Self-Composing Policies for Scalable Continual Reinforcement Learning

Mikel Malagón [1] **Josu Ceberio** [1] **Jose A. Lozano** [1 2]

Abstract

This work introduces a growable and modular neural network architecture that naturally avoids catastrophic forgetting and interference in continual reinforcement learning. The structure of each module allows the selective combination of previous policies along with its internal policy, accelerating the learning process on the current task. Unlike previous growing neural network approaches, we show that the number of parameters of the proposed approach grows linearly with respect to the number of tasks, and does not sacrifice plasticity to scale. Experiments conducted in benchmark continuous control and visual problems reveal that the proposed approach achieves greater knowledge transfer and performance than alternative methods.[1]

1. Introduction

The real world is non-stationary. Continuously learning, acquiring new knowledge, and fine-tuning existing skills are vital for Reinforcement Learning (RL) agents operating within our world (Hassabis et al., 2017). Dating back to Mnih et al. (2013), deep RL has demonstrated the ability to outperform humans in a constantly increasing number of tasks and domains (Badia et al., 2020; Perolat et al., 2022). These powerful algorithms are usually trained *from scratch* to operate in a stationary environment where the goal is to solve a single well-delimited problem. On the contrary, humans and other animals greatly benefit from previous experiences to efficiently solve novel tasks (Lawrence, 1952; Elio & Anderson, 1984). In this realm, the Continual Reinforcement Learning (CRL) field aims to develop agents that incrementally develop complex behavior based on previously acquired knowledge. Ideally, such agents should learn, adapt, reuse, and transfer knowledge in a never-ending stream of tasks (Hadsell et al., 2020).

It is well known that Neural Networks (NNs) can benefit from the experience obtained in simple problems to approach new and more complex challenges that otherwise would hardly be solvable or would require extreme computational resources (Wang et al., 2019; Bauer et al., 2023). However, as described by Bengio et al. (2009) and Graves et al. (2017), NNs are highly sensitive to the order of appearance and complexity of tasks. Learning a new task can easily harm the performance of the model in previously learned or future tasks due to the well-known phenomena of *catastrophic forgetting* and *interference* (McCloskey & Cohen, 1989; French, 1999; Kumaran et al., 2016). To overcome the mentioned issues, growable NN architectures (Rusu et al., 2016; Hung et al., 2019; Gaya et al., 2023) incorporate new NN modules every time a new task is presented. By retaining parameters learned in previous tasks, these methods naturally overcome forgetting and interference, while knowledge is transferred between modules by sharing hidden layer representations. However, by increasing the number of parameters, the memory cost of these models also increases. Indeed, many of these approaches grow quadratically in the number of parameters with respect to the number of tasks, greatly limiting their scalability (Terekhov et al., 2015; Rusu et al., 2016; 2017).

In this paper, we present a growable NN architecture that leverages the composition of previously learned policy modules instead of sharing hidden layer representations. This approach significantly reduces the memory and computational cost per task required by the model. Contrarily to many compositional NN approaches (Rosenbaum et al., 2019), the method we introduce eliminates the need for a dedicated NN to learn to compose the modules. Instead, modules autonomously learn to compose themselves, hence the name *self-composing policies*. Illustrated in Figure 1, the architecture, called CompoNet, adds a new module to the network each time a new task is introduced while retaining modules learned in previous tasks. Within the NN architecture of each module, policies from previous modules are selectively composed together with an internal policy, accelerating the learning process for new tasks. Direct access to previously

[1]Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, Donostia-San Sebastian, Spain [2]Basque Center for Applied Mathematics (BCAM), Bilbao, Spain. Correspondence to: Mikel Malagón <mikel.malagon@ehu.eus>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]Code available at https://github.com/mikelma/componet.

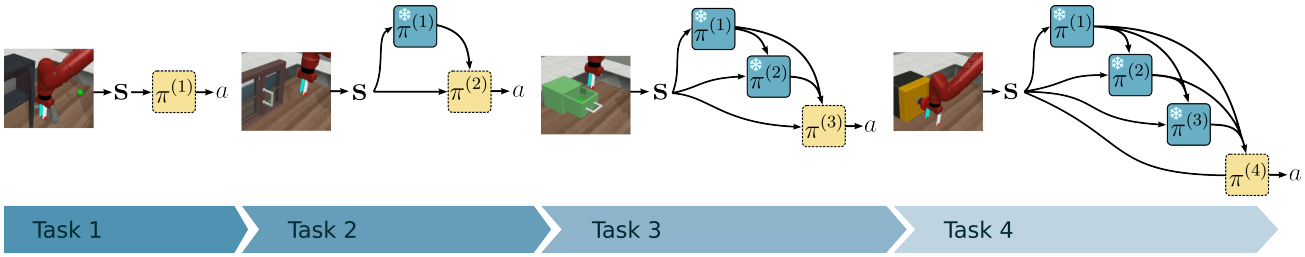


Figure 1: Evolution of the Self-**Compo**sing Policies **Net**work architecture (CompoNet) across multiple tasks. Trainable self-composing policy modules are represented by light yellow blocks, and frozen modules are denoted by dark blue blocks. The initial task involves training a single policy module from scratch. Following the completion of each task, the trained module is frozen, and a new trainable module is introduced for the subsequent task. This process is repeated in every transition from one task to another. Importantly, each module, after the initial one, benefits from access to the current state of the environment alongside the outputs generated by the preceding policy modules.

learned modules allows for easy reuse, leverage, and adaptation of stored knowledge. This procedure automatically creates a cascading structure of policies that grows in depth with the number of tasks, where each node can access and compose the outputs of previous ones to solve the current task.

Exhaustive empirical evaluation highlights that CompoNet greatly benefits from the knowledge acquired in solving previous tasks compared to other CRL methods. The latter is demonstrated for sequences of diverse robotic manipulation tasks from the Meta-World environment (Yu et al., 2020b) optimized using the Soft Actor-Critic (SAC) algorithm (Haarnoja et al., 2018), as well as in visual control tasks from the Arcade Learning Environment (ALE) (Bellemare et al., 2013) where Proximal Policy Optimization (PPO) (Schulman et al., 2017) is employed. We empirically demonstrate the robustness of the presented approach in scenarios where none of the previous policy modules offer information for solving the current task, enabling the learning of a policy from scratch with no interference. Conversely, when a function over the previous modules solves the current task, we show that CompoNet efficiently learns it after a few steps into the training. Nevertheless, knowledge transfer and robustness are not the only desirable characteristics of CRL agents: scalability is a critical feature and the potential weak point of growing-size NN approaches (Terekhov et al., 2015; Rusu et al., 2016). Despite the memory costs associated with growth, we demonstrate that the number of parameters required by CompoNet is linear with respect to the number of tasks, greatly enhancing its scalability. Furthermore, we show that CompoNet efficiently scales in inference time compared to growing NNs from the literature, which demand considerably more time and resources. Some methods address this challenge by growing only when strictly needed and/or consolidating the knowledge from multiple tasks in a single NN. However, they introduce a dilemma between scalability and the ability to acquire new knowledge (plasticity) (Mallya & Lazebnik, 2018; Hung et al., 2019). In contrast, the presented approach scales without sacrificing plasticity.

2. Related Work

The presented work builds upon the extensive literature on CRL, especially on works that explicitly retain knowledge via parameter storage (Khetarpal et al., 2022).

Growable neural networks increase their capacity every time a new task is encountered, optimizing the newly added parameters to learn the task, while retaining old parameters to avoid forgetting (Terekhov et al., 2015; Rusu et al., 2016; 2017; Czarnecki et al., 2018). This procedure allows knowledge transfer from previous NN modules to the current one, naturally avoiding catastrophic forgetting issues at the cost of computational complexity (Parisi et al., 2018; Hadsell et al., 2020). For example, the number of parameters in Rusu et al. (2016) grows quadratically with respect to the number of tasks. To address this challenge, recent efforts Yoon et al. (2018); Hung et al. (2019) prune and selectively retrain parts of old modules, while only adding new modules when needed. However, these methods introduce a trade-off between plasticity and memory cost.

Neural composition leverages the composition of specialized NNs to solve complex tasks (Rosenbaum et al., 2018; Cases et al., 2019; Tseng et al., 2021; Khetarpal et al., 2022; Mendez & Eaton, 2023). These learning systems have some similarities with the inner workings of the brain; Stocco et al. (2010) and Kell et al. (2018) provided evidence of their biological plausibility. Moreover, recent work has shown that previously learned NNs can be employed to accelerate the learning of new ones (Mendez et al., 2022), although it requires experience replay to avoid forgetting. However, as described by Rosenbaum et al. (2019) and Khetarpal et al. (2022), these methods require jointly learning the composing strategy and the NNs to compose. This is a

non-stationary problem by itself, as the composing strategy depends on the optimization of the NNs being composed and vice versa, making the training process difficult and unstable.

Avoiding forgetting by reducing plasticity has been a promising (Wolczyk et al., 2021) and popular (Khetarpal et al., 2022) approach for CRL. This line of research aims to optimize the parameters of an NN in such a way that learning a new task does not interfere with relevant parameters for solving other tasks (Kirkpatrick et al., 2017; Wortsman et al., 2020; Yu et al., 2020a; Liu et al., 2021). For example, Kirkpatrick et al. (2017) propose to selectively slow down the learning of parameters relevant to other tasks. Alternatively, Mallya & Lazebnik (2018) and Wortsman et al. (2020) decompose the parameters into subnetworks that can be retrieved according to the current task. Although these methods effectively overcome forgetting in NNs, this ability comes with the cost of limited plasticity, bringing about the *stability-plasticity dilemma* (Mermillod et al., 2013; Khetarpal et al., 2022).

3. Preliminaries

We start by introducing the notation employed in the rest of the paper and formalizing the problem to solve.

The classical RL environment (Sutton & Barto, 2018) is defined as a Markov Decision Process (MDP) in the form of $M = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, where $\mathcal{S}$ is the space of states, $\mathcal{A}$ is the space of actions, $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the state transition probability function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount rate. At every timestep, the agent receives a state $\mathbf{s} \in \mathcal{S}$ of the environment, and takes an action $a \in \mathcal{A}$ sampled from the probability function $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$, known as the policy. Then, a new state $\mathbf{s}'$ is sampled $\mathbf{s}' \sim p(\cdot|\mathbf{s}, a)$ and the reward r is computed. The objective is to find the optimal policy π^* that maximizes the expected sum of discounted rewards for all states in $\mathcal{S}$.

Aligned with the CRL definition by Khetarpal et al. (2022), we characterize non-stationary environments as MDPs whose components might exhibit some time dependence. We define a task k as a stationary MDP $M^{(k)} = \langle \mathcal{S}^{(k)}, \mathcal{A}^{(k)}, p^{(k)}, r^{(k)}, \gamma^{(k)} \rangle$, where k is discrete and changes over time, creating a sequence of tasks. Specifically, we consider that the agent has some limited budget of timesteps $\Delta^{(k)}$ to interact with a task $M^{(k)}$, to optimize the policy $\pi^{(k)}$. After consuming the budget, a new task $M^{(k+1)}$ is introduced, and the agent is limited to only interacting with this task. The aim is to accelerate and enhance the optimization of the policy $\pi^{(k)}$ to solve the task $M^{(k)}$ leveraging the knowledge of the previous policies $\{\pi^{(i)}\}_{i=1,\dots,k-1}$.

Assumptions. We take into consideration three usual assumptions on the types of variations that might occur between different tasks (Wolczyk et al., 2021; Khetarpal et al., 2022). First, the action space $\mathcal{A}$ remains constant across all tasks. We consider this assumption soft, as tasks with different groups of actions can be considered to share a single set of actions $\mathcal{A}$ by setting the probability of sampling the other actions to zero depending on the task. Secondly, task transition boundaries and their identifiers are known to the agent, as usually assumed in the literature (Wolczyk et al., 2021; 2022; Khetarpal et al., 2022). Finally, the variations between tasks mostly occur in the *underlying logic* of the tasks, mainly determined by p and r. Consequently, tasks should have a similar state space, $\mathcal{S}^{(i)} \approx \mathcal{S}^{(j)}$, as generalization across different state spaces is mostly related to the domain adaptation problem and invariant representation learning literature (Higgins et al., 2017; Zhang et al., 2020).

For this work, given a sequence of previously learned policies $\{\pi^{(i)}\}_{i=1,\dots,k-1}$, where the policy $\pi^{(i)}$ corresponds to task $M^{(i)}$, we identify three scenarios regarding the relationship between the current task $M^{(k)}$ and the set of previous policies: (i) $M^{(k)}$ can be directly solved[2] by a previously learned policy; (ii) $M^{(k)}$ can be solved by a function involving the previous policies and the current state; (iii) the current task cannot be solved by either a previous policy or a function based on the previous policies. Note that the first case is a specific instance of the second, where the function always returns the policy solving the task; we distinguish it as an especially relevant case. The design of CompoNet, presented in the next section, is specifically tailored to handle and exploit the mentioned scenarios.

4. Model Architecture

Concerning the three scenarios outlined in the preceding paragraph, our goal is to design a learning system that fulfills the following desiderata: in scenario (i) the model should exploit the previous policy that solves the current task; in (ii) the model should learn the specified function; in (iii) the agent should learn the policy that solves the task from scratch with minimal interference from previous policies.

The basic unit of the CompoNet architecture is the self-composing policy module, depicted as blocks in Figure 1. Whenever a new task $M^{(k)}$ is presented, the parameters of the policy from the preceding task $\pi^{(k-1)}$ are frozen, and a new learnable module corresponding to $\pi^{(k)}$ is introduced. While operating in the k-th task, states $\mathbf{s} \in \mathcal{S}^{(k)}$ are provided to all the previous modules, obtaining a set of $k-1$ output vectors, one for each module $j \in \{1, \dots, k-1\}$. For the sake of compact notation, we will refer to the mentioned set of outputs as the matrix $\Phi^{k;\mathbf{s}}$ with dimensions

[2] By *solve* we refer to achieving a sum of rewards in an episode (e.g., time until game over) above a certain threshold defined by the task at hand, not necessarily being the optimal policy.

$(k-1) \times |\mathcal{A}|$, where the j-th row $\Phi_j^{k;\mathbf{s}}$ corresponds to the output vector of the j-th module. Note that these vectors define the probability values of a categorical distribution over $\mathcal{A}$ when actions are discrete, and to the mean vector of a Gaussian distribution when actions are continuous. Then, we can recursively define any policy $\pi^{(k)}$ of the architecture as the probability distribution over $\mathcal{A}$ conditioned on the current state $\mathbf{s} \in \mathcal{S}^{(k)}$ and the matrix $\Phi^{k;\mathbf{s}}$, formally, $\pi^{(k)}(a|\mathbf{s}, \Phi^{k;\mathbf{s}})$.

Therefore, every time a new task is presented to the agent, its NN architecture is changed by adding a new policy module to the cascading graph structure (see Figure 1), increasing in depth by a unit. In turn, the new module is not limited to accessing only the current state $\mathbf{s}$, but it can also benefit from the policies of preceding modules. This allows the model to exploit the relations between the learned policies and the current task described in the last part of Section 3.

4.1. State Encoding

To ensure the effectiveness of CompoNet, the input of each module must remain consistent across tasks: the input state distribution of a module cannot change once its parameters are frozen. In this section, we contemplate different state encoding strategies depending on the nature of the tasks.

When states have a large dimensional representation (e.g., multiple RGB images), we consider an encoder for each module. Usually, these encoders might be defined by simple CNNs that reduce states $\mathbf{s} \in \mathcal{S}$ into a lower dimensional space $\mathbf{h_s} \in \mathbb{R}^{d_{\text{enc}}}$ where the most useful features are retained. Note that this approach is only viable when states are low-resolution images requiring simple CNNs; as is the case for many RL benchmarks such as the arcade learning environments (Bellemare et al., 2013).

Although less common, when states comprise high-dimensional images, or when dealing with an extreme number of tasks, a single encoder shared across all tasks can be used. For example, leveraging recent vision foundational models for generating representations without requiring fine-tuning or prior task-specific information (Oquab et al., 2023).[3]

Otherwise, when states are low dimensional real-valued vectors no encoder is needed for CompoNet, thus $\mathbf{h_s} = \mathbf{s}$.

4.2. Self-Composing Policy Module

In this section, the building block of the presented architecture is described: the *self-composing policy module*. As illustrated in Figure 2, it is divided into three main blocks; corresponding to the light-blue areas of the diagram: the output attention head, the input attention head, and the internal policy. The following lines describe and explain the rationale behind these three blocks.

Output Attention Head. This block proposes an output for the current module directly based on the preceding policies. Specifically, it generates a tentative vector $\mathbf{v}$ for the output of the module based on the matrix $\Phi^{k;\mathbf{s}}$ and the representation of the current state $\mathbf{h_s}$. In fact, the block employs an attention mechanism that, conditioned on $\mathbf{h_s}$, returns a linear combination of the outputs of the previous policies (the rows of $\Phi^{k;\mathbf{s}}$). The query vector is obtained as the linear transformation $\mathbf{q} = \mathbf{h_s} W_{\text{out}}^Q$, where $W_{\text{out}}^Q \in \mathbb{R}^{d_{\text{enc}} \times d_{\text{model}}}$ is a parameter matrix and d_{model} is the hidden vector size. The keys matrix is computed as $K = (\Phi^{k;\mathbf{s}} + E_{\text{out}}) W_{\text{out}}^K$, where $W_{\text{out}}^K \in \mathbb{R}^{|\mathcal{A}| \times d_{\text{model}}}$ is a parameter matrix, and E_{out} is a positional encoding matrix of the same size of $\Phi^{k;\mathbf{s}}$. The positional encoding method considered in this work is the cosine positional encoding of Vaswani et al. (2017). In the case of the values matrix, no linear transformation is considered, thus, $V = \Phi^{k;\mathbf{s}}$. The result of the output attention head is the scaled dot-product attention (Vaswani et al., 2017) of $\mathbf{q}$, K, and V:

$$\text{Attention}(\mathbf{q}, K, V) = \text{softmax}\left(\frac{qK^T}{\sqrt{d_{\text{model}}}}\right) V \qquad (1)$$

Input Attention Head. The purpose of this block is to retrieve relevant information from both the previous policies and the output attention head. It provides the necessary information for the decision-making process of the internal policy (the next block) by attending to the important features from past policies and the tentative vector $\mathbf{v}$ from the output attention head. Similarly to the previous block, it employs an attention head conditioned on $\mathbf{h_s}$, but unlike the preceding block, the attention head returns a linear combination over learnable transformations of its inputs. Specifically, the query vector is computed as $\mathbf{q} = \mathbf{h_s} W_{\text{in}}^Q$, where $W_{\text{in}}^Q \in \mathbb{R}^{d_{\text{enc}} \times d_{\text{model}}}$. Following Figure 2, the keys are computed as $(P + E_{\text{in}}) W_{\text{in}}^K$, where P is the row-wise concatenation of the output of the previous block ($\mathbf{v}$) and $\Phi^{k;\mathbf{s}}$, while E_{in} is a positional encoding matrix of the same size as P and $W_{\text{in}}^K \in \mathbb{R}^{|\mathcal{A}| \times d_{\text{model}}}$. In turn, the values matrix is obtained as the linear transformation $V = P W_{\text{in}}^V$, where $W_{\text{in}}^V \in \mathbb{R}^{|\mathcal{A}| \times d_{\text{model}}}$. Once $\mathbf{q}$, K, and V have been computed, the output of this block is the dot-product attention of these three elements, see Equation (1). Note that the learnable parameters of this block are W_{in}^Q, W_{in}^K, and W_{in}^V.

Internal Policy. This block is used to adjust, overwrite, or retain the tentative vector $\mathbf{v}$ from the output attention head, considering the contextual information provided by the input attention head and the representation of the current state. It is comprised of a feed-forward multi-layer

[3] Refer to Appendix A for preliminary results analyzing the feasibility of this approach.

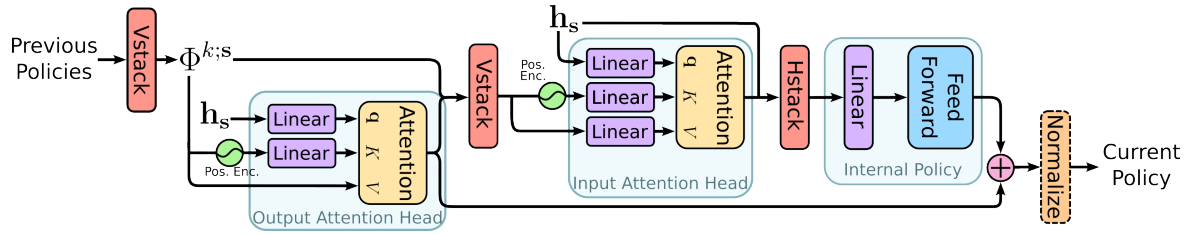


Figure 2: Diagram of the self-composing policy module. *Vstack* and *Hstack* successively represent row-wise and column-wise concatenation operations, while the normalization operation has been delimited with a dashed line to denote that it is optional and dependent on the nature of the action space. Finally, note that the only blocks with learnable parameters are the feed-forward block and the linear transformations.

perceptron network which takes the result of the previous block and $\mathbf{h_s}$ as input, generating a real-valued vector of size $|\mathcal{A}|$. Notably, this vector is not the direct output of the self-composing policy module; instead, it is added to the tentative vector $\mathbf{v}$ to form the final output of the module. Finally, depending on the nature of the task at hand, this addition might require normalization, as the output of the module usually represents a categorical distribution over $\mathcal{A}$ or continuous actions within some bounds.

In summary, the output attention head proposes an output for the module based on the current state and the information of the previous policies. Subsequently, the input attention head retrieves relevant information from both the previous policies and the output attention head. Then, the internal policy utilizes this information along with the current state representation to adjust, overwrite, or retain the tentative output from the output attention head. Note that the output attention head proposes outputs based on preceding policies and the current state representation, while the input attention head retrieves relevant information from previous modules for guiding the decision-making process of the internal policy.

In Section 3, we categorized three scenarios concerning the current task and previously learned policy modules that motivated the design of CompoNet. The subsequent lines review these scenarios within the described architecture:

(i) If a previous policy solves the current task, the output attention head assigns high attention to it, and the internal policy may output a vector of zeros to retain this result, akin to residual connections in deep learning.

(ii) If a function over the previous policies and the current state can solve the task at hand, then the three blocks of the module can be used to learn such a function.

(iii) When previous policies offer no relevant information for the task at hand, the internal policy independently learns a policy from scratch based on current state information, superseding the result of the output attention head in the last addition step.

4.3. Computational Cost

As mentioned in Section 2, computational cost stands as the primary drawback of growing NN architectures. Consequently, memory and inference costs are focal points in this study. First, the self-composing policy module is designed to mitigate the memory complexity of the model. As a result, CompoNet grows linearly in the number of parameters with respect to the number of tasks while being able to encompass the information of all previously learned modules (see Appendix B for further details). The rightmost plot in Figure 3 contrasts the memory costs of CompoNet with progressive NNs (ProgressiveNet). The latter method, introduced by Rusu et al. (2016) is one of the best-known growing NNs, and shares multiple similarities with CompoNet. Regarding the computational cost of inference, while the theoretical cost of CompoNet is quadratic with respect to the number of tasks (elaborated in Appendix C.1), the results presented in Figure 3 indicate that the empirical computational cost of CompoNet does not exhibit quadratic growth up to the 300 tasks tested, effectively scaling to very long task sequences in practice.

5. Experiments

In this section, we validate the presented architecture across sequences of tasks from multiple environments and domains. The central hypothesis of these experiments is that CompoNet should be able to benefit from forward knowledge transfer to solve the task at hand in the scenarios presented in Section 4.

5.1. Evaluation Metrics

We start by describing the CRL-relevant metrics commonly used in the literature (Wolczyk et al., 2021; 2022). Consider $p_i(t) \in [0, 1]$ to be the success rate[4] in task i at time t, indicating whether the task is solved[2], $p_i(t) = 1$, or not,

[4]We use $p_i(t)$ to denote success rate (performance) as it is the standard notation in the literature. Not to be confused with p, which is commonly used for probability functions.

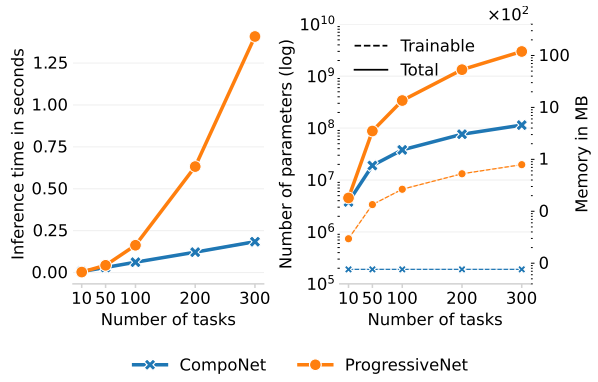


Figure 3: Empirical computational cost of inference (left) and growth in the number of parameters (right) with respect to the number of tasks for CompoNet and ProgressiveNet methods. Hyperparameters are: $d_{\text{enc}} = 64$, $|\mathcal{A}| = 6$, $d_{\text{model}} = 256$, and a batch size of 8. Measurements have been taken in a machine with an AMD EPYC 7252 CPU and an NVIDIA A5000 GPU.

$p_i(t) = 0$. Note that the metric is task-specific and defined by the problem itself. Moreover, the interaction of an agent with each task is limited to Δ timesteps, being the total number of timesteps $T = N \cdot \Delta$, where N is the number of tasks. Continuing the standard practice in CRL, we consider an agent trained from scratch in each task as the baseline for the following metrics (Díaz-Rodríguez et al., 2018; Wolczyk et al., 2021; 2022).

Average Performance. The average performance at timestep t is computed as $\mathrm{P}(t) = \frac{1}{N}\sum_{i=1}^{N} p_i(t)$. In the next sections, we report the final performance value $\mathrm{P}(T)$ as it is a commonly used metric in the CL literature (Wolczyk et al., 2021).

Forward Transfer. The forward transfer is defined as the normalized area between the training curve of the method and the training curve of the baseline. Considering $p_i^b(t) \in [0, 1]$ to be the performance of the baseline, the forward transfer FTr_i on task i is,

$$\mathrm{FTr}_i = \frac{\mathrm{AUC}_i - \mathrm{AUC}_i^b}{1 - \mathrm{AUC}_i^b}, \quad \mathrm{AUC}_i = \frac{1}{\Delta}\int_{(i-1)\cdot\Delta}^{i\cdot\Delta} p_i(t)\mathrm{d}t, \quad \mathrm{AUC}_i^b = \frac{1}{\Delta}\int_0^{\Delta} p_i^b(t)\mathrm{d}t \tag{2}$$

Under this metric, a key concept is the **Reference forward Transfer (RT)**. A CRL method should ideally perform at least as well as fine-tuning from the task with the highest transfer to the current one. Thus, RT is defined as follows:

$$\mathrm{RT} = \frac{1}{N}\sum_{i=2}^{N} \max_{j<i} \mathrm{FTr}(j, i) \tag{3}$$

where $\mathrm{FTr}(j, i)$ is the forward transfer obtained by training a model from scratch in the j-th task and fine-tuning it in the i-th task. Note that a model can outperform the RT by composing the knowledge from previous tasks (Wolczyk et al., 2021).

5.2. Experimental Setup

To validate the proposed method, we conducted experiments comparing CompoNet with other CRL methods from the literature across three sequences of tasks. The first sequence includes 20 robotic arm manipulation tasks (a sequence of 10 different tasks repeated twice) from Meta-World (Yu et al., 2020b), an established benchmark in meta-learning and multi-task learning communities. In these tasks, states and actions consist of low dimensional real-valued vectors and a budget of $\Delta = 1M$ timesteps per task has been used.[5] Following the common practice of the literature, the Soft Actor-Critic (SAC) (Haarnoja et al., 2018) algorithm has been used to optimize every method. The other two task sequences are selected from the Arcade Learning Environment (Machado et al., 2018). In this case, actions are discrete, and states consist of RGB images of 210×160 pixels. Thus, we employ a CNN encoder as described in Section 4.1 to encode images into a lower dimensional space (see Appendix E.1). The first sequence corresponds to the 10 playing modes of the *ALE/SpaceInvaders-v5* environment, while the last one to the 7 playing modes of the *ALE/Freeway-v5* environment, with $\Delta = 1M$ timesteps per task. The Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm is employed to optimize the methods in these tasks as commonly employed in the literature (Huang et al., 2022).

Methods. In addition to CompoNet, we include five methods for comparison. The baseline involves training a randomly initialized NN for each task, serving as the baseline reference. We expect CRL methods to perform at least as well as the baseline. Adhering to common practice in the literature (Wolczyk et al., 2021; 2022), FT-1 continuously fine-tunes a single NN model across all tasks; expecting more advanced CRL methods to at least match its forward transfer. FT-N adopts a similar approach but preserves the models trained at each task to prevent forgetting. ProgressiveNet (Rusu et al., 2016) is selected for its similarity to this work and for serving as a basis for more complex approaches. This method instantiates a new NN module every time the task changes, freezing the parameters of previous modules and adding lateral connections between the hidden layers of the modules. Finally,

[5] We utilize the same task sequence as in the CW20 benchmark proposed by Wolczyk et al. (2021), using the second version of the tasks from Yu et al. (2020b) due to code deprecation issues. See Appendix D.1.1 for further details.

Table 1: Summary of results across all task sequences and methods. Metrics from Section 5.1 are presented as averages and standard deviations from 10 random seeds, with the best results highlighted in bold. The last row indicates the reference forward transfer (RT) for each sequence. CompoNet, the method proposed in this paper, achieves superior performance and forward transfer in all three sequences.

	Meta-World		SpaceInvaders		Freeway	
Method	Perf.	Fwd. Transf.	Perf.	Fwd. Transf.	Perf.	Fwd. Transf.
Baseline	0.06±0.12	0.00±0.00	0.56±0.37	0.00±0.00	0.19±0.26	0.00±0.00
FT-1	0.03±0.09	-0.21±0.38	0.44±0.50	0.73±0.25	0.15±0.36	0.64±0.09
FT-N	0.37±0.48	-0.21±0.38	**0.99±0.01**	0.73±0.25	0.81±0.01	0.64±0.09
ProgNet	0.41±0.49	-0.04±0.04	0.71±0.25	0.10±0.07	0.47±0.28	0.30±0.18
PackNet	0.24±0.40	-0.67±1.38	0.63±0.33	0.36±0.31	0.51±0.20	0.31±0.25
CompoNet	**0.42±0.49**	**0.01±0.14**	**0.99±0.01**	**0.74±0.22**	**0.94±0.06**	**0.80±0.07**
RT	—	-0.06	—	0.70	—	0.67

PackNet (Mallya & Lazebnik, 2018), stores the parameters to solve every task of the sequence in the same network by building masks to avoid overwriting the ones used to solve previous tasks. PackNet has shown strong results in previous works (Hung et al., 2019; Wolczyk et al., 2022).

The complete description of each environment and task is available in Appendix D, while implementation details and hyperparameters are provided in Appendix E.

5.3. Results

Table 1 shows the results for all methods in each of the task sequences under the metrics described in Section 5.1, including the RT for every sequence (computed from the transfer matrices in Appendix D.5).

Regarding performance in the Meta-World sequence, both growing NN methods exhibit the highest performance, with CompoNet outperforming ProgressiveNet. For the SpaceInvaders sequence, CompoNet and FT-N show the highest performance, achieving identical scores, while ProgressiveNet trails with a 0.28 lower score. In the Freeway sequence, CompoNet significantly outperforms all other methods by a considerable margin, followed by FT-N with 0.13 lower performance.

In terms of forward transfer, CompoNet stands out as the only method achieving positive forward transfer (i.e. has no interference) in the challenging Meta-World sequence. This highlights the robustness of CompoNet given the high interference nature of the sequence demonstrated by the negative RT value. In SpaceInvaders, CompoNet achieves the highest forward transfer, followed by FT-1 and FT-N, the only three methods surpassing the RT in this sequence. Finally, in the Freeway sequence, CompoNet notably outperforms other methods and the RT. Across all sequences, CompoNet consistently improves the RT, demonstrating that its knowledge transfer capabilities go beyond fine-tuning the previous task of highest transfer, also leveraging knowledge composition.

Refer to Appendix F.1 for the success rate curves in each task. In Appendix F.2 we describe and provide results for forgetting, another common metric in the CRL literature. Note that this metric has been omitted from this section as the only affected methods are the baseline and FT-1.

5.4. Architectural Validation

Returning to the scenarios presented in the first lines of Section 4, in (i) CompoNet should be able to efficiently reuse previous policies; in (ii) it should extract useful information from previous policies to accelerate the learning process on the current task; and in (iii), when previous tasks have no relation with the current one, it should learn a policy from scratch. The ability of CompoNet to leverage forward knowledge transfer to more efficiently solve unseen tasks (second scenario) has been demonstrated in Section 5.3, here, we verify that CompoNet fulfills the expected behavior in the first and last scenarios.

Regarding the first objective, we train CompoNet on the fifth task of the SpaceInvaders sequence with a previous policy already trained on this task (referred to as *Inf. Mod.* in Figure 4) and four random modules that output a random policy sampled from a uniform Dirichlet distribution at every timestep. Figure 4.a shows the episodic return[6] curves (greater is better), where CompoNet clearly benefits from the information of the previous policy trained in the current task. Observing the behavior of the output attention head in Figure 4.d, we see it assigns high attention scores to the previous policy trained in the task at hand (*Inf. Mod.*, marked with triangles), letting the output from this policy to pass through the attention head to the other blocks of the current module. In turn, the input attention head (see Figure 4.c) attends to the previous policy that solves the task and to the output attention head, whose output is equal to the output of the previous policy that solves the current task

[6]Episodic return is defined as the sum of all rewards for an episode (e.g., time until the game is over).

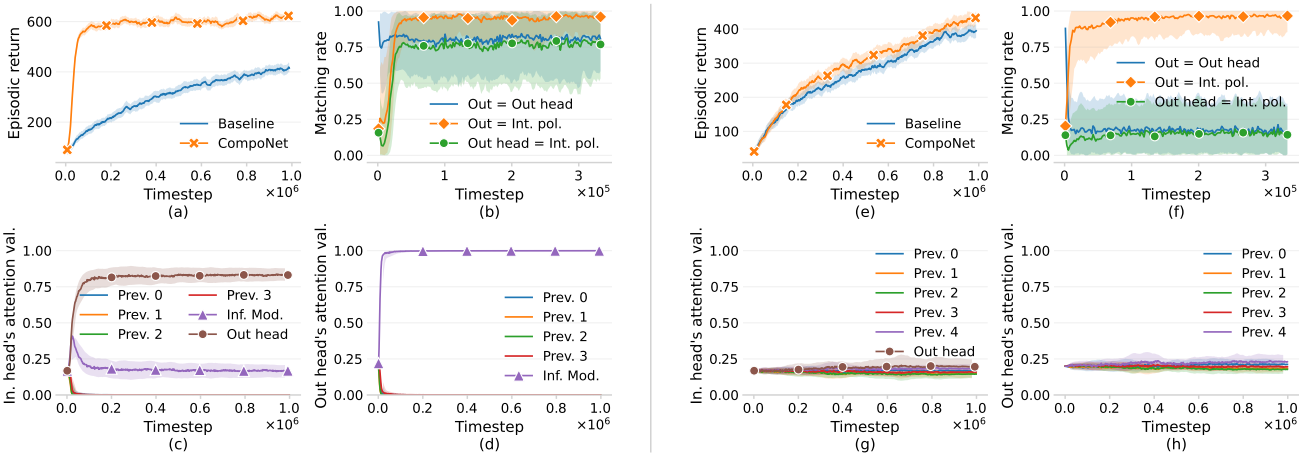


Figure 4: Empirical results on the fulfillment of objectives (i) and (iii) from Section 4 that motivated the design of CompoNet. In the leftmost figures, CompoNet is trained on the fifth task of SpaceInvaders with four non-informative previous policies that sample their output from a uniform Dirichlet distribution, and one policy trained to solve the current task (*Inf. Mod.*). In the rightmost figures, CompoNet is trained on the sixth task of SpaceInvaders with five non-informative previous policies. Results aggregate 10 random seeds. The matching rate indicates the frequency with which the action of the highest probability from the output of the last module, or some of its components, matches with each other. For clarity, the X axis of Figures 4.b and 4.f are shortened to the first 3×10^5 timesteps.

at this point on the training. Finally, Figure 4.b shows the rate at which the action of the highest probability of two outputs are equal, computed for the output of the internal policy (*Int. Pol.*), output attention head (*Out head*), and the final output of the model (*Out*). In the initial timesteps, the output of the model matches with the result of the output attention head, and the internal policy is barely used. After several timesteps, the internal policy learns to imitate the result of the output attention head, which is mostly used as the final output of the model.

To validate CompoNet in the last scenario, in the rightmost plots of Figure 4 CompoNet is trained to solve the sixth task of SpaceInvaders, having five previous modules that output a random policy in the same way as in the previous experiment. Observing Figure 4.e, we see that CompoNet matches (and slightly improves, possibly due to the extra stochasticity introduced by the previous policies) the performance of training a policy from scratch. Moreover, Figure 4.f shows that the final output of the model is completely determined by the internal policy after a few training steps, effectively overwriting the result of the output attention head. Finally, in Figures 4.g and 4.h we can see that both attention heads give uniform attention to all of their inputs. This behavior is expected as none of the previous modules provides any useful information for the task at hand.

As additional evidence on the validity of the architecture described in Section 4.2, Appendix G presents ablation studies to verify the functionalities of the different blocks that constitute the architecture of the self-composing policy module. Finally, Appendix H demonstrates the scalability of CompoNet to handle hundreds of previous modules.

6. Conclusions

We propose CompoNet, a modular growing NN architecture for CRL that naturally avoids catastrophic forgetting and interference while leveraging the knowledge obtained in previous tasks to address new ones, as demonstrated in experiments across task sequences of different natures. Unlike other growable NN approaches, CompoNet grows linearly in the number of parameters with the number of tasks and shows substantially better scalability in terms of inference. These properties allow CompoNet to scale to many sequential tasks in the practice, without sacrificing plasticity and performance. Contrary to most compositional NN methods, CompoNet does not require a dedicated model to compose learned modules, as modules learn to compose themselves. Moreover, experiments demonstrate that CompoNet is able to utilize the knowledge obtained in previous tasks while learning without interference when the current task is unrelated to prior ones.

We believe that this work is a step forward in the development of CRL agents that benefit from learning from numerous sequential tasks in the practice, however, several challenges remain. Although CompoNet already scales to many tasks in practice, improving its computational cost is key for considering never-ending continual learning problems. We think that quantization (Xiao et al., 2023) and policy distillation (Rusu et al., 2015) could allow this issue

to be addressed. Finally, likewise to most works in the CRL literature (Rusu et al., 2016; Hung et al., 2019; Wolczyk et al., 2021; 2022), our work assumes agent knowledge of task boundaries and identifiers. Incorporating a method to extract such information in CompoNet would allow new modules to be automatically added. Although this is beyond the scope of this work, we believe that future research on systems that extract such information is pivotal for advancing CRL research.

Acknowledgements

We are grateful to Jose A. Pascual for the technical support. We also thank Jon Vadillo and Ainhize Barrainkua for reading the preliminary versions of the paper.

Mikel Malagón acknowledges a predoctoral grant from the Spanish MICIU/AEI with code PREP2022-000309, associated with the research project PID2022-137442NB-I00 funded by the Spanish MICIU/AEI/10.13039/501100011033 and FEDER, EU.

This work is also funded through the BCAM Severo Ochoa accreditation CEX2021-001142-S/MICIN/AEI/10.13039/501100011033; and the Research Groups 2022-2025 (IT1504-22), and Elkartek (KK-2021/00065 and KK-2022/00106) from the Basque Government.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the Atari human benchmark. In *Proceedings of the 2020 International Conference on Machine Learning (ICML)*, pp. 507–517, 2020.

Bauer, J., Baumli, K., Behbahani, F., Bhoopchand, A., Bradley-Schmieg, N., Chang, M., Clay, N., Collister, A., Dasagi, V., Gonzalez, L., Gregor, K., Hughes, E., Kashem, S., Loks-Thompson, M., Openshaw, H., Parker-Holder, J., Pathak, S., Perez-Nieves, N., Rakicevic, N., Rocktäschel, T., Schroecker, Y., Singh, S., Sygnowski, J., Tuyls, K., York, S., Zacherl, A., and Zhang, L. M. Human-timescale adaptation in an open-ended task space. In *Proceedings of the 2023 International Conference on Machine Learning (ICML)*, volume 202, pp. 1887–1935. PMLR, 2023.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 47:253–279, jun 2013.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 2009 International Conference on Machine Learning (ICML)*, pp. 41–48, 2009.

Cases, I., Rosenbaum, C., Riemer, M., Geiger, A., Klinger, T., Tamkin, A., Li, O., Agarwal, S., Greene, J. D., Jurafsky, D., et al. Recursive routing networks: Learning to compose modules for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3631–3648, 2019.

Czarnecki, W., Jayakumar, S., Jaderberg, M., Hasenclever, L., Teh, Y. W., Heess, N., Osindero, S., and Pascanu, R. Mix & match agent curricula for reinforcement learning. In *Proceedings of the 2018 International Conference on Machine Learning (ICML)*, pp. 1087–1095, 2018.

Díaz-Rodríguez, N., Lomonaco, V., Filliat, D., and Maltoni, D. Don't forget, there is more than forgetting: new metrics for continual learning. *ArXiv preprint arXiv:1810.13166*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the 2021 International Conference on Learning Representations (ICLR)*, 2021.

Elio, R. and Anderson, J. R. The effects of information order and learning mode on schema abstraction. *Memory & Cognition*, 12(1):20–30, 1984.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

Gaya, J.-B., Doan, T., Caccia, L., Soulier, L., Denoyer, L., and Raileanu, R. Building a subspace of policies for scalable continual learning. In *Proceedings of the 2023 International Conference on Learning Representations (ICLR)*, 2023.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. In *Proceedings of the 2017 International Conference on Machine Learning (ICML)*, pp. 1311–1320, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 2018 International Conference on Machine Learning (ICML)*, pp. 1861–1870, 2018.

Hadsell, R., Rao, D., Rusu, A. A., and Pascanu, R. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 24(12):1028–1040, 2020.

Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. DARLA: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 2017 International Conference on Machine Learning (ICML)*, pp. 1480–1490, 2017.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research (JMLR)*, 23(274):1–18, 2022.

Hung, C.-Y., Tu, C.-H., Wu, C.-E., Chen, C.-H., Chan, Y.-M., and Chen, C.-S. Compacting, picking and growing for unforgetting continual learning. In *Proceedings of the 2019 Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Kell, A. J., Yamins, D. L., Shook, E. N., Norman-Haignere, S. V., and McDermott, J. H. A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy. *Neuron*, 98(3):630–644, 2018.

Khetarpal, K., Riemer, M., Rish, I., and Precup, D. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research (JAIR)*, 75:1401–1476, 2022.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. In *Proceedings of the 2017 National Academy of Sciences*, pp. 3521–3526. National Acad Sciences, 2017.

Kumaran, D., Hassabis, D., and McClelland, J. L. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in Cognitive Sciences*, 20(7):512–534, 2016.

Lawrence, D. H. The transfer of a discrimination along a continuum. *Journal of Comparative and Physiological Psychology*, 45 6:511–16, 1952.

Liu, B., Liu, X., Jin, X., Stone, P., and Liu, Q. Conflict-averse gradient descent for multi-task learning. In *Proceedings of the 2021 Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 18878–18890, 2021.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 61:523–562, 2018.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the 2018 IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7765–7773, 2018.

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

Mendez, J. A. and Eaton, E. How to reuse and compose knowledge for a lifetime of tasks: A survey on continual learning and functional composition. *Transactions on Machine Learning Research (TMLR)*, 2023.

Mendez, J. A., van Seijen, H., and EATON, E. Modular lifelong reinforcement learning via neural composition. In *Proceedings of the 2022 International Conference on Learning Representations (ICLR)*, 2022.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., et al. DINOv2: Learning robust visual features without supervision. *ArXiv preprint arXiv:2304.07193*, 2023.

Parisi, G. I., Ji, X., and Wermter, S. On the role of neurogenesis in overcoming catastrophic forgetting. In *NIPS Workshop on Continual Learning*, 2018.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Perolat, J., De Vylder, B., Hennes, D., Tarassov, E., Strub, F., de Boer, V., Muller, P., Connor, J. T., Burch, N., Anthony,

T., et al. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*, 378(6623): 990–996, 2022.

Rosenbaum, C., Klinger, T., and Riemer, M. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *Proceedings of the 2018 International Conference on Learning Representations (ICLR)*, 2018.

Rosenbaum, C., Cases, I., Riemer, M., and Klinger, T. Routing networks and the challenges of modular and compositional computation. *ArXiv preprint arXiv:1904.12774*, 2019.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

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

Rusu, A. A., Večerík, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. Sim-to-real robot learning from pixels with progressive nets. In *Proceedings of the 2017 Conference on Robot Learning (CoRL)*, pp. 262–270, 2017.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *ArXiv preprint arXiv:1707.06347*, 2017.

Stocco, A., Lebiere, C., and Anderson, J. R. Conditional routing of information to the cortex: A model of the basal ganglia's role in cognitive coordination. *Psychological Review*, 117(2):541, 2010.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Terekhov, A. V., Montone, G., and O'Regan, J. K. Knowledge transfer in deep block-modular neural networks. In *Proceedings of the 2015 International Conference on Biomimetic and Biohybrid Systems*, pp. 268–279. Springer, 2015.

Tseng, W.-C., Lin, J.-S., Feng, Y.-M., and Sun, M. Toward robust long range policy transfer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9958–9966, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Proceedings of the 2017 Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. *ArXiv preprint arXiv:1901.01753*, 2019.

Wolczyk, M., Zajac, M., Pascanu, R., Kucinski, L., and Miloś, P. Continual World: A robotic benchmark for continual reinforcement learning. In *Proceedings of the 2021 Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 28496–28510, 2021.

Wolczyk, M., Zajkac, M., Pascanu, R., Kucinski, L., and Miloś, P. Disentangling transfer in continual reinforcement learning. In *Proceedings of the 2022 Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 6304–6317, 2022.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., and Farhadi, A. Supermasks in superposition. In *Proceedings of the 2020 Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 15173–15184, 2020.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 2023 International Conference on Machine Learning (ICML)*, pp. 38087–38099, 2023.

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

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. In *Proceedings of the 2020 Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 5824–5836, 2020a.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the 2020 Conference on Robot Learning (CoRL)*, pp. 1094–1100, 2020b.

Zhang, A., McAllister, R., Calandra, R., Gal, Y., and Levine, S. Learning invariant representations for reinforcement learning without reconstruction. In *Proceedings of the 2020 International Conference on Learning Representations (ICLR)*, 2020.

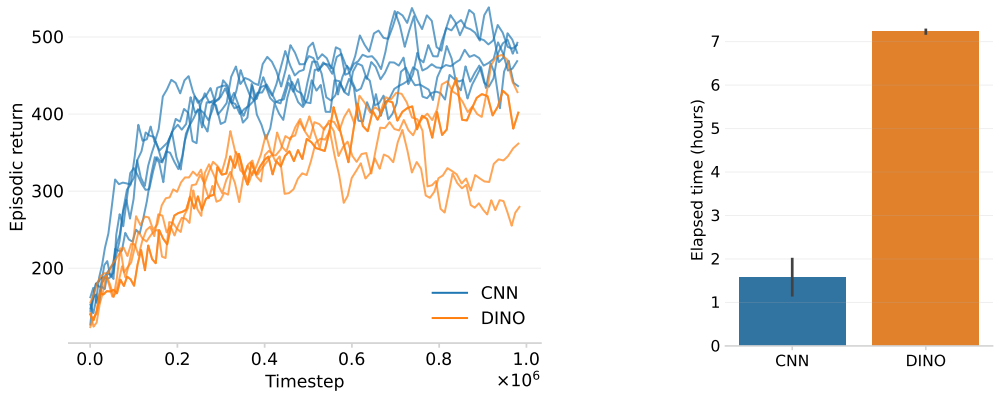
A. Vision Foundation Models for Visual Control Tasks in RL

In the context of Section 4.1, where different state encoding strategies for CompoNet are discussed, in this appendix we show preliminary results on the possibility of using a single pre-trained visual foundation model for generating low dimensional representations of high dimensional images.

By employing a single encoder, the number of parameters required by each module of the CompoNet architecture could be considerably reduced in some cases. In fact, this strategy can be of special interest in task sequences where states are composed of high-dimensional images or in extremely large sequences, where the cost of maintaining a CNN for each module is unaffordable. Note that this is not the case for the image-based sequences considered in this work (SpaceInvaders and Freeway) and that this appendix only aims to shed some light on alternative strategies to follow in the mentioned hypothetical cases.

Specifically, for this experiment, we employ the recent DINOv2 vision foundation model by Oquab et al. (2023), which demonstrated the ability to generate all-purpose visual features from images that can be used for multiple tasks such as image classification, segmentation, or depth recognition.

For this appendix, we have employed the smallest model from Oquab et al. (2023), a 21M parameter vision transformer (Dosovitskiy et al., 2021) with an embedding size of 384 (referred to as d_{enc} in Sections 4.1 and onwards) and a patch size of 14 pixels. As the model is a vision transformer, its input consists of a sequence of non-overlapping patches of the input image and a *CLS* token. The output is a sequence of the same length that includes patch tokens and the class token (corresponding to *CLS*), that serves as the representation of the whole image. Consequently, for an image of 224×224 pixels, the DINOv2 ViT-S model outputs a single class token and 256 patch tokens, where the dimension of each one is 384.



(a) (b)

Figure A.1: Comparison of training an agent in the first task of the SpaceInvaders sequence using the representations generated by a pre-trained DINO model, and a CNN encoder that is trained with the agent. The leftmost figure compares both approaches in terms of episodic return, while the rightmost compares the total execution times to reach 1M timesteps. Both figures contain the results from 5 runs per setting.

Figure A.1 shows the comparison of training an agent (the baseline method using the hyperparameters from Table E.2) to solve the first task of the SpaceInvaders sequence using the representations encoded by the DINOv2 pre-trained model, and a CNN that is trained alongside the agent. As depicted, in most cases, the final episodic return obtained by both approaches is similar. However, the CNN-based agent can reach higher scores faster and the DINO-based agent obtains considerably lower values for a couple of repetitions. Although the analysis is preliminary and results can be improved, the similarity in the final performance demonstrates that the representations generated by the pre-trained DINO model can be used to successfully train an RL agent to solve this type of visual task. Moreover, note that the DINO model has not been trained with the RL agent, and despite not having data from this game in its training, it is still able to extract useful information to solve the task at hand.

Remember that using visual foundation models such as DINO, would only be needed in extremely large task sequences, or

in tasks where states are composed of considerably high dimensional images. In fact, the computational cost of using such a large encoder model (the smallest DINO model has 21M parameters) is counter-productive for the small image-based tasks considered in this work. This is supported by the training time comparison presented in Figure A.1b, where the training times of both approaches are shown. As depicted, the CNN-based models require less than half of the training time of the DINO-based ones.

B. Memory Cost of CompoNet

Memory cost has been a major concern in this work, as it is one of the major issues of growing NNs, greatly limiting their scalability. In this section, we show that the presented architecture grows linearly in the number of parameters with respect to the number of tasks, being scalable in terms of memory.

CompoNet requires adding a new policy module every time a new task arrives, however, as described in Section 4.2 and illustrated in Figure 2, each module only consists of a few components that require parameters. Specifically, six linear transformations and a feed-forward block, the rest of the operations have no parameters. As specified in Section 4.2, the size of the matrices that correspond to these linear transformations is determined by the output dimension of the encoder d_{enc}, the hidden vector size of the model d_{model} and the number of actions $|\mathcal{A}|$. Additionally, the number of parameters of the feed-forward block is only dependent on the number of layers and the hidden vector dimension (assumed to be equal to d_{model}). Note that the size of the matrices used in the attention heads and the feed-forward network are only dependent on fixed hyperparameters. Therefore, the number of parameters of a self-composing policy module is constant and independent of the number of tasks addressed. Considering m to be the number of parameters of a self-composing module, the number of parameters of the CompoNet architecture is $\mathcal{O}(m \cdot n)$, where n is the number of tasks. As m is constant, the memory complexity of CompoNet with respect to the number of tasks is linear, $\mathcal{O}(n)$.

Figure B.1 provides a comparison of the growth of CompoNet and ProgressiveNet in the number of parameters and the required memory in Megabytes. The hyperparameters are set according to those employed in the Meta-World sequence (detailed in Appendix E and Table E.1).

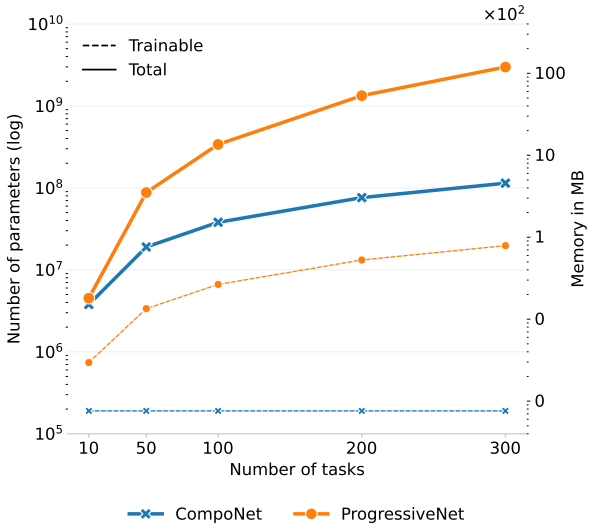


Figure B.1: Growth in memory of CompoNet and ProgressiveNet models as the number of tasks (assuming an NN module per task) increases. The count is given in the total number of parameters, depicted with solid lines, and in the number of trainable parameters (not frozen), in dashed lines. Note that we assume 32-bit floats are used to represent the parameters of the models. Hyperparameters correspond to the ones utilized in the Meta-World sequence: $d_{\text{enc}} = 39$, $d_{\text{model}} = 256$, and $|\mathcal{A}| = 4$.

To get a sense of the possible scalability to extremely large task sequences, Figure B.2 shows the memory cost (in gigabytes) of CompoNet for up to 10K tasks and different d_{model} values. As shown, in the case of $d_{\text{enc}} = 256$ (which corresponds to the one used in the experimentation) the proposed architecture can grow to more than 10k modules in a single NVIDIA A5000 GPU with 24GB of VRAM memory.

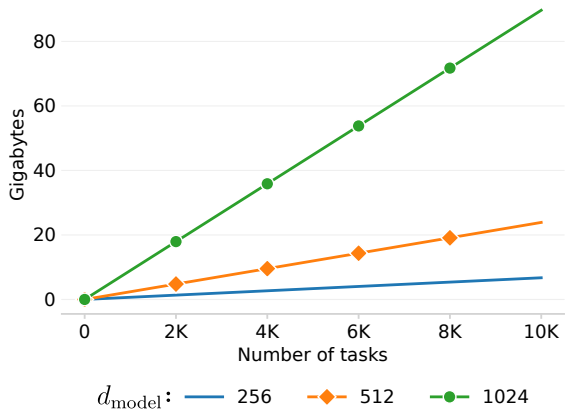


Figure B.2: Memory size in Gigabytes required by CompoNet with respect to the number of tasks for different d_{model} values. Hyperparameters correspond to the ones from the Meta-World sequence: $d_{\text{enc}} = 39$, and $|\mathcal{A}| = 4$.

C. Computational Cost of Inference in CompoNet

In this section, we analyze the computational cost of inference with respect to the number of tasks for the CompoNet architecture. First, we describe the computational complexity of the model in big-O-notation. Then, we study the empirical computational cost of inference of CompoNet for hundreds of tasks.

C.1. Computational Complexity of Inference

According to the diagram presented in Figure 2, the self-composing policy module can be divided into three parts: the output attention head, the input attention head, and the internal policy. In the following lines, we respectively develop the computational complexity of each of the three parts with respect to the number of self-composing policy modules (n).[7]

Output Attention Head. The input of this block comprises the current state representation $\mathbf{h_s}$, the matrix with the results of the previous modules $\Phi^{k;\mathbf{s}}$, and the same matrix with the positional encoding $\Phi^{k;\mathbf{s}} + E_{\text{out}}$. The first element, $\mathbf{h_s}$, has a constant dimension d_{enc}, while the $\Phi^{k;\mathbf{s}}$ matrix has a size of $(n-1) \times a$, where the second dimension a is constant with respect to the number of modules n but the first is not. This implies that the computational cost of calculating the key K matrix and the dot product required to compute the attention depends on the number of modules n. [8] Formally,

$$T_{\text{input}}(n) = \underbrace{d_{\text{enc}} \cdot d_{\text{model}}}_{\text{Compute } \mathbf{q}} + \underbrace{(n-1) \cdot a \cdot d_{\text{model}}}_{\text{Compute } K} + \underbrace{d_{\text{model}} \cdot (n-1)}_{\mathbf{q}K^T} + \underbrace{O(n-1)}_{\text{Cost of softmax}} + \underbrace{a \cdot (n-1)}_{\text{Mult. att. and } V} \tag{4}$$

Note that all the terms in $T_{\text{input}}(n)$ except the first depend on n. As there is no term with higher order than n, then the computational complexity of the output attention head is $T_{\text{input}}(n) = O(n)$.

Input Attention Head. This block requires the representation of the current state $\mathbf{h_s}$, and the row-wise concatenation of the result from the previous block with $\Phi^{k;\mathbf{s}}$. Note that similarly to the output attention head, this block also adds a positional encoding to the keys matrix, however, the values matrix (V) is computed via a linear transformation (unlike the previous block, where $V = \Phi^{k;\mathbf{s}}$). In this case, the computational complexity is,

$$T_{\text{output}}(n) = \underbrace{d_{\text{enc}} \cdot d_{\text{model}}}_{\text{Compute } \mathbf{q}} + \underbrace{2 \cdot n \cdot a \cdot d_{\text{model}}}_{\text{Compute } K \text{ and } V} + \underbrace{d_{\text{model}} \cdot n}_{\mathbf{q}K^T} + \underbrace{O(n)}_{\text{Softmax}} + \underbrace{n \cdot d_{\text{model}}}_{\text{Mult. att. and } V} \tag{5}$$

Likewise the previous attention head, the highest order in $T_{\text{output}}(n)$ is n, and thus, its computational complexity is $T_{\text{output}}(n) = O(n)$.

[7] The computational complexity of multiplying an $n \times m$ and an $m \times p$ matrix is $O(nmp)$.

[8] Remember from Section 4 that $\mathbf{q} = \mathbf{h_s} W^Q_{\text{out}}$, $K = (\Phi^{k;\mathbf{s}} + E_{\text{out}}) W^K_{\text{out}}$, and $V = \Phi^{k;\mathbf{s}}$ where $W^Q_{\text{out}} \in \mathbb{R}^{d_{\text{enc}} \times d_{\text{model}}}$, $W^K_{\text{out}} \in \mathbb{R}^{|\mathcal{A}| \times d_{\text{model}}}$, and E_{out} is a positional matrix of the same size as $\Phi^{k;\mathbf{s}}$. Moreover, attention is computed as $\text{Attention}(\mathbf{q}, K, V) = \text{softmax}\left(qK^T / \sqrt{d_{\text{model}}}\right) V$.

Internal Policy. The input of the internal policy consists of a vector of size $(d_{\text{enc}} + d_{\text{model}})$ and outputs a vector of length $a = |\mathcal{A}|$. Assuming that the internal policy is constructed from linear layers, its computational complexity is defined as the following,

$$T_{\text{int}}(n) = \underbrace{d_{\text{model}} \cdot (d_{\text{enc}} + d_{\text{model}})}_{\text{First layer}} + \underbrace{(l-2) \cdot d^2_{\text{model}}}_{\text{Intermediate layers}} + \underbrace{d_{\text{model}} \cdot a}_{\text{Last layer}} \tag{6}$$

Note that none of the terms of $T_{\text{int}}(n)$ depends on n (the number of policy modules). Therefore, the computational cost of the internal policy is constant and independent of the number of modules (i.e., number of tasks), $T_{\text{int}}(n) = O(1)$.[9]

Total Complexity. Considering the computational complexity of inference of computing the three parts that constitute the self-composing policy module, the total complexity of a single module is $T_{\text{output}}(n) + T_{\text{input}}(n) + T_{\text{int}}(n) = O(n) + O(n) + O(1) = O(n)$. However, considering a CompoNet model of n policy modules, obtaining its final output requires computing the results of all n modules sequentially, as each module depends on the previous ones. Consequently, although the complexity of inference for a policy module is linear, the total complexity of CompoNet is $n \cdot O(n) = O(n^2)$.

C.2. Empirical Computational Cost

The previous section focused on the theoretical computational complexity of inference for the CompoNet architecture, in this section, we empirically measure the time required to perform these computations in practice.

Specifically, we run inference in CompoNet and ProgressiveNet models with an increasing number of modules, simulating inference in models that would have faced hundreds of tasks, maintaining a module for each one. To provide practical and realistic measurements, we set the same hyperparameters employed in the Meta-World sequence for this experiment (see Appendix E and Table E.1).

Results are shown in Figure C.1. Although the computational complexity of both approaches is quadratic, the quadratic trend of CompoNet grows substantially slower compared to ProgressiveNet. This might be caused by the fact that ProgressiveNet requires the information of all previous modules in every layer of each module (or NN column as referred to in Rusu et al. (2016)). In contrast, although CompoNet also requires the information of all previous modules, the keys and values for the attention heads are computed in parallel.

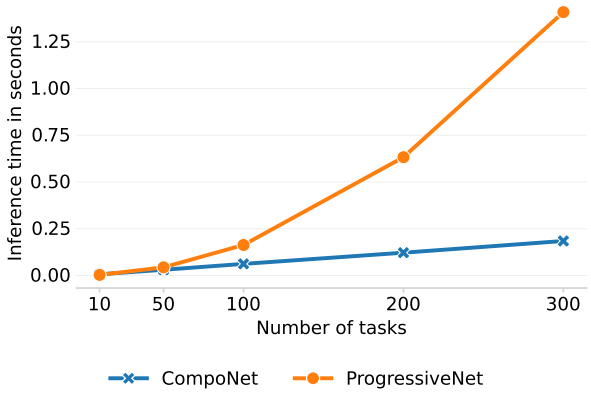


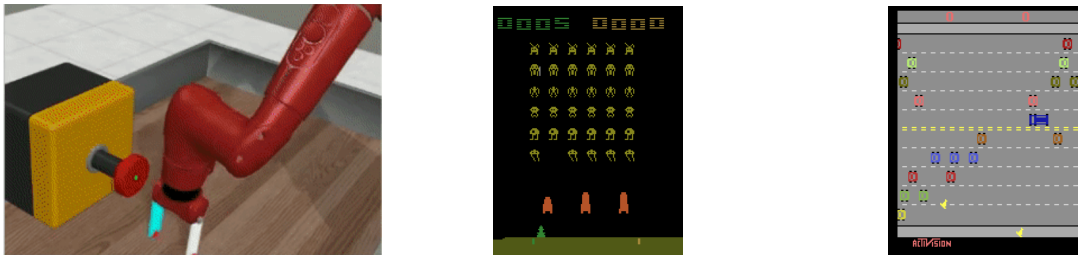
Figure C.1: Empirical computational cost of inference of CompoNet and ProgressiveNet with respect to the number of tasks (assuming one module is instantiated for every task). Results depict the average inference time over one minute of recording. This ensures a minumum of 40 inferences to compute the estimate ($40 = 60/1.5$). Hyperparameters are: $d_{\text{enc}} = 39$, $d_{\text{model}} = 256$, and $|\mathcal{A}| = 4$. Results were measured in a machine with an AMD EPIC 7252 CPU and an NVIDIA A5000 GPU.

[9]For the sake of simplicity, this definition of the internal policy ignores possible activation and normalization layers.

D. Environments and Tasks

The experimentation is conducted across three task sequences: one consisting of continuous control robotic tasks from Meta-World presented by Yu et al. (2020b), while the other two correspond to the different playing modes of the SpaceInvaders and Freeway games from the Arcade Learning Environment (Bellemare et al., 2013; Machado et al., 2018). An illustration of a task from each sequence is depicted in Figure D.1.

Appendix D.1, D.2, and D.3 describe the environments and tasks of the mentioned sequences. Next, Appendix D.4 provides details on the calculation of the success rate in the SpaceInvaders and Freeway sequences. Finally, Appendix D.5 gives the forward transfer matrices used to compute the RT values shown in Table 1.



(a) Meta-World (b) SpaceInvaders (c) Freeway

Figure D.1: An example of a frame for each of the considered task sequences.

D.1. Meta-World

This sequence is composed of tasks from Meta-world, an open-source suite that includes a variety of robotic arm tasks proposed in Yu et al. (2020b). Although Yu et al. (2020b) proposed 50 different tasks, in this work we have selected a sequence of 10 tasks, that we repeated twice making 20 tasks in total $(10 + 10)$, following the same practice as Wolczyk et al. (2021).

In these tasks, states are 39-dimensional real-valued vectors, $\mathcal{S} \subset \mathbb{R}^{39}$, where information on the positions, rotations, and states of the different parts of the robotic arm and the objects in the environment are encoded. In turn, the action space consists of a 4-dimensional real-valued vector in the range $[-1, 1]$, which represents the change in the position of the arm and the torque that the gripper fingers should apply. The following lines describe the 10 different tasks in order:

hammer-v2. Hammer a screw into the wall, with random hammer and screw positions every episode.

push-wall-v2. Bypass a wall and push a puck to a goal, with random positions of the puck and goal in every episode.

faucet-close-v2. Rotate the faucet clockwise, random faucet position every episode.

push-back-v2. Push a mug under a coffee machine, with random positions of the mug and the machine every episode.

stick-pull-v2. Grab a stick and pull a box with the stick, random stick position every episode.

handle-press-side-v2. Press a handle down sideways, random handle position every episode.

push-v2. Push the puck to a goal, with random puck and goal positions every episode.

shelf-place-v2. Pick and place a puck onto a shelf, with random positions of the puck and the shelf every episode.

window-close-v2. Push and close a window, with a random position of the window every episode.

peg-unplug-side-v2. Unplug a peg sideways, random peg position every episode.

D.1.1. A Note on the Selection of the Tasks in the Meta-World Sequence

The selection of tasks and their order in our Meta-World sequence is directly influenced by the Continual World (CW) benchmark proposed in Wolczyk et al. (2021), specifically the CW20 benchmark, which comprises a sequence of 20 Meta-World tasks (a sequence of 10 different tasks repeated twice) for evaluating CRL methods. While we adopt the exact sequence from the benchmark, we use the second version of the environments available in the up-to-date version of Meta-

World at the time of writing (e.g., `hammer-v1` replaced by `hammer-v2`). This decision is motivated by several factors. Firstly, the second version of the library addresses issues present in the first version (e.g., non-Markovian observations in `v1`[10]). Moreover, the source code of the CW benchmark has not been updated since 2021 and seems to lack maintenance.[11] Additionally, CW relies on a deprecated version of the `gym` library, with `gymnasium`[12] now recommended by `gym` authors as a modern replacement.[13] In fact, `gymnasium` is now under the Farama Foundation[14], a non-profit organization that promotes the standardization and maintenance of RL libraries. Notably, from the second version of their library, the Meta-World project is also under the Farama Foundation, ensuring ongoing maintenance and sustainability.

D.2. SpaceInvaders

This sequence is based on the *ALE/SpaceInvaders-v5* environment (see Figure D.1b) of the `gymnasium` library.[15] In SpaceInvaders, all tasks share the same objective: shoot space invaders before they reach the Earth. The game ends if the player is hit by enemy fire (falling laser bombs) or when invaders reach the ground. Observations consist of 210×160 RGB images of the frames, and actions are: *do nothing*, *fire*, *move right*, *move left*, *combination of move right and fire*, and *combination of move left and fire*. Tasks correspond to the different playing modes of the game. The player has three lives, and destroying space invaders is rewarded (hitting the invaders in the back rows gives more reward). The considered tasks are the following:

Mode 0. The default playing mode described at the beginning of this section.

Mode 1. Shields (orange blocks between the player and the invaders, see Figure D.1b) move back and forth on the screen, instead of staying in a fixed position. Using them as protection becomes unreliable.

Mode 2. The laser bombs dropped by the invaders *zigzag* as they come down the screen, making it more difficult to predict the place they are going to land.

Mode 3. This task combines modes 1 and 2.

Mode 4. Same as mode 0 but laser bombs fall considerably faster.

Mode 5. Same as mode 1 but laser bombs fall considerably faster.

Mode 6. Same as mode 2 but laser bombs fall considerably faster.

Mode 7. Same as mode 3 but laser bombs fall considerably faster.

Mode 8. Same as mode 0 but invaders become invisible for a few frames.

Mode 9. Same as mode 1 but invaders become invisible for a few frames.

D.3. Freeway

Like the previous sequence, Freeway is based on the *ALE/Freeway-v5* environment from the same library. On Freeway, the objective is to guide a chicken to cross a road with busy traffic (see Figure D.1c). [16] In the same way as SpaceInvaders, observations consist of the RGB frames of the game. There are only three possible actions: *do nothing*, *move up*, and *move down*. A reward is only given when the chicken reaches the top of the screen after crossing all the lanes of traffic, resulting in an environment of particularly sparse reward. The considered tasks corresponding to the different game modes are the following:

Mode 0. Corresponds to the default playing mode of the game.

Mode 1. Traffic is heavier and the speed of the vehicles increases, the upper lane closest to the center has trucks. Trucks are longer vehicles, and thus, more difficult to avoid.

[10] See the following issue for more details: https://github.com/Farama-Foundation/Metaworld/issues/392.

[11] The source code can be found at https://github.com/awarelab/continual_world. Although the last commit at the time of writing is from 2023, the commit did not modify any code-related files.

[12] The homepage of the project can be found at https://gymnasium.farama.org/.

[13] See the README file of the original repository at https://github.com/openai/gym.

[14] See the homepage of the organization at https://farama.org/.

[15] More information at https://gymnasium.farama.org/environments/atari/space_invaders/.

[16] More information at https://gymnasium.farama.org/environments/atari/freeway/.

Mode 2. Trucks move faster than the fastest vehicles of the previous modes, and traffic is heavier.

Mode 3. There are trucks in all lanes, and trucks move as fast as in mode 2 in some of the lanes.

Mode 4. Similar traffic to previous modes, there are no trucks, but the velocity of the vehicles is randomly increased or decreased.

Mode 5. Same as mode 1 with the speed of the vehicles randomly changing and some vehicles come in groups of two or three very close to each other.

Mode 6. Same as the previous mode but with heavier traffic.

D.4. Success Scores for the SpaceInvaders and Freeway Sequences

As described in Section 5.1, the CRL metrics we use in this work are defined over the concept of *performance*, which is in turn defined in terms of the *success rate*. The success rate is based on the binary metric that indicates whether the task is solved[2] (1), or not (0). In the case of the Meta-World sequence, the success rate is given by the environments themselves (Yu et al., 2020b). As the SpaceInvaders and Freeway sequences are generated in this work, we define success as 1 if the episodic return is greater or equal to the *success score*, otherwise zero. In both environments, we aim to have a high success rate for approaches with high performance, and low success rate for lower performing methods. For this purpose, we compute the success score for each task as 90% of the average final episodic return (i.e., the performance once trained) of all 8 methods (10 random seeds per method). For reproducibility, the success scores are given in Tables D.1a and D.1b. Note that in the case of both sequences, the success score is considerably greater than the episodic return obtained by an agent that acts randomly[17], 148.0 and 0.0 for SpaceInvaders and Freeway respectively (Badia et al., 2020). This ensures that a successful policy must at least learn a more sophisticated behavior than a completely random policy.

Table D.1: Success score values for the SpaceInvaders and Freeway task sequences. If the episodic return in a task is greater or equal to this score, the task is considered solved and success is 1, otherwise 0.

(a) SpaceInvaders

	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9
Success score	340.94	366.762	391.16	386.99	379.41	383.73	393.83	367.98	484.23	456.19

(b) Freeway

	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7
Success score	16.65	15.1	8.27	17.09	18.54	9.43	9.14	13.96

D.5. Forward Transfer Matrices

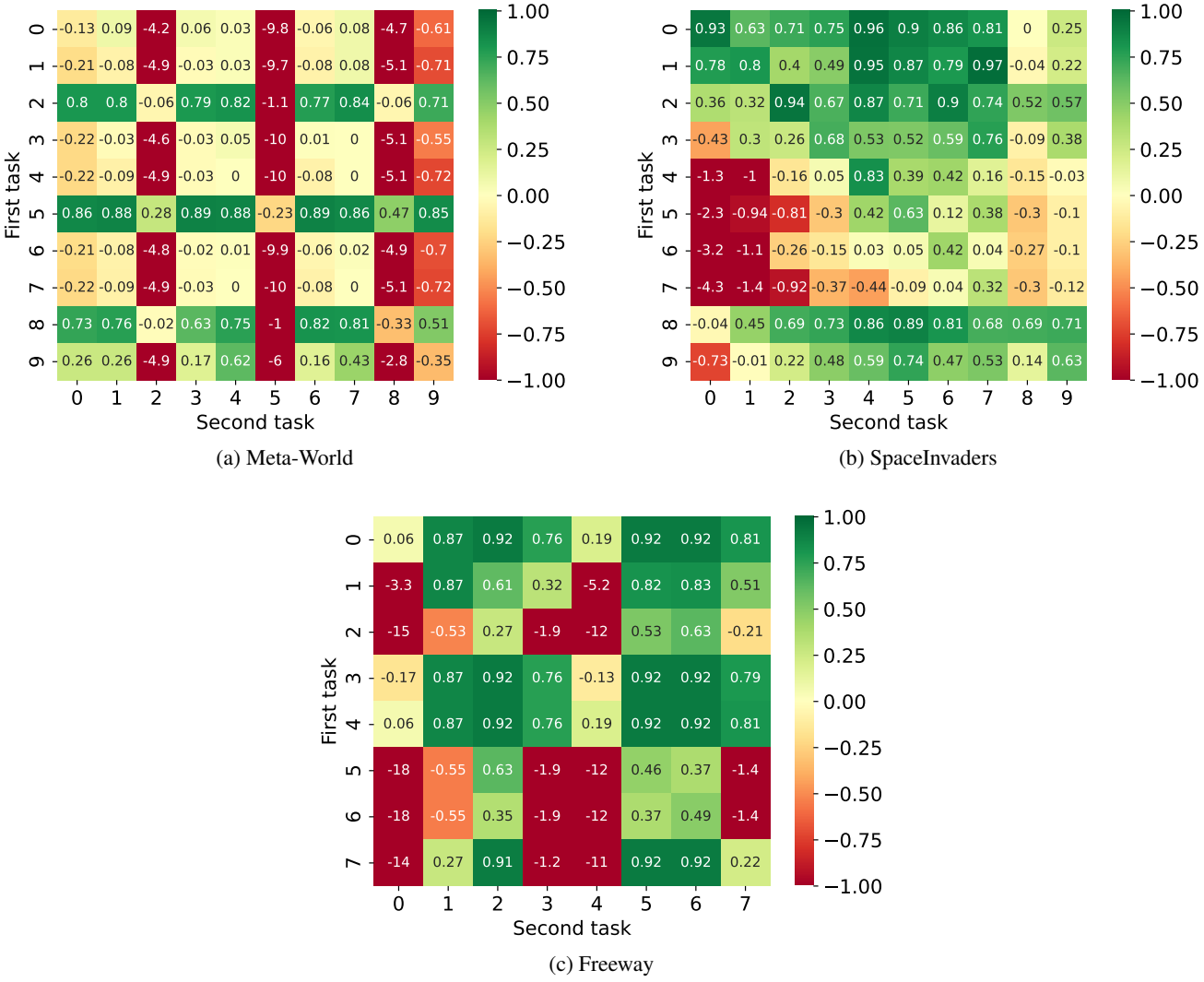
Figure D.2 presents the forward transfer (FTr) matrices used to compute the RT of each sequence, described in Equation (3) from Section 5.1 and provided in Table 1.

Starting from the matrix corresponding to the Meta-World sequence in Figure D.2a, we see that most of the forward transfer values are negative, including the RT (reported in Table 1), indicating considerable interference between the tasks that comprise the sequence. However, few tasks that greatly transfer to others, indicated by the mostly green rows when the first task is 2, 5, 8, or 9. Contrarily, looking at the columns of the matrix, we see that when the second task is 2, 5, or 8, the transfer to these tasks from practically any other task is negative, implying that other tasks provide almost no information to solve the mentioned ones. Finally, note that the diagonal of the matrix has no especially positive transfer values. Even though this might appear to be counterintuitive (as fine-tuning an agent trained from scratch in the same task might result in high FTr), resetting the replay buffer in SAC and learning from instances collected from a mostly uniform policy in the first steps of training (the output head is re-initialized at the beginning of each task) has been shown to cause this effect, as described by Wolczyk et al. (2021).

[17] Whose actions are always sampled from a uniform distribution over $\mathcal{A}$.

Regarding SpaceInvaders, in Figure D.2b, we observe considerably higher transfer values compared to the previous sequence. In fact, SpaceInvaders is the task with the highest RT, as shown in Table 1. Looking at the transfer matrix, we see that the rows corresponding to tasks 0 and 8 have especially higher transfer values, showing that these tasks have a high transfer to the other ones. Moreover, in the area of negative values in the center leftmost part of the matrix, we see that tasks of the middle of the sequence (4-7) transfer poorly to the first tasks of the sequence (0-2). Note that, in this case, the optimization algorithm is PPO, which has no replay buffer, and thus, the values of the diagonal are all positive and have a considerably high value, although not all are close to 1. We believe that this might be caused by the stochasticity introduced when re-initializing the last layer of the actor and the critic network every time the task changes.

Figure D.2c provides the FTr matrix of the last sequence, Freeway. Likewise SpaceInvaders, we see that, in general, the transfer values in this sequence are mostly positive, however, this sequence shows particularly low values when the transfer is negative. The latter corresponds to the values in the column corresponding to task 0, where the only tasks from which this one benefits are task 4 and itself. Concerning the diagonal of the matrix, we observe the same behavior as in the previous task.



(a) Meta-World

(b) SpaceInvaders

(c) Freeway

Figure D.2: Forward transfer matrices for all sequences. Each element in the matrices is computed as the average forward transfer of training a model from scratch in the first task (Y-axis) and fine-tuning it in the second (X-axis). Results aggregate values from 3 different random seeds. Note that Figure D.2a is a 10×10 matrix and not 20×20, corresponding with the 10 different tasks that comprise the sequence, as the remaining 10 are repetitions of these.

E. Implementation Details

To make the presented experiments as fair and reproducible as possible, our implementations are based on those from Huang et al. (2022), which provides high-quality implementations of RL algorithms. Specifically, we maintain modifications to the original SAC and PPO implementations minimal, only changing the definition of the agent to hold the architecture described in Section 4 and the methods from Section 5.2. Note that both SAC and PPO require an actor-critic setting, where the actor is the model from which actions are sampled, while the critic is trained to predict the sum of future rewards given a state. Following common practice in the literature (Wolczyk et al., 2022) the CRL methods are only applied to the network of the actor while the critic is restarted at the beginning of each task. All methods share the same common hyperparameters in every task sequence.

In the Meta-World sequence, where SAC is used to optimize all methods, the complete list of hyperparameters is provided in Table E.1. Following the settings by Yu et al. (2020b), as with the rest of the hyperparameters for this sequence, the actor and critic networks have been defined as a two-layer MLP, followed by two separate output heads of a single layer, for the mean and the logarithm of the standard deviation of the normal distribution.

Table E.2 shows the hyperparameters shared by every method in the case of the SpaceInvaders and Freeway sequences under the PPO algorithm. In this case, the same encoder architecture from Appendix E.1 is used for every method to extract low-dimensional feature vectors from image observations. Finally, unless otherwise specified, two single-layer output heads are used to generate the logits of the categorical distribution over the action space (actor) and to compute the value function (critic). Note that these hyperparameters are set according to the ones employed in the reference implementations from Huang et al. (2022).

Table E.1: Hyperparameters shared by all methods in the Meta-World task sequence under the SAC algorithm.

	Description	Value
Common	Optimizer	*Adam*
	Adam's β_1 and β_2	(0.9, 0.999)
	Discount rate (γ)	0.99
	Max Std.	e^2
	Min Std.	e^{-20}
	Activation Function	*ReLU*
	Hidden Dimension (d_{model})	256
SAC Specific	Batch size	128
	Buffer size	10^6
	Target Smoothing Coef. (τ)	0.005
	Entropy Regularization Coef. (α)	0.2
	Auto. Tuning of α	Yes
	Policy Update Freq.	2
	Target Net. Update. Freq	1
	Noise Clip	0.5
	Number of Random Actions	10^4
	Timestep to Start Learning	5×10^3
Networks	Target Net. Layers	3
	Critic Net. Layers	3
	Actor's Learning Rate	10^{-3}
	Q Networks's Learning Rate	10^{-3}

E.1. Encoder for Visual Control Tasks

As mentioned in Section 5.2, the ALE environments (SpaceInvaders and Freeway) define an observation space of 210×160 pixels, requiring an encoder network (see Section 4.1) to process the images. Specifically, we define the same Convolutional Neural Network (CNN) architecture for all methods. Following the implementation from Huang et al. (2022), the CNN is composed of three convolutional layers with 32, 64, and 64 channels, with filter sizes being 8, 4, and 3 respectively. The last layer is a dense layer with an output dimension of 512. The rest of the hyperparameters of the encoder network are the ones provided in Table E.2.

Table E.2: Hyperparameters shared by all methods in the SpaceInvaders and Freeway task sequences under the PPO algorithm.

	Parameter	Value
Common	Optimizer	Adam
	AdamW's β_1 and β_2	(0.9, 0.999)
	Max. gradient norm	0.5
	Discount rate (γ)	0.99
	Activation Function	*ReLU*
	Hidden Dimension (d_{model})	512
	Learning rate	$2.5 \cdot 10^{-4}$
PPO Specific	PPO Value function coef.	0.5
	GAE λ	0.95
	Num. parallel environments	8
	Batch size	1024
	Mini-batch size	256
	Num. mini-batches	4
	Update epochs	4
	PPO Clipping coefficient	0.2
	PPO Entropy coefficient	0.01
	Learn. Rate Annealing	Yes
	Clip Value Loss	Yes
	Normalize Advantage	Yes
	Num. Steps per Rollout	128

E.2. Method Specific Details

The following lines describe all of the considered CRL methods in the experimentation from Section 5, also providing method-specific implementation details.

Baseline. This method involves training an NN initialized with random parameters for each task. Although not a CRL method *per se*, we believe that training from scratch is a valid approach for CRL, however, we expect CRL-specific methods to at least match its performance. Moreover, this method serves as the baseline for the forward transfer metric presented in Equation (2) from Section 5.1.

FT-1 and FT-N. FT-1 consists of continuously fine-tuning the same NN from the first to the last task of a sequence. Although this approach usually achieves great results in forward transfer, the final performance is usually relatively low and suffers from considerable forgetting (Wolczyk et al., 2022; 2021; Parisi et al., 2018). This is mainly driven by the fact that FT-1 does not protect the parameters learned in previous tasks when learning new ones, overwriting relevant parameters for other tasks and causing catastrophic forgetting (see Appendix F). Alternatively, FT-N follows the same procedure as the latter method but saves the model trained in each of the N tasks to alleviate the aforementioned issues at the sacrifice of increasing memory cost. Following the implementation by Wolczyk et al. (2021), the output heads have been re-initialized at the beginning of every task, fine-tuning the preceding layers of the networks.

Progressive Neural Networks. ProgressiveNet proposed in Rusu et al. (2016) is one of the best-known growing NN approaches (Rusu et al., 2017; Parisi et al., 2019; Hung et al., 2019; Hadsell et al., 2020; Khetarpal et al., 2022; Gaya et al., 2023). Every time the task changes ProgressiveNet instantiates a new network, adding lateral connections between the current network and the ones learned in previous tasks (whose parameters are frozen, similarly to CompoNet). Specifically, the input of every layer of the new network will not only consist of the output of the preceding layer but also include the outputs of the layers from the networks learned in previous tasks. Although ProgressiveNet has shown great performance and significant forward transfer abilities across tasks of multiple domains (Rusu et al., 2016; Hung et al., 2019; Gaya et al., 2023), the number of parameters required by the approach grows quadratically with respect to the number of tasks, as does its computational cost of inference.

Note that the ProgressiveNet architecture consitutes the whole actor network except the final output layer (i.e., the *head*). After the end of each task, the last layer is saved and a new layer (with random initial parameters) is instantiated for the new

task.

PackNet. Proposed by Mallya & Lazebnik (2018), this method retains the parameters learned in several tasks in the same network while avoiding catastrophic forgetting. Specifically, once a task is learned, the method prunes the network selecting the parameters relevant to solve the current task, and retrains it to preserve performance. Then, the selected parameters are kept frozen (immutable) for the rest of the future tasks. In each task, the number of iterations to retrain the model is set to 20% of the timestep budget of each task ($200K$ steps, being $\Delta = 1M$).

Note that this method requires knowledge of the total number of tasks beforehand, as the number of parameters to be kept for each task (p/N, where p is the total number of parameters, and N the number of tasks in the sequence) has to be known in advance. This requirement is one of the main issues of PackNet in the context of CRL, as continually learning agents should be able to learn in a variable or an unknown number of tasks (Hadsell et al., 2020). Moreover, note that the number of trainable parameters decreases with every task, as p/N parameters are frozen every time a task ends. In fact, after the N-th task, no trainable parameters are left in the network.

As with ProgressiveNet, and following the procedure by Wolczyk et al. (2021), we apply PackNet to all layers except the last one, which is saved at the end of each task, and re-initialized before starting a new one.

CompoNet. Regarding the specific implementation details for CompoNet, we have applied the presented architecture to the actor of the SAC and PPO algorithms, using a separate network for the critic. In the case of SAC, the internal policy of CompoNet has been defined with the parameters in Table E.1. The output of the module defines the mean vector of the normal distribution employed by the agent, while a separate network has been used for the logarithm of the standard deviation. In the image-based task sequences, SpaceInvaders and Freeway, we use a single encoder per module as described in Section 4.1, where the encoder of a new module is initialized with the weights of the previous one. Moreover, the output of the encoder, $\mathbf{h_s}$, served both for the CompoNet module (the actor), and as the input to the value function, defined as a single fully connected layer. Finally, the internal policy employs a fully connected network with two layers following the hyperparameters in Table E.2.

E.3. Hardware Resources

The experimentation has been conducted on two cluster nodes, one containing eight RTX3090 GPUs, an Intel Xeon Silver 4210R CPU, and 345GB of RAM, while the second comprises eight Nvidia A5000 GPUs with an AMD EPYC 7252 CPU and 377GB of RAM. Execution times for the SpaceInvaders and Freeway sequences were approximately 1.5 hours per task, and approximately 3 hours in the case of Meta-World, times mostly varied depending on the method.

F. Further Experimental Results

In this section, we present additional experimental results that complement observations discussed in the main body of the paper, providing a more comprehensive understanding of the performance and behavior of the considered methods.

F.1. Success Rate Curves

The success rate curves of the methods in the Meta-World, SpaceInvaders, and Freeway sequences are provided in Figures F.1, F.2, and F.3 respectively. The figures show the results from which the CRL relevant metrics from Section 5.1 have been derived to obtain the values presented in Table 1 of the main body of the paper. Each figure aggregates the results of 10 random seeds per method in every task. The first row illustrates the average curves of all methods, while the rest of the rows depict each method separately including a contour with the standard deviation. Note that the FT-1 and FT-N methods share the same success rate curves, referred to as FT in the figures. The latter is driven by the fact that both methods share the exact same learning method and hyperparameters, only differing in how they handle forgetting (not measured with this metric).

Note that in Figure F.1, corresponding to Meta-World, in tasks 4, 7, 14, and 17 (corresponding to `stick-pull-v2` and `shelf-place`, as tasks 14 and 17 are repetitions of these), none of the methods can obtain a consistent positive success rate. Although we employ the hyperparameters in Yu et al. (2020b) for Meta-World, we believe that these results might be caused by the differences in the implementations of the SAC algorithm, as our implementation is based on the one from Huang et al. (2022). Nevertheless, we think that considering a few considerably difficult tasks is of special interest in the

context of CRL: methods should be robust to any type of scenario, and should not assume that consistent positive success will be achieved for every task.

Regarding the curves of the SpaceInvaders sequence in Figure F.2, we observe that every method obtains some positive success rate in each task. In the curves of the baseline method, we see that from task 6 and onwards, tasks are especially hard to approach with no information from previous ones. In fact, the rest of the methods, which consider information from preceding tasks, achieve substantially higher success rate values. Also note that in the first tasks of the sequence (mainly tasks 0 and 1), PackNet reaches high success values, falling to very low values at the end of these tasks. This behavior is caused by the heavy pruning of the network in the first tasks of the sequence, where a high percentage of the parameters are pruned to consolidate the most important parameters for the task at hand. For a description of the method refer to Appendix E.2.

Figure F.3 provides the success rate curves of the last sequence, Freeway. From the curves corresponding to the baseline, we identify that tasks 2, 4, 5, and 7 are of special difficulty within this sequence. This might be caused by the fact that Freeway is a scenario of very sparse reward, as the agent obtains a positive reward only when it reaches its objective, otherwise, the reward is zero. Therefore, the mentioned tasks might be particularly hard scenarios for a random initial policy to obtain positive feedback from which to optimize the policy. Refer to Appendix D.3 for a detailed description of Freeway and its tasks. Concerning PackNet, although the success rate drops when pruning the network (mostly noticeable in the first three tasks of the sequence), the retraining phase effectively updates the network to the same success values obtained before pruning.

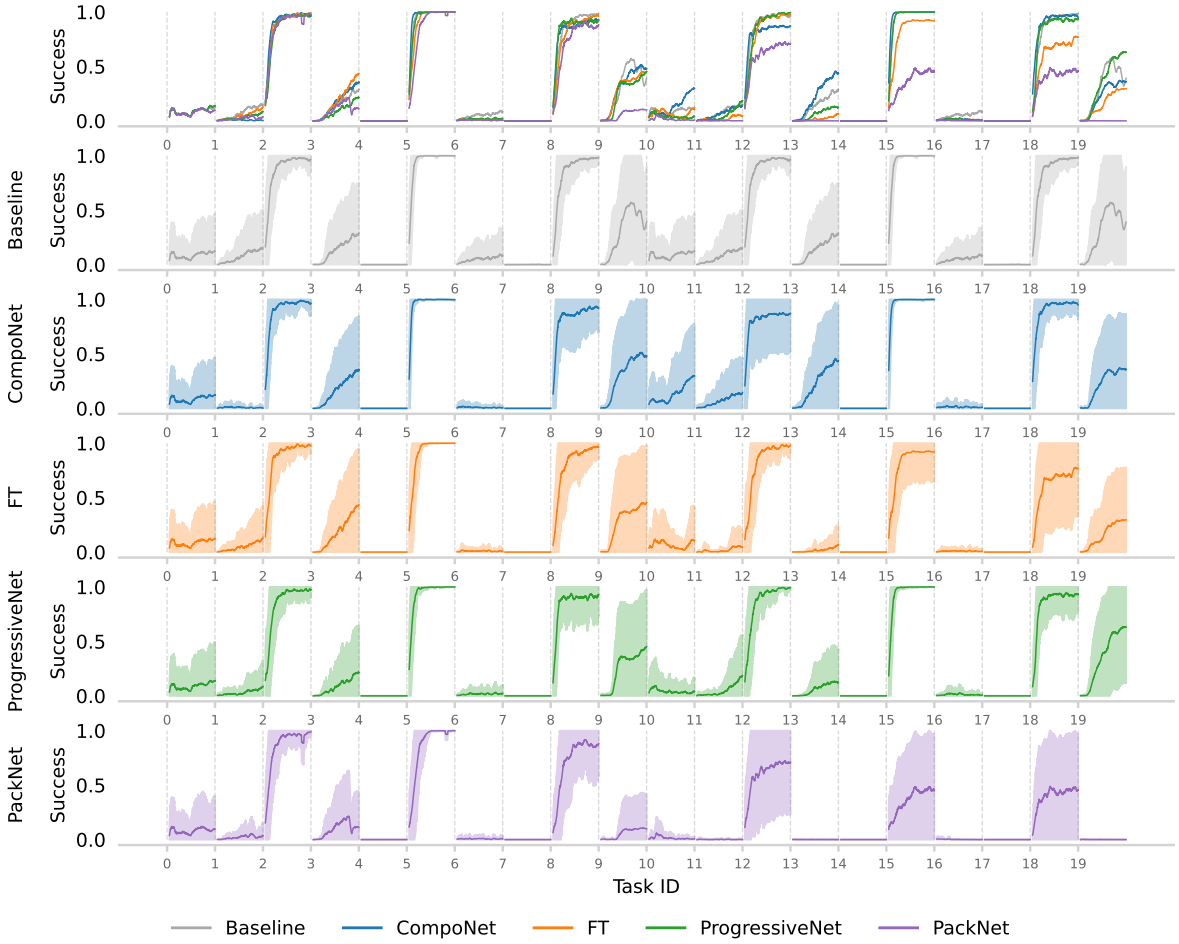


Figure F.1: Success curves of all methods in the Meta-World sequence. Results from 10 seeds are aggregated.

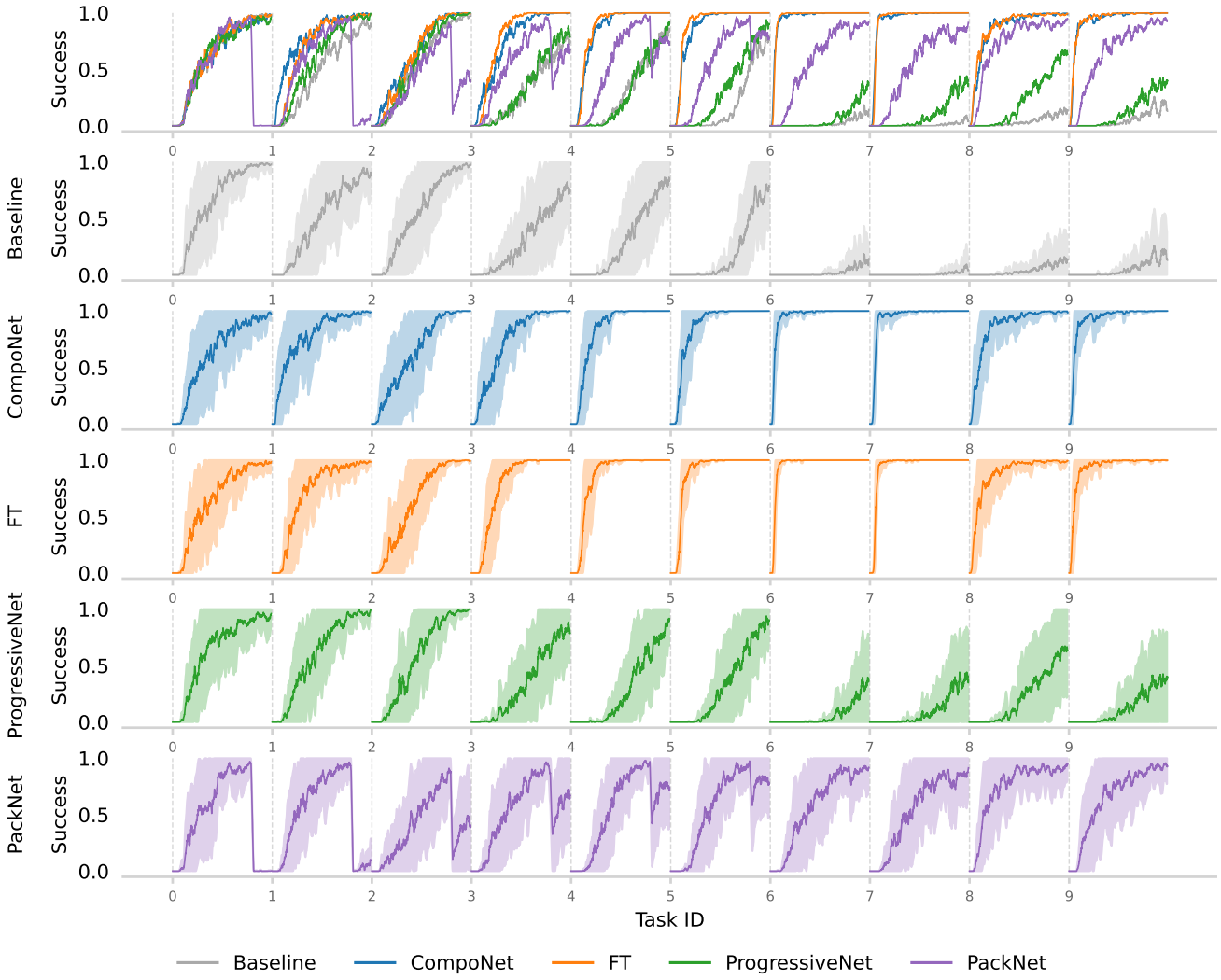


Figure F.2: Success curves of the different methods in the SpaceInvaders sequence. Results from 10 seeds are aggregated.

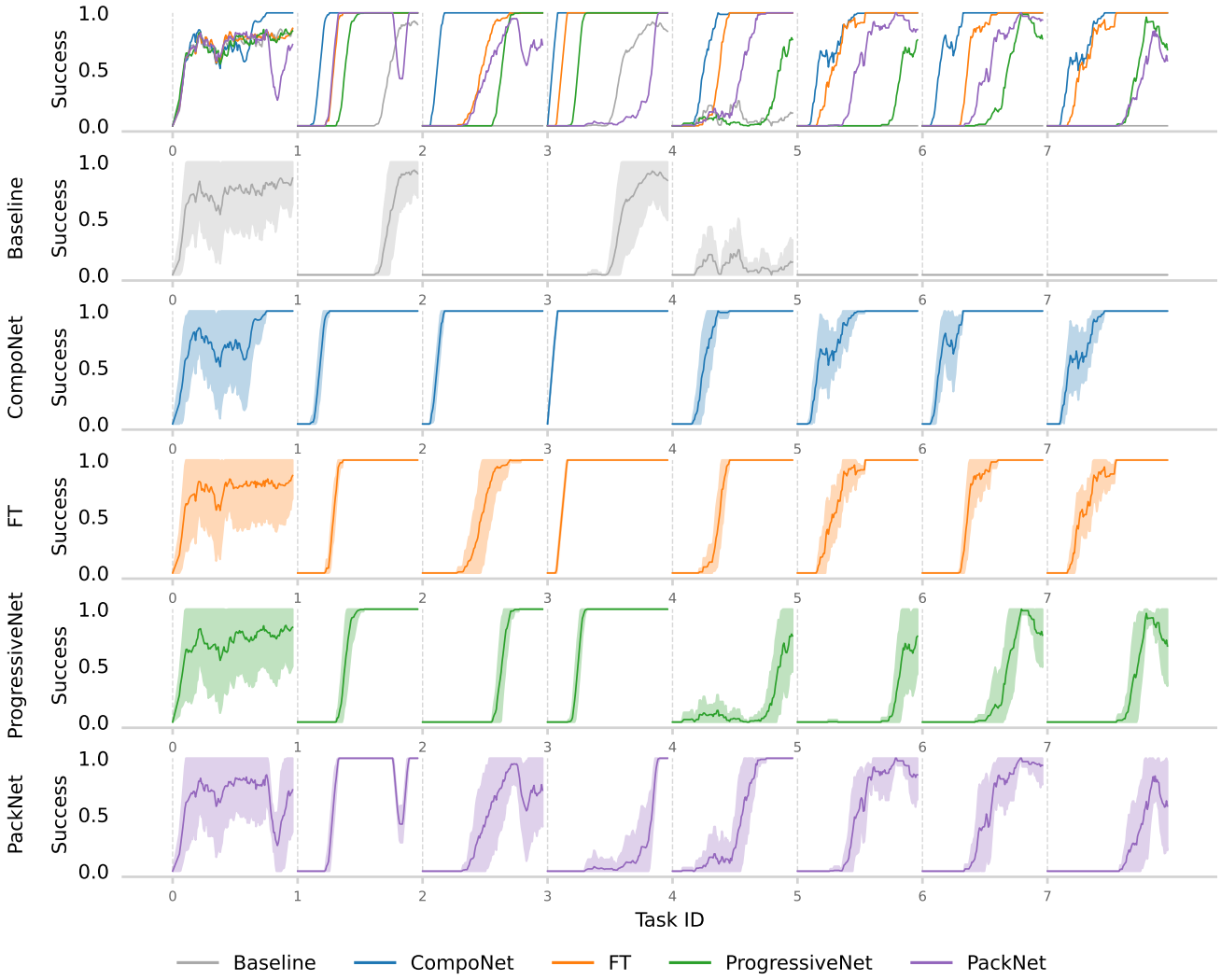


Figure F.3: Success curves of the different methods in the Freeway sequence. Results from 10 seeds are aggregated.

F.2. Forgetting

In addition to the metrics described in Section 5.1, forgetting is also commonly analyzed in the CRL literature. Following the notation from Section 5.1, we define the forgetting in the i-th task as,

$$\mathrm{F}_i = p_i(i \cdot \Delta) - p_i(T) \tag{7}$$

Intuitively, forgetting is the difference between the performance of the model at the end of a task, and the performance in the same task after the end of the whole sequence. Therefore, if a model has successfully retained the knowledge obtained in a task, it should achieve near-zero forgetting values. Contrarily, positive forgetting values correspond to scenarios where the model forgets previously learned information. Moreover, when forgetting is negative, the model has been able to acquire information in later tasks that help to solve the previous one. This phenomenon is called *backward knowledge transfer*.

Under the settings assumed in this work (task boundaries and identifiers are known to the agents, see Section 3), the only methods that can theoretically achieve backward knowledge transfer are FT-1 and the baseline, which precisely suffer from positive forgetting values as shown in the following lines. The rest of the methods store the parameters learned in each task, effectively avoiding forgetting (see Appendix E.2 for further details). Forgetting values for the affected methods are given in Table F.1, aggregating the results of 10 random seeds per method in every task. As shown, both methods suffer from considerable forgetting, and no backward transfer case is present.

Table F.1: Forgetting of the baseline and FT-1 methods in all of the considered task sequences. Note that the rest of the methods are omitted as their forgetting is zero under the assumptions from Section 3. Results from 10 random seeds are aggregated.

(a) Meta-World

Method	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Avg.
Baseline	$0.02_{\pm 0.30}$	$0.15_{\pm 0.00}$	$0.89_{\pm 0.27}$	$0.29_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.97_{\pm 0.17}$	$0.10_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.98_{\pm 0.00}$	$0.01_{\pm 0.49}$	$0.34_{\pm 0.46}$
FT-1	$0.12_{\pm 0.00}$	$0.19_{\pm 0.00}$	$0.99_{\pm 0.00}$	$0.45_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.95_{\pm 0.22}$	$0.02_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.88_{\pm 0.10}$	$0.17_{\pm 0.45}$	$0.38_{\pm 0.42}$

(b) SpaceInvaders

Method	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Avg.
Baseline	$0.66_{\pm 0.47}$	$0.85_{\pm 0.34}$	$0.81_{\pm 0.39}$	$0.85_{\pm 0.36}$	$0.99_{\pm 0.10}$	$0.99_{\pm 0.10}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.91_{\pm 0.26}$	$0.74_{\pm 0.44}$	$0.88_{\pm 0.32}$
FT-1	$0.58_{\pm 0.49}$	$0.57_{\pm 0.49}$	$0.19_{\pm 0.39}$	$0.32_{\pm 0.47}$	$0.56_{\pm 0.50}$	$0.61_{\pm 0.49}$	$0.60_{\pm 0.49}$	$0.67_{\pm 0.47}$	$0.67_{\pm 0.47}$	$0.70_{\pm 0.46}$	$0.36_{\pm 0.48}$

(c) Freeway

Method	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Avg.
Baseline	$0.02_{\pm 0.44}$	$0.87_{\pm 0.00}$	$0.49_{\pm 0.35}$	$1.00_{\pm 0.00}$	$0.51_{\pm 0.44}$	$0.80_{\pm 0.24}$	$0.70_{\pm 0.24}$	$0.60_{\pm 0.43}$	$0.62_{\pm 0.42}$
FT-1	$0.49_{\pm 0.45}$	$0.75_{\pm 0.33}$	$0.62_{\pm 0.11}$	$0.88_{\pm 0.33}$	$0.54_{\pm 0.42}$	$0.69_{\pm 0.37}$	$0.62_{\pm 0.34}$	$0.72_{\pm 0.33}$	$0.58_{\pm 0.43}$

G. Ablation Study

In Section 4 CompoNet is validated on several edge cases, showing that it fulfills the expected behavior in the scenarios that guided its development (introduced at the beginning of Section 4). This appendix aims to provide additional evidence on the fact that the three blocks that build the self-composing policy module (see Figure 2) cooperate to achieve the expected behavior.

G.1. Output Attention Head

This appendix analyzes the contribution of the output attention head to the overall performance and forward transfer abilities showed by CompoNet in the experiments in Section 5.

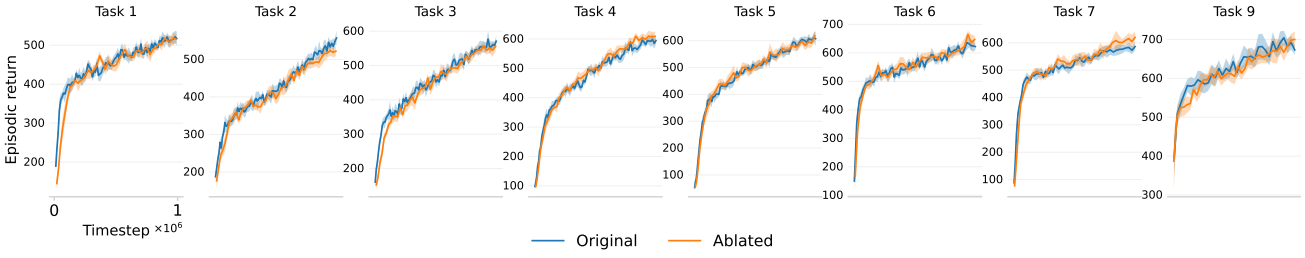
Results are shown in Figure G.1, where the performance of the original CompoNet design and an ablated version without the output attention head are compared in the SpaceInvaders and Freeway sequences, Figures G.1a and G.1b respectively. The performance of the original and the ablated versions do not considerably differ in the SpaceInvaders sequence, except for slightly better results for the original CompoNet design in the first timesteps of some tasks, see Figure G.1a. These results greatly contrast with the ones obtained in the Freeway sequence (see Figure G.1b), where the ablated version shows considerably lower performance in every task except the first.

Due to the design of CompoNet (see Figure 2 and Section 4.2), the output attention head should be of special interest in tasks where reusing previous modules is the best-performing strategy. In this sense, we believe that results from Figure G.1 are mainly affected by the fact that the Freeway environment is a scenario highly sparse reward (see the description of the sequence in Appendix D.3). This fact makes the reuse of previously learned policies a strategy of special interest. This hypothesis is supported by the low performance obtained by the baseline in a great part of the tasks depicted in Figure F.3.

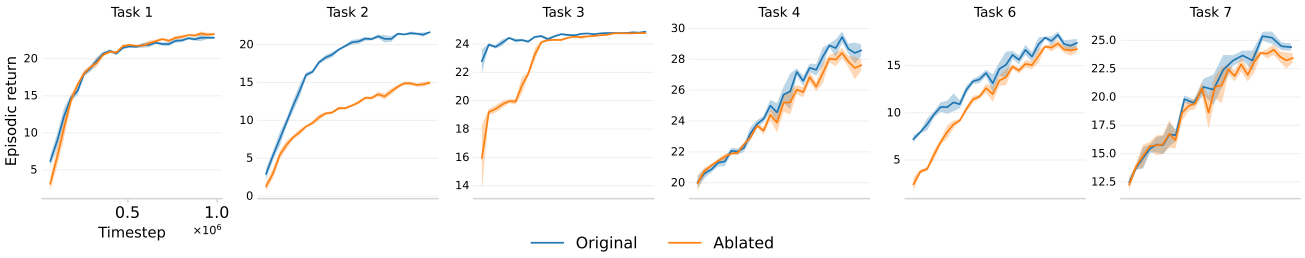
G.2. Input Attention Head

In this appendix, we evaluate the influence of the input attention head on the performance and forward transfer capabilities exhibited by CompoNet in the experiments presented in Section 5.

For this purpose, we have designed an experimental setting where CompoNet should employ the input attention head to gather information to solve the current task, while this information can not be directly used to solve the task via imitation using the output attention head. Specifically, we have trained a CompoNet agent in the fifth task of the SpaceInvaders sequence, where the module operating in the current task has access to five previous modules: four non-informative (sample



(a) SpaceInvaders



(b) Freeway

Figure G.1: Episodic return curves for the SpaceInvaders and Freeway sequences. Blue lines correspond to the original CompoNet architecture presented in Section 4, while orange lines represent the same architecture with no output attention head (referred to as *Ablated*). Each curve is computed from the average result of 5 seeds and contours indicate standard error.

their output from a uniform Dirichlet distribution), and one trained to solve the current task. This setting is equivalent to the one presented in the leftmost plots of Figure 4, although in this case the probability vectors outputted by all previous modules have been shifted one element to the left.[18] This implies that the probability originally assigned to each action is assigned to another, nullifying the strategy of directly using previously learned policies as the output of the current module.

Results are presented in Figure G.2, where the module trained to solve the current task is named *Inf. Mod.* (marked with stars). The figure aggregates the results of 10 random seeds for the baseline, the original CompoNet architecture, and an ablated version of CompoNet that has no input attention head.[19] Observing the episodic return curves in Figure G.2.i, we can see that the original CompoNet architecture successfully employs the information at hand to reach substantially higher episodic returns faster. Moreover, in Figure G.2.iii, we see that CompoNet attends to the module with information on the current task (*Inf. Mod.*) and to the output attention head, which in turn only attends to *Inf. Mod.* (see Figure G.2.iv). Finally, although in both versions of CompoNet, the output attention head learns to attend *Inf. Mod.*, the final output of the model is defined by the internal policy (see Figure G.2.ii), especially in the case of the non-ablated version, where the final output of the model completely matches the output of the internal policy.

In summary, the ablated version of CompoNet is capable of learning a policy from scratch with minimal interference, as shown by the similar episodic return curve to the baseline. However, leveraging the input attention head, CompoNet successfully extracts information from previous policies to substantially enhance the training in the current task.

H. Performance of the attention heads in extremely long task sequences

In this appendix, we focus on another aspect of scalability when dealing with extremely long task sequences beyond the computational costs (already analyzed in detail in Appendixes C and B). Specifically, we analyze the ability of the proposed method to efficiently capture information from previous modules when the number of these is extremely large. For this purpose, we incorporate a previous module trained to solve the current task and multiple non-informative ones that sample

[18] By shifting one element to the left we mean that each output vector $\mathbf{v} = (v_1, v_2, \ldots, v_{n-1}, v_n)$ is modified as $\mathbf{v}' = (v_2, v_3, \ldots, v_n, v_1)$.

[19] $\mathbf{h_s}$ is the only input of the internal policy in the ablated CompoNet model.

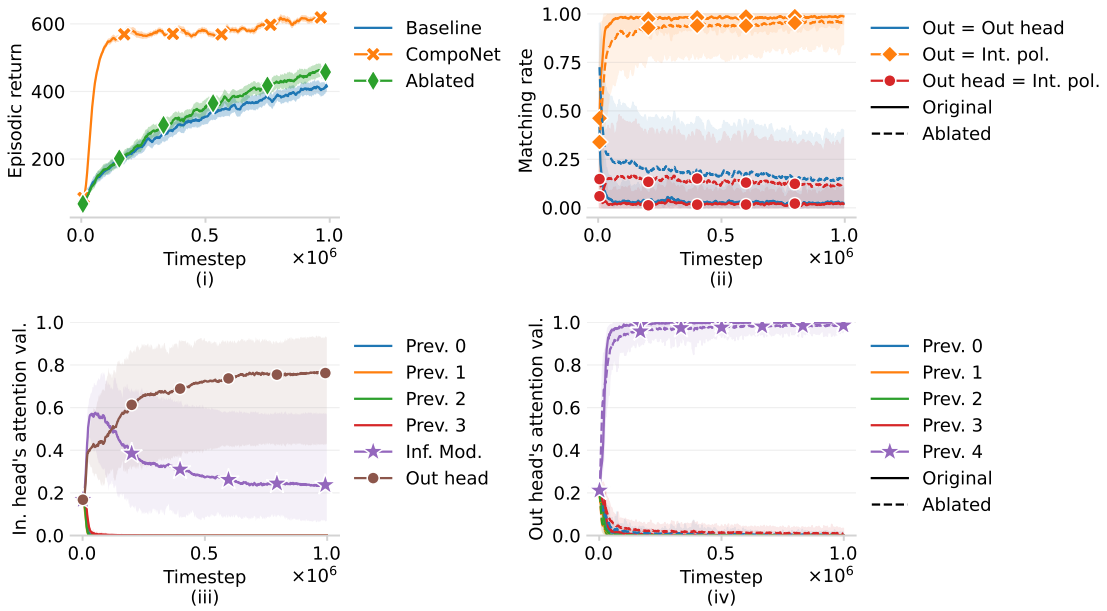


Figure G.2: Comparison of the original CompoNet design, an ablated version of CompoNet without the input attention head, and the baseline. The methods have been trained to solve the fifth task of the SpaceInvaders sequence using the hyperparameters from Appendix E, where the results of 10 seeds have been aggregated. In the case of the CompoNet models, five previous modules have been provided: four non-informative, and one trained to solve the current task (called *Inf. Mod.* and marked with stars). However, the output vectors of these modules have been shifted one element to the left, making the direct usage of *Inf. Mod.* an unviable approach although still maintaining useful information to solve the current task. In (i) the performance of the methods is compared in terms of episodic return; (ii) depicts the rate in which the outputs of different parts of the models match; finally, (iii) and (iv) respectively illustrate the evolution of the attention values for the input and output heads of the module being trained. Results show that the original version of CompoNet successfully employs the input attention head to extract information for the internal policy to solve the task at hand, greatly outperforming the ablated version and the agent trained from scratch.

their outputs from a uniform Dirichlet distribution. This setting resembles the experiment presented in the first part of Section 5.4 and is illustrated in the leftmost plots of Figure 4, but in this case, the number of non-informative previous modules is increased and is substantially larger.

Results are shown in Figure H.1. The first observation is that the episodic return curves of the CompoNet agents grow faster and to greater values than the agent trained from scratch to solve the task at hand (referred to as the baseline method). This demonstrates that the attention heads capture relevant information from the informative previous module despite the much larger number of non-informative ones, showing the robustness of CompoNet to a large number of modules.

Moreover, when comparing the curves corresponding to CompoNet, we observe that there is no clear correspondence with the number of non-informative previous modules and performance. For instance, the curve corresponding to 511 non-informative previous modules is almost consistently above the one regarding 99 of such modules. This suggests that the number of non-informative modules could be further increased without a substantial loss in performance. Note that, despite these results, we expect the number of non-informative previous modules will affect the performance of CompoNet when the number of previous modules is increased by orders of magnitude to the ones tested in this experiment. However, note that the 512 previous modules tested in this experiment would correspond to a sequence of 512 previous tasks, which is much larger than the usual sequences in the literature, where 20 tasks are considered rather a large sequence, and 100 tasks have been considered at most to the best of our knowledge (Wolczyk et al., 2021).

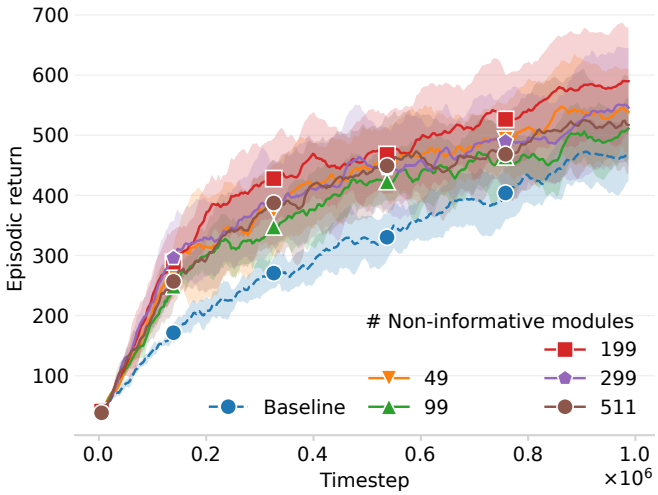


Figure H.1: Scalability of the attention heads to retrieve useful information from previous modules in extremely long task sequences. The figure compares the episodic return curve of the baseline method (trained from scratch) to CompoNet with an increasing number of non-informative previous modules and a single module that is trained to solve the current task. Specifically, the task at hand is the sixth task of the SpaceInvaders sequence. Lines and contours respectively represent the mean and standard deviation of 5 runs with different random seeds. Note that the total number of previous modules for each curve is the number of non-informative modules, shown in the legend, plus one, corresponding to the informative (i.e., trained) module.