5 restarts in a training run. Since we run longer experiments, we searched over the grid $[5 \times 10^4, 1 \times 10^5, 2.5 \times 10^5, 5 \times 10^5]$ gradient steps for the reset period which corresponds to having 50, 25, 10 and 5 restarts per training (10M frames, replay ratio 1). The best found period is 1×10^5. For SAC, we reset agent's networks entirely every 2×10^5 environment steps, following the original paper.

Table C.2: Hyper-parameters for DQN.

Parameter	Value
Optimizer: Learning rate	6.25×10^{-5}
Initial collect steps	20000
n-step	1
Training iterations	Default setting: 40, otherwise: 10
Training environment steps per iteration	250K
(Updates per environment step, Target network update period)	(0.25, 8000)
	(0.5, 4000)
	(1, 2000)
	(2, 1000)

Table C.3: Hyper-parameters for DrQ(ϵ).

Parameter	Value
Optimizer: Learning rate	1×10^{-4}
Initial collect steps	1600
n-step	10
Training iterations	40
Training environment steps per iteration	10K
Updates per environment step	1, 2, 4, 8

Replay ratio. For DQN, we evaluate replay ratio values: {0.25 (default), 0.5, 1, 2}. Following [VHHA19], we scale the target update period based on the value of the replay ratio as shown in Table C.2. For DrQ(ϵ), we evaluate the values: {1 (default), 2, 4, 8}.

ReDo **hyper-parameters.** We did the hyper-parameter search for DQN trained with $RR = 1$ using the nature CNN architecture. We searched over the grids [1000, 10000, 100000] and [0, 0.01, 0.1] for the recycling period and τ-dormant, respectively. We apply the best values found to all other settings of DQN, including the ResNet architecture and DrQ(ϵ), as reported in Table C.1.

Dormant neurons in supervised learning. Here we provide the experimental details of the supervised learning analysis illustrated in Section 5.2. We train

Table C.4: Hyper-parameters for SAC.

Parameter	Value
Initial collect steps	10000
Discount factor	0.99
Training environment steps	10^6
Replay buffer size	10^6
Updates per environment step (Replay Ratio)	1, 2, 4, 8
Target network update period	1
target smoothing coefficient τ	0.005
Optimizer	Adam [KB15]
Optimizer: Learning rate	3×10^{-4}
Minibatch size	256
Actor/Critic: Hidden layers	2
Actor/Critic: Hidden units	256
Recycling period	200000
τ-Dormant	0
Minibatch size for estimating neurons score	256

a convolutional neural network on CIFAR-10 [KH+09] using stochastic gradient descent and cross-entropy loss. We select 10000 samples from the dataset to reduce the computational cost. We analyze the dormant neurons in two supervised learning settings: (1) training a network with *fixed targets*, the standard single-task supervised learning, where we train a network using the inputs and labels of CIFAR-10 for 100 epochs, and (2) training a network with *non-stationary targets*, where we shuffle the labels every 20 epochs to generate new targets. Table C.5 provides the details of the network architecture and training hyper-parameters.

Learning ability of networks with dormant neurons. Here we present the details of the regression experiment provided in Section 5.2. Inputs and targets for regression come from a DQN agent trained on DemonAttack for 40M frames with the default hyper-parameters. The pre-trained network was trained for 40M frames using a replay ratio of 1.

Table C.5: Hyperparameters for CIFAR-10.

Parameter	Value
Optimizer	SGD
Minibatch size	256
Learning rate	0.01
Momentum	0.9
Architecture:	
Layer	(channels, kernel size, stride)
Convolution	(32, 3, 1)
Convolution	(64, 3, 1)
MaxPool	(-, 2, 2)
Convolution	(64, 3, 1)
MaxPool	(-, 2, 2)
Dense	(128, -, -)

C.2 The Dormant Neuron Phenomenon in Different Domains

In this appendix, we demonstrate the dormant neuron phenomenon on DrQ(ϵ) [YKF21] on the Atari 100K benchmark [KBM+19] as well as on additional games from the Arcade Learning Environment on DQN. Additionally, we show

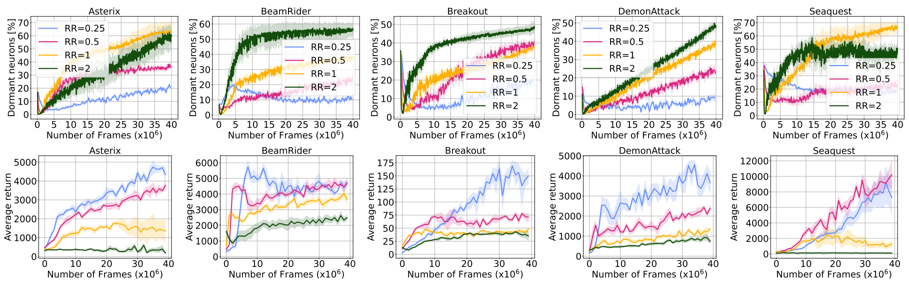


Figure C.1: Effect of replay ratio in the number of dormant neurons for DQN on Atari environments (experiments presented in Figure 5.7).

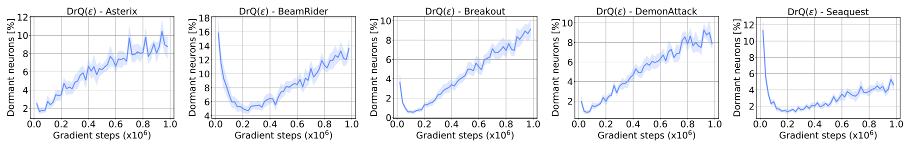


Figure C.2: The dormant neuron phenomenon becomes apparent as the number of training steps increases during the training of DrQ(ϵ) with the default replay ratio on Atrai 100K.

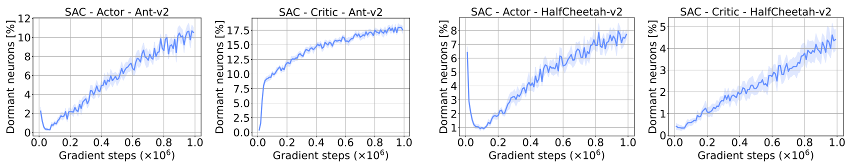


Figure C.3: The number of dormant neurons increases over time during the training of SAC on MuJoCo environments.

the phenomenon on continuous control tasks and analyze the role of dormant neurons in performance. We consider SAC [HZAL18] trained on MuJoCo environments [TET12]. Same as our analyses in Section 5.2, we consider $\tau = 0$ to illustrate the phenomenon.

Figure C.1 shows that across games, the number of dormant neurons consistently increases with higher values for the replay ratio on DQN. The increase in dormant neurons correlates with the performance drop observed in this regime. We then investigate the phenomenon on a modern valued-based algorithm DrQ(ϵ). As we see in Figure C.2, the phenomenon emerges as the number of training steps increases.

Figure C.3 shows that the phenomenon is also present in continuous control tasks. An agent exhibits an increasing number of dormant neurons in the actor and critic networks during the training of SAC on MuJoco environments. To analyze the effect of these neurons on performance, we prune dormant neurons every 200K steps. Figure C.4 shows that the performance is not affected by pruning these neurons; indicating their little contribution to the learning process. Next, we investigate the effect of *ReDo* and the studied baselines (Reset [NSD$^+$22] and weight decay (WD)) in this domain. Figure C.5 shows that *ReDo* maintains the performance of the agents while other methods cause a per-

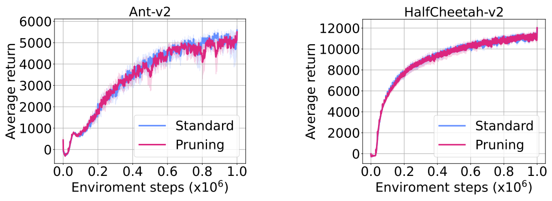


Figure C.4: Pruning dormant neurons during the training of SAC on MuJoCo environments does not affect the performance.

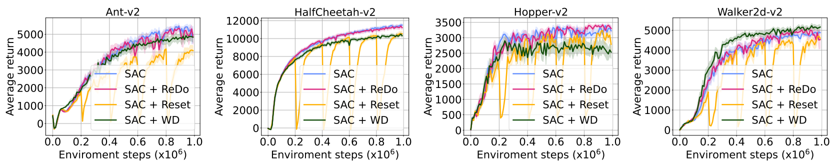


Figure C.5: Comparison of the performance of SAC agents with *ReDo* and two different regularization methods.

formance drop in most cases. We hypothesize that *ReDo* does not provide gains here as the state space is considerably low and the typically used network is sufficiently over-parameterized.

To investigate this, we decrease the size of the actor and critic networks by halving or quartering the width of their layers. We perform these experiments on the complex environment Ant-v2 using 5 seeds. Table C.6 shows the final average return in each case. We observe that when the network size is smaller, there are some gains from recycling the dormant capacity. Further analyses of the relation between task complexity and network capacity would provide a more comprehensive understanding.

C.3 Recycling Dormant Neurons

Here we study different strategies for recycling dormant neurons and analyze the design choices of *ReDo*. We perform these analyses on DQN agents trained with $RR = 1$ and $\tau = 0.1$ on Atari games. Furthermore, we provide some additional insights into the effect of recycling the dormant capacity on improving

Table C.6: Performance of SAC on Ant-v2 using using half and a quarter of the width of the actor and critic networks.

Width	SAC	SAC+ReDo
0.25	2016.18 ± 102	**2114.52** ± 212
0.5	3964.04 ± 953	**4471.61** ± 648

the sample efficiency and the expressivity of the network.

C.3.1 Effect of Activation Function

In this section, we attempt to understand the effect of the activation function (ReLU) used in our experiments. The ReLU activation function consists of a linear part (positive domain) with unit gradients and a constant zero part (negative domain) with zero gradients. Once the distribution of pre-activations falls completely into the negative part, it would stay there since the weights of the neuron would get zero gradients. This could be the explanation for the increased number of dormant neurons in our neural networks. If this is the case, one might expect activations with non-zero gradients on the negative side, such as leaky ReLU, to have significantly fewer dormant neurons.

In Figure C.6, we compare networks with leaky ReLU to original networks with ReLU activation. As we can see, using leaky ReLU slightly decreases the number of dormant neurons but does not mitigate the issue. *ReDo* overcomes the performance drop that occurs during training in the two cases.

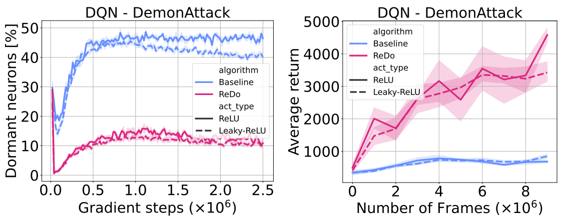


Figure C.6: Training performance and dormant neuron characteristics of networks using leaky ReLU with a negative slope of 0.01 (default value) compared to original networks with ReLU.

C.3.2 Recycling Strategies

Outgoing connections. We investigate the effect of using random weights to reinitialize the outgoing connections of dormant neurons. We compare this strategy against the reinitialization strategy of *ReDo* (*zero weights*). Figure C.7 shows the performance of DQN on five Atari games. The random initialization of the outgoing connections leads to a lower performance than the zero initialization. This is because the newly added random weights change the output of the network.

Incoming connections. Another possible strategy to reinitialize the incoming connections of dormant neurons is to scale their weights with the average norm of non-dormant neurons in the same layer. We observe that this strategy has a similar performance to the random weight initialization strategy, as shown in Figure C.8.

C.3.3 Effect of Batch Size

The score of a neuron is calculated based on a given batch $\mathcal{D}$ of data (Section 5.2). Here we study the effect of the batch size in determining the percent-

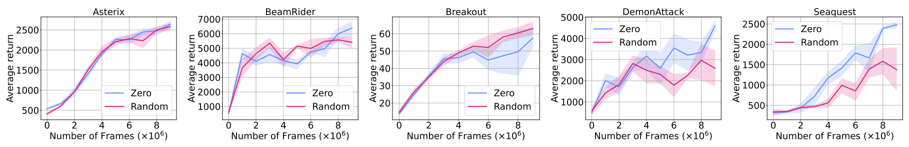


Figure C.7: Comparison of performance with different strategies of reinitializing the outgoing connections of dormant neurons.

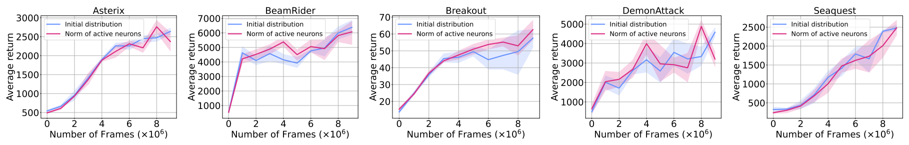


Figure C.8: Comparison of performance with different strategies of reinitializing the incoming connections of dormant neurons.

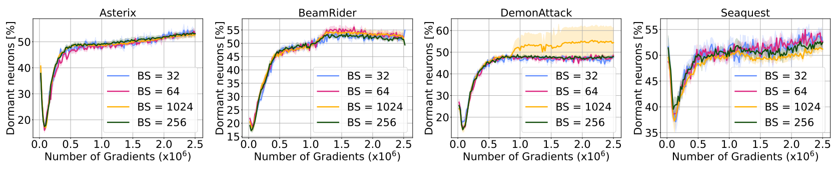


Figure C.9: Effect of the batch size used to detect dormant neurons.

age of dormant neurons. We study four different values: {32, 64, 256, 1024}. Figure C.9 shows that the identified percentage of dormant neurons is approximately the same using different batch sizes.

C.3.4 Comparison with Continual Backprop

Similar to the experiments in Figure 5.15, we use a fixed recycling schedule to compare the activation-based metric used by *ReDo* and the utility metric proposed by Continual Backprop [DMS21]. Results shown in Figure C.10 show that both metrics achieve similar results. Note that the original Continual Backprop algorithm calculates neuron scores at every iteration and uses a running average to obtain a better estimate of the neuron saliency. This approach requires additional storage and computing compared to the fixed schedule used by our algorithm. Given the high dormancy threshold preferred by our method (i.e., more neurons are recycled), we expect better saliency estimates to have a limited impact on the results presented here. However, a more thorough analysis is needed to make general conclusions.

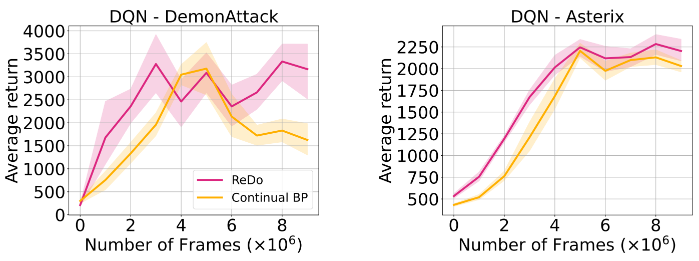


Figure C.10: Comparison of different strategies for selecting the recycled neurons.

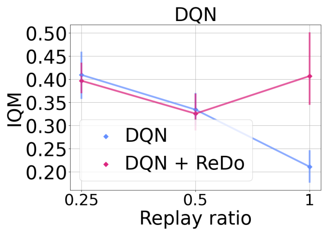


Figure C.11: Comparison of agents with varying replay ratios, while keeping the number of gradient updates constant.

C.3.5 Effect of Recycling the Dormant Capacity

Improving Sample Efficiency. To examine the impact of recycling dormant neurons on enhancing the agents' sample efficiency, an alternative approach is to compare agents with varying replay ratios, while keeping the number of gradient updates constant during training. Consequently, agents with a higher replay ratio will perform fewer interactions with the environment.

We perform this analysis on DQN and the 17 Atari games. Agents with a replay ratio of 0.25 run for 10M frames, a replay ratio of 0.5 run for 5M frames, and a replay ratio of 1 run for 2.5M frames. The number of gradient steps is fixed across all agents. Figure C.11 shows the aggregated results across all games. Interestingly the performance of *ReDo* with $RR = 1$ is very close to $RR = 0.25$, while significantly reducing the number of environment steps by four. On the other hand, DQN with $RR = 1$ suffers from a performance drop.

Improving Networks' expressivity. Our results in Chapter 5 show that recycling dormant neurons improves the learning ability of agents measured by their performance. Here, we do some preliminary experiments to measure the effect of neuron recycling on the learned representations. Following [KAGL21], we calculate the effective rank, a measure of expressivity, of the feature learned in the penultimate layer of networks trained with and without *ReDo*. We perform this analysis on agents trained for 10M frames on DemonAttack using DQN. The results are averaged over 5 seeds. The results in Table C.7 suggest recycling dormant neurons improves the expressivity, shown by the increased rank of the learned representations. Further investigation of expressivity metrics and analyses on other domains would be an exciting future direction.

Table C.7: Effective rank [KAGL21] of the learned representations of agents trained on DemonAttack.

Agent	Effective rank
DQN	449.2 ± 5.77
DQN + *ReDo*	**470.8** ± 1.16

C.4 Performance Per Game

Here we share the training curves of DQN using the CNN architecture for each game in the high replay ratio regime ($RR = 1$) (Figure C.12) and the default setting ($RR = 0.25$) (Figure C.13). Similarly, Figure C.14 and C.15 show the training curves of DrQ(ϵ) for each game in the high replay ratio regime ($RR = 4$) and the default setting ($RR = 1$), respectively.

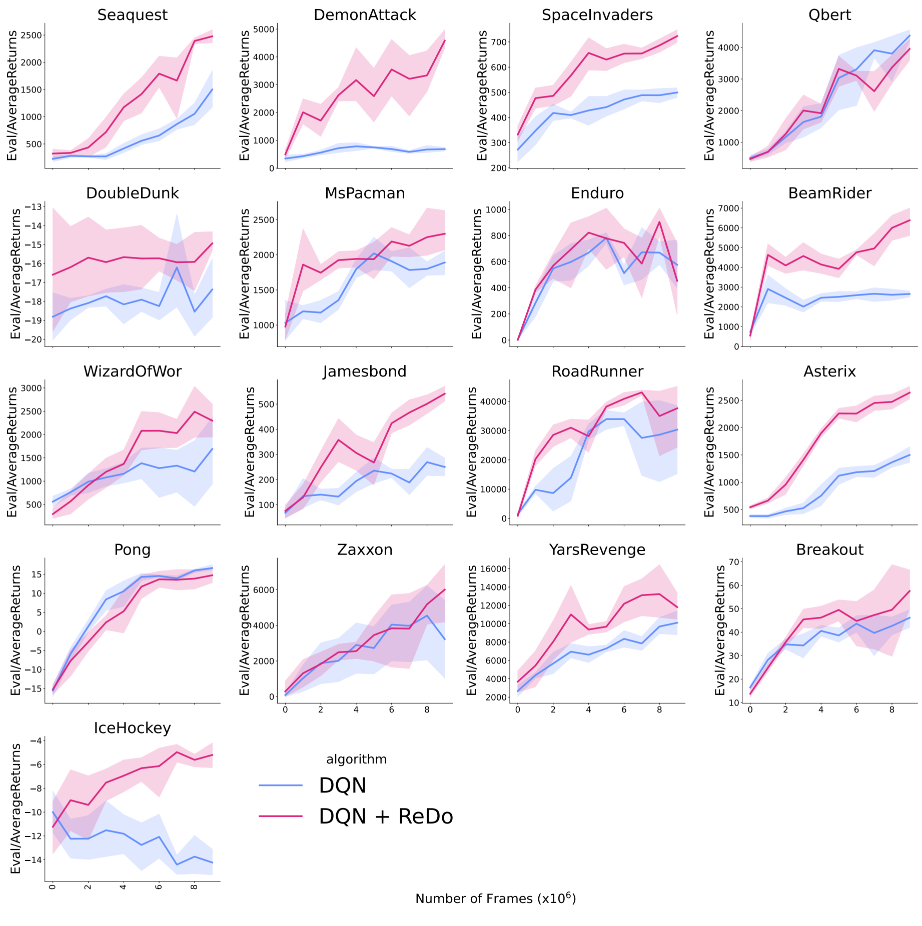


Figure C.12: Training curves for DQN with the nature CNN architecture ($RR = 1$).

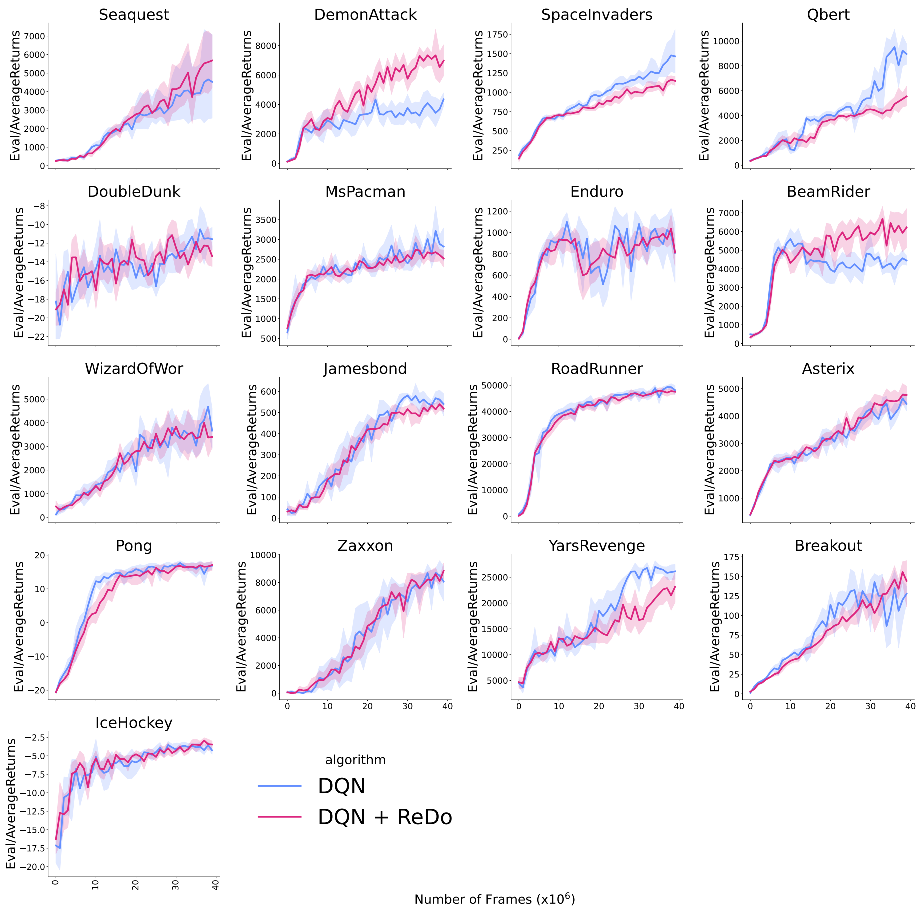


Figure C.13: Training curves for DQN with the nature CNN architecture ($RR = 0.25$).

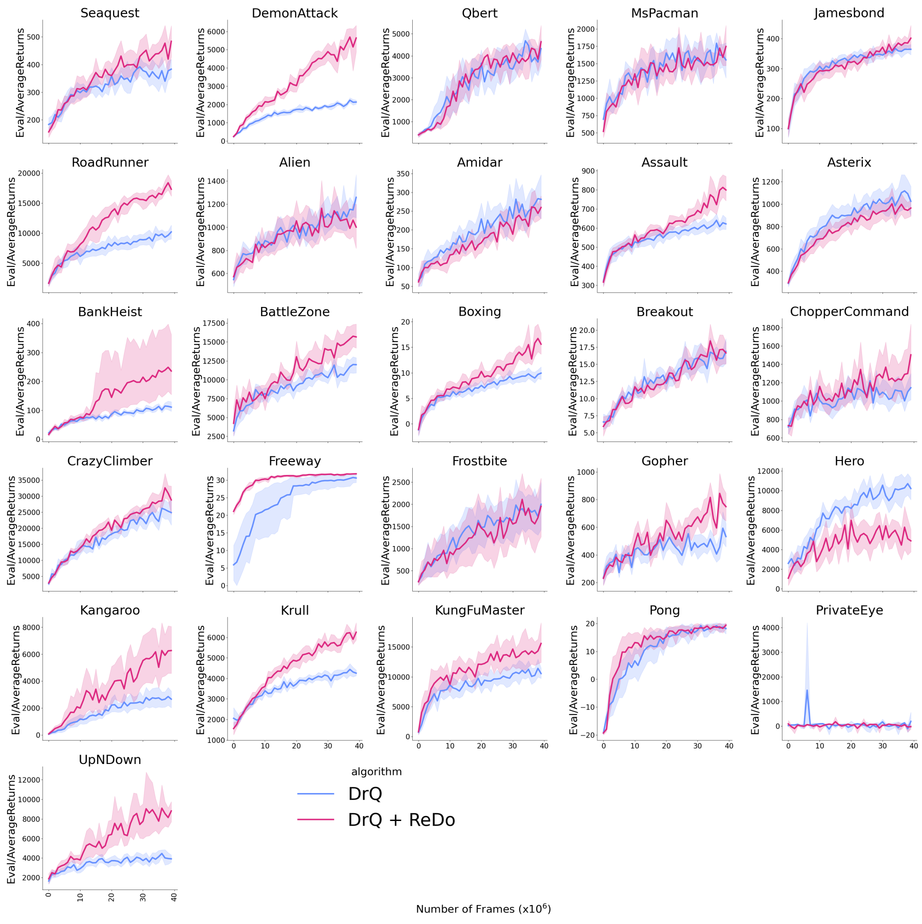


Figure C.14: Training curves for DrQ(ϵ) with the nature CNN architecture ($RR = 4$).

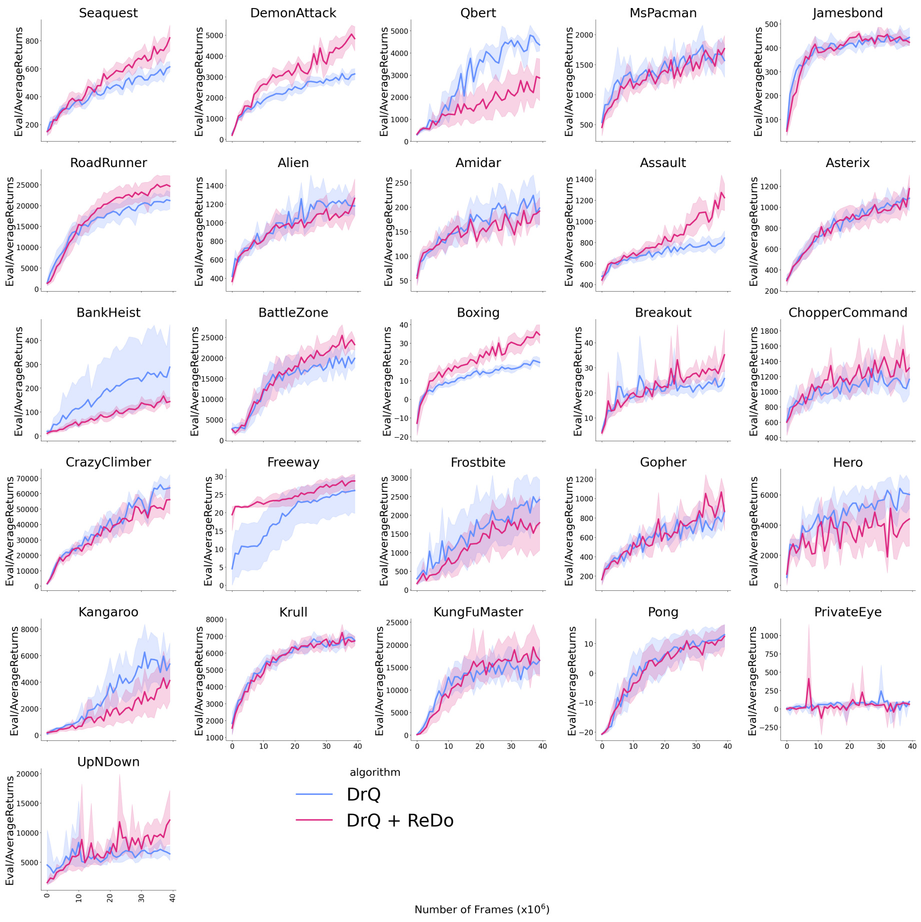


Figure C.15: Training curves for DrQ(ϵ) with the nature CNN architecture ($RR = 1$).

Bibliography

[AA20] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in Neural Information Processing Systems*, 33:3884–3894, 2020. (Cited on page 91.)

[ABB$^+$19] Mike Ashby, Christiaan Baaij, Peter Baldwin, Martijn Bastiaan, Oliver Bunting, Aiken Cairncross, Christopher Chalmers, Liz Corrigan, Sam Davis, Nathan van Doorn, et al. Exploiting unstructured sparsity on next-generation datacenter hardware, 2019. (Cited on page 98.)

[ABE$^+$18] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018. (Cited on pages 17 and 40.)

[ACC21] João Guilherme Madeira Araújo, Johan Samir Obando Ceron, and Pablo Samuel Castro. Lifting the veil on hyper-parameters for value-based deep reinforcement learning. In *Deep RL Workshop NeurIPS*, 2021. (Cited on page 76.)

[AMK21] Ibrahim Alabdulmohsin, Hartmut Maennel, and Daniel Keysers. The impact of reinitialization on generalization in convolutional neural networks. *arXiv preprint arXiv:2109.00267*, 2021. (Cited on page 91.)

[AOPP21] Samin Yeasar Arnob, Riyasat Ohib, Sergey Plis, and Doina Precup. Single-shot pruning for offline reinforcement learning. *Offline Reinforcement Learning Workshop at Neural Information Processing Systems*, 2021. (Cited on page 91.)

[ASC$^+$21] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021. (Cited on pages 76, 85, 86, and 116.)

[ASN20] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020. (Cited on page 80.)

[AZM$^+$23] Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. Loss of plasticity in continual deep reinforcement learning. *arXiv preprint arXiv:2303.07507*, 2023. (Cited on page 5.)

[BCB15] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*, 2015. (Cited on page 41.)

[BCC$^+$20] Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020. (Cited on page 76.)

[BCD$^+$21] Tudor Berariu, Wojciech Czarnecki, Soham De, Jörg Bornschein, Samuel L. Smith, Razvan Pascanu, and Claudia Clopath. A study on the plasticity of neural networks. *CoRR*, abs/2106.00042, 2021. (Cited on page 91.)

[BCD22] Bruno Benedetti and Sebastien Couillard-Despres. Why would the brain need dormant neuronal precursors? *Frontiers in Neuroscience*, 16, 2022. (Cited on page 92.)

[BDK$^+$20] Bruno Benedetti, Dominik Dannehl, Richard König, Simona Coviello, Christina Kreutzer, Pia Zaunmair, Dominika Jakubecova, Thomas M Weiger, Ludwig Aigner, Juan Nacher, et al. Functional integration of neuronal precursors in the adult murine piriform cortex. *Cerebral cortex*, 30(3):1499–1515, 2020. (Cited on page 92.)

[BKH15] Zuo Bai, Liyanaarachchi Lekamalage Chamara Kasun, and Guang-Bin Huang. Generic object recognition with local receptive fields based extreme learning machine. 2015. (Cited on page 54.)

[BKML18] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. (Cited on page 17.)

[BMR$^+$20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. (Cited on page 2.)

[BNVB13] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. (Cited on pages 76, 79, 85, and 115.)

[BPP20] Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *International Conference on Machine Learning*, pages 767–777. PMLR, 2020. (Cited on page 76.)

[BRF13] Omri Barak, Mattia Rigotti, and Stefano Fusi. The sparseness of mixed selectivity neurons controls the generalization–discrimination trade-off. *Journal of Neuroscience*, 33(9):3844–3856, 2013. (Cited on page 54.)

[CL18] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018. (Cited on page 38.)

[CMG$^+$18] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. (Cited on pages 85 and 115.)

[CMP21] Selima Curci, Decebal Constantin Mocanu, and Mykola Pechenizkiyi. Truly sparse neural networks at scale. *arXiv preprint arXiv:2102.01732*, 2021. (Cited on page 98.)

[CPK$^+$17] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017. (Cited on pages 14 and 38.)

[CRO$^+$20] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Page-Caccia, Issam Hadj Laradji, Irina Rish, Alexandre Lacoste, David Vázquez, and Laurent Charlin. Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. In *Advances in Neural Information Processing Systems*, 2020. (Cited on page 41.)

[CRRE18] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations*, 2018. (Cited on pages 68 and 108.)

[CWZR20] Xinyue Chen, Che Wang, Zijian Zhou, and Keith W Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2020. (Cited on page 91.)

[CYES19] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019. (Cited on page 98.)

[CZX$^+$17] Long Chen, Hanwang Zhang, Jun Xiao, Liqiang Nie, Jian Shao, Wei Liu, and Tat-Seng Chua. Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5659–5667, 2017. (Cited on page 41.)

[DBK+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020. (Cited on pages 1 and 100.)

[DMS21] Shibhansh Dohare, A Rupam Mahmood, and Richard S Sutton. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021. (Cited on pages 5, 91, 92, and 124.)

[DRLFM18] Natalia Díaz-Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. 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. (Cited on pages 4, 38, and 52.)

[DSD+13] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, pages 2148–2156, 2013. (Cited on page 62.)

[DSP+19] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5138–5146, 2019. (Cited on page 41.)

[DYJ19] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019. (Cited on page 92.)

[DZ19] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019. (Cited on page 17.)

[EGM+20] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020. (Cited on pages 17, 62, and 109.)

[ER+60] Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960. (Cited on page 64.)

[ESM+18] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018. (Cited on pages 85 and 86.)

[EvMU+21] Utku Evci, Bart van Merrienboer, Thomas Unterthiner, Fabian Pedregosa, and Max Vladymyrov. Gradmax: Growing neural networks using gradient information. In *International Conference on Learning Representations*, 2021. (Cited on page 92.)

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017. (Cited on page 42.)

[FBB+17] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017. (Cited on page 18.)

[FC18] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018. (Cited on page 62.)

[FG19] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. In *Privacy in Machine Learning and Artificial Intelligence workshop, ICML*, jun 2019. (Cited on pages 14, 18, 19, and 25.)

[FHM18] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. (Cited on pages 61, 63, 68, 107, and 108.)

[FKSL19] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *International*

Conference on Machine Learning, pages 2021–2030. PMLR, 2019. (Cited on pages 76, 88, and 91.)

[FL18] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Representations*, 2018. (Cited on page 42.)

[FLT+19] Jun Fu, Jing Liu, Haijie Tian, Yong Li, Yongjun Bao, Zhiwei Fang, and Hanqing Lu. Dual attention network for scene segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3146–3154, 2019. (Cited on page 41.)

[FMR16] Stefano Fusi, Earl K Miller, and Mattia Rigotti. Why neurons mix: high dimensionality for higher cognition. *Current opinion in neurobiology*, 37:66–74, 2016. (Cited on page 54.)

[FRA+20] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pages 3061–3071. PMLR, 2020. (Cited on page 6.)

[Fre91] Robert M French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. In *Proceedings of the 13th annual cognitive science society conference*, volume 1, pages 173–178, 1991. (Cited on page 30.)

[FRKL19] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. (Cited on page 41.)

[FWH+21] Linxi Fan, Guanzhi Wang, De-An Huang, Zhiding Yu, Li Fei-Fei, Yuke Zhu, and Animashree Anandkumar. Secant: Self-expert cloning for zero-shot generalization of visual policies. In *International Conference on Machine Learning*, pages 3088–3099. PMLR, 2021. (Cited on page 91.)

[GD22] Mustafa B Gurbuz and Constantine Dovrolis. Nispa: Neuro-inspired stability-plasticity adaptation for continual learning in sparse networks. In *International Conference on Machine Learning*, pages 8157–8174. PMLR, 2022. (Cited on page 96.)

[GEEC22] Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In *International Conference on Machine Learning*, pages 7766–7792. PMLR, 2022. (Cited on pages 91, 93, 97, and 116.)

[GKC19] Siavash Golkar, Michael Kagan, and Kyunghyun Cho. Continual learning via neural pruning. *arXiv preprint arXiv:1903.04476*, 2019. (Cited on page 18.)

[GKR$^+$18] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. `https://github.com/tensorflow/agents`, 2018. [Online; accessed 25-June-2019]. (Cited on page 115.)

[GLO$^+$16] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016. (Cited on pages 14 and 38.)

[GMD23] Mustafa Burak Gurbuz, Jean Michael Moorman, and Constantine Dovrolis. Sharp: Sparsity and hidden activation replay for neuro-inspired continual learning. *arXiv preprint arXiv:2305.18563*, 2023. (Cited on page 96.)

[GSD$^+$23] Bram Grooten, Ghada Sokar, Shibhansh Dohare, Elena Mocanu, Matthew E Taylor, Mykola Pechenizkiy, and Decebal Constantin Mocanu. Automatic noise filtering with dynamic sparse training in deep reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 1932–1941, 2023. (Cited on page 10.)

[GSS$^+$22] Caglar Gulcehre, Srivatsan Srinivasan, Jakub Sygnowski, Georg Ostrovski, Mehrdad Farajtabar, Matthew Hoffman, Razvan Pascanu, and Arnaud Doucet. An empirical study of implicit regularization in deep offline rl. *Transactions on Machine Learning Research*, 2022. (Cited on page 91.)

[GXGP23] Tomer Galanti, Mengjia Xu, Liane Galanti, and Tomaso Poggio. Norm-based generalization bounds for compositionally sparse neural networks. *arXiv preprint arXiv:2301.12033*, 2023. (Cited on page 98.)

[HABN$^+$21] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(241):1–124, 2021. (Cited on pages 60, 73, 98, and 109.)

[HBKV15] Guang-Bin Huang, Zuo Bai, Liyanaarachchi Lekamalage Chamara Kasun, and Chi Man Vong. Local receptive fields based extreme learning machine. *IEEE Computational intelligence magazine*, 10(2):18–29, 2015. (Cited on page 54.)

[HH49] Donald Olding Hebb and DO Hebb. *The organization of behavior*, volume 65. Wiley New York, 1949. (Cited on page 25.)

[HIH$^+$21] Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning. In *International Conference on Learning Representations*, 2021. (Cited on page 91.)

[HLRK18] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. In *NeurIPS Continual learning Workshop*, 2018. (Cited on pages xxv, 4, 14, 18, and 28.)

[HMI09] Verena Heidrich-Meisner and Christian Igel. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, 64(4):152–168, 2009. Special Issue: Reinforcement Learning. (Cited on page 61.)

[HMVH$^+$18] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018. (Cited on pages 7 and 76.)

[HNA$^+$17] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang

Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017. (Cited on page 76.)

[Hoo21] Sara Hooker. The hardware lottery. *Communications of the ACM*, 64(12):58–65, 2021. (Cited on pages 98 and 110.)

[HRRP20] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020. (Cited on page 4.)

[HSRN$^+$19] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P Sadayappan. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, pages 300–314, 2019. (Cited on page 98.)

[HSS18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. (Cited on pages 41 and 44.)

[HSW21] Nicklas Hansen, Hao Su, and Xiaolong Wang. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3680–3693. Curran Associates, Inc., 2021. (Cited on page 91.)

[HVD15a] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015. (Cited on page 17.)

[HVD15b] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. (Cited on page 40.)

[HWCW19] Lun Huang, Wenmin Wang, Jie Chen, and Xiao-Yong Wei. Attention on attention for image captioning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4634–4643, 2019. (Cited on page 41.)

[HZAL18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. (Cited on pages 61, 63, 76, 79, 85, 90, 108, 112, 115, and 120.)

[HZH$^+$18] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018. (Cited on pages 68, 110, and 112.)

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. (Cited on pages 14 and 95.)

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. (Cited on pages 1 and 2.)

[HZS04] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)*, volume 2, pages 985–990. IEEE, 2004. (Cited on page 53.)

[IFL$^+$20] Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2020. (Cited on pages 6, 76, and 91.)

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. (Cited on pages 26 and 46.)

[JFZL19] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019. (Cited on page 91.)

[JPR$^+$20] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33:20744–20754, 2020. (Cited on page 62.)

[JW19] Khurram Javed and Martha White. Meta-learning representations for continual learning. In *Advances in Neural Information Processing Systems*, pages 1820–1830, 2019. (Cited on page 41.)

[JYP$^+$17] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017. (Cited on page 60.)

[JZR$^+$19] LIU Junjie, XU Zhe, SHI Runbin, Ray CC Cheung, and Hayden KH So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2019. (Cited on page 17.)

[KAGL21] Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021. (Cited on pages xxvi, 7, 76, 82, 83, 85, 91, 115, 125, and 126.)

[KAM$^+$21] Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. Dr3: Value-based deep reinforcement learning requires explicit regularization. In *International Conference on Learning Representations*, 2021. (Cited on pages 6, 85, 91, and 115.)

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. (Cited on pages 116 and 118.)

[KBM$^+$19] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2019. (Cited on pages 85, 115, 116, and 119.)

[KH$^+$09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. (Cited on pages 25, 46, 80, and 118.)

[KH23] Tommi Kärkkäinen and Jan Hänninen. Additive autoencoder for dimension estimation. *Neurocomputing*, 551:126520, 2023. (Cited on page 93.)

[KMA$^+$18] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018. (Cited on pages 14 and 18.)

[KMH$^+$20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. (Cited on page 76.)

[KMM$^+$22] Haeyong Kang, Rusty John Lloyd Mina, Sultan Rizky Hikmawan Madjid, Jaehong Yoon, Mark Hasegawa-Johnson, Sung Ju Hwang, and Chang D Yoo. Forget-free continual learning with winning subnetworks. In *International Conference on Machine Learning*, pages 10734–10750. PMLR, 2022. (Cited on page 96.)

[KNH$^+$22] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022. (Cited on page 1.)

[KPR$^+$17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017. (Cited on pages 14, 17, 19, 40, and 47.)

[KT19] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019. (Cited on pages 14 and 38.)

[KZGR21] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021. (Cited on page 91.)

[LAM$^+$19] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks, 2019. (Cited on page 5.)

[LBV$^+$18] Chen Liu, Guillaume Bellec, Bernhard Vogginger, David Kappel, Johannes Partzsch, Felix Neumärker, Sebastian Höppner, Wolfgang Maass, Steve B Furber, Robert Legenstein, et al. Memory-efficient deep learning on a spinnaker 2 prototype. *Frontiers in neuroscience*, page 840, 2018. (Cited on page 98.)

[LC20] Dor Livne and Kobi Cohen. Pops: Policy pruning and shrinking for deep reinforcement learning. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):789–801, 2020. (Cited on pages 60 and 61.)

[LDG$^+$17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. (Cited on pages 14 and 38.)

[LH17] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017. (Cited on pages 17, 19, 40, 41, and 47.)

[LHP$^+$16] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016. (Cited on page 91.)

[Lin92] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992. (Cited on page 6.)

[LKK+21] Juhyoung Lee, Sangyeob Kim, Sangjin Kim, Wooyoung Jo, and Hoi-Jun Yoo. Gst: Group-sparse training for accelerating deep reinforcement learning. *arXiv preprint arXiv:2101.09650*, 2021. (Cited on page 62.)

[LLS+20] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020. (Cited on page 38.)

[LLZY19] Janice Lan, Rosanne Liu, Hattie Zhou, and Jason Yosinski. Lca: Loss change allocation for neural network training. In *Advances in Neural Information Processing Systems*, pages 3619–3629, 2019. (Cited on page 23.)

[LM17] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning*, pages 17–26, 2017. (Cited on pages 18, 41, and 47.)

[LMM+20] Shiwei Liu, Decebal Constantin Mocanu, Amarsagar Reddy Ramapuram Matavalam, Yulong Pei, and Mykola Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33:2589–2604, 2020. (Cited on pages 62, 98, and 109.)

[LMPP21] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6893–6904. PMLR, 18–24 Jul 2021. (Cited on page 62.)

[LMW+22] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022. (Cited on page 1.)

[Lon18] Liangqu Long. Maml-pytorch implementation. `https://github.com/dragen1860/MAML-Pytorch`, 2018. (Cited on page 104.)

[LPR17] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, pages 6467–6476, 2017. (Cited on pages 27 and 52.)

[LRD21] Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2021. (Cited on pages 7, 76, 83, 88, and 91.)

[LSGT11] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011. (Cited on page 49.)

[LVdLY$^+$21] Shiwei Liu, Tim Van der Lee, Anil Yaman, Zahra Atashgahi, Davide Ferraro, Ghada Sokar, Mykola Pechenizkiy, and Decebal Constantin Mocanu. Topological insights into sparse neural networks. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*, pages 279–294. Springer, 2021. (Cited on page 17.)

[LWL$^+$17] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017. (Cited on pages 14 and 38.)

[LXA$^+$20] Xingjian Li, Haoyi Xiong, Haozhe An, Cheng-Zhong Xu, and Dejing Dou. Rifle: Backpropagation in depth for deep transfer learning through re-initializing the fully-connected layer. In *International Conference on Machine Learning*, pages 6010–6019. PMLR, 2020. (Cited on page 91.)

[LZN$^+$23] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. (Cited on page 7.)

[MBB13] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects, 2013. (Cited on page 5.)

[MC89] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989. (Cited on pages 14 and 40.)

[MDL18] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018. (Cited on pages 18 and 19.)

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. (Cited on pages 76, 79, 85, 91, and 115.)

[ML18] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018. (Cited on pages 18 and 19.)

[ML19] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73, 2019. (Cited on pages xviii, xix, 26, 29, 46, and 48.)

[MMN+16] Decebal Constantin Mocanu, Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. A topological insight into restricted boltzmann machines. *Machine Learning*, 104(2-3):243–270, 2016. (Cited on page 17.)

[MMO95] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995. (Cited on page 40.)

[MMP+21] Decebal Constantin Mocanu, Elena Mocanu, Tiago Pinto, Selima Curci, Phuong H Nguyen, Madeleine Gibescu, Damien Ernst, and Zita A Vale. Sparse training theory for scalable and efficient agents.

In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 34–38, 2021. (Cited on pages 60, 98, and 110.)

[MMS+18] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018. (Cited on pages 17, 62, 64, and 66.)

[MVE+16] Decebal Constantin Mocanu, Maria Torres Vega, Eric Eaton, Peter Stone, and Antonio Liotta. Online contrastive divergence with generative replay: Experience replay without storing data. *arXiv preprint arXiv:1610.05555*, 2016. (Cited on pages 14 and 18.)

[MW19] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pages 4646–4655, 2019. (Cited on pages 17 and 62.)

[NS18] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2:2, 2018. (Cited on pages 41 and 55.)

[NSD+22] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pages 16828–16847. PMLR, 2022. (Cited on pages xx, xxi, 6, 11, 76, 77, 82, 89, 90, 91, 116, and 120.)

[PGL+21] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021. (Cited on page 2.)

[PKP+19] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. (Cited on page 38.)

[Pog22] Tomaso Poggio. Compositional sparsity: a framework for ml. Technical report, Center for Brains, Minds and Machines (CBMM), 2022. (Cited on page 98.)

[PRV+19] Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In *Advances in Neural Information Processing Systems*, pages 68–80, 2019. (Cited on page 41.)

[Put14] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014. (Cited on page 6.)

[RBB+18] Peter Rotheneichner, Maria Belles, Bruno Benedetti, Richard König, Dominik Dannehl, Christina Kreutzer, Pia Zaunmair, Maren Engelhardt, Ludwig Aigner, Juan Nacher, et al. Cellular plasticity in the adult murine piriform cortex: continuous maturation of dormant precursors into excitatory neurons. *Cerebral Cortex*, 28(7):2610–2621, 2018. (Cited on page 92.)

[RBW+13] Mattia Rigotti, Omri Barak, Melissa R Warden, Xiao-Jing Wang, Nathaniel D Daw, Earl K Miller, and Stefano Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585–590, 2013. (Cited on page 54.)

[RCA+18] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2018. (Cited on pages 41 and 55.)

[RKSL17] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. (Cited on pages 18, 19, and 35.)

[RL16] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016. (Cited on page 47.)

[RRD+16] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. (Cited on pages 14, 18, 19, and 41.)

[SACE23] Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2023. (Cited on page 11.)

[SAPM22] Ghada Sokar, Zahra Atashgahi, Mykola Pechenizkiy, and Decebal Constantin Mocanu. Where to pay attention in sparse training for feature selection? *Advances in Neural Information Processing Systems*, 35:1627–1642, 2022. (Cited on page 10.)

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. (Cited on page 86.)

[SCL$^+$18] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, 2018. (Cited on pages 19 and 38.)

[SDX$^+$20] Karthik Abinav Sankararaman, Soham De, Zheng Xu, W Ronny Huang, and Tom Goldstein. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. In *International conference on machine learning*, pages 8469–8479. PMLR, 2020. (Cited on page 88.)

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. (Cited on page 76.)

[SLKK17] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017. (Cited on pages 14, 18, and 19.)

[SLXC20] Qian Shi, Hak-Keung Lam, Chengbin Xuan, and Ming Chen. Adaptive neuro-fuzzy pid controller based on twin delayed deep deterministic policy gradient algorithm. *Neurocomputing*, 402:183–194, 2020. (Cited on page 63.)

[SM02] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002. (Cited on page 61.)

[SMM$^+$22] Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. Dynamic sparse training for deep reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2022. (Cited on pages 10 and 91.)

[SMP21a] Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Self-attention meta-learner for continual learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1658–1660, 2021. (Cited on page 9.)

[SMP21b] Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Spacenet: Make free space for continual learning. *Neurocomputing*, 439:1–11, 2021. (Cited on pages 9 and 62.)

[SMP22] Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Avoiding forgetting and allowing forward transfer in continual learning via sparse networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 85–101. Springer, 2022. (Cited on page 9.)

[SSMK18] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557, 2018. (Cited on page 41.)

[Sta03] Kenneth O. Stanley. Evolving adaptive neural networks with and without adaptive synapses. In *In Proceeedings of the 2003 Congress on Evolutionary Computation (CEC 2003*, pages 2557–2564. Press, 2003. (Cited on page 61.)

[Sut88] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988. (Cited on page 6.)

[Tan18] Pranjal Tandon. Pytorch implementation of soft actor critic, 2018. (Cited on page 108.)

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012. (Cited on pages 77, 85, 115, and 120.)

[THP$^+$22] Yiqin Tan, Pihe Hu, Ling Pan, Jiatai Huang, and Longbo Huang. Rlx2: Training a sparse deep reinforcement learning model from scratch. In *The Eleventh International Conference on Learning Representations*, 2022. (Cited on pages 91 and 97.)

[TL19] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. (Cited on page 1.)

[TSD21] Ahmed Taha, Abhinav Shrivastava, and Larry S Davis. Knowledge evolution in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12843–12852, 2021. (Cited on page 91.)

[VBC$^+$19] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. (Cited on pages 7, 10, and 60.)

[vdVT18] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. In *Continual Learning Workshop NeurIPS*, 2018. (Cited on pages xxv, 3, 4, 14, 18, 28, and 48.)

[vHDS$^+$18] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *CoRR*, abs/1812.02648, 2018. (Cited on page 76.)

[VHHA19] Hado P Van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019. (Cited on pages 6 and 117.)

[VLS21] Marc Vischer, Robert Tjarko Lange, and Henning Sprekeler. On lottery tickets and minimal task representations in deep reinforcement learning. In *International Conference on Learning Representations*, 2021. (Cited on page 62.)

[WJH$^+$18] Peiqi Wang, Yu Ji, Chi Hong, Yongqiang Lyu, Dongsheng Wang, and Yuan Xie. Snrram: an efficient sparse neural network computation architecture based on resistive random-access memory. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018. (Cited on page 98.)

[WJQ$^+$17] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2017. (Cited on page 41.)

[WKSF20] Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7968–7978. Curran Associates, Inc., 2020. (Cited on page 91.)

[WLZ$^+$20] Hao-nan Wang, Ning Liu, Yi-yun Zhang, Da-wei Feng, Feng Huang, Dong-sheng Li, and Yi-ming Zhang. Deep reinforcement learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, pages 1–19, 2020. (Cited on page 60.)

[WS06] Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006. (Cited on page 61.)

[WWL19] Lemeng Wu, Dilin Wang, and Qiang Liu. Splitting steepest descent for growing neural architectures. *Advances in neural information processing systems*, 32, 2019. (Cited on page 92.)

[WWPR20] Jong Ha Woo, Lei Wu, Jong-Bae Park, and Jae Hyung Roh. Real-time optimal power flow using twin delayed deep deterministic policy gradient algorithm. *IEEE Access*, 8:213611–213618, 2020. (Cited on page 63.)

[WZG+22] Zifeng Wang, Zheng Zhan, Yifan Gong, Geng Yuan, Wei Niu, Tong Jian, Bin Ren, Stratis Ioannidis, Yanzhi Wang, and Jennifer Dy. Sparcl: Sparse continual learning on the edge. *Advances in Neural Information Processing Systems*, 35:20366–20380, 2022. (Cited on page 96.)

[WZSZ23] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*, 2023. (Cited on pages 3 and 5.)

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. (Cited on page 25.)

[YETM19] Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *International Conference on Learning Representations*, 2019. (Cited on page 62.)

[YKF21] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021. (Cited on pages 76, 79, 85, 86, and 119.)

[YYLH18] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018. (Cited on pages 14, 18, 19, 41, 49, and 92.)

[ZBK+23] Sheheryar Zaidi, Tudor Berariu, Hyunjik Kim, Jorg Bornschein, Claudia Clopath, Yee Whye Teh, and Razvan Pascanu. When does re-initialization work? In *Proceedings on "I Can't Believe It's Not Better! - Understanding Deep Learning Through Empirical Falsification" at NeurIPS 2022 Workshops*, pages 12–26, 2023. (Cited on page 91.)

[ZDS+18] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7151–7160, 2018. (Cited on page 41.)

[ZHL19] Hongjie Zhang, Zhuocheng He, and Jing Li. Accelerating the deep reinforcement learning with neural network compression. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019. (Cited on pages 60 and 62.)

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016. (Cited on page 35.)

[ZKHB22] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12104–12113, 2022. (Cited on page 76.)

[ZMZ$^+$20] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: M fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2020. (Cited on pages 98 and 109.)

[ZPG17] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017. (Cited on pages 14, 17, 19, 25, 26, 40, 46, 47, 48, and 55.)

[ZSL12] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. Online incremental feature learning with denoising autoencoders. In *Artificial intelligence and statistics*, pages 1453–1461. PMLR, 2012. (Cited on page 92.)

[ZVLC21] Hattie Zhou, Ankit Vani, Hugo Larochelle, and Aaron Courville. Fortuitous forgetting in connectionist networks. In *International Conference on Learning Representations*, 2021. (Cited on page 91.)

[ZVSL18] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. (Cited on pages 14 and 38.)

Curriculum Vitae

Ghada Sokar was born on January 7, 1993. She is Egyptian and raised in Cairo. In 2014, she received a bachelor's degree in Computer Engineering from Cairo University with the prestigious distinction of honor and ranked first in her class. She obtained her master's degree in Computer Engineering from Cairo University in 2017. During her MSc, she worked at AvidBeam where she applied deep learning techniques in many applications, including age and gender detection, and object recognition.

In the summer of 2017, she was a research intern at the visual computing & artificial intelligence lab at the Technical University of Munich (TUM) in Germany. Afterward, she joined Siemens where she worked on developing a tool for automatically generating fixes for hotspots in the layout of integrated circuits (ICs) using deep learning.

In July 2019, Ghada moved to the Netherlands, where she started her PhD research in the Data Mining group in the Mathematics and Computer Science Department at the Eindhoven University of Technology under the supervision of Dr. Decebal C. Mocanu and Prof. Dr. Mykola Pechenizkiy. She simultaneously worked as a teaching assistant in the department, gaining valuable teaching experience in AI, data science, and computer science.

Ghada is an active member of the research community. She served on program committees for top AI conferences, including ICML, NeurIPs, ICLR, and ECMLPKDD. She also led some initiatives, such as co-organizing workshops on continual learning and sparse neural networks (SNNs) at ICDM and ICLR, respectively, tutorials on SNNs at IJCAI and ECMLPKDD, and a conference on continual learning with ContinualAI. Toward the end of her PhD, she joined Google Brain as a research intern for 5 months, working on reinforcement learning. She is currently a research scientist at Google DeepMind.

SIKS Dissertations

2011	01	Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models
	02	Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
	03	Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems
	04	Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference
	05	Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
	06	Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage
	07	Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction
	08	Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues
	09	Tim de Jong (OU), Contextualised Mobile Media for Learning
	10	Bart Bogaert (UvT), Cloud Content Contention
	11	Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective
	12	Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining
	13	Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling
	14	Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets
	15	Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval
	16	Maarten Schadd (UM), Selective Search in Games of Different Complexity
	17	Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness
	18	Mark Ponsen (UM), Strategic Decision-Making in complex games

19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles
20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach
21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems
22 Junte Zhang (UVA), System Evaluation of Archival Description and Access
23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media
24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics
26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns
28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure
29 Faisal Kamiran (TUE), Discrimination-aware Classification
30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions
31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science
33 Tom van der Weide (UU), Arguing to Motivate Decisions
34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training
36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach
37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization

	39	Joost Westra (UU), Organizing Adaptation using Agents in Serious Games
	40	Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development
	41	Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control
	42	Michal Sindlar (UU), Explaining Behavior through Mental State Attribution
	43	Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge
	44	Boris Reuderink (UT), Robust Brain-Computer Interfaces
	45	Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection
	46	Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
	47	Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression
	48	Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
	49	Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality
2012	01	Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda
	02	Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
	03	Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories
	04	Jurriaan Souer (UU), Development of Content Management System-based Web Applications
	05	Marijn Plomp (UU), Maturing Interorganisational Information Systems
	06	Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks
	07	Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
	08	Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories

09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms
10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment
11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems
13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems
15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance
18 Eltjo Poort (VU), Improving Solution Architecting Practices
19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution
20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval
22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval
25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
26 Emile de Maat (UVA), Making Sense of Legal Text
27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

28 Nancy Pascall (UvT), Engendering Technology Empowering Women
29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval
30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making
31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning
33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)
34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications
35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics
36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes
37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation
38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms
39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks
40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia
41 Sebastian Kelle (OU), Game Design Patterns for Learning
42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning
43 Withdrawn
44 Anna Tordai (VU), On Combining Alignment Techniques
45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions
46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior
48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data

	49	Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
	50	Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering
	51	Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching
2013	01	Viorel Milea (EUR), News Analytics for Financial Decision Support
	02	Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
	03	Szymon Klarman (VU), Reasoning with Contexts in Description Logics
	04	Chetan Yadati (TUD), Coordinating autonomous planning and scheduling
	05	Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns
	06	Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
	07	Giel van Lankveld (UvT), Quantifying Individual Player Differences
	08	Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
	09	Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications
	10	Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.
	11	Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services
	12	Marian Razavian (VU), Knowledge-driven Migration to Services
	13	Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
	14	Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning
	15	Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications

16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation
17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid
18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification
19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling
20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval
21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation
22 Tom Claassen (RUN), Causal Discovery and Logic
23 Patricio de Alencar Silva (UvT), Value Activity Monitoring
24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning
25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning
27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance
28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience
29 Iwan de Kok (UT), Listening Heads
30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support
31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications
32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere
34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search
35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction
36 Than Lam Hoang (TUe), Pattern Mining in Data Streams
37 Dirk Börner (OUN), Ambient Learning Displays

	38	Eelco den Heijer (VU), Autonomous Evolutionary Art
	39	Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems
	40	Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games
	41	Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
	42	Léon Planken (TUD), Algorithms for Simple Temporal Reasoning
	43	Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts
2014	01	Nicola Barile (UU), Studies in Learning Monotone Models from Data
	02	Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method
	03	Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions
	04	Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
	05	Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability
	06	Damian Tamburri (VU), Supporting Networked Software Development
	07	Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior
	08	Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints
	09	Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
	10	Ivan Salvador Razo Zapata (VU), Service Value Networks
	11	Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support
	12	Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control
	13	Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains
	14	Yangyang Shi (TUD), Language Models With Meta-information

15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare
16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria
17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations
19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link
21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments
22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training
23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era
24 Davide Ceolin (VU), Trusting Semi-structured Web Data
25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction
26 Tim Baarslag (TUD), What to Bid and When to Stop
27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
28 Anna Chmielowiec (VU), Decentralized k-Clique Matching
29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software
30 Peter de Cock (UvT), Anticipating Criminal Behaviour
31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support
32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data
33 Tesfa Tegegne (RUN), Service Discovery in eHealth
34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.
35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach

	36	Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models
	37	Maral Dadvar (UT), Experts and Machines United Against Cyber-bullying
	38	Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.
	39	Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital
	40	Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education
	41	Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text
	42	Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models
	43	Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments
	44	Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.
	45	Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach
	46	Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity
	47	Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval
2015	01	Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response
	02	Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls
	03	Twan van Laarhoven (RUN), Machine learning for network data
	04	Howard Spoelstra (OUN), Collaborations in Open Learning Environments
	05	Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding
	06	Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
	07	Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis

08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems
10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning
11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins
12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13 Giuseppe Procaccianti (VU), Energy-Efficient Software
14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations
15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation
16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork
17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners
20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination
21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning
22 Zhemin Zhu (UT), Co-occurrence Rate Networks
23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage
24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection
26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure
27 Sándor Héman (CWI), Updating compressed colomn stores
28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO

	29	Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
	30	Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning
	31	Yakup Koç (TUD), On the robustness of Power Grids
	32	Jerome Gard (UL), Corporate Venture Management in SMEs
	33	Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources
	34	Victor de Graaf (UT), Gesocial Recommender Systems
	35	Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction
2016	01	Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
	02	Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
	03	Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
	04	Laurens Rietveld (VU), Publishing and Consuming Linked Data
	05	Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
	06	Michel Wilson (TUD), Robust scheduling in an uncertain environment
	07	Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
	08	Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
	09	Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
	10	George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
	11	Anne Schuth (UVA), Search Engines that Learn from Their Users
	12	Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
	13	Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
	14	Ravi Khadka (UU), Revisiting Legacy Software System Modernization

	35	Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
	36	Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
	37	Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
	38	Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
	39	Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
	40	Christian Detweiler (TUD), Accounting for Values in Design
	41	Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
	42	Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
	43	Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
	44	Thibault Sellam (UVA), Automatic Assistants for Database Exploration
	45	Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
	46	Jorge Gallego Perez (UT), Robots to Make you Happy
	47	Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
	48	Tanja Buttler (TUD), Collecting Lessons Learned
	49	Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
	50	Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains
2017	01	Jan-Jaap Oerlemans (UL), Investigating Cybercrime
	02	Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
	03	Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
	04	Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store

05 Mahdieh Shadi (UVA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TUE), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UVA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VU), Logics for causal inference under uncertainty
23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search

26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VU), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (UvT), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VU), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support

	47	Jie Yang (TUD), Crowd Knowledge Creation Acceleration
	48	Angel Suarez (OU), Collaborative inquiry-based learning
2018	01	Han van der Aa (VUA), Comparing and Aligning Process Representations
	02	Felix Mannhardt (TUE), Multi-perspective Process Mining
	03	Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
	04	Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
	05	Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
	06	Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
	07	Jieting Luo (UU), A formal account of opportunism in multi-agent systems
	08	Rick Smetsers (RUN), Advances in Model Learning for Software Systems
	09	Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
	10	Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
	11	Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
	12	Xixi Lu (TUE), Using behavioral context in process mining
	13	Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
	14	Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
	15	Naser Davarzani (UM), Biomarker discovery in heart failure
	16	Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
	17	Jianpeng Zhang (TUE), On Graph Sample Clustering
	18	Henriette Nakad (UL), De Notaris en Private Rechtspraak
	19	Minh Duc Pham (VUA), Emergent relational schemas for RDF
	20	Manxia Liu (RUN), Time and Bayesian Networks
	21	Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games

	22	Eric Fernandes de Mello Araujo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
	23	Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
	24	Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
	25	Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
	26	Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
	27	Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
	28	Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
	29	Yu Gu (UVT), Emotion Recognition from Mandarin Speech
	30	Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
2019	01	Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
	02	Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
	03	Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
	04	Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
	05	Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
	06	Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
	07	Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
	08	Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
	09	Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
	10	Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
	11	Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs

12 Jacqueline Heinerman (VU), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VU),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games

	33	Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
	34	Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
	35	Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
	36	Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
	37	Jian Fang (TUD), Database Acceleration on FPGAs
	38	Akos Kadar (OUN), Learning visually grounded and multilingual representations
2020	01	Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
	02	Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
	03	Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
	04	Maarten van Gompel (RUN), Context as Linguistic Bridges
	05	Yulong Pei (TUE), On local and global structure mining
	06	Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
	07	Wim van der Vegt (OUN), Towards a software architecture for reusable game components
	08	Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
	09	Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
	10	Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
	11	Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
	12	Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
	13	Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
	14	Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases

	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines:Integrating Social Practice Theory in Agent-Based Models
	03	Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with Smart Devices
	04	Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
	05	Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
	06	Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
	07	Armel Lefebvre (UU), Research data management for open science
	08	Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
	09	Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Childrens Collaboration Through Play
	10	Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
	11	Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
	12	Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
	13	Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
	14	Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
	15	Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
	16	Esam A. H. Ghaleb (UM), BIMODAL EMOTION RECOGNITION FROM AUDIO-VISUAL CUES
	17	Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks

	18	Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
	19	Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
	20	Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
	21	Pedro Thiago Timb Holanda (CWI), Progressive Indexes
	22	Sihang Qiu (TUD), Conversational Crowdsourcing
	23	Hugo Manuel Proena (LIACS), Robust rules for prediction and description
	24	Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing
	25	Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
	26	Benno Kruit (CWI & VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
	27	Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
	28	Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs
2022	1	Judith van Stegeren (UT), Flavor text generation for role-playing video games
	2	Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
	3	Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
	4	Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
	05	Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
	06	Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
	07	Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
	08	Maikel L. van Eck (TU/e), Process Mining for Smart Product Design

09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
14 Michiel Overeem (UU), Evolution of Low-Code Platforms
15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
16 Pieter Gijsbers (TU/e), Systems for AutoML Research
17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
25 Anna L.D. Latour (LU), Optimal decision-making under constraints and uncertainty
26 Anne Dirkson (LU), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences

	27	Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
	28	Onuralp Ulusoy (UU), Privacy in Collaborative Systems
	29	Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
	30	Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
	31	Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
	32	Cezara Pastrav (UU), Social simulation for socio-ecological systems
	33	Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
	34	Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
	35	Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction
2023	01	Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
	02	Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
	03	Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
	04	Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
	05	Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
	06	António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
	07	Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
	08	Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
	09	Xu Wang (VUA) Scientific Dataset Recommendation with Semantic Techniques
	10	Dennis J.N.J. Soemers (UM) Learning State-Action Features for General Game Playing