Continual Robot Learning: Benchmarks and Modular Methods

Sam Powers

CMU-RI-TR-23-74

August 2023

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee

Abhinav Gupta	Carnegie Mellon University (*chair*)
Chris Atkeson	Carnegie Mellon University
Shubham Tulsiani	Carnegie Mellon University
Chris Paxton	Meta AI Research

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

Abstract

Humans adapt continuously to the world around us, allowing us to acquire new skills and explore diverse environments seamlessly. Current AI methods, however, cannot attain this versatility. Instead, they are typically trained with vast datasets, and learn all tasks simultaneously. However, the trained models have limited ability to adapt to changing contexts, and are limited by available data. This challenge is particularly pronounced in robotics, where real world interaction data is scarce.

Instead, we envision a robot capable of continuously learning from both the environment and human interactions, quickly acquiring new information without overwriting past knowledge, and capable of adapting to a user's specific needs.

In this thesis, we apply continual learning to robotics, with the goal of enabling crucial capabilities, including: the ability to apply prior information to new settings, maintain old information, sustain capacity for new skills, and understand context. We explore these across two learning modes: continual reinforcement learning (CRL), where the agent learns from experience, and continual imitation learning (CIL), where it learns from demonstrations.

However, substantial barriers hinder progress, including limited open-source resources, resource-intensive benchmarks, and impractical metrics for robotics. To address these challenges, we present CORA (COntinual Reinforcement Learning Agents), an open-source toolkit with benchmarks, baselines, and metrics to enhance CRL accessibility. CORA extends beyond catastrophic forgetting, evaluating models for forward transfer and generalization.

With this foundation, we introduce SANE (Self-Activating Neural Ensembles) to create a dynamic library of adaptable skills. SANE's ensemble of independent modules learns and applies skills as needed, reducing forgetting. We demonstrate this method on several Procgen reinforcement learning task sets.

We then adapt SANE to a physical robot, the Stretch, with SANER (SANE for Robotics) using CIL. Leveraging our novel Attention-Based Interaction Policies (ABIP), SANER excels in few-shot learning, showcasing its effectiveness at generalization across various tasks.

SANERv2 further advances this capability, integrating natural language and achieving strong performance over a diverse set of 15 manipulation tasks in a simulated environment, RLBench. Remarkably, SANERv2 was also able to display the potential of independent modules, demonstrating that a node could be moved between agents without loss of performance, promising possible future composable ensembles.

Acknowledgments

First and foremost, I extend my earnest gratitude to my advisor, Abhinav Gupta, whose guidance, support, and patience have been invaluable. Thank you for enabling me to explore my offbeat research interests and encouraging me to take risks. I'm also sincerely indebted to Chris Paxton for his help in applying my research to real robots and for offering guidance regardless of the domain: algorithms, hardware, and communication alike. I also wish to express my appreciation to the other members of my committee, Chris Atkeson and Shubham Tulsiani, whose feedback and insight helped shape the direction of my research.

My sincere appreciation also goes to Eliot Xing for his myriad contributions to this work, from ideation to code review to scaling up the experiments. This work would not be what it is without you. I owe a debt of gratitude as well to my labmates, especially Sudeep Dasari, Victoria Dean, Helen Jiang, Adithya Murali, Lerrel Pinto, Senthil Purushwalkam, and Abhinav Shrivastava for their assistance with compute, robots, and maintaining my sanity.

I'm immensely thankful for the mentorship and collaboration of those I had the privilege of working with, including Tim Rocktäschel, Mikayel Samvelyan, Priyam Parashar, Jay Vakil, Eric Kolve, and Roozbeh Mottaghi. My gratitude extends to all the colleagues with whom I've exchanged ideas, received feedback, or shared a dinner break during a tight deadline. This PhD is the culmination of all those moments, as much as it is of the work itself.

Above all, I'd like to thank my cat, Loki, for her (mostly) unconditional support and for being my steadfast companion. And, of course, my deep gratitude goes to my friends and family for their unwavering support, which means the world to me.

This work was supported by the CMU Presidential Fellowship, ONR MURI, ONR Young Investigator Program, DARPA MCS, and Meta AI.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

From birth, humans receive a continuous stream of information from the world around us. Remarkably, we are capable of responding reasonably in nearly every circumstance, even in the presence of significant novelty. Consider, for instance, visiting a foreign country for the first time: despite potentially stark differences, we can still accomplish tasks like locating food or purchasing forgotten items. Even in unfamiliar environments, we are able to recognize patterns that enable us to leverage existing skills.

Critically, however, initial attempts at solving a problem are unlikely to be optimal; learning and improvement will instead occur over time, in a variety of ways. We might make mistakes and learn from them, or observe how others do things, or consult a friend.

Similarly, a robot operating in the real world will encounter a continuous stream of new situations it must be able to respond to and learn from. Being able to adapt to new information presented over time is referred to as **continual learning (CL)**, in contrast with multi-task learning [33] where many tasks are learned, but they are specified up-front and dynamic adaptation does not occur. This work envisions a robot that can adapt as humans do, pursuing the development of a robot capable of continual learning.

While the ultimate goal is to train a single agent that can learn in a variety of ways, this work makes the problem more tractable by focusing on 1) **continual reinforcement learning (CRL)**, where skills are learned by trial and error, and 2) **continual imitation learning (CIL)**, where demonstrations of skills are provided. These modes of learning are treated separately, though we discuss in Chapter 7 how they can be combined into a dynamic learning system, and

extended to other forms of learning (e.g. instruction).

In the remainder of this chapter, I will discuss the specific challenges addressed in this work, then discuss how the rest of the thesis addresses them.

1.2 Goals of the Work

Improved CL Capability. In this thesis, we say that a model is an effective continual learner if it is capable of learning skills or representations that are:

1. **Broadly useful (Transfer/Generalization)** Skills should be effectively leveraged in new situations, or enable new, related skills to be learned more quickly.

2. **Learnable at any Time (-Intransigence)** [25] Prior continual learning models, e.g. [86], struggled with this, becoming less capable of learning new skills over time.

3. **Maintained over time (-Forgetting)** [112, 143] This was the initial challenge that created the field of continual learning; it was observed that when a neural net was presented with a new task, catastrophic forgetting of its previous skills would occur.

A continual learning model that can achieve all of these will not only be capable of learning new skills, but be able to do so increasingly effectively over time, as more skills are learned on which new ones can build.

Additionally, I specify the following desirable abilities for a model, that would considerably broaden the scope over which a model could be used:

1. **Leverage rich context**, e.g. language. Language provides a natural way to specify goals, and can potentially provide information that helps the model with skill transfer. For example, "put the fruit in the bowl" and "put the carrot in the box" might contain information that indicate they can leverage similar skills, even if they are not visually similar. Other rich contexts, e.g. images, can provide transferable information as well.

2. **Share information** amongst models easily, without disrupting existing information. This ability would ideally reduce data requirements for individual robots to learn new skills.

Through the course of this work, we will demonstrate improvement on all of these desired capabilities, via the development of a novel ensemble method, SANE.

Accessibility. Unfortunately, continual learning has a number of significant barriers to progress:

1. **Missing code.** Research code is not publicly available and reproducibility remains difficult. This creates a high barrier to entry as any new entrant must re-implement and tune baselines, in addition to designing their own algorithm.

2. **Benchmarking gaps.** There are few benchmarks and metrics used for evaluation, with no set standards.

3. **High resource barrier.** Existing baselines, benchmarks, and metrics require extensive compute resources, limiting their accessibility. Since continual learning scales all costs (time, compute, expense) by the number of tasks, efficiency is particularly critical.

These issues have made continual policy learning broadly inaccessible. This thesis attempts to mitigate them by providing new benchmarks, efficient metrics, and open source baselines.

1.3 Outline

Figure 1.1: An example image from each of the settings used in this work. Each represents a different suite of tasks; a total of 72 training tasks were used.

Chap.	Method(s)	Mode	Seq	# Tasks		Metrics				
					context	Gen?	$\mathcal{Z}$	$\Delta\mathbf{R}$	$-\mathcal{F}$	$-\mathcal{I}$
3	CLEAR, EWC, onl-EWC, P&C [86, 150, 160]	CRL	Atari [16]	6	implicit	✗	✓	✗	✓	✗
			Procgen [31]	6	implicit	✓	✓	✗	✓	✗
			MiniHack [156]	15	implicit	✓	✓	✗	✓	✗
			CHORES x4 (ALFRED) [168]	3x4	image	✓	✓	✗	✓	✗
4	SANE	CRL	Progcen x3	{4, 4, 5}	implicit	✗	✗	✗	✓	✗
5	SANER	CIL	Stretch [81]	4	point cloud	✓	✓	✓	✓	✓
6	SANERv2	CIL	RB15 (RLBench [70]	15(+1)	point cloud	✓	✓	✓	✓	✓

Table 1.1: An at-a-glance overview of the work, summarizing what method(s) the chapter focuses on, what learning mode it represents (CIL/CRL), task sequence description, and for each task sequence: how many tasks it included, whether it evaluated generalization, and what metrics were measured.

In the first part of this work, Chapters 2 & 3, I describe the progress made on overcoming the barriers to progress, making continual learning more accessible, with particular application to CRL. In the second part, Chapters 4- 6, I leverage the improved accessibility to develop a novel continual learning method: SANE, and its improved variants SANER & SANERv2.

A summary of the experiments presented in this work is provided in Table 1.1.

1.3.1 Barriers

Metrics. In Chapter 2, I formalize an overall framework for continual policy learning agents, based upon work done as part of both CORA and SANER, and use this to define a set of metrics that is feasible to apply to real robots. In particular, these new metrics are only $\mathcal{O}(n)$ in the number of tasks, whereas the prior metrics were $\mathcal{O}(n^2)$. Additionally, the new metrics enable researchers

to add more tasks dynamically. In the prior metrics, they would need to re-run from the beginning.

Benchmarks, Baselines. In Chapter 3 we present a set of four reinforcement learning benchmarks based upon the following environments, Atari [115], Procgen [31], MiniHack [156], and ALFRED [168]. We additionally implemented the baselines presented in Chapter 2.6, and evaluated them across all of the benchmarks.

Between these, our aim was to select benchmarks that were less computationally intensive than than Atari, provide open source implementations of several key continual learning methods, and evaluate

Sample Efficiency. Efficiency is particularly applicable to the robotics setting, as collecting demonstrations is time-consuming. A novel architecture, Attention-Based Interaction Policies (ABIP) is presented as part of Chapter 5. This model is capable of learning robust policies from only two demonstrations, making a larger number of tasks feasible.

1.3.2 Approach: SANE

Having provided a foundation for approaching continual learning in robotics, the next step is to attempt a solution. I envision a home robot that learns an ever-growing library of skills. These skills can then be expanded, built upon, re-used, and even shared between robots in the future. They should be automatically created and utilized, dynamically adapting over time as situations arise.

Towards this goal, this work presents SANE, Self-Activating Neural Ensembles. The core concept is the creation of an ensemble of independent, "Self-Activating", modules. Every module is responsible for two things: knowing a skill, and knowing when to use it. Their independence not only allows for the potential of a skill library, it also side-steps catastrophic forgetting, while still enabling forward transfer.

The basic method for SANE is presented in Chapter 4; its ability to mitigate forgetting in the reinforcement learning setting is demonstrated on three task sequences. In Chapter 5, it is extended to the imitation learning setting and applied to a real robot, demonstrating the capacity of the method for efficient generalization; I refer to this version as SANER (SANE for Robotics). Finally, in Chapter 6, SANER is applied to a longer sequence of tasks (RB15) that utilizes language; this variant is referred to as SANERv2. This setting allows us to evaluate several capabilities: 1) skill transfer, 2) plasticity, 3) language-based contextualization, and 4) composing modules trained separately.

Chapter 2

Background

In this chapter, we formalize the concepts common to both continual reinforcement learning and continual imitation learning, and develop them into a set of metrics that can be effectively applied in both domains. We then discuss each of CRL and CIL, with more details on how the general definitions are applied to each. Finally, we give a brief overview of the baselines used throughout this work.

2.1 Definitions

Task. Formally, we consider each task $\mathcal{T}$ as a finite, discrete-time Markov decision process (MDP), represented by the tuple $\langle S, A, T, r, \rho_0 \rangle$, with state space S, action space A, state transition probability function T, reward function r (optional), and probability distribution ρ_0 on the initial states $S_0 \subset S$. The goal is to learn a policy, $\pi(a|s)$, that predicts an action, $a \in A$, given a state $s \in S$ [170].

Episode. A single complete execution of the policy for a particular task, which occurs when s is a terminal state, is referred to as an episode; the trajectory of (s, a) pairs produced is denoted τ. A score is provided per-episode to evaluate the performance of the policy; we refer to this as the **episode return**, by analogy to its usage in reinforcement learning. The return is indicated with $\mathbf{R}_\pi$.This will be discussed more specifically in Sections 2.4 & 2.5.

It can be convenient to refer to an optimal policy, which we indicate with π^*. This refers to a policy that always takes the correct action to maximize the $\mathbf{R}$ it will obtain.

Context. A state s can be decomposed as (c, s'),where $s' \in S'$ is the underlying

state and $c \in C$ is the context, such that $S = C \times S'$. We refer the reader to [52, 85, 203] for further discussion of Contextual MDPs (CMDPs) [58].

The context c is information that remains fixed throughout an episode, and provides context about the goals of the task. The context can be implicit in the representation, as in a game of Pong, where an image-based s is sufficient to understand the goal. It can be automatically captured from the observation and maintained, as with a home robot that is cleaning a table: the context might be the initial dirty table, allowing the agent to maintain focus even if the table is not observable at a particular timestep. It can also be a language specification, provided by the user, that indicates what is desired.

Distribution Shift. The initial contextual state distribution may be factorized as $\rho_0(s) := p(c)\rho_0(s'|c)$, where $p(c)$, the context distribution, can be used to define collections of environments. These collections generally share a desired behavior, but vary in dimensions we wish to evaluate generalization over. Formally, we consider context sets C_{train} and C_{test}, where the policy is trained on training task set $\mathcal{T}_{C_{train}}$ and evaluated on the testing task set $\mathcal{T}_{C_{test}}$.

2.2 Continual Learning: Definition

While the true test of a continual learning robot is for it to learn from the natural interactions that occur while it inhabits a home, this is, for the moment, infeasible.

Technically, the only strict requirement is that the environment can provide a $\mathbf{R}$ of some nature that provides a signal for agent performance. Additionally, if the environment can't be run multiple times in almost a repeatable configuration, Intransigence becomes infeasible.

However, the goal when defining a less-natural continual setup should ideally be to adequately exercise the intended capabilities of an algorithm, as described in Section 2.3. In particular, the setup should ideally possess the following attributes:

1. **Optimal policy shift.** The optimal policy, π', changes over time. Historically [86], this has usually co-occurred with domain shift, e.g. switching video games, but this is not necessary. Policy shift without domain shift occurs when doing the a different task in the same environment (e.g. tasks in the same kitchen). This tends to elicits catastrophic forgetting, as it drives the result of the neural net to new values.

2. **Task similarity.** Some tasks share similarities in the behavior desired, for example learning to grasp a cup may aid in grasping a pitcher. This

can take the form of curriculum learning, where later skills build on earlier ones. This is useful for evaluating forward transfer.

3. **Multiple variations.** The same task can be run in several variations, ideally procedurally if simulated. Unlike in the natural setting, where generalization is largely equivalent to performance, in the artificial setting certain variations can be held out, so we can observe how a model's ability to generalize varies.

4. **Numerous tasks.** The task sequence should be as long as is feasible for the setting, to evaluate intransigence.

5. **Dimensional consistency.** Tasks are drawn from some world collection $\mathcal{W}$, where the dimensions of the observation space and action space are consistent across all tasks from $\mathcal{W}$. This is desirable so that the same networks can be used for all tasks.

There are a variety of paradigms that meet these criteria; for example [86] defines a randomized (but fixed) schedule of 10 games, where games are seen concurrently, for varying durations, and re-visited at varying intervals. [205] defines a setting where the physics parameters underlying the tasks shift over time. However, we have opted for the following approach, as in [150, 160], which we see as a simple and effective way to induce a *non-stationary* learning process:

Task Sequence. We assume N tasks are presented as a sequence $\mathcal{S}_N := (\mathcal{T}_0 \dots \mathcal{T}_{N-1})$, and each task $\mathcal{T}_i$ is given a budget of B_i timesteps. The agent trains on task $\mathcal{T}_i$ at timesteps in the interval $[A_i, B_i)$, where A_i and B_i are the task boundaries denoting the start and end, respectively, of task $\mathcal{T}_i$. We cycle through the tasks M times, so the full task sequence $\mathcal{S}_{NM}$ has length $N \cdot M$.

Offline evaluation. While a continual agent operating in the real world would not pause for evaluation, offline evaluation is a useful tool enabled by simulation to understand agent performance. Offline evaluation helps answer questions such as: "How does the agent currently perform on tasks learned in the past?", "How does experience the agent has acquired help it learn new tasks?", or "How well would the agent generalize if it were asked to perform the task in a unseen environment?".

2.3 Metrics

In this section we first outline the specific metrics used, then provide definitions and discuss a few considerations one should make when utilizing them. This section attempts to unify the knowledge gained while doing the rest of the work presented. As such, some specifics of the metrics used vary in future sections from what is presented here, and differences will be noted as such.

2.3.1 Overview

We formalize the goals described in Chapter 1.2 into the following metrics:

1. **Final performance ($\mathbf{R}_{final}$: Eqn 2.1)** The robot's ability to perform all tasks at the end of training.

2. **Zero-Shot Forward transfer ($\mathcal{Z}_C$: Eqn 2.4)** The agent's ability to leverage previous skills when learning a new one, before it has seen any training data for the new task [105].

3. **Performance Improvement ($\Delta\mathbf{R}$: Eqn 2.5)** How much the agent learns on a given task, while training on it.

4. **Forgetting ($-\mathcal{F}_C$: Eqn 2.7)** Historically the original goal of continual learning [112, 143], catastrophic forgetting occurs in neural networks when they are trained sequentially. We report this as $-F$ (see below).

5. **Intransigence ($-\mathcal{I}$: Eqn 2.8)** Some models struggle to learn new skills after many have already been learned, for example due to insufficient model capacity or algorithm plasticity. This difficulty is referred to as intransigence [25]. As with Forgetting, we report this as $-I$.

6. **Generalization ($\mathbf{R}_{gen}$: Eqn 2.2)** The robot should be able to generalize its abilities to unseen settings, for example variations of the environment.

7. **Continual Evaluation (Sec 2.3.3)** This metric evaluates performance on all tasks periodically during training[160]. We make a few significant alterations for more efficient use in the robotics setting.

We present Forgetting and Intransigence in their negated forms so that for all metrics, a larger positive value is preferred. We do not change the wording (e.g. "negative forgetting" could be "backwards transfer") as these terms are standard.

2.3.2 Setup

We define here several expected episode returns and associated terms that will be used when defining our metrics.

$$\mathbf{R}(i, K) = \frac{1}{s} \sum_{s} \mathbf{R}(s, i, K) \qquad \text{Expected return on task } \mathcal{T}_i \text{ at timestep } K$$

$$\mathbf{R}_*(i) = \frac{1}{s} \sum_{s} \mathbf{R}_*(s, i, K) \qquad \text{Expected return on } \mathcal{T}_i \text{ for policy } \pi* \text{ trained only on } \mathcal{T}_i$$

$$\mathbf{R}_{max}(i) := \max_{K \in [A_0, B_N)} \mathbf{R}(i, K) \qquad \text{Max return on } \mathcal{T}_i \text{ after all tasks, for the current run.}$$

$$\mathbf{R}_{all,max}(i) := \max_{\pi \in \mathcal{A}} \max_{K \in [A_0, B_N)} \mathbf{R}_\pi(i, K) \qquad \text{Max return on } \mathcal{T}_i, \text{ across all policies.}$$

Where s represents the number of separate runs (seeds), and $\mathcal{A}$ indicates the set of all agent policies, across all algorithms, including individually trained policies.

Normalization. It can be advantageous to normalize the returns prior to using them for computing certain metrics. Using the max for only the current episode ($\mathbf{R}_{max}(i)$) has the benefit that it gives the metrics as a ratio of what was achieved; for example, $ZSFT$ communicates, of the total amount of skill the agent achieved overall, how much of it was learned before the task was started?

This is useful for understanding an algorithm, but does not always produce easily intrepretable results when comparing against other agents. An agent that learns very little will have likely learned much of it before training started, making it, in that case, more a metric of the noise present than any true learning. By contrast, normalizing with ($\mathbf{R}_{all,max}(i)$), as in [25], can be seen as simply bringing a variety of tasks, with differing reward scales or levels of difficulty, into parity, to make aggregate metrics more sensible.

2.3.3 Continual Evaluation

Continual evaluation is the basic data collection strategy on which all other metrics presented are built, providing insight into the performance over time on all tasks, and is presented as a set of graphs, to observe how each task is impacted by training on the others.

Standard Approach. The standard approach collects data every k_{eval} timesteps while training. Training is paused, to ensure all evaluation hap-

pens on a consistent state, and all tasks are evaluated. Returns are estimated by averaging over n_e episodes.

Unseen Settings. Our first adaptation is to evaluate on environments drawn from C_{test}, which allows us to compare the generalization ability of various methods.

Optimization for Robotics. The standard method grows in the number of tasks as $\mathcal{O}(n^2)$, which is only compounded by performing evaluation frequently during each task. This volume of evaluation is infeasible for the robotics setting, where an individual episode can take on the order of minutes for even the simplest tasks.

We simplify our data collection for robotics by evaluating each task at only three points: immediately before task i is trained: $\mathbf{R}(T_i, A_i)$; immediately after the task is trained: $\mathbf{R}(T_i, B_i)$; and at the end of training: $\mathbf{R}(T_i, B_N)$. Our evaluation strategy now grows as $\mathcal{O}(n)$, which makes extending to longer sequences significantly more viable.

It also has the benefit that new tasks can be readily added; in the standard approach, if a new task is added, the experiment needs to be restarted from the beginning. With our method, adding additional tasks requires only one additional evaluation of the other tasks, at the new end. This enables training to be dynamic, for example by adding more challenging tasks if the agent is succeeding at easier ones, while still remaining quantitative.

We additionally train one agent on each task individually, π_*, which gives us a reference point, used in measuring intransigence.

2.3.4 Robust Representation Metrics

The term "forward transfer" has been used to refer to several separate metrics: 1) the zero-shot capabilities of a model [105], 2) how learning speed (measured as area under the learning curve) compares to a non-continual referent [186], and 3) how easily a model's representation can be used to learn a new task using an independent, "probe", network [27].

The first is effectively a form of evaluating generalization. While in this work we usually use "generalization" to refer to tasks where the agent is expected to execute a known policy in a different domain, and "forward transfer" to refer to tasks that require learning a new policy by leveraging prior experience, the grey area between them is substantial.

A clear distinction does not need to be drawn; both are, fundamentally, two forms of evaluating the same underlying capability of the robot: how robust are the representations it has learned, and how capable is it of utilizing them?

Additionally, intransigence can be seen as equivalent to the second definition, in the limit where only two data points for evaluation are collected. This metric, too, contributes to our understanding of the utility prior tasks provided for learning later ones.

Therefore, instead of seeing all three as strictly independent, we can see them instead as windows into the underlying capability of interest.

2.3.5 Final Performance & Generalization ($\mathbf{R}_{final}, \mathbf{R}_{gen}$)

Final performance is given by the return for each task, after training is completed on all tasks:

$$\mathbf{R}_{final}(i) = \frac{\mathbf{R}(i, B_N)}{|\mathbf{R}_{all,max}(i)|} \tag{2.1}$$

When evaluated on C_{test}, this is also our generalization metric:

$$\mathbf{R}_{gen}(i) = \frac{\mathbf{R}(i, B_N, C_{test})}{|\mathbf{R}_{all,max}(i, C_{test})|} \tag{2.2}$$

This metric can be thought of as aggregating several other factors being measured: $\mathbf{R}_{final,i} = Z + PI - F$.

Generalization can also be measure via $\mathbf{R}(C_{train}) - \mathbf{R}(C_{test})$, which

2.3.6 Zero-Shot Forward Transfer ($\mathcal{Z}$)

Isolated Zero-Shot Forward Transfer ($\mathcal{Z}_I$):

We begin with a partial simplification of the standard metric. In this formulation, Z compares the expected return achieved for later task $\mathcal{T}_i$ before and after training on earlier task $\mathcal{T}_j$, where $i >= j$:

$$\mathcal{Z}_I(i, j) = \frac{\mathbf{R}(i, B_j) - \mathbf{R}_{i,B_{j-1}}}{|\mathbf{R}_{max}(i)|} \tag{2.3}$$

When $\mathcal{Z}_{i,j} > 0$, the agent has become better at later task $\mathcal{T}_i$ having trained on earlier task $\mathcal{T}_j$, indicating forward transfer has occurred by *zero-shot* learning [105, 149]. When $\mathcal{Z}_{i,j} < 0$, the agent has become worse at task i, indicating negative transfer has occurred. We normalize by the absolute value of the maximum expected return observed for task $\mathcal{T}_i$ within the run.

Cumulative ZSFT ($\mathcal{Z}_c$):

We further simplify the metric, by accumulating the transfer metric across all prior tasks, rather than separating them out per-task. From this we obtain the cumulative version, that we use for robotics experiments:

$$\mathcal{Z}_c(i) = \frac{\mathbf{R}_{i,i-1} - \mathbf{R}_{i,A_0}}{|\mathbf{R}_{all,max}(i)|} \tag{2.4}$$

In this formulation, reward achieved at the very start of a task i, before any explicit training on it is done, is compared to the score obtained by the fully untrained agent. Additionally, for this formulation we scale by the maximum observed over all agents, as discussed in Section 2.3.2, to prioritize comparison between methods over algorithm analysis.

The former gives more insight, but scales as $\mathcal{O}(n^2)$, so for the domain of robotics we utilize the latter definition.

2.3.7 Performance Improvement (ΔR)

Performance improvement (PI) represents how effectively the agent learned the task due to training on it.

$$\Delta R(i) = \frac{\mathbf{R}(i,i) - \mathbf{R}(i,i-1)}{|\mathbf{R}_{all,max}(i)|} \tag{2.5}$$

This metric is not always straightforward to interpret; if an agent experiences significant ZSFT, there is simply less for the agent to learn and its PI will consequently be lower.

2.3.8 Forgetting ($-\mathcal{F}$)

Isolated Forgetting ($\mathcal{F}_I$):

Originally inspired by [105, 183], Isolated Forgetting represents how much is forgotten from a learned task during later tasks. It compares the expected return achieved for earlier task $\mathcal{T}_i$ before and after training on later task $\mathcal{T}_j$, where $i < j$:

$$-\mathcal{F}_I(i,j) = \frac{\mathbf{R}(i,B_j) - \mathbf{R}(i,B_{j-1}))}{|\mathbf{R}_{max}(i)|} \tag{2.6}$$

When $-\mathcal{F}_{i,j} < 0$, the agent has become worse at past task $\mathcal{T}_i$ while training on new task $\mathcal{T}_j$, indicating forgetting has occurred. Conversely, when $-\mathcal{F}_{i,j} > 0$,

the agent has become better at task $\mathcal{T}_i$, indicating *backward transfer* [105] has been observed. We normalize by the absolute value of the maximum expected return observed for task $\mathcal{T}_i$ within the run. Tasks can have varying reward scales, and normalization helps for comparing between tasks, without the added burden of your metric depending on others.

Cumulative Forgetting ($-\mathcal{F}_C$):
As with $\mathcal{Z}$, we introduce a cumulative version of F, in addition to the isolated version.

$$-\mathcal{F}_C(i) = \frac{\mathbf{R}_{i,N} - \mathbf{R}_{i,i}}{|\mathbf{R}_{all,max}(i)|} \tag{2.7}$$

This metric analyzes how much forgetting is induced cumulatively after the time when it was trained. As we express it as $-F_C$, it has the interpretation of backwards transfer, as $-F_I$ does.

2.3.9 Intransigence ($-\mathcal{I}$)

Intransigence is the relative inability to learn compared to a stand-alone model [25]. We present its negative:

$$-I(i) = \frac{\mathbf{R}(i, B_i) - \mathbf{R}_*(i, B_i)}{|\mathbf{R}_{all,max}(i)|} \tag{2.8}$$

which instead measures how much learning sequentially contributed to more effectively learning the task.

2.4 Continual Reinforcement Learning

The key defining feature of the reinforcement learning setting is the availability of a reward function during training. This indicates to the agent the utility of the actions it has taken, which is then used to train the policy.

For the RL setting, we define the return used for evaluation, $\mathbf{R}$, as the sum of (undiscounted) rewards over a finite-length episode: $\mathbf{R} = \sum_{i \in \tau} r_i$.

Artificial vs Natural. Continual reinforcement learning has adopted the paradigm of standard reinforcement learning [86]; namely, that many parallel environments are used in parallel for data collection. This has several benefits

that are not generally applicable to robotics: 1) the generation of a diversity of data, which helps to stabilize training, 2) the ability to leverage existing RL work, and 3) solving more challenging environments, as RL methods are notoriously sample-inefficient.

While this assumption is unlikely to hold when applied to robots, all three are significant advantages, allowing us to spend less time on a single task, and more time on the continual aspect of the problem.

2.5 Continual Imitation Learning for Robotics

As reinforcement learning is defined by the existence of rewards, so imitation learning is defined by the existence of demonstrations. The CIL setting is established by defining tasks via the collection of k_{demo} demonstrations, where each demonstration is given as a trajectory of state, action pairs: $(\langle s, a \rangle)$.

To allow us to assume the Markov property, we additionally augment s with context c, taken either as the state of the environment at the beginning of the episode (as in Chapter 5), or the language annotation provided (as in Chapter 6). Each task is trained on this dataset for T timesteps, using samples taken randomly from the demonstration trajectories. Evaluation is performed by executing our learned policy on the robot, in k_{unseen} settings.

For reinforcement learning, the sum of episode rewards is naturally utilized for $\mathbf{R}$. For imitation learning, however, the choice can be less clear. Simulated environments, like that of Chapter 6, are often designed to determine rewards for the actions the agent takes. For our experiments with the real robot, however, we scored each run manually, based on pre-specified conditions.

2.6 Baselines

The continual learning methods selected for our baselines were chosen to cover the categorizations described by [99, 118, 128]: regularization, architectural, and rehearsal:

- **CLEAR** [150] is a replay buffer method that utilizes a large buffer size and reservoir sampling, and consistently performs well across domains.
- **Elastic Weight Consolidation (EWC)** [86] is a regularization method that uses the diagonal of the Fisher matrix to estimate the importance of parameters for past tasks, and slows updates to those parameters when learning new tasks. EWC uses task IDs to determine when to save parameter information.

- **Online EWC** is a variant of EWC introduced by [160] which adapts it into a task ID-free method.
- **Progress & Compress (P&C)** [160] is a dynamic architecture approach that uses an online variant of EWC to consolidate learned behavior between dual networks, after each task is learned. This method also uses explicit task IDs.

Chapter 3

CORA: A Platform for Continual Reinforcement Learning Agents

3.1 Introduction

In this chapter, we aim to democratize the field of continual RL by reducing the barriers to entry and enable more research groups to develop algorithms for continual RL. To this end, we introduce CORA, a platform that includes benchmarks, baselines, and metrics for **Co**ntinual **R**einforcement Learning **A**gents.

In this chapter, we first present a set of benchmarks, each tailored to measure progress toward a different goal of continual learning. Our benchmarks include task sequences designed to: test generalization to unseen environment contexts (Procgen), evaluate scalability to the number of tasks being learned (MiniHack), and exercise scalability in realistic settings (CHORES), in addition to a standard, proven benchmark (Atari). Second, we release open-source implementations of previously proposed continual RL algorithms in a shared codebase, including CLEAR [150], a state-of-the-art method. We demonstrate that while CLEAR outperforms other baselines in Atari and Procgen, there is still significant room for methods to improve on our benchmarks.

3.2 Related Work

Evaluating continual reinforcement learning. While continual learning is most commonly addressed in the context of supervised learning for image classification such as in [62, 62, 72, 105, 108, 155], here we focus our discussion on continual reinforcement learning, and as such, simulation environments and

tasks to benchmark RL agents. For an overview on continual learning applied to neural networks in general, we refer the reader to [128] and [118].

With CORA, we introduce benchmarks designed to evaluate continual RL algorithms that can be used in more challenging, realistic scenarios. Continual RL for policies or robotic agents [99] is more nascent, although several benchmarks have been proposed. As mentioned above, continual RL has typically been evaluated on a sequence of Atari games [86, 150, 160] and we leverage these prior results to validate our baselines. Other video game-like environments proposed to evaluate continual learning include StarCraft [159] and VizDoom [103].

Procgen [31] and MiniHack [90, 156], two of our other benchmarks, are procedurally-generated, like Jelly Bean World [134] which is a procedurally-generated 2D gridworld proposed as a testbed for continual learning agents. Beyond video game environments, [186] evaluate continual RL using task boundaries in a multi-task robot manipulation environment, while [84] discuss home simulations as a potentially suitable environment to benchmark continual RL. In this work we present CHORES in AI2-THOR [87]: task sequences for an agent in a home simulation to evaluate continual RL methods in the visually realistic scenes offered.

Our work is conceptually similar to `bsuite` [123], which curates a collection of toy, diagnostic experiments to evaluate different capabilities of a standard, non-continual RL agent. Concurrent with our work, Sequoia [120] introduces a software framework with baselines, metrics, and evaluations aimed at unifying research in continual supervised learning and continual reinforcement learning. While both are valuable benchmarking tools, they focus predominantly on simpler tasks like MNIST and CartPole. The most complex environments that Sequoia uses are Meta-World, with simple state-based manipulation tasks, and MonsterKong, composed of 8 hand-designed platformer levels. In contrast to both, CORA presents challenging task sequences for vision-based, procedurally-generated environments that evaluate generalization and scability for continual RL. Concurrent with our work, Avalanche RL [106] also introduces a library for continual RL. However, Avalanche RL does not present any experimental results on baseline methods. In this paper, we evaluate several continual RL methods across four different environments.

Environments and tasks. Historically, RL agents were evaluated on simple control tasks with state-based inputs from the OpenAI Gym [18] or DeepMind Control Suite [174]. Some of these tasks have been shown to be easily solvable by random search algorithms [111] and thus should not be considered as sufficiently difficult for comparing algorithms. Leveraging physics simulators,

many environments have been proposed that involve robots, fixed in place, for object manipulation tasks of varying complexity [55, 70, 76, 97, 121, 133, 142, 193, 197, 208]. Learning policies for robot manipulation is challenging, compounded by the exploration difficulty of the task, continuous action spaces, and the sample inefficiency of RL algorithms.

In this work, we choose environments with discrete action spaces, in order to use a single output policy across multiple tasks, rather than use separate output heads for different tasks. We make this decision considering the capabilities of current methods. For environments with continuous action spaces, task-agnostic single-headed architectures may be infeasible currently, as in [186], so we leave this for future work. Furthermore, it is beneficial for continual RL for tasks to share a consistent observation space, allowing for the creation of common policies that can leverage task similarities. If environments rely on state-based inputs such as the positions of objects, it is usually the case that the state space changes for different tasks. To ensure a consistent observation space, we pick environments which offer image-based observation spaces.

RL is also frequently evaluated on video game-like environments, most commonly Atari [16], among others [14, 30, 31, 56, 73, 82, 90, 181].

In this work, we reproduce prior continual RL results on Atari [150, 160]. We also define new task sequences using Procgen [31] and MiniHack [156]. MiniHack is designed to isolate more tractable subproblems in the highly challenging NetHack [90] environment. We believe Procgen and MiniHack, with fast, procedurally-generated, stochastic environments to be better testbeds for continual RL onwards.

Beyond video game environments, many home environment simulators have been proposed recently [19, 49, 139, 157, 158, 195] which offer visually-realistic scenes for evaluating Embodied AI. Among these types of environments, we highlight AI2-THOR [87], Habitat 2.0 [172], iGibson [164], Sapien [190], and ThreeDWorld [47], which feature a wide range of household objects and scene-level interaction tasks. In particular, AI2-THOR, Habitat 2.0, and iGibson provide multiple home scenes based on real-world data. These different scenes are useful for applying realistic domain shift between tasks and evaluating forward transfer when learning later tasks. We choose AI2-THOR as a simulation environment in this benchmark because it offers a higher-level discrete action space, compared to Habitat 2.0 or iGibson at the time of development, along with a diverse set of demonstrations released in ALFRED [168].

We also note that recent work using AI2-THOR has done evaluations with object manipulation tasks [13, 38, 168], paving the way for more complex action spaces.

3.3 Task Sequences for Benchmarking CRL

The goal of continual reinforcement learning is to develop an agent that can learn a variety of different tasks in non-stationary settings. To this end, prior work has primarily focused on preventing catastrophic forgetting [86, 150, 160] and maintaining plasticity [114] so that the agent can learn new tasks. While simple tasks are useful for debugging, skill on them does not necessarily translate to more complex tasks. In this chapter, we begin to address more ambitious goals. In particular, we believe continual RL methods should address the following problems: (a) showing positive forward transfer by leveraging past experience; (b) generalizing to unseen environment contexts; (c) learning similar tasks through provided goal specifications; (d) improving sample efficiency; in addition to (e) mitigating catastrophic forgetting; and (f) maintaining plasticity (mitigating intransigence).

While a single benchmarking environment that suitably deals with each of these features may be ideal, over the course of development we have found this to be impractical with the tools currently available. For example, visually-realistic, physics-based environments are generally not fast enough for the longer sequences of tasks that we use to test resilience to forgetting. Furthermore, it may be overbearing for new algorithms to sufficiently address every continual RL goal, whereas a modular set of evaluations allows for researchers to focus on areas to best highlight particular contributions of their new methods.

Instead, we present four benchmarks which continual RL reseachers may utilize:

- Atari [16], 6 task sequence: A standard, proven benchmark used by [160] and [150], particularly to demonstrate resilience to catastrophic forgetting.
- Procgen [31], 6 task sequence: Designed to test resilience to forgetting and in-distribution generalization to unseen contexts in procedurally-generated, visually-distinct environments.
- MiniHack [90], 15 task sequence, based on NetHack [156]: Designed to train agents on a long sequence of tasks in environments that are stochastic, procedurally-generated, and visually-similar, in order to demonstrate resilience to forgetting, maintenance of plasticity, forward transfer, and out-of-distribution generalization (extrapolation along different environment factors).
- 4 different CHORES, utilizing ALFRED [168] and AI2-THOR [87]: Designed to test agents in a visually realistic domain where sample efficiency is key. Unlike other environments where different tasks may be easily identified visually, CHORES tasks explicitly provide a goal image. CHORES

also present an opportunity to test forward transfer due to task similarity. For example, the ability to pick up a hand towel ideally should transfer from one bathroom to another.

We direct the reader to Chapter 2 for formalism and background on the continual RL setting, including more precise definitions of generalization for these benchmarks and how continual RL applies to these task sequences.

Our goals include reducing the compute costs of continual RL experiments for the new benchmarks, as compared to the Atari experiments. Indeed, we observed a speedup of 7x for MiniHack, 6x for Procgen, and 2x for CHORES.

3.3.1 Atari tasks

Figure 3.1: Examples of initial observations for each task in the 6 task Atari sequence.

Building off the work of [86] that evaluated a random set of ten Atari [16] games, recent work in continual reinforcement learning [150, 160] evaluates

continual learning on six Atari games: [0-SpaceInvaders, 1-Krull, 2-BeamRider, 3-Hero, 4-StarGunner, 5-MsPacman], as visualized in Figure 3.1. They train agents on each of the six tasks for 50M frames, cycling through the sequence 5 times, for a total of 1500M frames seen. This results in 250M frames per task, which is five times as many frames as is standard in the single task setting. The primary focus of algorithms that were developed and evaluated on this Atari task sequence was to reduce catastrophic forgetting. This setting is particularly suitable for catastrophic forgetting due to the lack of overlap between tasks, in regards to both observations and skills required. More details are given in Section 3.5.1.

Following [86, 150, 160], we use the original Atari settings, meaning that the games are deterministic. Modifications such as "sticky actions" [107] have become more standard to overcome simulator determinism, and are an option for increasing task difficulty in future work. In this work, we use Atari to validate our baseline implementations, preferring procedurally-generated environments (Procgen, MiniHack) to test generalization. We note that different Atari game modes may also be used to produce variation and assess generalization capability, as proposed by [41].

3.3.2 Procgen tasks

We use Procgen [31] to define a new sequence of video game tasks, with the intention of replacing the Atari tasks used previously to evaluate continual learning methods. We chose Procgen because its procedural generation allows for evaluating generalization on unseen levels, unlike Atari. Like Atari however, the tasks are all visually distinct and the task sequence is well-suited to evaluating catastrophic forgetting. As with Atari, this is due to a general lack of overlap between tasks. From the full set of available Procgen environments, the specific set of tasks was chosen by [65] to ensure the existence of a nontrivial generalization gap and to ensure generalization actually improves during training.

Procgen is also significantly faster to run (our experiments take several days for Procgen vs. weeks on Atari). To improve sample efficiency and reduce compute costs, we use the easy distribution mode for these Procgen games. We use a sequence of six tasks [0-Climber, 1-Dodgeball, 2-Ninja, 3-Starpilot, 4-Bigfish, 5-Fruitbot], training for 5M frames on each task with 5 learning cycles. This results in 25M frames per task and 125M frames total. Note that we are not increasing the number of training frames per task compared to the original paper, unlike the Atari task sequence.

The observation space is (64, 64) RGB images and is not framestacked.

Figure 3.2: Examples of initial observations for each task in the 6 task Procgen sequence

The 15-dim action space is the same across Procgen tasks. As recommended in the original paper, we train the agent on 200 levels, while evaluation uses the full distribution of levels that Procgen can procedurally generate. What is randomized varies depending on the game environment, but covers textures, enemies, objects, and room layouts.

3.3.3 MiniHack's NetHack tasks

Most prior continual RL work evaluates on a relatively small number of tasks, but the recently introduced MiniHack [156] environment is fast enough to enable scaling up. MiniHack is based on the NetHack Learning Environment [90], a setting that is procedurally generated like Procgen and has stochastic dynamics (such as when attacking monsters). As with Procgen, the variation over which levels are randomized differs by environment, but includes objects, enemies,

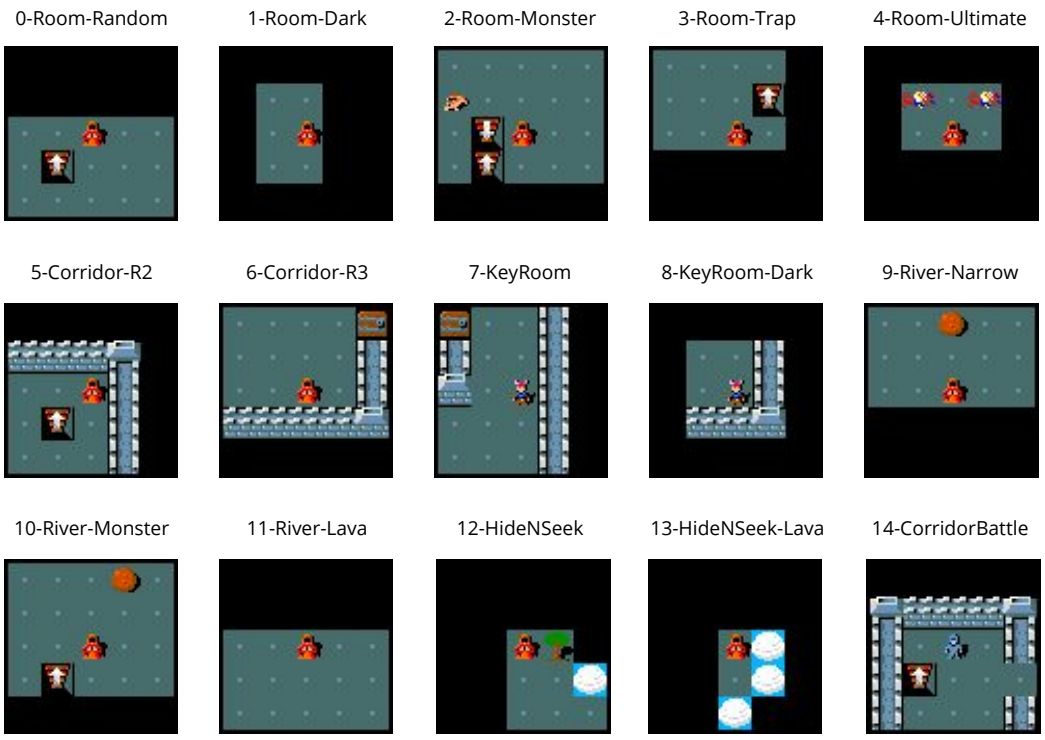


Figure 3.3: Examples of initial observations for each task in the 15 task MiniHack sequence. Observations are shown for the training task of each pair.

start & goal locations, and room layouts. The larger number of tasks enables MiniHack to more extensively test an agent's ability to prevent forgetting and to maintain plasticity. Additionally, while the Procgen tasks are easy to tell apart visually, the MiniHack tasks use the same texture assets and are more challenging to distinguish. This makes task identification and boundary detection more difficult.

To create the MiniHack task sequence, we define 15 (train, test) task pairs with a total of 27 different navigation-type tasks. The training environments are the easier versions, and we evaluate the agent on the harder environment variant. We select from the navigation-type tasks introduced by MiniHack, ordering by how MiniHack presents their tasks, and only omit the tasks that require episodic memory and deep exploration. Three evaluation environments are each used twice, because each has two related training tasks, the impact of which we discuss further in Section 3.5.3.

MiniHack also provides skill acquisition tasks, which could be used in future

work for an even more challenging task sequence.

When reporting results on this task sequence in Figure 3.9, we use the training environment name to refer to each task. We use only the pixel-based input for the agent. MiniHack renders an (80, 80) RGB image which we zero-pad to (84, 84) for convenience. All tasks share an 8-dim action space.

3.3.4 CHORES benchmark suite using ALFRED and AI2-THOR

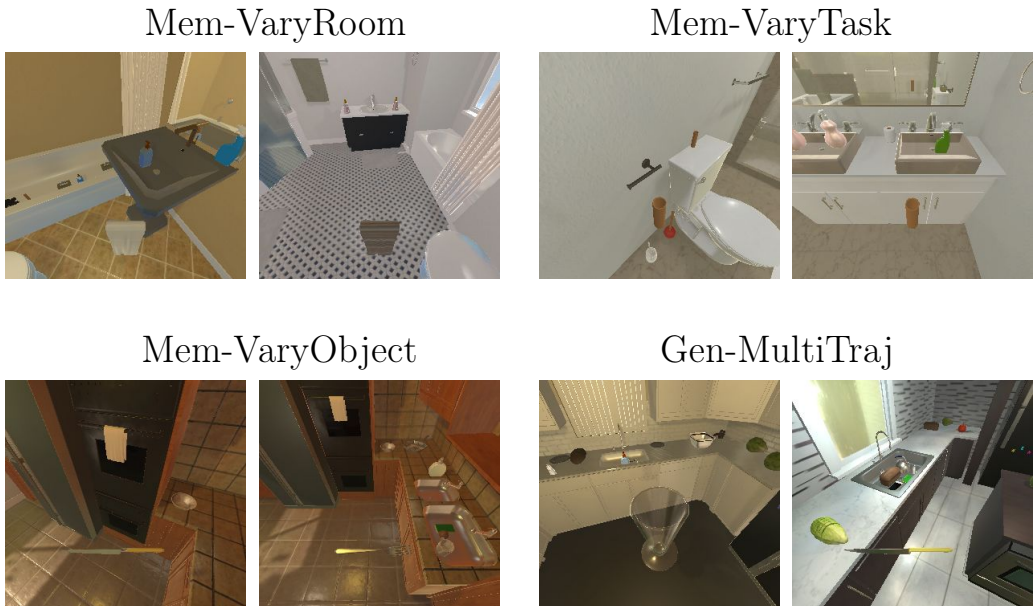


Figure 3.4: Examples for CHORES that show the variation within each task sequence.

AI2-THOR [87] is a visually realistic simulation environment that provides a variety of rooms for an agent to act in, with 30 layouts each of bedrooms, living rooms, kitchens, and bathrooms. ALFRED [168] is a benchmark for embodied vision-and-language agents which provides demonstrations for extended sequences of complex, tool-based tasks defined using AI2-THOR.

Using the demonstration trajectories and task definitions from ALFRED, we define a set of environments and task sequences for continual RL, which we refer to as **C**ontinual **H**ousehold **R**obot **E**nvironment **S**equences (CHORES). We do not provide ALFRED demonstration trajectories to the agent; instead, we leverage the demonstration data to initialize an AI2-THOR environment and generate subgoal images for the agent, which communicate the intended

task for the agent to perform. The initial state of the environment is set to the initial state of the demonstration trajectory. The usage of these demonstrations enables us to have a variety of initializations for robot location, object locations, and room instance without explicitly setting the simulation parameters or hand-defining distributions over these parameters. ALFRED also defines reward functions for its tasks based on achieving its subgoals, which we use.

Our CHORES benchmark extends continual RL into a visually realistic domain, where sample efficiency is key and where tasks bear similarities that make forward transfer particularly useful. Sample efficiency is critical because we designed CHORES to use a tight frame budget, as an initial attempt to mirror what would be feasible in the real world.

We first define three CHORES that shift the environment context in well-defined ways: **Mem-VaryRoom** changes the room scene, **Mem-VaryTask** changes the task type, and **Mem-VaryObject** changes the object with which the agent interacts. The fourth CHORES, **Gen-VaryTraj**, is considerably harder than the first three: it varies both the object and the scene, in addition to testing generalization on unseen contexts from heldout demo trajectories. Figure 3.4 visualizes CHORES and shows examples of variation within each task sequence. We note that the CHORES protocol is not exclusive to AI2-THOR and can also be applied using any home simulation with a diverse dataset of demonstrations.

We use an action space of 12 discrete actions (e.g. LookDown, MoveAhead, SliceObject, PutObject, etc.). For an action that interacts with an object, we take the action with the correct task-relevant object. Note that this differs from agents evaluated in ALFRED originally, which generate interaction masks to select one object from those in view to interact with. We use an observation size of (64, 64, 6), with 3 channels for the current RGB image observation and 3 channels for an RGB goal image.

3.3.4.1 CHORES design objectives

Goal communication. All CORA benchmarks other than CHORES use video game environments, where the visual differences between the tasks may have been sufficient for the agent to know what they are supposed to do, in order to receive reward. For instance in Atari, 0-SpaceInvaders is distinct enough in appearance from 2-BeamRider that no further task specification is required, Figure 3.1. However, since all CHORES take place in a fixed set of rooms, the observation that the agent receives on its own is insufficient to distinguish task boundaries with. In this work, we use subgoal images in CHORES to communicate task intentions to the agent. In real-world settings,

	Difficulty	Test Type	Num traj. per task	Scene	Task	Object
Mem-VaryScene	easier	memorize	1	Δ, bath	put in bathtub	hand towel
Mem-VaryTask	easier	memorize	1	bath, Room 402	Δ	toilet paper (TP)
Mem-VaryObject	easier	memorize	1	kitchen, Room 24	clean object	Δ
Gen-MultiTraj	harder	generalize	3	Δ, kitchen	cool & put in sink	Δ

	Task A	Task B	Task C
Mem-VaryScene	Room 402 ($r = 12$)	Room 419 ($r = 12$)	Room 423 ($r = 12$)
Mem-VaryTask	hang TP ($r = 12$)	put 2 TP in cabinet ($r = 24$)	put 2 TP on counter ($r = 24$)
Mem-VaryObject	fork ($r = 18$)	knife ($r = 18$)	spoon ($r = 18$)
Gen-MultiTraj	Room 19, cup ($r = 18$)	Room 13, sliced potato ($r = 31$)	Room 2, sliced lettuce ($r = 31$)

Table 3.1: Summary of the four CHORES benchmarks. The first three are memorization tasks, and are evaluated on the training environment. The fourth is a harder generalization task, with 3 trajectories per task to initialize the scene and task parameters. We also summarize which scene each task is in, what task it performs, and what objects it utilizes. We categorize each CHORES by what the task sequences varies. The r values in parentheses show the minimum return for solving the task.

this could be achieved by a human demonstrating a task and taking pictures at critical points during the task to give to a robotic agent. Future work may leverage the language annotations ALFRED provides with each demonstration trajectory for alternate as more convenient forms of communication would be useful for robotic agents to employ.

Task constraints. To make the benchmark as accessible for the community, our aim was for each task used by CHORES to be individually solvable in under five hours using a machine with 16 vCPUs, 64 GB of RAM, and a Titan X GPU. Given the nature of simulating realistic environments, this corresponds to a budget of around 1 million frames per task. Additionally, since continual RL ultimately should be deployed onto robotic agents in the real world, modest sample budgets align with what will likely be feasible with real world learning.

Most existing policies may not be sample efficient enough to learn complex tasks in this amount of time. However, by providing sequences of simple tasks that are at the edge of what is currently achievable, we hope to move beyond this boundary and encourage the development of algorithms that are successful under these conditions. We also provide one complex task as an example of what is possible moving forward and for what we hope will be achievable in

future CHORES benchmarking.

Task selection. The CHORES tasks were (by necessity) somewhat more hand-picked. These were selected in the following way:

1. We used ALFRED to generate a new set of trajectories for the latest AI2-THOR version (needed for headless rendering to use on our cluster) using ALFRED's defined set of tasks.

2. Based on our defined axes of variation (e.g. varying objects), we filtered successfully generated tasks into clusters that met our criteria.

3. From this filtered set, we selected tasks to maximize diversity (e.g. more than just pick-and-place).

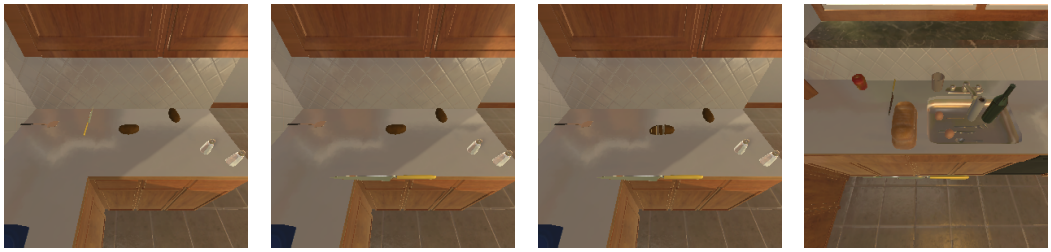
The selection process was born more out of necessity than the ideal, but we believe the tasks cover the desired goals of the benchmark more than adequately.

3.3.4.2 CHORES details

To start, we propose three memorization-based CHORES, each of which tests an agent's robustness to a particular type of domain shift. Each task within these three CHORES is intended to be relatively easy, using only one trajectory to set the environment parameters and evaluating in the same scene as during training. We additionally propose one harder CHORES to evaluate generalization. This last CHORES has more complex tasks, with a set of three trajectories per task to initialize the environment from, and evaluation is also done in unseen settings from a different 3 trajectories. These benchmarks are summarized in Table 3.1.In all cases, the locations of movable, interactable objects are randomized between trajectories.

The first task sequence, which we refer to as **Mem-VaryRoom**, keeps the task type and task object the same, while changing the room scene the agent interacts in to different bathrooms. The agent is trained to find a hand towel and place it in the bath tub of Room 402 for 1M steps, then in Room 419, then in Room 423. We then cycle through the environments again, to evaluate how much faster learning each environment is the second time.

The second task sequence, **Mem-VaryTask**, follows the same pattern but holds the current room and object constant, while changing the task. The agent is trained in the same bathroom to change a roll of toilet paper on a hanger, then to put two rolls of toilet paper in the cabinet, and finally to place two rolls on the countertop.



(a) Go to counter (b) Pick up knife (c) Slice potato (d) Go to sink

(e) Put knife in sink (f) Go to counter (g) Pick up potato (h) Go to fridge

(i) Cool potato (j) Go to sink (k) Put potato in sink

Figure 3.5: Visualization of one ALFRED trajectory used to define a task in CHORES.

The third task sequence, **Mem-VaryObject**, holds the current room and task constant but changes the object. In kitchen 24 the agent is tasked to clean a fork, then clean a knife, then clean a spoon. Cleaning is done by putting an object under running water from a faucet. For the first two tasks, after cleaning the agent must put the object on the counter top, and in the third it must put it in the cabinet.

The fourth task sequence, **Gen-MultiTraj**, uses a task where an agent takes an object, puts it in the fridge to cool it, removes it, and then places it in the sink. With this base task, the task sequence is as follows: (a) in kitchen 19, the agent performs the task with a cup; (b) in kitchen 13, the agent must slice a potato, then perform the task with the sliced potato; (c) in kitchen

2, the agent must slice lettuce, then perform the task with the sliced lettuce. The key difference from the previous task sequences is that each task in the fourth CHORES is evaluated on unseen settings initialized from three possible heldout demonstrations trajectories, testing an agent's ability to generalize.

In Figure 3.5, we visualize all subgoal images for one trajectory of the Gen-MultiTraj potato task (task 2).

Reward details. Unlike ALFRED which reports the number of subgoals achieved, we report the episode returns for consistency with the other benchmarks, clipped to a minimum value of -10. Extremely negative values occur when the agent performs a particularly suboptimal action for the duration of the episode, until the maximum step limit of 1000 is hit. Without clipping, this occasional negative behavior completely drowns out the agent's successes, both in visualization and metrics.

3.3.5 Summary of Tasks

In Procgen, we consider *in-distribution* generalization on unseen environment contexts, where C_{train} and C_{test} are disjoint sets composed of i.i.d samples of C. In particular, c is a random seed which determines how the game level procedurally generates. For the easy difficulty setting of Procgen, C_{train} is composed of 200 fixed seeds, while $C_{test} = C$ is uniform over all seeds.

In MiniHack, we consider *out-of-distribution* generalization, namely extrapolation along different environment factors. In addition to a random seed, any MiniHack environment instance is also determined by its `des-file`, which controls map layout as well as placement of environment features, monsters, and objects. For example, the MiniHack task Corridor-R5 has one `des-file` associated with it, while KeyRoom-S5 has defined its own separate one. Thus, each (train, test) task pair tests extrapolation along environment variations such as room size, number of rooms, obstacles, or lighting. We refer the reader to Appendix C of the MiniHack paper for further details on variation [156], and the provided code for the full list of MiniHack task pairs we use. Similarly, CHORES Gen-MultiTraj evaluates out-of-distribution generalization on unseen factors such as room scene and object of interest.

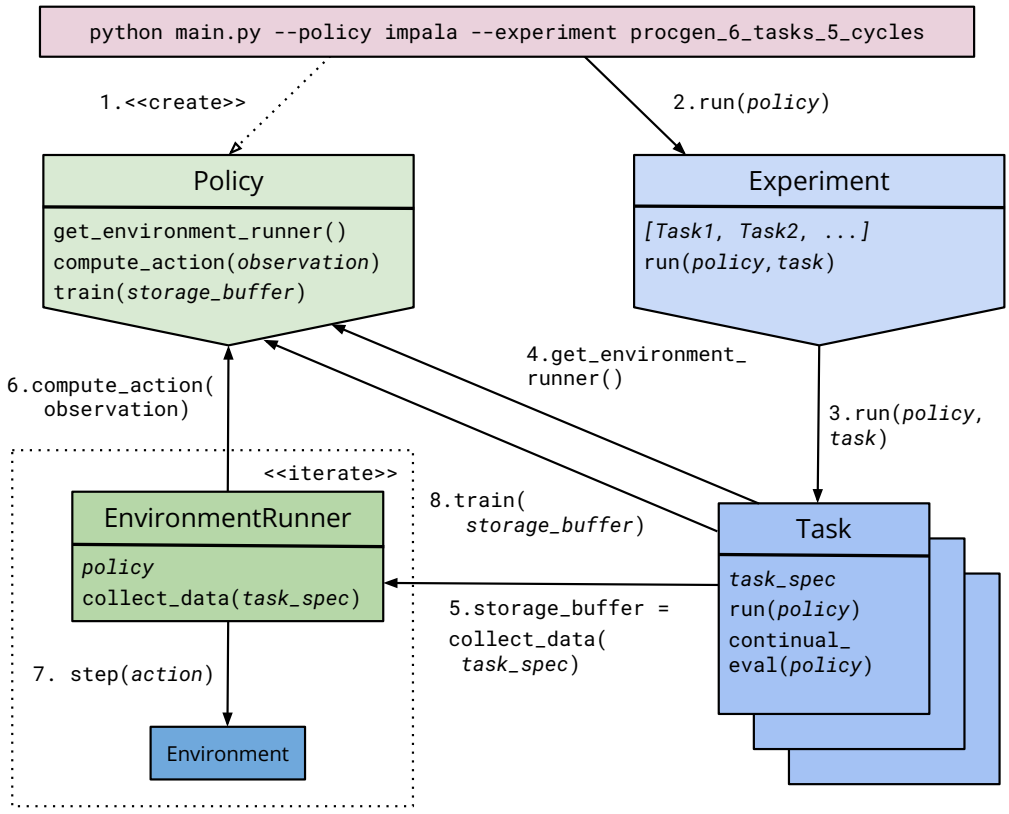


Figure 3.6: Sequence diagram representing the most basic flow of the `continual_rl` package. Blue represents components defined by `Experiment`, and green represents components defined by `Policy`.

3.4 CORA: A Platform for Continual Reinforcement Learning Agents

3.4.1 Code Structure

3.4.1.1 Architecture diagram

An overview of our code package architecture can be seen in Figure 3.6. The two fundamental components of the package are `Experiment` and `Policy`. `Experiment` conceptually encapsulates everything that should remain the same between runs, such as task specification, ordering, duration, and observation dimensionality. `Policy` encapsulates everything an algorithm has control over and can change as tasks are learned.

To train and evaluate agents on our benchmarks, these two things must be

specified. They may be specified either via command line, or by configuration file, along with any hyperparameter changes from the defaults. We recommend referring to the README provided with the source code for more details on running experiments and implementing new policies and experiments.

3.4.1.2 Policies

Any `Policy` must implement: (i) computing an action given an observation and (ii) training in response to collected experience. Any existing code that does these can be integrated into the `continual_rl` package by implementing a simple adapter wrapper. We provide an example of doing this for PPO based on the `pytorch-a2c-ppo-acktr` repository [88]. This enables easier integration of agents from outside our codebase, so the experiments and metrics provided by `continual_rl` can be leveraged.

The other thing a `Policy` must specify is how it should be run (i.e. its training loop), which we encapsulate in modules we refer to as EnvironmentRunners. In most simple cases, an existing EnvironmentRunner will suffice, such as EnvironmentRunnerBatch for standard, synchronous RL. However, in highly asynchronous or distributed cases, a user of CORA may wish to write their own. We provide more details on EnvironmentRunners in Section 3.4.1.3.

The final two steps to using a policy in CORA are the specification of configuration parameters, by extending `ConfigBase`, and adding the new policy to `continual_rl/available_policies.py`, so it can be used identically to existing ones, either via config file or via command line.

Additionally, since the policies are independent modules, it is also easy to use the provided policy implementations in a separate code base. The framework is installable as a pip package, which can be imported directly.

3.4.1.3 EnvironmentRunners

EnvironmentRunners have one function they must implement: `collect_data()`. Given the task specification, the EnvironmentRunner must collect any number of steps worth of data from the environment and return the results of what it has collected. The function will be called repeatedly until the total number of steps for the task have been satisfied. Data collection for continual evaluation occurs between calls to `collect_data()`, so care should be taken when selecting how much data to collect at a time. If too many timesteps are collected at once, the metrics will not be able to be computed as often as desired.

EnvironmentRunners can also be viewed as a higher-level API for more advanced policies. One example of how this is useful is for IMPALA [39].

IMPALA's key feature is how it learns asychronously by decoupling collecting data with actors from training policies, so the simple `Policy` structure of `compute_action()` and `train()` are insufficient. Instead, we define ImpalaEnvironmentRunner and implement a custom `collect_data()` method that returns new results that have accumulated every fixed number of seconds to support the actors and learners working asynchronously.

3.4.1.4 Experiments

Any `Experiment` defines a sequence of tasks. Every task contains full specifications (available in `continual_rl/task_spec.py`) for what environment should be created, how many frames it is given as a budget, and so on. Each task also provides common preprocessing features for convenience. For instance, we can define an ImageTask that scales the observation image, stacks frames, and converts the observation to a PyTorch tensor.

Experiments use this sequence of tasks to handle collecting metrics such as the Continual Evaluation metric described in Chapter 2.3. The Forgetting and Transfer metrics are computed in a post-processing step using the collected data from continual evaluation.

3.4.2 Baselines

We re-implemented four continual RL methods with baseline results on the Atari sequence from Section 3.3.1, which are not publicly available to the best of our knowledge. We prioritized methods which had been demonstrated on Atari before, in order to reference such results and appropriately validate our implementations. The methods are those described in Chapter 2.6: EWC, online EWC, Progress & Compress (P&C), and CLEAR.

Our implementations of these baselines all build off the IMPALA [39] architecture and use the open-source TorchBeast code [89]. In Section 3.5.1, we validate the performance of our baseline implementations compared to that of the original implementations on Atari.

3.4.3 Code package

We release our `continual_rl` codebase[1] as a convenient way to run continual RL baselines on the benchmarks we outlined in Section 3.3 and to use the continual RL evaluation metrics we defined in Chapter 2.3. The package is designed modularly, so any component may be used separately elsewhere, and

[1]Our code: https://github.com/AGI-Labs/continual_rl

new benchmarks or algorithms may be integrated in. Hyperparameters for all experiments are made available as configuration files in the codebase.

	IMPALA	EWC	Online EWC	P&C	CLEAR
Atari	-2.3 ± 0.1	-0.3 ± 0.3	-1.6 ± 0.1	-1.8 ± 0.1	-0.7 ± 0.1
Procgen	-1.2 ± 0.0	-0.7 ± 0.0	-1.1 ± 0.0	-0.5 ± 0.0	-0.0 ± 0.0
MiniHack	-0.3 ± 0.0	-	-	-	-0.1 ± 0.0
C-VaryRoom	-	0.6 ± 0.6	-	0.0 ± 0.0	1.7 ± 1.7
C-VaryTask	-	-2.1 ± 1.4	-	2.2 ± 2.2	-1.5 ± 0.2
C-VaryObj	-	1.0 ± 1.1	-	-2.0 ± 2.0	3.4 ± 0.2
C-MultiTraj	-	-0.7 ± 1.2	-	0.1 ± 0.1	0.3 ± 2.1

(a) Isolated Forgetting ($-\mathcal{F}_{\mathcal{I}}$) summary statistics for all experiments.

	IMPALA	EWC	Online EWC	P&C	CLEAR
Atari	0.1 ± 0.0	0.1 ± 0.2	-0.0 ± 0.0	-0.0 ± 0.1	0.0 ± 0.0
Procgen	-0.1 ± 0.0	-0.2 ± 0.1	-0.1 ± 0.1	0.1 ± 0.1	-0.1 ± 0.0
MiniHack	0.6 ± 0.0	-	-	-	0.5 ± 0.1
C-VaryRoom	-	-0.0 ± 0.0	-	3.2 ± 1.9	-1.1 ± 1.1
C-VaryTask	-	-4.0 ± 2.6	-	0.2 ± 0.1	-3.2 ± 0.0
C-VaryObj	-	2.6 ± 2.9	-	5.4 ± 1.3	-4.6 ± 0.8
C-MultiTraj	-	-4.0 ± 0.5	-	0.4 ± 0.1	-4.7 ± 0.7

(b) Transfer ($\mathcal{Z}$) summary statistics for all experiments.

Table 3.2: Summary statistics for all benchmarks and for all methods evaluated on them.

3.5 Experimental Results

In this section, we present results on Atari (Section 3.5.1), Procgen (Section 3.5.2), MiniHack (Section 3.5.3), and CHORES (Section 3.5.4). Metric summary statistics for all methods can be seen in Table 3.2. The final performance table for Atari is shown in Table 3.3. All metric diagnostic tables and other final performance tables are available in [138]. To estimate expected return and compute metrics, we use the following values for parameters described in Section 2.3: Procgen: $N = 0.25e6$, $s = 20$; MiniHack: $N = 1e6$, $s = 10$; CHORES: $N = 5e4$, $s = 3$, Atari: $N = 0.25e6$, $s = 5$.

On the Continual Evaluation plots, solid lines represent evaluation on unseen testing environments, while dashed lines show evaluation on the training

environments. Shaded grey rectangles are used to indicate which task is being trained during the indicated interval. We plot the mean as each line and the standard error as the surrounding, shaded region. For these experiments, we used the negated isolated forgetting metric, as discussed in Chapter 2, to make positive values imply the desirable outcome.

We proceed to discuss experimental results with CORA using two perspectives. First, from the viewpoint of benchmark analysis, we empirically discuss what each benchmark is evaluating and give examples of how the metric tables may be used. Second, from the view of algorithm design, we examine the performance of the baselines to identify axes which can be improved on by future algorithms. We frame this section through these two lenses in order to show how CORA may be used by end-users.

3.5.1 Atari results

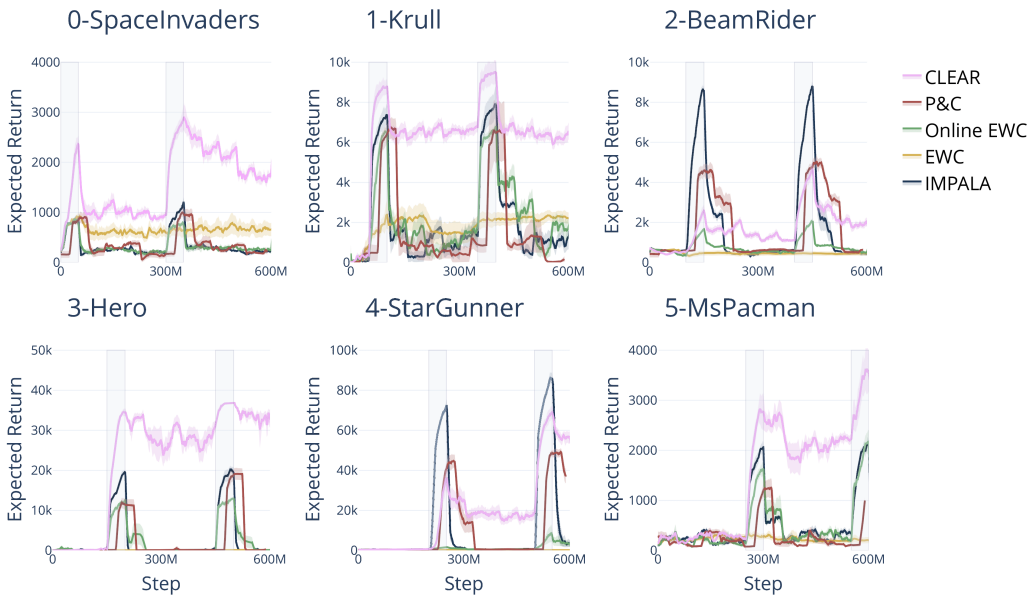


Figure 3.7: Results for Continual Evaluation ($\mathcal{C}$) on the 6 Atari task sequence from [150, 160]. Due to compute constraints, we only train for 2 cycles compared to the original experiments which used 5 learning cycles. IMPALA is the baseline learning algorithm that the other methods for continual RL build off. Gray shaded rectangles show when the agent trains on each task.

We include the standard, proven Atari task sequence as a benchmark, in order to validate our baseline implementations on an existing standard. The

Task	CLEAR	P&C	Online EWC	EWC	IMPALA
0-SpaceInvaders	**1767 ± 89**	209 ± 56	240. ± 65	654 ± 134	248 ± 39
1-Krull	**6543 ± 410.**	157 ± 142	1714 ± 532	2211 ± 291	1250 ± 593
2-BeamRider	**2003 ± 212**	628 ± 36	492 ± 68	459 ± 110.	426 ± 108
3-Hero	**33604 ± 1488**	289 ± 289	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
4-StarGunner	**56366 ± 2882**	37515 ± 3693	3422 ± 1453	140. ± 87	3496 ± 1774
5-MsPacman	**3536 ± 400.**	996 ± 58	2172 ± 230.	202 ± 58	2104 ± 104

Table 3.3: Comparison of final performance (mean±SEM) between methods in the environments for each Atari task.

reproduction of these Atari results were developed over hundreds of hours, including time spent analyzing papers for algorithm details, corresponding with the original authors, tuning hyperparameters, and running many seeds of Atari experiments, each of which takes hundreds of millions of frames. These results were reproduced using a university server cluster and several thousand dollars of AWS credits, compared to the industry-level compute that the original authors (from DeepMind) [160] and [150] had access to. This is one of the primary reasons we are advocating for more compute-friendly continual RL benchmarks. It is also the reason that we were only able to run 2 learning cycles for these Atari results instead of the intended 5 cycles.

We use the full 18-dim action space for this task sequence. The observation space is (84, 84) grayscale images, and the agent receives a framestack of 4. The Atari games used are fully deterministic, and following the prior continual RL work on Atari, we do not apply sticky actions [107].

Atari results are shown in Figure 3.7, and we compare them against the results presented in [150]. Notably, on almost all Atari tasks, our implementations outperform the results reported in CLEAR. This may be because we use the TorchBeast [89] implementation of IMPALA, while the results in [150] and [160] use an earlier, pre-release version of IMPALA.

Benchmark analysis. Summary metrics are available in Table 3.2. From these, we observe that Atari does effectively test for robustness to catastrophic forgetting, but exhibits nearly no transfer. By looking at the diagnostic transfer table for Atari, we observe no transfer, likely because the six Atari tasks used are too distinct from each other.

Algorithm design. From the continual evaluation results in Figure 3.7, we can see that CLEAR outperforms the other baselines at both at recall and plasticity on Atari, which matches the original results by [150]. EWC maintains a flat return curve for early tasks, which is consistent, losing plasticity and

failing to learn the later tasks. P&C largely maintains its plasticity, but we observe considerably more forgetting than was reported.

3.5.2 Procgen results

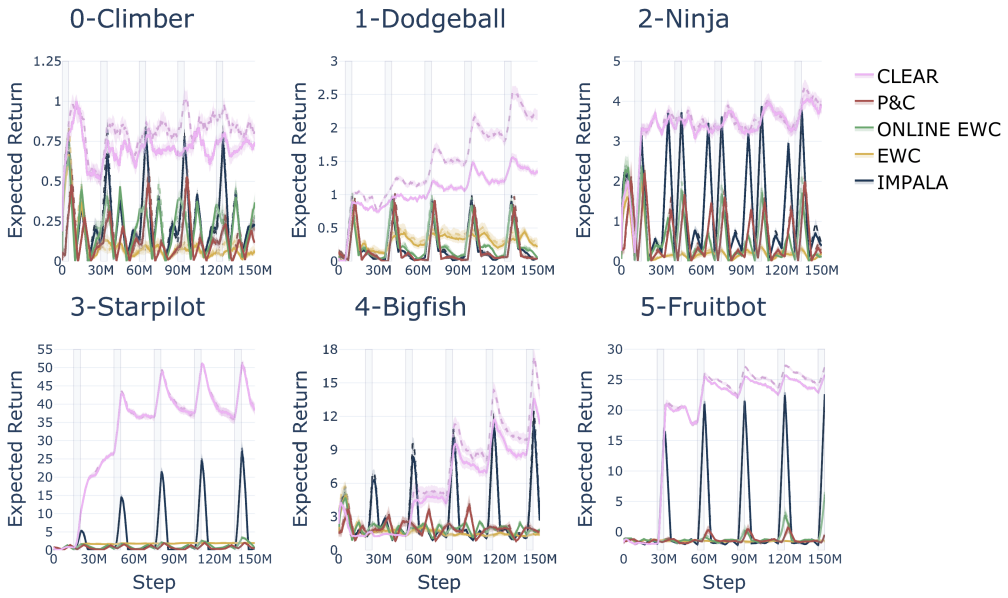


Figure 3.8: Results for Continual Evaluation ($\mathcal{C}$) on the 6 Procgen tasks, based on recommendations by [65]. The solid line shows evaluation on unseen testing environments; the dashed line shows evaluation on training environments. Gray shaded rectangles show when the agent trains on each task.

Benchmark analysis. From the summary statistics in Table 3.2, we can see that Procgen tests for catastrophic forgetting, but shows little forward transfer overall, which aligns with our expectations for this benchmark. Using the Transfer metric diagnostic tables for Procgen, we see that that forward transfer is not uniform across tasks. For example, we observe that 0-Climber transfers reasonably well to 2-Ninja and 4-Bigfish. Intuitively, as 0-Climber and 2-Ninja are both platformer games, transfer is expected. Transfer to 4-Bigfish is less obvious but may be explained by both games using side-view perspectives or by sharing useful skills like object gathering. In particular, 0-Climber involves collecting stars, while 4-Bigfish tasks the agent with eating other fish.

Algorithm design. From the Continual Evaluation results in Figure 3.8, we observe that CLEAR is a strong baseline for avoiding catastrophic forgetting

on all tasks, reliably outperforming every other method. However, there is still room for improvement: maximum scores obtained by CLEAR fall significantly short of the maximum achievable scores reported in Appendix C of the Procgen paper [31], particularly on 0-Climber (1 vs 12.6), 1-Dodgeball (2.5 vs 19), 2-Ninja (4 vs 10), and 4-Bigfish (18 vs 40). Additionally, by comparing the training (dashed) and testing (solid) lines, we observe that CLEAR generalizes well to unseen contexts on all tasks, except 1-Dodgeball. The summary statistics in Table 3.2 show that transfer is overall low for Procgen. Using the more detailed diagnostic tables, we can see that this varies by task. For instance with EWC, training on 1-Dodgeball improves performance on 3-Starpilot but reduces performance on all other tasks. CLEAR shows some transfer from 0-Climber to 2-Ninja, but essentially none anywhere else, even showing negative transfer from 1-Dodgeball to 2-Ninja. These failures represent opportunities for investigation and for new algorithms to improve on.

3.5.3 MiniHack results

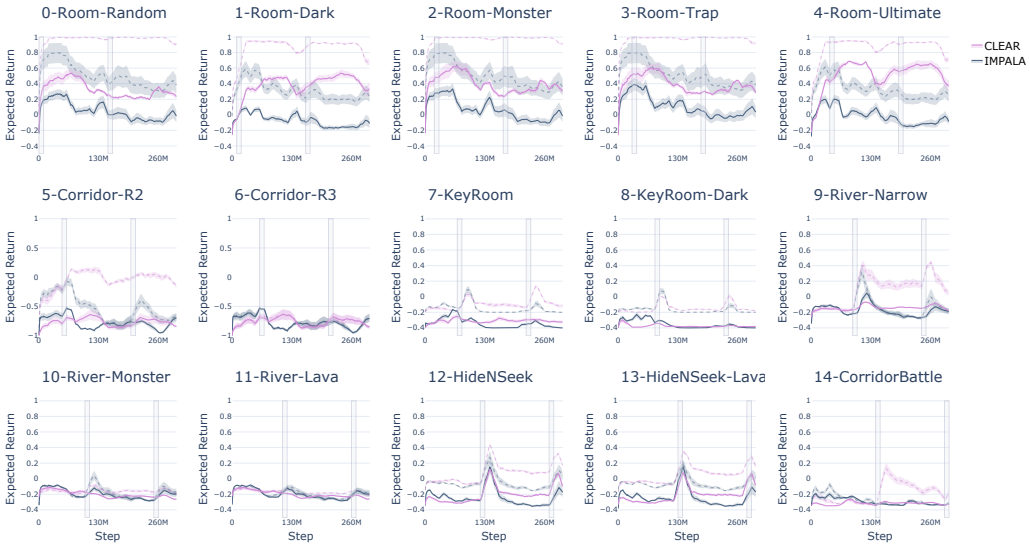


Figure 3.9: Results for Continual Evaluation ($\mathcal{C}$) on the 15 MiniHack task pairs sequence. The solid line shows evaluation on unseen testing environments; the dashed line shows evaluation on training environments. Gray shaded rectangles show when the agent trains on each task.

Benchmark analysis. From the summary statistics in Table 3.2, we observe that IMPALA and CLEAR show minor transfer on MiniHack tasks compared

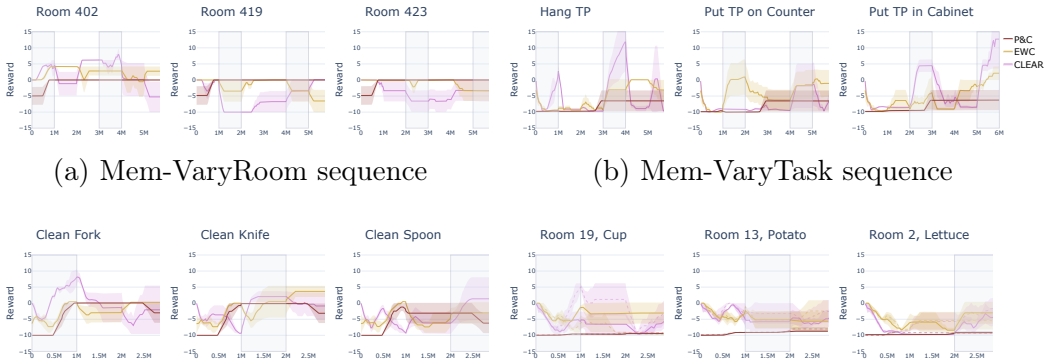
to Atari and Procgen. Looking at the Transfer metric diagnostic tables, we can see that the first task 0-Room-Random transfers significantly to all other tasks, which can be interpreted as the agent learning the basics of moving around a MiniHack environment. Furthermore, we observe transfer from environments to others of the same type. For example, the Room environments generally positively transfer to each other, while mostly negatively transferring to the later River and HideNSeek environments. When training tasks share the same testing task, such as 12-HideNSeek and 13-HideNSeek-Lava, the transfer metric is noticeably high, as expected. Finally, from the Continual Evaluation results in Figure 3.9, we observe that MiniHack effectively tests for plasticity as well, as later experiments fail to learn effectively.

Algorithm design. From the Continual Evaluation results in Figure 3.9, we can see CLEAR generally performs well at learning tasks and mitigating catastrophic forgetting for the first five tasks. However, we observe that the agent struggles to learn later tasks (fails to maintain plasticity), and that there is a significant out-of-distribution generalization gap, in performance on test (solid) compared to train (dashed) environments for all tasks. Additionally, inspecting the forgetting metric diagnostic tables, we see that the HideNSeek tasks exhibit particularly high forgetting. These results and shortcomings present important areas for new algorithms to pursue.

3.5.4 CHORES results

We show Continual Evaluation results for the four CHORES in Figure 3.10. On the memorization sequences, the testing environment is the same as the training environment, and evaluation is represented with a solid line. In our generalization experiment, the solid line represents evaluation on held-out testing environments, and the dotted line represents performance on the training environments. We report 2 cycles for each of (a) Mem-VaryRoom and (b) Mem-VaryTask, and 1 for each of (c) Mem-VaryObject and (d) Gen-MultiTraj, due to time constraints.

Benchmark analysis. From the Continual Evaluation results, we can see that CHORES are challenging, and current agents achieve low returns overall. Some learning occurs for the first task of every sequence, but nearly none in later tasks, with the exception of (a) Mem-VaryTask. We also observe no generalization to unseen contexts from held-out demo trajectories in (d) Gen-MultiTraj. The Transfer and Forgetting metrics are less meaningful with low returns, but there is some indication of forward transfer, particularly on (c) Mem-VaryObj. Taken together, we can see CHORES as the reach-goal; a



(a) Mem-VaryRoom sequence

(b) Mem-VaryTask sequence

(c) Mem-VaryObject sequence

(d) Gen-MultiTraj sequence

Figure 3.10: Results for Continual Evaluation ($\mathcal{C}$) on the CHORES suite of benchmarks. For (d) Gen-MultiTraj, the solid line shows evaluation on unseen testing environments; the dashed line shows evaluation on training environments. Gray shaded rectangles show when the agent trains on each task.

set of tasks current methods cannot solve, and that will truly test the sample efficiency of future methods.

Algorithm design. CLEAR achieves the highest returns overall, but there is still significant room to improve in learning these tasks. We observe significant Forgetting, particularly on (c) Mem-VaryObj, illustrating one such area for improvement. Advances in sample efficiency and exploration are likely required for agents to make progress on this challenge.

3.6 Summary

In this chapter, we presented CORA, a platform designed to reduce the barriers to entry for continual reinforcement learning. CORA provides a set of benchmarks, open-sourced implementations of several baselines, evaluation metrics, and the modular `continual_rl` package to contain it all. Each benchmark is designed to exercise different aspects of continual RL agents: a standard, proven Atari benchmark for catastrophic forgetting and sample efficiency; a Procgen benchmark to test forgetting and generalization to unseen environment contexts; a MiniHack benchmark to test generalization, plasticity, and transfer; and the new, challenging, CHORES benchmark to test capability in a visually-realistic environment where sample-efficiency is key. With these benchmarks, we demonstrate the strengths and weaknesses of the current state-of-the-art

continual RL method, CLEAR. While CLEAR generally outperforms the other baselines at learning tasks and mitigating catastrophic forgetting, significant improvements are needed for generalization, forward transfer, and maintaining plasticity over a long sequence of tasks.

Limitations. We study task sequences that share a high-dimensional observation space (images) and have a discrete action space. These RL tasks are finite-horizon, with episodic resets. In this work, we also primarily study video-game environments, with procedurally-generated variation. These assumptions are shaped by our perspective of the continual RL problem and current state of the field. We acknowledge that different points of view exist, backed by design choices which may differ from the conditions we study. For instance, we consider task cycling in this work, which may favor replay-based methods such as CLEAR that can retain data from all tasks in the sequence, after the first cycle. This protocol does not apply to a pure online learning setup, where no assumptions may be made on the structure and similarities of incoming data. We intend to relax these assumptions as new methods are developed and CORA evolves.

Chapter 4

SANE: Self-Activating Neural Ensembles

4.1 Introduction

The core goal of continual learning is to learn new skills efficiently by leveraging prior knowledge without forgetting old behaviors. However, when placed into continual learning settings, current deep reinforcement learning approaches do neither: the forward transfer properties of these systems are negligible, and they suffer from catastrophic forgetting [44, 112].

The core issue of catastrophic forgetting is that a neural network trained on one task starts to forget what it knows when trained on a second task, and this issue only becomes exacerbated as more tasks are added. The problem ultimately stems from sequentially training a single network in an end-to-end manner. The shared nature of the weights and the use of backpropagation to update them mean that later tasks overwrite earlier ones [112, 143].

To handle this, past approaches have proposed a wide variety of ideas: from task-based regularization [86], to learning different sub-modules for different tasks [153], and dual-system slow/fast learners inspired by the human hippocampus [160]. The fundamental problem of continual learning, which few methods address, is that the agent should autonomously determine how and when to adapt to changing environments, or stabilize existing knowledge, without explicit task specification. It is infeasible for a human to indefinitely provide agents with task-boundary supervision, and doing so side-steps the core problem.

There are a few existing task-agnostic [201] methods, though most have only been demonstrated on classification or behavior cloning: for example [7]

addresses the problem by detecting plateaus and using those as boundaries, [95] adaptively creates new clusters using Dirichlet processes, and [180] replaces backpropagation completely. Methods that have been demonstrated on reinforcement learning are rarer; exceptions include [150], which utilizes a large replay buffer, and [103] which uses the error in the value estimate to determine when to consolidate.

We approach the problem by introducing a system that continuously, dynamically adapts to changing environments. Our ensemble-based method, Self-Activating Neural Ensembles (SANE), depicted in Figure 4.1, is the core of our proposal. Each module in the ensemble is a separate, task-agnostic network. Periodically, a single module from the ensemble is activated to determine which policy to use. Only activated modules are updated, leaving unused modules unchanged and therefore protected from catastrophic forgetting. Crucially, our ensemble is dynamic: new modules are created when existing modules are found to be insufficient. In this way, modules are created when novel scenarios are encountered, preventing destructive updates to other modules. Additionally, SANE is simple; modules control their own relevance, activating when the situation to which they are specialized is encountered, and remaining untouched the rest of the time. SANE provides the following desirable properties for continual reinforcement learning: (a) It mitigates catastrophic forgetting by only updating relevant modules; (b) Because of its task-agnostic nature, unlike previous approaches, it does not require explicit supervision with task IDs; (c) It achieves these targets with bounded resources and computation. We demonstrate SANE on three visually rich, challenging level sequences based on Procgen [31] environments. Additionally, we analyze the behavior of SANE at a more fine-grained level on 2 individual runs, to gain more understanding into the dynamics of training SANE.

4.2 Related Work

Continual learning. Any continually learning system must balance *stability* (the extent to which existing knowledge is retained) and *plasticity* (how readily new knowledge is acquired) [2, 54, 114]. Stability has posed a substantial challenge due to *catastrophic forgetting*, by which neural networks trained by backpropagation abruptly forget learned behavior for solving old tasks when presented with new tasks [44, 80, 100, 112, 143]. Broadly, methods for continual learning can be categorized under Regularization, Rehearsal, or Architectural approaches, as well as combinations of them. We refer the reader to the survey papers by [99, 118, 128], and review methods for continual learning relevant to

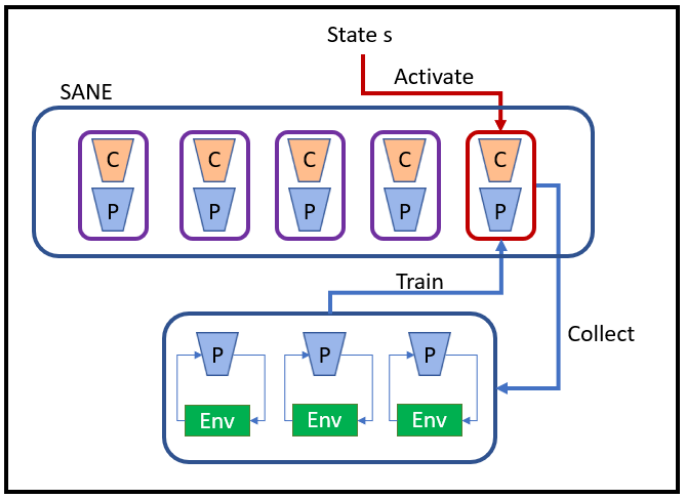


Figure 4.1: The overall structure of the SANE system. Each module contains an actor and a critic. Upon activation, collection occurs from several environments in parallel.

discussion of our approach.

Recent strategies for mitigating catastrophic forgetting such as Elastic Weight Consolidation (EWC) [86], among other Regularization approaches [4, 7, 25, 60, 69, 94, 101, 130, 147, 162, 200], constrain updates to network parameters important for past tasks when learning new tasks. However, these methods fundamentally run into the stability-plasticity dilemma, as over-constraining updates can hinder the learning of new tasks. To improve plasticity, dynamic architectures [146, 153, 176, 206] incorporate additional network parameters to help learn new tasks. Furthermore, to prevent model size from growing unbounded, such approaches [32, 77, 109, 110, 160, 178, 192, 194, 196] use distillation [22, 61, 152, 175], pruning, and related techniques to consolidating learned behavior while reducing parameter count. Similarly, Rehearsal and (generative) memory-based approaches [5, 23, 24, 26, 37, 43, 46, 51, 66, 75, 79, 98, 105, 129, 144, 145, 148, 165, 169, 179, 188, 191] must also balance data storage and memory network constraints when determining which examples are needed to preserve previously learned behavior. We build our ensemble approach off of CLEAR [150], a state-of-the-art asynchronous continual RL method which uses Rehearsal, by maintaining a replay buffer that uniformly preserves past experience via reservoir sampling [66], along with Regularization, via behavioral cloning and a KL penalty to preserve prior learned behavior.

Ensemble methods. Falling under Architectural approaches, aggregation

ensembles [29, 180, 185] combine predictions from multiple models to produce a final output. These types of ensembles are also commonly used for uncertainty estimation [91], exploration [131], or reducing overestimation bias such as in double Q-learning [45, 59]. In contrast, modular ensembles [6, 42, 83, 95, 125] use a subset of the entire ensemble's parameters to select an appropriate expert model for the task presented. Selectively updating a subset of parameters or specific modules instead of the entire ensemble can circumvent catastrophic forgetting while bounding compute costs; this is a feature we utilize in SANE, which is a type of modular ensemble rather than the former, aggregation ensemble. Our method is similar to Multiple Choice Learning [57, 93, 96, 161], which chooses and updates only the best expert from an ensemble, encouraging specialization. However, Multiple Choice Learning uses fixed-size static ensembles, while SANE is a dynamic ensemble that merges similar modules and works with a given resource budget. For supervised continual learning, LMC [124] also proposes a modular ensemble approach, although LMC assumes access to task IDs at training time and can only add modules, meaning that its computational footprint is linear relative to the number of tasks learned. In contrast, SANE is completely task-agnostic at train and test time, while also creating and merging modules to meet a given compute budget.

Hierarchical RL can be seen as a hierarchy of meta-policies that control access to an ensemble of (often hand-designed) sub-policies that act at differing temporal resolutions [21, 171, 177, 204]. Analogous to our own value-based activation score, some hierarchical RL methods use predicted Q-values to select amongst their ensemble, as in [34, 36]. [53] demonstrates the utility of avoiding meta-policies, instead relying on primitives that independently determine their own relevance, similar to self-activation in our approach. However, their primitives distinguish themselves by factorizing a state space, placing strong assumptions on the learnable policies. Additionally, their primitives are not created over time, so the method relies on regularization to ensure primitives in their ensemble are used.

4.3 Background

Traditional neural networks suffer from catastrophic forgetting because weights in the network are changed by backpropagation every update [112], causing information learned in a new scenario to overwrite prior behavior. Instead of learning and updating a single neural network for policy π across multiple tasks, we propose using an dynamic ensemble of *self-activating modules*. Our approach partitions, allocates, and manages parameters for separate modules,

so that each module may handle different situations without interfering with others.

If a module is relevant to the current situation, it activates during inference and is updated during training. If a module is irrelevant, it is unused and remains unchanged. One way of viewing these modules is as latent behaviors, each specialized to a particular circumstance. For example, if in one context an agent must carefully wait to allow an enemy to pass, we don't want this to disrupt a behavior where moving quickly to dodge an enemy is the best action.

How may we know when to use which module, when task boundaries are ambiguous and not given by human supervision? Each module in our ensemble predicts an *activation score*, which estimates the relevancy of a given module's behavior to the current situation, and the module with the highest activation score is selected from the ensemble. An appropriate activation score will protect modules against catastrophic forgetting, and can also enable forward transfer, by activating modules with prior learned behaviors that are advantageous in new settings.

How should such an ensemble be structured? Pre-defining a static fixed-size ensemble will be ineffective for module-based behavior specialization. In such a static ensemble, one module will tend to perform well at a task, leading to that module being chosen as the starting point for future tasks which results in catastrophic forgetting. Regularizing with additional losses would be necessary to distribution activation across the ensemble's experts, as in [53, 68, 163]. Instead, we design SANE as a *dynamic ensemble*, in which modules are created and merged together as necessary. Intuitively, modules are created when existing latent behaviors fail to perform as expected, and the ensemble determines that a new latent behavior is needed. Modules may also be merged to conserve resource consumption and meet a given compute budget.

Bringing self-activating modules and a dynamic ensemble together, we present Self-Activating Neural Ensembles (SANE). To summarize, our approach differs from traditionally-used ensembles in two ways: (i) We do not aggregate results across modules, in order to keep modules isolated from one another. This circumvents catastrophic forgetting, by not backpropogating through the entire ensemble. (ii). The ensemble itself is dynamic, in that modules are being created and merged throughout training.

4.4 Self-Activating Neural Ensembles for Continual RL

We now proceed to formally describe SANE in full detail. SANE is a dynamic collection of modules $\{M_1, \ldots, M_k\}$ where, based the context, one module M_t activates and is used for inference. Subsequently, given transitions from collected episodes, only the selected module $\mathcal{M}_t$ is updated. We describe an individual SANE module in Section 4.4.1, including how activation scores are computed to determine which module to use. We present the learning process to manage a dynamic ensemble in Section 4.4.2. Pseudocode is provided in Algorithm 1.

Algorithm 1 SANE

Require: timestep t, total task timesteps T, state at episode start s_0, max allowed module count N, modules $\mathcal{M} = \{M_1, \ldots, M_k\}$ where M_i contains actor π_i, critic V_i, static anchor critic a_i, and replay buffer R_i.

while $t < T$ **do**
  $M_{max} = \arg\max_i(v_{UCB,i}(s_0))$ ▷ select active module
  Collect t_{new} timesteps of data using M_{max}
  $t := t + t_{new}$
  if $v_{UCB,M_{max}}(s) < v_a(s)$ **then** ▷ negative drift detected, so add module
    $M_{new} := clone(M_{max})$
    $a_{new} := clone(V_{max})$
    $\mathcal{M} := \{M1, ..., M_k, M_{new}\}$
  else if $v_{LCB,m}(s) > v_a(s)$ **then** ▷ positive drift detected, so update module
    $a_{max} := clone(V_{max})$
  end if
  if $|M| > N$ **then** ▷ merge modules
    $g_i := avg(batch(R_i)['observation'])$ ▷ compute an avg observation for each module
    M_i, $M_j = \arg\min_{i,j} ||g_i, g_j||_2$
    M_{keep}, M_{remove} = which of M_i or M_j has been used more and fewer times, respectively
    $R_{keep} = \{R_i, R_j\}$
    $\mathcal{M} = \mathcal{M} - M_{remove}$

  end if
end while

4.4.1 Self-Activating Module

Every module $\mathcal{M}_i$ is an actor-critic algorithm represented by: a policy $\pi_i(a|s)$, a critic $V_i(v, u|s)$, as well as a replay buffer $\mathcal{B}_i$ that holds experience transitions. We modify the critic V_i from the standard formulation in the following way. Given a state s at timestep t, the critic V_i predicts two scalars: $v_i(s)$, the value estimate of the return R_t received if module $\mathcal{M}_i$ is activated, and $u_i(s)$, an uncertainty estimate of the absolute error:

$$u_i(s) \approx |R_t - v_i(s)| \tag{4.1}$$

We proceed by defining an optimistic estimate v_i^{UCB} (upper confidence bound) and a pessimistic estimate v_i^{LCB} (lower confidence bound) for the return that the module $\mathcal{M}_\rangle$ can achieve from state s:

$$v_i^{UCB}(s) = v_i(s) + \alpha_u * u_i(s) \tag{4.2}$$

$$v_i^{LCB}(s) = v_i(s) - \alpha_l * u_i(s) \tag{4.3}$$

where $\alpha_u, \alpha_l > 0$ are hyperparameters which represent how wide a margin around the expected value to allow. We use these margins to: (i) choose which module to activate during inference; (ii) decide when to create a new module during Structure Update (Section 4.4.2).

In all, each module $\mathcal{M}_i$ can be considered as a tuple $\langle \pi_i, V_i, \mathcal{B}_i, \overline{V}_i, A_i \rangle$, where $\overline{V}_i$ and A_i are two other versions of the critic V_i, which we proceed to describe.

The target network $\overline{V}_i$ is used for the confidence bounds estimates (Equation 4.2 and 4.3) instead of V_i. Target networks are commonly used in Q-learning [8, 45, 102, 116] to stabilize training by reducing variance from approximation error. Similarly, we update $\overline{V}_i$ with an exponential moving average [67, 135, 151]. Denote V_i's parameters by θ_i, $\overline{V}_i$'s parameters by θ'_i, and the update rate by τ_V; we use the update: $\theta'_i \leftarrow \tau_V \theta_i + (1 - \tau_V)\theta'_i$.

The anchor A_i is a frozen instance of the critic V_i from when the module $\mathcal{M}_i$ was created. We describe how we use the anchor A_i to measure drift in Section 4.4.2.1.

Module update. SANE can be applied to any actor-critic algorithm; we describe specifics of our implementation in Section 4.4. Let $L_{\mathcal{M}_i}$ denote the loss function of the active SANE module and L_{rl} be the loss of the actor-critic RL algorithm, with components associated with module $\mathcal{M}_i$. We perform a module update by optimizing $L_{\mathcal{M}_i} = L_{rl} + \mu L_{ue}$, where L_{ue} is MSE loss to estimate uncertainty from Equation 4.1.

Inference (Self-Activation). SANE consists of several modules where each module represents the behavior for a particular situation. Activating the right module for the right situation is key to the success of the SANE method. In an RL setting, the critic predicts a value estimate, which can serve as an effective proxy for how successful a module $\mathcal{M}_i$ may be in obtaining high return. At the beginning of the episode, we compute v_i^{UCB} for each module in the ensemble $\{\mathcal{M}_1, \ldots, \mathcal{M}_k\}$ using the target network critic $\overline{V}_i$. Then, we greedily select the module whose critic predicts the highest such value, and use that module for the whole episode.

4.4.2 Dynamic Ensemble

We propose a process to dynamically update the structure of the ensemble in SANE. If the current set of modules behave in an expected manner (returns are within the expected range) then the current set is sufficient. However at some point in training, if the returns are outside the expected range, then we know the current set of modules is insufficient. We create new modules to handle the new situation, and merge modules together to stay within a given compute budget.

4.4.2.1 Measuring drift

The key to successfully updating the SANE structure lies in our ability to detect that we have moved outside this expected range. Our main assumption here is that the change in rewards received is sufficient for distinguishing relevant changes in setting. Therefore, we detect change in setting by measuring drift in rewards. Drift describes when an environment is non-stationary, e.g. when the reward distribution or the state transition distribution is changing over time. Often where drift occurs, catastrophic forgetting follows because networks update to the new setting, forgetting the old.

To recognize drift with SANE, at the time of their creation modules have their critic cloned and frozen, creating a static critic called the *anchor*. We compare the prediction of a module's critic to the prediction of its anchor. We say that sufficient change has occurred when the bounds of the expected return, as defined in Section 4.4.1, predicted by a module's critic do not include the value predicted by its anchor, which serves as a static baseline.

Let v_{A_i} denote the value estimate of the anchor A_i. Formally, we say that sufficient change has occurred when for a given state s, critic V_i, and anchor A_i, either of the following inequalities hold:

$$v_i^{UCB}(s) < v_{A_i}(s) \tag{4.4}$$

$$v_i^{LCB}(s) > v_{A_i}(s) \tag{4.5}$$

In practice, we use the target network critic $\overline{V}_i$ to predict $v_i^{UCB}(s)$ and $v_i^{LCB}(s)$, instead of V_i.

4.4.2.2 Creating a new module

When drift occurs such that the returns are better than anticipated, we expect that this corresponds to the case that the policy has simply improved in the current setting, as intended by standard module policy training. In this

setup, we just update the anchor to improve expectation. However, the case of negative drift, where the UCB falls below the estimate of the anchor, requires a different strategy. This situation occurs when the behavior (policy) starts under-performing expectation, which can occur when the task has been changed and the old policy is no longer as effective as it had been. What we do in this case is create a new module that is a clone of the one that was activated. We empty the replay buffer and update the anchor at the time of creation of the new module. The goal is for the new module to be activated by the new setting while the old one continues to be activated by the old setting, splitting the input space to more effectively handle the two desired behaviors that are not well handled by a single policy.

4.4.2.3 Merging modules

To prevent unlimited memory consumption, we limit the the total number of modules in our ensemble by merging modules. To execute a merge, we start by finding the two modules in the ensemble that are closest by the L2 distance between frames averaged from a sample of trajectories from the replay buffers. We then keep the more frequently used module and drop the less frequent module from the ensemble. Before dropping, we combine the replay buffer of the two modules and run a module update (Section 4.4.1) on the combined module.

Note that in combining the replay buffers of the two modules we use the reservoir sampling technique from CLEAR [150]. We maintain a reservoir value for each trajectory, defined as a random value between 0 and 1, that allows every trajectory to have an equal chance of being stored in the buffer, regardless of when it was collected. The trajectories from the module being dropped are added to the replay buffer of the module being kept using the reservoir values that were originally generated.

4.4.3 Implementation Details

Leveraging CLEAR and IMPALA. We have chosen to base our modules on IMPALA-based CLEAR as implemented in CORA, as it allows us to get several useful features for free: a. Learning is done efficiently, in a highly parallel manner. b. The policy is updated using vtrace, an effective credit assignment method, as described in [39]. c. The CLEAR replay buffers are maintained using reservoir sampling. d. CLEAR provides auxiliary losses that maintain consistency of both the policy and critic with the replay buffer.

Model Architecture. The base implementation uses the Nature CNN model from [116]. We augment the baseline network with 2 hidden linear layers of dimension 32 with ReLU nonlinearities to increase its representational capacity. All other hyperparameters for the experiments are provided in the available code.

Parallelism. By collecting data from multiple environments in parallel, training is considerably faster, but it requires us to make one key assumption: the activated module must be guaranteed to be applicable to all actors being run at the same time. This requires that all actors be running the same task.

4.5 Experiments

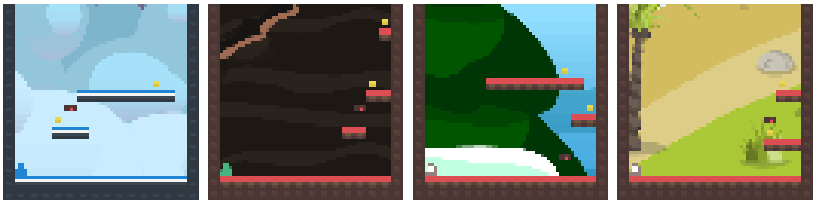
Task sequences. We choose three procedurally-generated game environments (Climber, Miner, Fruitbot) from Procgen [31]. We construct three task sequences using each of these game environments, by isolating sequences of levels that are likely to cause catastrophic forgetting and where approaches like CLEAR would perform poorly on. We selected four levels for Climber and Miner. For Fruitbot, we added an easier fifth level at the start as a simple curriculum. We run each set of levels for three cycles, to see how learning evolves as the levels are seen again. The first frame of the selected levels are visualized in Figure 4.2.

Baselines. We compare our approach to three of the baselines described in Chapter 2.6. We also perform an ablation showing the importance of the dynamic ensemble compared to a static set. The baselines we selected are:

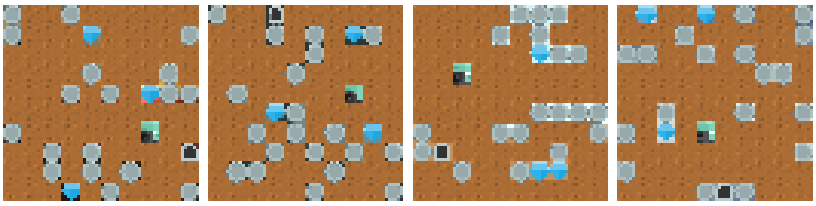
- **CLEAR** In addition to comparing to CLEAR with the default number of parameters (the same as each module in the SANE ensemble), we also compare to a version of CLEAR with as many total parameters as we use in our SANE ensemble. We refer to this as "CLEAR 8x".

  Worth noting is that while SANE takes around 14 hours to run our Fruitbot sequence and standard CLEAR takes around 10 hours, these larger models take longer to run: CLEAR 8x took 4.5 days. We would have liked to compare to a CLEAR 32x as well, but such an experiment was on track to take more than 2 weeks. This exemplifies another benefit of SANE: the effective usage of more parameters without such a dramatic increase in runtime.

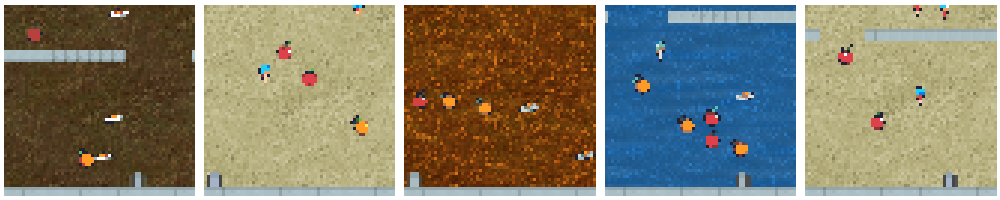
- **Elastic Weight Consolidation (EWC)**



(a) Climber Levels



(b) Miner Levels



(c) Fruitbot Levels

Figure 4.2: The first frame of each sequence of levels used in our experiments.

- **Progress & Compress (P&C)**
- **Static SANE Ensemble** To validate the utility of our dynamic SANE ensemble, we compare to a SANE ensemble that is static: all modules are initialized upfront, and no creation or merging occur.
- **SANE Oracle** We also compare against an Oracle version of SANE, where each task has its own pre-specified module, which is looked up by task ID.

Experimental Setup and Metric. All implementations for baselines are based on those described in Chapter 3. We use Continual Evaluation to generate plots for each task in the task sequence, which show how well each task was learned and how well each task was remembered. For fairness of comparison we hold constant the number of replay frames each method has access to in total, at 400k frames. Every method was run for 5 seeds, and the mean and standard

error of the mean are shown in the graphs. Gray shaded rectangles show when the agent trains on each task. We also report the Forgetting metric introduced in Chapter 2.3. Hyperparameters and further details for the methods used are given in [137].

We compute the Isolated Forgetting statistic for each seed and take the average across tasks, multiplied by 10 for readability. We report the average and standard error of the mean across seeds for the Forgetting summary statistic. See Chapter 2.3 for more details.

4.5.1 Results

We present the Forgetting summary statistics for all methods in Table 4.1 and the Continual Evaluation graphs, which present the average and standard error of the mean of the returns received from the environment versus steps taken in the environment, in each section.

Climber. First we demonstrate SANE on Climber, a side-view task where the agent must ascend a series of platforms while collecting coins and dodging bats. The selected levels are particularly challenging because avoiding the bats requires relatively precise timing; a slight decay in policy performance results in significantly reduced reward.

We start by analyzing the Continual Evaluation results in Figure 4.3. SANE and Static SANE both learn the tasks, but we can see that our dynamic model consistently learns and remembers, while Static SANE overall shows more inconsistent performance, doing particularly poorly on Envs 0 and 2. Both versions of CLEAR learn the tasks but readily forget them, indicating that SANE is not improving by merely adding more parameters. EWC has mixed results; it does worse than SANE uniformly on all cycles of Env 0 and the first cycle of the other Envs, but approximately ties it on the other cycles of Envs 2 and 3, and exceeds it on the other cycles of Env 1. P&C largely fails to learn the tasks at all, with some exception on Env 2.

These results are further validated by looking at the Forgetting summary statistics presented in Table 4.1. By this metric EWC does the best, likely aided by poorer learning during the first cycle of Envs 0 and 1 and the particularly good later performance on Env 1. Of the four methods that learned all tasks immediately (SANE, Static SANE, CLEAR, and CLEAR 8x), SANE exhibits the least forgetting.

Miner. We additionally demonstrate SANE on Miner, a task where the agent must dig through dirt in two dimensions, collecting diamonds and going to a specified end-goal without getting crushed by rocks.

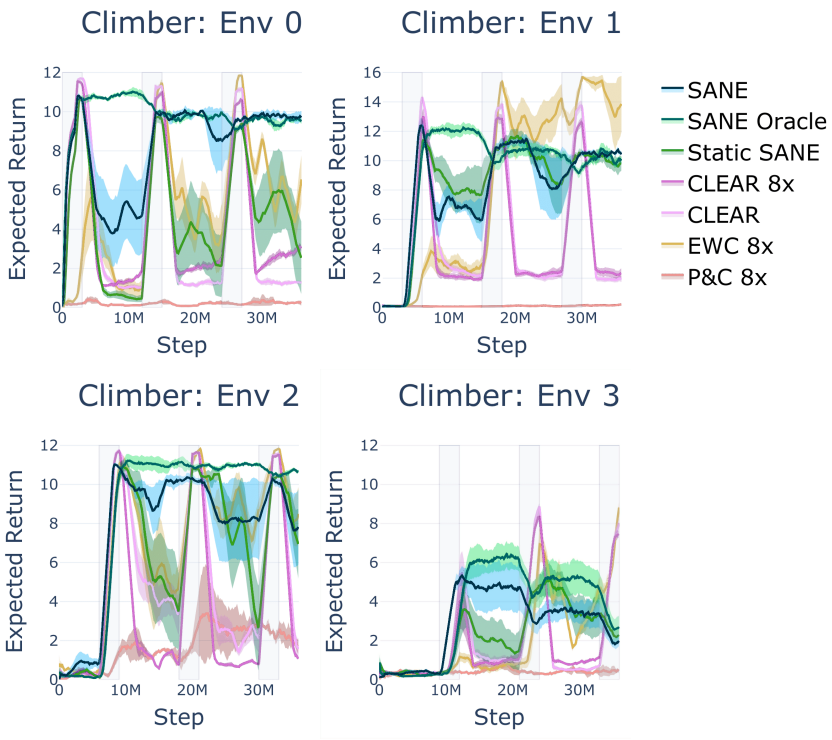


Figure 4.3: Results for Continual Evaluation on the Climber sequence of tasks. We observe that SANE consistently learns and recalls the tasks. Gray shaded rectangles show when the agent trains on each task.

Continual Evaluation results are shown in Figure 4.4. SANE overall outperforms the baselines on the first three environments; we see CLEAR and EWC learning and forgetting, static SANE showing more recall than CLEAR but less than SANE, and largely little learning from P&C with the exception of Env 1. However, on Env 2 one of SANE's seeds fails to learn the task, and on Env 3 all seeds did. Perhaps CLEAR's larger buffer effectively provides more exploration, as there is more randomness amongst the batches selected to be trained upon.

Table 4.1 again demonstrates numerically these qualitative results. We see that of the methods that learned the tasks, SANE not only did the best, it also exhibited some backwards transfer (negative forgetting).

Fruitbot. The final Procgen sequence we use is based on Fruitbot, where the environment continuously scrolls and the agent must move left and right to collect fruit, avoid vegetables, and make it through gaps in the wall. Continual Evaluation results, shown in Figure 4.5, are less clear-cut than the previous two

	SANE	Static SANE	CLEAR	CLEAR 8x	EWC 8x	P&C 8x	SANE Oracle
Climber	-2.2 ± 0.7	-2.5 ± 0.7	-5.6 ± 0.2	-5.4 ± 0.4	2.4 ± 0.1	-1.2 ± 1.1	0.5 ± 0.1
Miner	1.1 ± 0.4	-0.4 ± 0.6	-3.5 ± 0.4	-3.9 ± 0.3	-2.5 ± 1.1	0.0 ± 0.2	1.5 ± 0.2
Fruitbot	-2.7 ± 0.3	-4.0 ± 0.2	-5.5 ± 0.5	-4.5 ± 0.4	-3.3 ± 0.8	-1.7 ± 0.3	0.1 ± 0.2

Table 4.1: Isolated Forgetting ($-\mathcal{F}_{\mathcal{I}}$) summary statistics for all experiments. EWC and P&C exhibit little forgetting because they also exhibit little learning. Of the methods that learned the tasks, we see SANE performs best.

experiments. SANE clearly exceeds baselines on recall on Envs 1 and 3, but remains comparable to the CLEAR 8x baseline on Envs 2 and 4, and struggles on Env 0, only exceeding the baseline in the final cycle. Furthermore for the most part SANE receives a lower maximum score than the CLEAR baselines, with the exception of Env 3. Table 4.1 shows that despite the mixed qualitative results, SANE again exceeds baselines quantitatively.

4.6 Analysis of SANE

We generate two additional types of plot to help us analyze our SANE ensembles. The first is a "Module ID" plot. On creation we assign every module a unique identifier: an integer that increases per module created. This ID is constant through the lifetime of the module, including when other modules get merged into it. We can plot what module is active by plotting its module ID. This allows us to see when there are periods of rapid creation (steep regions of the graph), when older modules are re-used, and when modules are being stably activated.

The second plot is a lineage plot, showing a graph that represents the history of the ensemble, with each node in the graph representing a module. A blue line indicates that one module spawned another, and a red line indicates that a module was merged into another. Light blue nodes represent modules that are current available to be activated at the current time. An example lineage plot is shown in Figure 4.6.

4.6.1 Single Run: Climber

We start by analyzing a single (non-hand-picked) run of SANE in Climber, to demonstrate the dynamics of learning in a simple environment where behaviors are readily separable. In Figure 4.7 we've aligned a graph of the ID of the currently active module with the reward received at that time.

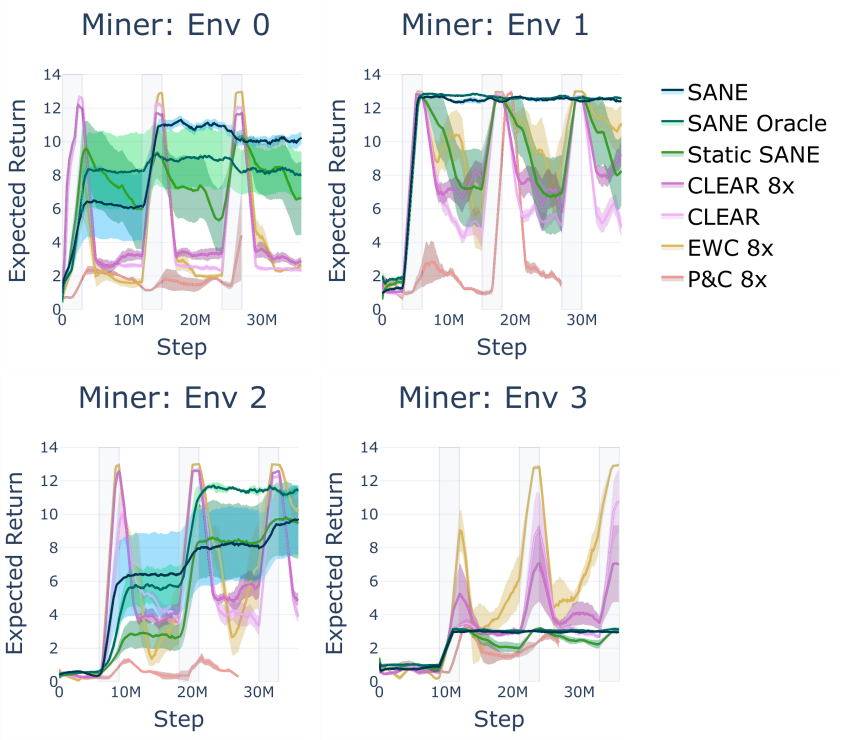


Figure 4.4: Results for Continual Evaluation on the Miner task sequence. We again observe that SANE improves on the baselines at recall across the tasks. Gray shaded rectangles show when the agent trains on each task.

We can see the desired behavior in this case: several new modules are created (a sharp increase in module ID is observed) as performance successively fails to meet expectation, until a suitable module is created. Additionally, we can observe that before a task is trained upon, it is likely to use the best module for the current task. E.g. Env 3 uses Env 0's active module (module 0) for the first 3M steps, then Env 1's module (15) for most of the next 3M, then Env 2's module (18) for the next 3M, until during its own training period drift is detected and a custom module is created.

It is also worth remarking on the decline in Env 3's performance, which occurs particularly while the task is being trained. It occurs while a consistent module is being activated, so is not related to the ensembling behaviors of module creation or merging. Observing the behavior indicates that the agent is jumping into a bat rather than avoiding it, so it would seem it is overfitting to the jumping behavior, possibly as a result of the decreased replay buffer size.

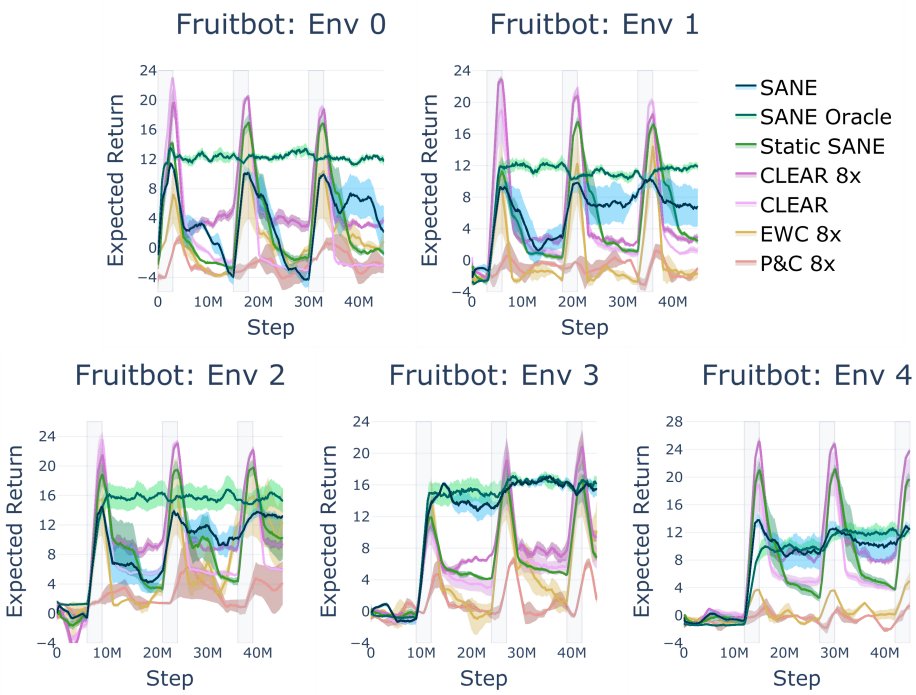


Figure 4.5: Results on the Fruitbot sequence. SANE performs particularly well on Envs 1 and 3, comparable to CLEAR 8x on Envs 2 and 4, and struggles some with Env 0. Gray shaded rectangles show when the agent trains on each task.

4.6.2 Fruitbot Analysis

Fruitbot performed least well of our experiments, so we dive in further to understand the dynamics at play.

Module Count Ablation. We first discuss the difference in expected return when we vary the number of modules for SANE, as visualized in Figure 4.8. Overall, we observe that the fewer the modules, the higher the maximum scores received. The one exception is Env 3, where 8 and 16 modules both receive comparable scores. However, in general the fewer the modules the more forgetting is observed as well. This is particularly noticeable on Envs 0, 1, and 4, with more ambiguity on Envs 2 and 3.

As we observed in the analysis of Climber, when a new task is switched to, we don't create just a single module. Rather, the critic steadily learns to adapt to the new task; each time it passes the v^{LCB} threshold, a new module is created. Once the maximum number of modules has been reached, this triggers a merge. Since a merge combines the replay buffers of the two modules, when

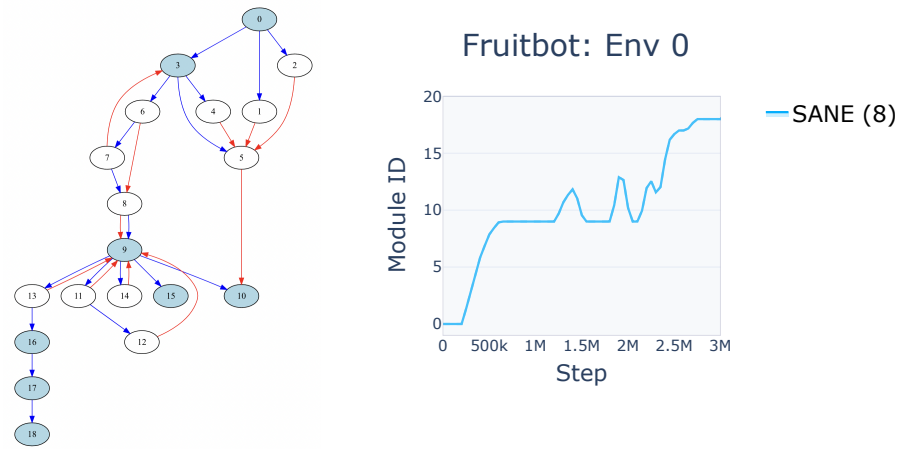


Figure 4.6: An example Lineage plot, showing how nodes are created and merged while training on Env 0 of Fruitbot.

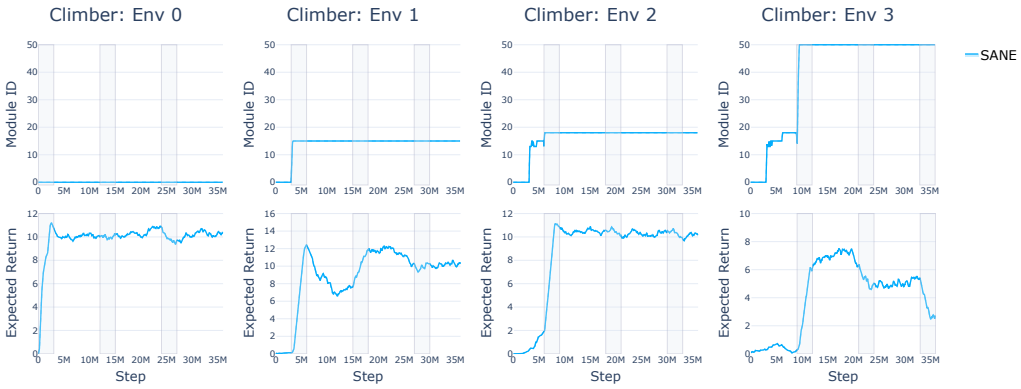


Figure 4.7: Module ID plot and Expected return plots aligned by timestep, to show module activation during a single run of Climber. Gray shaded rectangles show when the agent trains on each task.

two "compatible" modules are merged, the resulting policy is more robust than that of the individual modules. However when two modules representing conflicting behaviors are merged, we see a reduction in performance. Taken together, this means that as merging is occurring, more modules in the ensemble will generally be more stable over time, but might be slower to learn. We see a concrete example of this in Section 4.6.2.1 below.

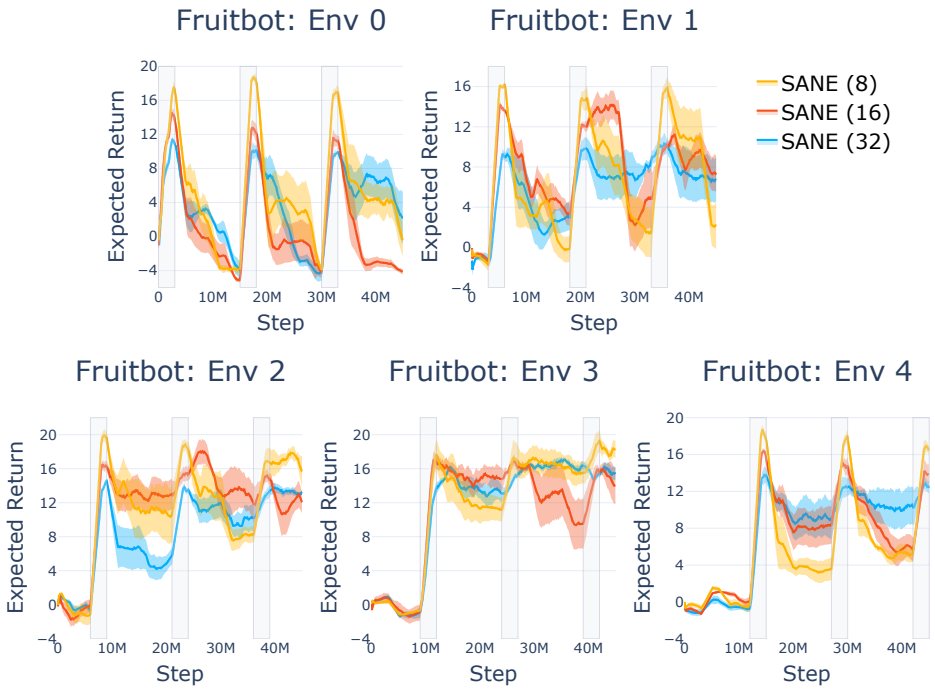


Figure 4.8: Comparison of SANE variations on the Fruitbot task sequence. The number in parentheses indicates the number of modules in the ensemble, ie. SANE (8) has 8 modules. Gray shaded rectangles show when the agent trains on each task.

4.6.2.1 Single Run

Here we analyze a single run of Fruitbot, which allows us to see in more detail the dynamics of SANE. We use an ensemble with 8 modules to simplify analysis. We focus on three important points, labeled A, B, and C in Figure 4.9. In all three cases the module the Environment is using switches to an older module.

At A (21M timesteps), Env 1 switches from using Module 193 to Module 10. By analyzing the Lineage plot (not shown here due to its large size) we see that 193 merged into Module 150, which then merged into Module 10. Thus the continued high performance on this task can be explained by a successful sequence of merges.

At B (27M timesteps), Env 2 switches from using Module 252 to Module 9. In this case, Module 252, which is a direct descendent of Module 9, merged into Module 197. Module 197 is at this point still a module available in the ensemble, meaning Env 2 began to activate a high-performing previous module (Module 9) instead of the result of the merge, implying that the critic value of

252 decayed as a result of its merger into Module 197. However, performance was rescued by return to a previous module and performance remains high.

At C (30M timesteps), Env 4 switches from using Module 261 to Module 10. Module 10, as we saw in case A, is a module that is well-suited for Env 1. In this case, Module 261 merged into Module 262, a descendent of Module 9, which as we saw in case B is well-suited for Env 2. Essentially, our 8 module ensemble lacks the capacity to adequately represent all of the behaviors necessary for this sequence of tasks, and start combining policies destructively, resulting in forgetting. This is mitigated, as we saw in Figure 4.8 by introducing more modules into our ensemble.

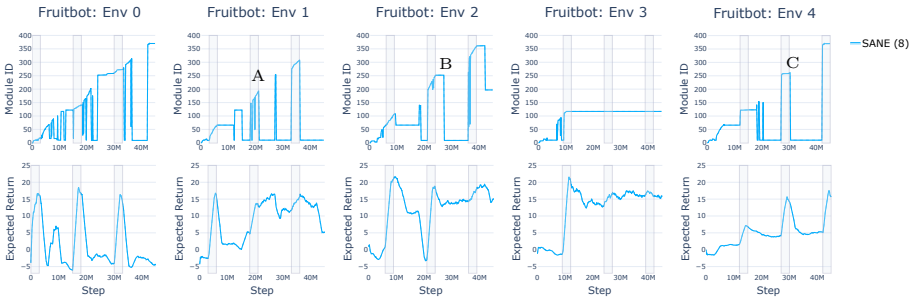


Figure 4.9: The Module ID and Expected Return plots for a single run of Fruitbot, aligned by timestep to see how modules are getting used and created while Fruitbot is training.

4.7 Summary

Inspired by the fact that catastrophic forgetting is caused by updating all neurons in a network for all tasks, we propose the creation of self-activating modules to break up a network into modular components that only get updated when they are used. Our experimental results suggest that a dynamic ensemble, which creates modules as necessary and merges them to conserve resources, performs better than a static ensemble where all modules are created up-front. By combining these two novel features, we present SANE (Self-Activating Neural Ensembles) for continual reinforcement learning. We demonstrate SANE on sequences of Procgen levels that prove particularly challenging for the current state-of-the-art (CLEAR), showing that our method reliably improves the mitigation of catastrophic forgetting. Furthermore, we present a thorough analysis of SANE, showing how modules are created, used, and merged on individual runs of Climber and Fruitbot, to provide a more comprehensive view into the system.

Chapter 5

SANER: SANE for Robotics

5.1 Introduction

Robotics is a challenging field for even a single task, as collecting data is time-intensive, supervision is costly [127], and significant randomness can cause undesirable damage to both the home and the robot. Additionally, large amounts of resetting would be a burden in a home setting [40, 207]. Finally, the ability to generalize is critical since real-world settings are never exactly the same; sensor noise, error in robot control, shifts in lighting, and more all result in variation, even in otherwise static scenes. As a result, learning on a real robot for a large enough set of tasks to validate new continuous learning methods has been out of reach. Existing work in the continual learning setting for robotics is limited and largely speculative [99].

To resolve these issues, we propose SANER [1], an adaptation of the SANE algorithm for the robotics imitation setting, which can be used to learn an ensemble of new skills given a handful of unstructured demonstrations of different tasks. To be sufficiently general for robotics, SANER must use a policy that is sample-efficient and robust to noise. Additionally, SANER introduces several significant modifications to enable learning from imitation.

The policy we introduce with SANER is a simple, highly sample-efficient point-cloud-based policy network we call Attention-Based Interaction Policies (ABIP). ABIP is based on PointNet++ [140], and was designed to efficiently learn via imitation. ABIP takes inspiration from prior work on perceptually-grounded action spaces for robots [71, 117, 167, 189, 198, 199], which has shown better data efficiency and robustness than directly learning to predict motor commands. Using ABIP, our agents are capable of acquiring new skills with only

[1] Code is available at https://github.com/AGI-Labs/continual_rl

Figure 5.1: On the left we show the entire set of training settings for four tasks: picking and placing a bottle into and out of a sink, and opening and closing an oven. On the right are our evaluation settings. We show how continuous learning techniques like CLEAR [150] and SANE can be extended to a low-data, learning-from-demonstration setting on a real robot, and can succeed on tasks that vary considerably from the original demonstrations.

two demonstrations, minimizing burden on the end-user and robot alike. While we present ABIP as part of SANER, we believe that its sample efficiency and generalization capacity make it a useful component of any continual learning system for use with robotics.

We evaluate SANER on four kitchen tasks using a low-cost mobile robot: two pick-and-place tasks and two tasks manipulating an articulated object. We demonstrate our method's ability to generalize to unseen object locations, unseen object types, and to clutter, all with very little data. We compare our new method to two other continual learning methods, modified to use ABIP: CLEAR [150] and EWC [86]. We demonstrate that SANER is more effective at both learning without forgetting and forward transfer in the robotics domain, and that ABIP is a useful component for learning general interaction policies from a small number of examples.

To summarize, in this paper we show for the first time how to modify existing state-of-the-art continual learning methods to a few-shot, real robot imitation learning domain, and describe the necessary algorithmic changes and evaluations. We also describe the requirements on a policy architecture which will make these experiments feasible, and provide a first version of such an architecture in ABIP.

5.2 Related Work

Relevant prior work spans several fields, including continual learning [128], robotics, and imitation learning. We look both at prior work in continuous learning, reinforcement learning, and in learning generalizable robot policies.

Continual learning for robotics. Continual learning began as the study and mitigation of catastrophic forgetting [143, 148]. It has since expanded to include forward transfer [105] and maintaining plasticity [114], and more recently, generalization and sample efficiency [138]. In this work, we focus on generalization and sample efficiency in the context of visuomotor policy learning, in addition to forgetting, as these are critical pieces for robotics.

While a significant amount of work has focused on continual learning for supervised image classification [92], this too has expanded with time to include unsupervised learning [11] and reinforcement learning [86]. Continual learning in robotics has received some attention [99], but while there have been a few works in simulation [50, 186, 202], little has been put into practice. Key exceptions include work by [154], which uses Progressive Networks [153] to apply continual learning to the sim2real problem; [48] utilize deep generative replay for the domain of continual imitation learning for robotics, using 15 demonstrations per real-world task, but their method requires using generative models to create whole new datasets in the new environment in order to avoid forgetting. [28] propose an approach which constructs policy mixtures for continuous control tasks, but can't learn visuomotor policies or generalize to different scenes.

One other route to continual learning for robotics would be, instead of learning visuomotor skills, to focus on grasp estimation and planning. In general, this sort of pipelined approach is brittle, failing due to occlusions or interference [78], and existing open-set grasp estimation methods do not do task-oriented grasping [119]. [104] proposed a continuous object detection dataset, which could have some overlap for our methods, but uses only RGB data, which is far less useful for robust grasping [119]. Others have used continuous learning for object detection on real robots [9, 10], but have not looked into reactive skill learning. However, ideas from continuous learning for object detection could be used in conjunction with SANER in the future to improve semantic learning and generalization.

Modular RL for robotics. SANE, on which SANER is based, is an ensemble method based on the idea that having separate modules side-steps the problem of catastrophic forgetting. The idea of re-usable robotic skill libraries from demonstrations has been seen in [15, 35, 132, 173], amongst many others [202].

Additionally, there has been work to apply modular methods to the continual robotics domain, such as in [113], which looks at modular reinforcement learning in the simulated RoboSuite [208] environment, and [17], which looks at lifelong reinforcement learning in a variety of simulated environments. These are relevant but not directly comparable to our work because much more data is available to train policies in these settings. Others have noted zero-shot generalization as an important problem for RL, for example [85] looks at avoiding overfitting to training environments through various means. In our case, we avoid overfitting through a mixture of augmentation and design of an attention-based, perception-driven robot policy.

Perception-driven robot learning. Recent work in robot learning focuses on building general, multi-purpose policies which can scale to a variety of scenarios based on sensor data. Much of this work has likewise focused on big-data settings. In Say-Can [3] and RT-1 [20], for example, the authors use a large dataset to train a multi-task, language-conditioned transformer model. Given that these works require large amounts of data to train, they are not necessarily suitable for online teaching of robot skills in a home environment.

Conversely, there is a thread of work which functions on much smaller amounts of data but still achieves strong performance [63, 71, 166, 167, 198, 199]. These works focus on a perceptual action space, where actions taken directly correlate with visual features, but often focus on a 2D vision of the world as a result [63, 166, 198, 199].

Recent work extended this sort of approach to deal with 3D environments more suitable for robotics tasks [167], but still uses very large transformer models that take a long time to train. An alternative is proposed by work like Where2Act [117] and VAT-Mart [189], which learn a point-cloud based policy which predicts an *interaction point* and a corresponding trajectory. These are based on the powerful Pointnet++ [140] backbone, which provides convolutional-equivalent operators that work in 3d space. However, these policies are *open-loop*, meaning that they cannot recover from failures. We build off this work, using a similar representation but modified to be slightly more general.

5.3 Method

One goal of this thesis is to empower robots with a growing library of skills, where modules can be easily created and learned. To achieve this, we propose SANER, an adaptation of SANE for Robotics.

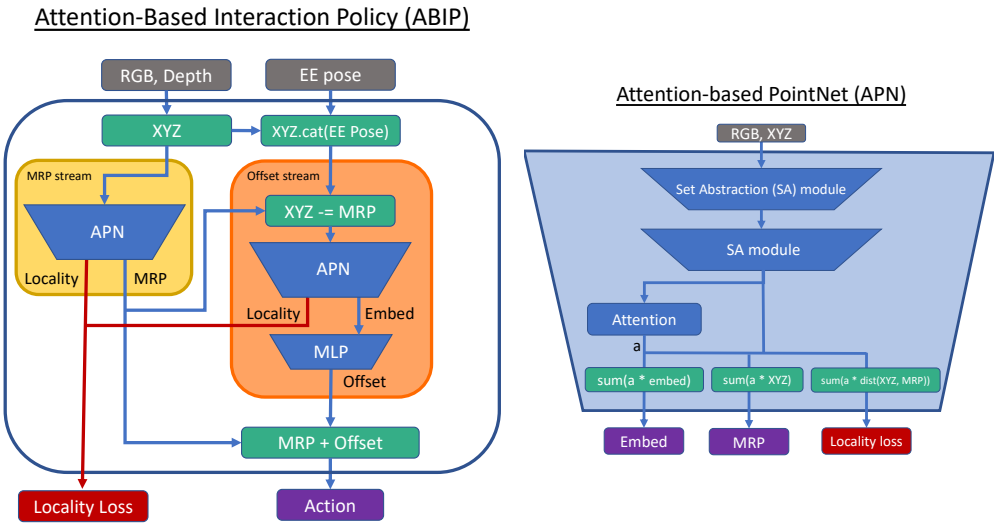


Figure 5.2: Attention-based interaction policies (ABIP). We build an architecture which ignores variations and distractors by first predicting a *Most Relevant Point* (MRP), then predicting an offset from this point. ABIP is trained with a *locality loss* which penalizes attending to points far from the robot's current position.

SANE is an algorithm for the automatic creation and re-use of skill modules; we provide an overview in Section 5.3. To be suitable in the imitation learning setting, SANER makes several key changes. We break the necessary modifications up as follows: first, we discuss the new policy architecture used by each module, *Attention-Based Interaction Policies* (ABIP), in Section 5.3.2; second, we discuss changes to the critic in Section 5.3.3; and third, we discuss changes to module creation in Section 5.3.4. Additionally, we modify two other continual learning methods, CLEAR and EWC, to use ABIP as well. We describe these adaptations in Section 5.3.5.

5.3.1 Preliminaries

For the experiments presented in this paper, the state is composed of an RGB-D image and the robot joint state, and the action is specified as the position and orientation (as a quaternion) of the end effector in world coordinates, plus the state of the gripper as a continuous variable in the range (-0.2, 0.2). Additionally, we include the fraction of the task completed in the action, which is used to determine the end of the episode. Refer to Chapter 2.3.2 for more details. use $N = 4$, $k_{demo} = 2$, k_{unseen}, and $T = 10000$.

5.3.2 Attention-Based Interaction Policy (ABIP)

Intuitively, in a cluttered and constantly changing setting like a human home, there will be many irrelevant details in the background of any demonstration that a user provides to SANER. Therefore, we split the action prediction problem into two steps: (1) we predict a *Most Relevant Point*, or MRP, which tells us which region of the world the policy must attend to; and (2) we reactively predict *actions* which determine where the robot should move in relation to that MRP: for example, how to approach the handle of an oven and when to close the gripper to grasp it.

These two operations are performed sequentially using a modified PointNet++ [140] model that we refer to as *Attention-based PointNet* (A-PointNet), shown in Figure 5.2. The *MRP Predictor* can then be agnostic to the position of the robot, instead focusing on the features of the object relevant to the overall task, while the *Action Predictor* can learn to focus on features relevant just to what the next action should be. For example, the MRP Predictor might learn to focus on the handle while the Action Predictor focuses on the angle of the oven door.

Image Pre-Processing. First we convert the RGB and depth images into a point cloud. We augment the point cloud of the current timestep with our context c, the point cloud from the beginning of the episode. This aids both in combating occlusion, as well as in disambiguating between similar observations that occur during different trajectories. To reduce compute, we crop the working area to 1m, and down-sample using grid pooling, with a resolution of 1cm for the current timestep and 2.5cm for the context. Specifically, we select a random point in each voxel, to reduce overfitting.

5.3.2.1 Attention-Based PointNet (A-PointNet)

Our Attention-Based PointNet module (A-PointNet), a core component of both our MRP and Action Prediction streams, augments PointNet++ via the addition of an attention mechanism, allowing us to learn to ignore irrelevant details of the scene.

In more detail, A-PointNet passes the point cloud through two set aggregation modules that process information at different scales, as in PointNet++ [140]. This allows us to harness both structural information from the scene as well as the finer details necessary for manipulation.

The output from the second set aggregation module [140] is a reduced point cloud P. We concatenate the embedding, m_i, and position, p_i, of each point i and pass these into an attention network, which is an MLP with output

dimension 1, and softmax the resulting values to compute the attention value, a_i. This attention is then used to produce both a weighted average position, $\bar{p} = \sum_{i \in P} a_i p_i$ and a weighted average embedding, $\bar{m} = \sum_{i \in P} a_i m_i$.

This version of attention can be seen as analogous to the spatial attention maps used by CBAM [187], using set aggregation modules in place of convolution. The context of each point, which determines its importance, is determined by the structure of points in its neighborhood.

5.3.2.2 Dual-Stream Architecture

Most Relevant Point (MRP) Prediction. Ideally, finding the most relevant point for a particular task would allow us to focus only on high-level task relevant information, ignoring background objects and distractors. To encourage this, we remove points from the point cloud in a rectangular prism surrounding the known location of the end-effector. Due to the fact that when using only a small number of demonstrations the current position of the end-effector is a highly accurate signal for the next position; as discussed in Sec. 5.5.2, an MRP predictor without this augmentation tends to overfit to these points. This point cloud is then passed into an A-PointNet module to compute a weighted average position, our Most Relevant Point (MRP).

Action Prediction. We then predict actions relative to the MRP. In this case the position of the end-effector is highly relevant to the computation of the next desired action. Therefore, we again remove the points around the end-effector, but now we add a rectangular prism in the position and orientation of the end-effector. This is because the end-effector is often occluded, and here we want to ensure it's available for use.

As before, this point cloud is then passed into a separate A-PointNet module. The resulting $\bar{p}$ and $\bar{m}$ are concatenated, along with the current state of the gripper, then passed into four separate MLPs, which compute the position offset (Δp), rotation (as a quaternion), and gripper state of the end-effector, as well as a prediction for the fraction of the task completed. The offset is added to the MRP to compute the final end-effector position.

5.3.2.3 Training ABIP

Imitation Loss. To train the policy, we use the action from the demonstration to supervise the position, gripper state, and completion fraction using a mean-squared error loss. We represent orientation as a quaternion and use $1 - (q_{true} \cdot q_{pred})^2$ as the loss, based on [64].

Offset Loss. We additionally compute a loss that drives the computed offset toward 0, to further encourage the MRP to encode information relevant to the interaction: $L_{offset} = ||\bar{p}_{off}||_2$

Locality Loss. We encourage generalization with a *locality loss* which encourages relevant points to be close together, defined in Equation 5.1:

$$L_{local} = \sum_{i \in P} a_i ||p_i, \bar{p}_i||_2 \tag{5.1}$$

Since these points must also contain information relevant to the task, this allows for a natural convergence of attention around a salient point in the observed point cloud. In effect, this allows us to more quickly learn to ignore irrelevant features of the environment, such as walls or the table surface, allowing for improved generalization.

Removal of Cloning Losses. While SANE uses the policy and value cloning losses from CLEAR, which help maintain old behavior in the presence of new data, with SANER this is unnecessary, as we are already training on the true actions observed from the demonstrations.

Removal of Policy Augmentation. Unlike in SANE, SANER trains the policy only on new data (B_{new}), not a batch augmented from the replay buffer (B_{aug}). This allows the module to specialize the new policy to the new setting as quickly as possible. This is related to policy freezing, discussed in Section 5.3.4.

5.3.3 Critic

SANE uses a reinforcement learning critic to estimate two values, a score and an uncertainty, that are used to determine which module to activate and to detect when drift has occurred. For SANER, both of these need to be changed to work in the imitation learning setting.

Critic Architecture. In addition to the ABIP policy network, SANER uses a separate A-PointNet encoder for the critic, plus a 3 layer MLP with hidden dimension 64 to predict the value and uncertainty. Unlike in SANE, SANER does not share weights between the actor and the critic. Note that the critic does not utilize a locality loss; we found that earlier tasks might focus too strongly on features that were insufficient signals for future tasks, and the critic's performance would suffer.

Activation Score. Conceptually, the goal of SANE's activation score is to evaluate how well each module will perform in the current context. When

applied to the reinforcement learning setting, the standard critic is a natural choice, as it is already designed for that purpose. However, in the imitation learning setting, the choice is less clear.

An activation score should: (1) always be greater than 0, to minimize unnecessary module creation at the beginning of training; (2) have a consistent maximum value, for consistency between tasks and to make hyperparameter selection easier – we choose a maximum of about 3; and (3) be highly sensitive to policy errors on about the same scale as the hardware and tasks permit, and less sensitive to the difference between large errors in action prediction.

Our last criteria is both the most important and the least trivial. Say we have a module that succeeds at a particular task with an error in the predicted-end effector position of about 1cm. If updates to the policy that induce an error of 2cm would cause the module to start failing at the task, then the score function should drop considerably. However for the same task, if a different module has a policy error of, say, 18cm, then an increase to 20cm should have little effect on the score; both are clear failures, and should be near 0.

While there are a number of options for functions that meet this criteria, we found it most straightforward to compute a target score based on the distance between the true action and the predicted action. Specifically, we use: $s_{target} = softplus(\beta)(a||p_{true}, p_{pred}||_2) + b$, where p_{true} is the true end-effector position, p_{pred} is the current prediction of the policy, and a, b, and β are constants. We use essentially this same metric for the orientation and gripper pose as well, and sum the results to achieve our final score. More details on the selection of these constants is described in [136]. We then train the critic to predict this target score using an L1 loss.

Uncertainty Estimation. In imitation learning, the policy learns much more quickly than in reinforcement learning, due to the fact that RL relies upon credit assignment instead of direct supervision, causing learning to take orders of magnitude more time. While faster training is generally beneficial, we found that our prediction of uncertainty tended to lag behind the performance of the policy, particularly at the beginning of a task when the policy is changing rapidly. To resolve this, we introduce a multiplicative factor on the uncertainty, for both the new node and the source node, that increases when drift is detected, and decays as the module is used. This gives the uncertainty estimator more time to converge accurately.

More specifically, we defined our modified uncertainty as $u' = (1 + f) * u$, where u is the standard uncertainty predicted by SANE, and f is our uncertainty factor. When a drift event is detected and a node is created, f is incremented

by k_s for the source node and k_n for the new node. When a node is activated, its f decays according to: $f_{t+n} = f_t * \gamma^n$, where n is the number of timesteps seen by the module during activation.

In SANE, uncertainty is used for node activation and upper bound estimation. We can therefore define f_{act} and f_{ub} separately, with separate k and γ values for each. This is useful because, for example, while we want the source node's uncertainty to be reflected in its upper bound to mitigate unnecessary node creation, we do not need the source node to be more likely activate. Uncertainty is also used for lower bound estimation; however, a sensitive lower bound poses little issue, and we therefore neglect f for this case.

Replay augmentation. In reinforcement learning it is not feasible to mine negative examples for one module from the others, because it is challenging to estimate what outcome a policy will have in an environment without actually doing it. However, with imitation learning, mining negative examples is trivial. We leverage this advantage by, at each activation step, giving the active module a 10% chance to exchange a few samples with each other module.

Handling noisy predictions. While ABIP is training, the point cloud augmentations, particularly the random downsampling before the first set aggregation module, can result in critic predictions with significant noise. We found that the slow critic SANE uses to smooth training did little to impact this source of noise. We opt to remove the slow critic, and instead average all critic computations over 5 runs of the same observation.

5.3.4 Module Creation

The main way that catastrophic forgetting is mitigated in SANE is by the creation of new modules, as described in Chapter 4, which allows one to persist prior behavior, while the other learns the new task. Creation occurs when an active module's critic predicts an upper bound on predicted value that is lower than the prediction of its anchor. With imitation learning however, once that has happened, the policy has already changed considerably, and significant forgetting has already occurred.

We mitigate this issue in SANER with a naïve strategy: once a module is cloned, we reset both the policy and the critic to their states at the last time the anchor was updated, which is effectively our last known good state. Furthermore, we freeze the policy, preventing it from training further.

5.3.5 Adapting Existing Baselines

Existing continual-learning methods require a few changes to be applied to the learning-from-demonstration context we want to use for our home robotics setting. In particular, we adapted CLEAR and EWC using same change in loss made for SANER: using the changes to the loss from Section 5.3.2.3 in place of V-trace, with the exception that we do not remove policy augmentation for CLEAR, and it is not applicable for EWC.

5.4 Experimental Setup

We demonstrate continual learning in robotics in a kitchen environment by doing the following 4 tasks: picking up a bottle and putting it in the sink, taking a bottle out of the sink, opening a toaster oven, and closing a toaster oven. For each task we collected and train on only two demonstrations. Further details on the experiments are provided in [136].

We perform three evaluations: first, we demonstrate the generalization ability of ABIP on each task independently, second we ablate our key novel contributions on ABIP, and finally we evaluate our continual learning methods on the tasks trained in sequence.

Robotic platform. We utilize the Hello Robot Stretch[2] [81] in all our experiments, shown in Figure 5.1. The Stretch robot is a "low cost" robot, composed of an arm that can independently move vertically and horizontally, a wrist with 3 rotational degrees of freedom, and a pinching gripper. The base is also capable of motion, but for our experiments we keep the robot stationary, leaving this for future work. The camera used to collect observation data is a RealSense D435 mounted on the head of the robot, and remained stationary during all episodes. Demonstration collection, training, and inference were done off-robot on a dedicated desktop, and communication with the robot was done via ROS.

Task selection. The tasks were chosen to highlight a variety of kitchen-relevant sub-skills, under the constraint that there are 5 degrees of freedom in the motion of the arm. The two primary skills exercised are pick-and-place and manipulation of an articulated object. The entirety of our train and test settings can be seen in Figure 5.1, where the opening and closing of the oven are two tasks using the same settings. Notice that with only small perturbations of the object of interest, we observe significant ability to generalize.

[2] hello-robot.com/

Specifically, picking the bottle from the sink requires a precise approach, as pushing the bottle is risky and may tip the bottle. During our evaluations, we move the bottle in all three dimensions, to evaluate the agent's ability to generalize accurately. The choice of a bottle partially filled with water is convenient, as it allows for tuning the task's sensitivity to imprecise actions. Picking the bottle from the counter is easier in both these respects, allowing us to use this setting to test more significant displacements in all dimensions, as well as robustness to clutter.

The toaster oven, by contrast, requires learning the curved trajectory of a closing door. An agent that cannot execute the oven interactions accurately will often get stuck, and will need to retreat and re-grasp.We train on two different ovens, and demonstrate generalization using an unseen oven in differing positions, and in the presence of clutter.

Beyond evaluating these specific sub-skills, the tasks we have chosen provide an efficient test-bed for continual learning specifically: multiple interactions with the same object provides opportunity for forward transfer, while similar observations with conflicting behaviors is an efficient way to elicit catastrophic forgetting.

We collected two demonstrations for each task, and execute 3 out-of-distribution trials. While training the agent on a task, we sample transitions randomly from both demonstrations.

Collecting demonstrations. We guided the robot through the trajectory using a controller, recording actions at critical points (key points) as in prior work [167]. At each key point, we collect the RGB and depth images, as well as the robot's joint states. The joint states are converted into end-effector position and orientation using forward kinematics.

Implementation Details. SANER is based on an implementation of SANE utilizing IMPALA [39], as provided by CORA.Each module has a replay buffer size of 625. Since four modules were created during training, SANER uses a total number of 2500 stored frames. CLEAR and EWC were also implemented using IMPALA. For consistency we set CLEAR's total number of replay frames to be 2500.

EWC. Unfortunately, while we tried training EWC using λ in [1000, 10000, 100000], we found no value that worked reasonably in our setting, and opted not to run EWC on the robot.

5.4.1 Evaluation

Evaluation was run by randomly selecting one of the two demonstrations, and initializing the robot to its starting configuration. We scored performance according to a rubric, with partial task completion earning partial credit. Roughly speaking, the robot earns a higher reward the more of the trajectory it successfully executes.

For ABIP performance and ablations, we present the mean of reward over three trials. For the continual learning results presented in Section 5.5.3, we captured performance data before and after training on each task, as well as final performance at the end of each run. While it might be preferred to collect data for *all* tasks at the end of each task, or even significantly more frequently as per [86], this is infeasible due to the time-consuming nature of real-robot experiments. Additionally, we never test in comparison with a randomly-initialized policy as in [25], to avoid damage to the robot or environment.

5.5 Experiments

5.5.1 Evaluating Attention-Based Interaction Policies

First, we examined the ability of ABIP to generalize to out-of-distribution variations of each task. We trained on each task separately using CLEAR [150], and then evaluated on each of the training settings, as well as in the out-of-distribution generalization settings visualized in Figure 5.1. Results are shown in Table 5.1.

Our specific goal is to demonstrate that this model is sufficiently capable of learning skills in the few-shot setting, in order to be useful as a building block of a continual learning system. For this to be the case: (1) ABIP must learn well enough that catastrophic forgetting would be observable, (2) the domain must be challenging enough for forward transfer to be meaningful, and (3) general representations and policies must be attainable, so that the skills learned are practical.

We see generally high performance across both in-distribution and out-of-distribution settings, with a few exceptions. While performance is not directly comparable, we see that our method compares favorably to similar settings presented by other work using the Stretch in the home environment [12, 126, 127] As discussed above, ABIP's performance on each task highlights different strengths and weaknesses of the method. The method performs well on the *Bottle To Sink* task, particularly on the unseen settings, which indicates that

Task	In Distribution: Demo 0	In Distribution: Demo 1	Out of Distribution
Bottle To Sink	0.80 ± 0.34	0.87 ± 0.23	1.0 ± 0.0
Bottle From Sink	0.60 ± 0.40	1.0 ± 0.0	0.47 ± 0.23
Open Oven	0.20 ± 0.20	0.80 ± 0.20	0.87 ± 0.11
Close Oven	0.53 ± 0.50	0.33 ± 0.12	1.0 ± 0.0
Average	0.53	0.75	0.83

Table 5.1: Performance of CLEAR [150] on "in distribution" and "out of distribution" tasks, using ABIP as the underlying policy representation. We observe significant generalization ability.

the method is robust to significant displacements of the target object, and to distractors – key features of our MRP-based method. It suffers a bit on the *Bottle From Sink* generalization task, however. This task is highly sensitive to small errors in grasp trajectory, which indicates that while the MRP is capable of identifying our object, the action prediction is an area of improvement.

Overall, it is clear that ABIP can capture our task, and that the task is challenging enough to be interesting. Additionally, we specifically demonstrated both that the method is capable of generalizing to objects in unseen, out-of-distribution positions, as well as to unseen objects. Our policies were able to adapt to significant shifts in position and scene composition, including being able to open and close an unseen oven.

5.5.2 ABIP Ablations

Having demonstrated that ABIP is a capable building block for continual learning, we proceed to analyze the effects of the design decisions made in its creation. In particular, we ablate by comparing the ABIP to methods that: 1) remove the locality loss for the MRP, 2) remove the locality loss for the offset, 3) use a single-stream variant of the model where the MRP is predicted in the same stream as the offset, 4) use a version with no MRP at all. Results are shown in Table 5.2. All results are provided as the average over three generalization trials.

ABIP outperforms the ablations significantly, with three of the four failing to achieve anything better than a partial grasp. When trained without the locality loss on the offset-prediction head, by contrast, the model succeeded in completing two full trajectories, but failed entirely at the third. However, the case where it failed is informative: it was the setting with distractor objects.

In summary, ABIP has qualitatively better representations for generalization, finds the object of interest more reliably, and is more capable of executing

	ABIP	No MRP Locality	No Offset Locality	Single Stream	No MRP
Bottle to Sink	1.0 ± 0.0	0.0 ± 0.0	0.67 ± 0.57	0.0 ± 0.0	0.067 ± 0.12

Table 5.2: Ablation experiments on the generalization (out-of-distribution) setting. ABIP provides better generalization from very few examples, including robustness to unseen, out-of-distribution object poses and new objects.

trajectories to completion.

5.5.3 Sequential Tasks

We compared the performance of SANER and CLEAR in our continual learning setting, where we train sequentially on 4 tasks. We evaluate only on the generalization settings; results are given in Table 5.3. While CLEAR tends to learn more via direct task training (ΔR), it also exhibited significant forgetting, particularly on the bottle tasks By contrast, SANER uniformly maintains what it has learned, even exhibiting increased performance on the "Open Oven" task after training on later tasks.

Additionally, whereas CLEAR observed *some* zero-shot forward transfer, SANER experienced a considerable amount, on the "Close Oven" task in particular. This likely contributes to SANER's lower ΔR score, as it was largely capable of solving the task prior to training on it. Due to SANER's ensemble nature, it is likely that the separation of representations enabled a more seamless transfer than was possible with CLEAR's single network.

Qualitatively, CLEAR's behavior was somewhat erratic, tending to first approach every object as if it were an oven. In one instance, it executed a seamless oven opening trajectory against the side of a toaster, and in another it grabbed the handle of the bottle and pulled it down in a similar way. SANER's behavior, while imperfect, is more consistent and interpretable. Since SANER modularizes its behaviors, one can roughly know what SANER will do by observing which module it activated. However, the generally lower performance as compared to the single task setting indicates significant room for improvement. We show an example failure in Fig. 5.3. Its attention shifts first to the edge of the sink, then to the bottle itself as it attempts a grasp. The grasp, however, fails, due to inaccurate timing of gripper closure. Similar grasp-timing related issues were seen in many of the failures in other tasks as well.

The issue may come down to the fact that CLEAR trains on effectively seven times as much data as SANER. SANER only trains the active module's

	Single	CLEAR					SANERv2				
	$\mathbf{R_i}$	$\mathbf{R_{i,\,N}}$	$\mathbf{\Delta R}$	ZSFT	-F	-I	$\mathbf{R_{i,\,N}}$	$\mathbf{\Delta R}$	ZSFT	-F	-I
Bot To Sink	1.0	0.067	0.87	—	-0.80	-0.13	0.33	0.33	—	0	-0.67
Bot From Sink	0.47	0.13	0.067	0.20	-0.13	-0.33	0.13	-0.067	0.13	0.067	-0.40
Open Oven	0.87	0.33	0.20	0.0	0.13	-0.60	0.33	0.33	0.0	0.0	-0.53
Close Oven	1.0	0.87	0.73	0.13	0.0	0.13	0.87	0.20	0.67	0.0	-0.13
Average	0.84	0.35	**0.47**	0.11	-0.20	**-0.30**	**0.42**	0.20	**0.26**	**0.017**	-0.44

Table 5.3: Comparison of two methods for continual learning: CLEAR and SANER, as demonstrated on our setting given only a handful of demonstrations, across a number of different metrics. We have negated Forgetting (F) and Intransigence (I) so that in all cases a larger number is preferred. "Ind." indicates the task score when trained individually.

actor on the current batch of data, instead of an augmented batch, as SANER's critics and CLEAR both do. As such, we see that after training the first task, SANER's loss is twice CLEAR's, giving further support to this hypothesis. We will investigate solutions in future work. Overall, however, SANER has demonstrated utility in the robotics setting, by being able to learn general behavior from only two examples, with essentially no forgetting and a non-trivial amount of forward transfer.

Figure 5.3: Visualization of the attention for the MRP for SANER as it evolves over the episode, showing convergence to the object of interest. The MRP is represented, where visible, with a blue sphere.

5.6 Summary

In order to enable robots to operate in the home setting, they need to be able to learn continually from small amounts of data. To this end, we proposed SANER, an ensemble method adapted to the robotics setting. We demonstrated, on a set of 4 kitchen skills, utilizing a Stretch robot, that it is capable of learning new skills and generalizing to unseen settings with forward transfer and while mitigating the effects of catastrophic forgetting, out-performing a

strong baseline on these metrics. SANER is built on top of ABIP, which can function as a simple building block capable of learning highly useful, very generalizable interaction policies. Finally, we presented the fundamentals of our environment design and a set of continual learning metrics simplified to be viable for the robotics case. In conclusion, we demonstrate how we can deploy continuous learning techniques like SANE and CLEAR [150] to a limited-data robotics context.

Chapter 6

Extending SANER

6.1 Introduction

The ability to learn and adapt to changing circumstances over an extended period of time is the most fundamental goal of continual learning. All of the challenges focused on so far have been necessary sub-problems towards this end, but the true measure of a continual learning algorithm is in its effectiveness over time.

SANE has so far only been demonstrated on relatively short task sequences, primarily to evaluate its ability to mitigate forgetting. In this chapter, we extend it to a longer sequence to analyze its behavior as we scale up; this allows us to more effectively analyze other attributes, such as intransigence and forward transfer. We utilize a robotics simulator, RLBench [70], to learn 15 manipulation tasks, as shown in Figure 6.1.

The tasks, described further in Section 6.3, require grasping and manipulating a wide variety of objects: umbrellas, shoes, drawers and more. However, the tasks also share similarities, allowing for learning previous tasks to aid in accomplishing later ones. While we did not select the sequence to be an explicit curriculum, later tasks are generally more challenging, requiring a combination of the skills learned in previous tasks. We refer to the task sequence used as RB15.

Additionally, in all previous experiments, the agent has been able to understand what is desired from visual observation alone. However, this will often not be viable; instead, the user may want to specify what it is they need done via natural language. We have thus included language descriptions, which give the robot additional necessary information. Tasks were also selected to require using this information: several are visually indistinguishable from each other

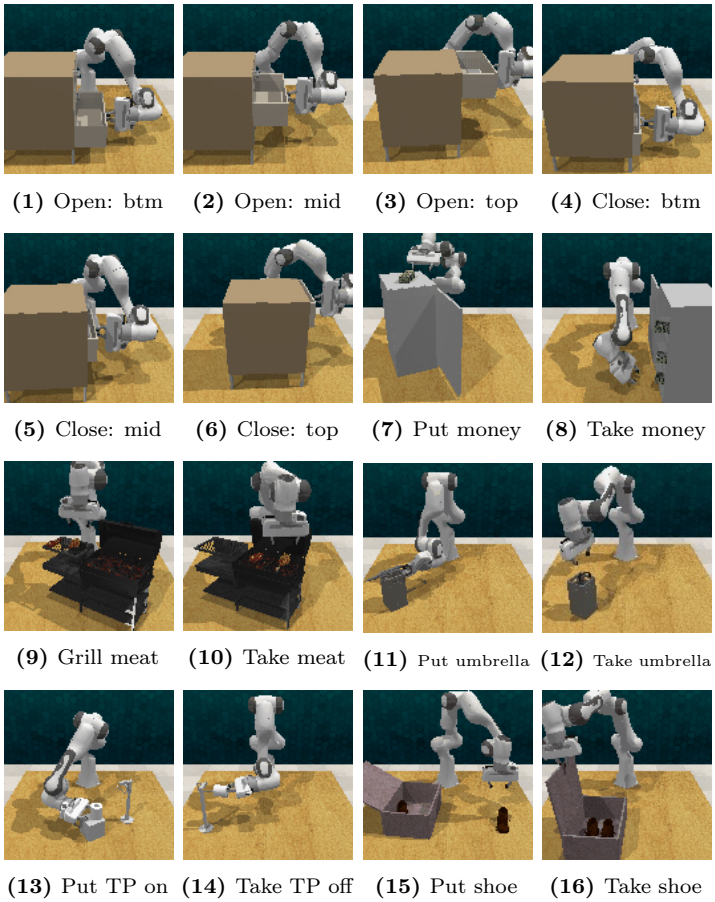


Figure 6.1: Images on each of the 16 tasks, demonstrating the spectrum of skills necessary: opening different drawers (tasks 1-3), closing them (tasks 6-9), and placing objects and retrieving them. The box image was not used in initial training, but used for ensemble-composition

(opening/closing different drawers from the same dresser).

As with SANER, we utilize imitation learning to efficiently learn the tasks. We train on 10 demonstrations per task, and evaluate its generalization capabilities on a set of 10 random variations.

Improving over Reservoir Sampling. During the course of this work, a number of improvements to SANER have been made. The primary concept that underlies the SANE approach is that if modules learn to specialize, they can learn more efficiently than both 1) multi-task models, as different skills compete for model weights, and 2) single-task models, where common structure cannot be discovered. While multi-task learning has proven effective in some

areas [122, 141], it requires a vast quantity of data. Significant investment is required to apply it to robotics, and even then the task diversity has been limited [74].

One of the ways SANE-based models accomplish this goal is by the distribution of data in the modules' replay buffers. In previous chapters, a favorable distribution was readily achieved, due to the limited number of tasks. However, the probability of a new sample being added to a replay buffer using reservoir sampling decreases over time as $\frac{1}{t}$, and the ratio of samples for a given task similarly decreases as new tasks are added, as $\frac{1}{|T|}$. Both of these work against us: the former means that new information will be slow to enter the buffer, and the latter means little of it will be present in our batches. We present Priority Reservoir Sampling (PRS) instead, which guarantees new samples will be added immediately, but with limited disruption of prior samples. We additionally present an analysis of the method in Appendix A.

6.2 Method

For all SANE-based methods, the more accurate each critic is, the better the ensemble performs. We are thus constantly making improvements to that end. The other changes for SANERv2 largely fall into two categories: 1) general improvements, which we believe SANE-based methods in general would benefit from; and 2) improvements specifically for the imitation learning from language setting.

Impact of Environment. However, there is one aspect to using the environment that catalyzed some of these changes. The workspace is relatively large, and cannot be effectively clipped due to high utilization. However, due to compute limitations, the number of points that can be reasonably used is relatively small (3500).

The result is that the same point cloud can produce significantly different results. This is a problem for the critic in particular. Some of the changes made were to stabilize network predictions, while others were to make the ensemble itself more robust.

From our experience with SANER and the initial RB15 experiments, we have made several observations that we believe result in general improvements:

1. **Replay Buffer Improvements.** Both the actor and the critic must have access to the information necessary. However, their goals are contrary: non-relevant samples negatively impact the actor, but are necessary for the critic. Additionally, both, at certain points, strongly benefit from

having a high density of new samples. We therefore introduce changes that improve the circumstances of both networks.

2. **Global state awareness.** The ensemble, as a collective, can experience two types of event based on information received from the environment: that a node has been activated, and that drift has been detected.

   Thus far in the development of SANER, modules have been treated entirely separately. However, neither of these two events are truly isolated to a particular node. As the ensemble together forms a unified agent, if one experiences drift, they have all experienced drift. And if one is activated, that means the others were not.

3. **In drift, activation should not be overly optimistic.** In SANER, we increased the amount by which a node can be optimistic, to stabilize training when a critic is most likely to be producing inaccurate values. However, when drift has occurred, optimism is undesirable; it will cause nodes to be created based on modules other than the one best-suited for the new circumstance. Modules must, therefore, be unbiased during determination of node duplication.

6.2.1 Replay Buffer Enhancements

Two changes were made to our handling of the replay buffer that dramatically improve its behavior over long task sequences: 1) labeling samples in the replay buffer with a "negative example" label, and 2) modifying the reservoir sampling algorithm we use.

6.2.1.1 Negative Examples

Negative example labeling is used to extract non-negative entries from the replay buffer for two purposes: 1) augmenting the actor batch, and 2) triggering drift detection based on both current samples and non-negative samples from the replay buffer.

Effectively, these examples are entries that are valid for use by the critic, but invalid for use by the policy or to detect drift. Note that negative examples are not used in any special way other than to filter them out in those specific use cases.

Samples in the replay buffer are labeled as "negative" in the following critical cases:

1. Samples from the source node get copied over to the new node as negative examples.

2. When a node has been activated, but is currently marked "inactive", the data it collects during training are marked as negative examples.

In SANER, when a module encountered drift, its policy would freeze. SANERv2 no longer requires this; instead, modules are now marked "inactive" instead of "frozen". Unlike frozen nodes, inactive nodes still train; however, they do not train on the current batch being observed, only on batches composed of non-negative samples from the buffer.

When a new node is created, all nodes are marked inactive. For the source node, this is permanent; for all others it is temporary. This gives them an opportunity, since they are in the presence of drift, to collect negative examples to keep their critics accurate.

There is one other case that results in node inactivation: with probability k_{deact} at the start of a training cycle, active node selection is based on the raw value predicted by the critic, with no uncertainty estimation augmentation at all. This is based on observation #3. If the current best module by raw activation is not the same as the one by augmented activation, then one of the two is possibly incorrect, and they both need to train on more data points. Node duplication is unavailable during this time as well.

Additionally, since raw activation scores are used during evaluation, if there's a discrepancy during training, that increases the likelihood that the wrong module will be used during evaluation.

6.2.1.2 Priority Reservoir Sampling (PRS)

Problem: new samples are insufficiently represented.. When a module experiences drift, it is advantageous for both the source node and the new node to store relevant samples immediately. If the new node does not, it will learn inefficiently. If the source node does not (but the new node does), then as the new node's critic adjusts to the true value (which is low until the actor improves), the source node will be activated too often; in the worst case, another new node will be created and the process repeats.

One of the main causes of this problem is the use of reservoir sampling for the replay buffer, which maintains samples with equal probability, regardless of when they were collected [182]. With the current hyperparameters, if standard reservoir sampling were used (and replay buffers inherit in full from the source, as we did with SANER), the chance of any individual sample being added is

about 3% after two tasks. If a consistent buffer lineage is maintained, at the end of the 15th experiment the probability is about 0.5%.

We propose a simple modification more suited for this use-case. Specifically, we desire the diversity of data that comes from random replacement, as well as reasonable maintenance of old samples, while also having a way to ensure that new nodes are guaranteed to be significantly represented.

Proposed Solution. Our simple solution to this problem was to, when drift is detected, remove fraction f samples from the buffer, uniformly. For convenience, we refer to this variant as Priority Reservoir Sampling (PRS). We model and analyze its behavior in a simple setting in Appendix A, but essentially by removing this fraction of entries, we effectively create a partition in the buffer. Newly added entries will continue replacing only each other until they attain the reservoir value used by the remainder of the buffer. Thus we get a minimum of fB new samples (for buffer size B), without disrupting old entries, until equilibrium is reached. However, this comes at the cost of the inexorable decay of old entries, as shown in Figure A.1.

For SANERv2, this is a beneficial trade-off. No individual buffer should, if the ensemble is working as intended, need to maintain samples from the entirety of training. However, it is likely there are future improvements to be made, perhaps by utilizing the negative example concept.

In SANERv2, this removal occurs in both the source node and the new node created when a duplication occurs, to make room for negative examples (in the former case), and policy-relevant samples (in the latter case).

6.2.2 Module-Level Changes

Driven by observations 2 & 3, modules in the ensemble now have states that can determine aspects of their behavior. In particular:

1. **Activated nodes are more likely to be re-activated**; as variance in critic estimates is high, this ensures two or more nodes are not competing without a clear winner, which distributes the training inefficiently between them.

2. **Nodes can be ineligible for duplication**. This temporary state occurs when the added activation factor f_a exceeds some threshold ($k_{a,max}$); or in other words, the amount they were aided is too large for the activation to have been in parity with the rest.

3. **If a node encounters drift, but is ineligible for duplication**, all nodes have their uncertainty factors set to 0 to level the playing field.

Thus, often, duplication happens in a 2-step process: 1) drift is detected and uncertainties are reset, 2) all nodes are eligible, so the next one to activate is truly the best in the setting; either it will be used, or it will detect drift and be duplicated.

4. **After duplication, the new node's activation is increased, and the source node's decreased**; since they begin as clones, we want the new one to succeed. Both upper bounds are also increased; drift is known, so re-detecting it is unnecessary. Additionally, the new node's prototype is permitted to update with the main network, to establish a better baseline for performance.

Pseudo-code for most of the updates to SANERv2 are summarized in Algorithm 2.

6.2.3 Improvements for Imitation Learning

We additionally make the following improvements for the continual imitation learning setting, with language:

1. **Bootstrapping R.** During training, SANER used a unroll length of 1 to optimize for randomness. However, that meant there was no ability too bootstrap a full-episode value estimate for the critic, so activation, particularly during evaluation, would be rather myopic. In SANERv2, we use a rollout of length 2, and estimate bootstrapped value estimates. $decay = 1$.

2. **Rollout.** Related to the above, we sample 2 steps from the environment, and adjust sampling from the environment to unsure all points in the trajectory are still selected with uniform distribution.

3. **A new value estimate for the critic.** We use $softplus(B) = (a + bx)$ for the critic value estimate, as in SANER, but we changed the parameters to have a thicker tail ($B = 0.05, a = -58, b = -595$).

4. **Language.** SANERv2 uses natural language to contextualize the task for the agent. See below.

5. **Uncertainty L1.** We additionally switched from L2 to L1 for uncertainty.

Algorithm 2 SANERv2 Module: state update

Require: On self: UB factor self.f_{UB}, Activation factor self.f_a, Inactive self.inactive, temporarily inactive self.temp_inactive
Require: Ensemble-level: *event* received, set of all nodes $\mathcal{N}$

if event.node_activated **then** ▷ When any node is activated...
    $f_a = f_a + k_{act}$ ▷ Active node gets act boost
else
    $f_a = 0$ ▷ Other nodes get reset
end if

if self.detected_drift **then** ▷ When drift is detected...
    eligible_for_duplication = self.inactive or $f_a < k_{a,max}$
    if eligible_for_duplication **then** ▷ If the node is eligible, duplicate it
        self.duplicate()
        self.fire(event.new_created)
    else
        for $node \in \{\mathcal{N}\}$ **do** ▷ Otherwise, level the field for all
            $node.f_a = 0$
            $node.f_{UB} = 0$
        end for
        self.reset_to_prototype() ▷ Reset actor & critic to prototype either way
    end if
end if

if event.new_created **then** ▷ When any node is created...
    if event.is_new **then** ▷ New node: UB & act boosts, k_{new} replay removed
        self.$f_a = f_a + k_{a,new}$
        self.$f_{UB} = f_{UB} + k_{UB,new}$
        self.remove_replay(k_{new})
    else if event.duplicated **then** ▷ Source: -act, +UB boost, deactivated, k_{dupe} replay removed
        self.$f_a = f_a - k_{a,dupe}$
        self.$f_{UB} = f_{UB} + k_{k_{UB,dupe}}$
        self.$inactive = true$
        self.remove_replay(k_{dupe})
    else ▷ Other nodes: UB boost, temporarily deactivated
        self.$f_{UB} = f_{UB} + k_{other,dupe}$
        self.temp_inactive = true
    end if
end if
$f_{UB}* = \gamma_{UB}$
$f_a* = \gamma_a$

Incorporating Language. We embedded the phrases (examples given in Table 6.1) using chatGPT's "text-embedding-ada-002" [1] model, which produces an embedding size of 1536.

The attention module in the APN has been modified to output an embedding with the language embedding dimension. During use, the dot product is taken with the language embedding, to produce a language-conditioned attention over the points in the point cloud. The intention is for this to align the point cloud representation with the language embedding.

Algorithm 3 SANERv2 Ensemble: Node activation

Require: set of all nodes $\mathcal{N}$, observation s
max_node, max_activation = $\emptyset, 0$
deactivate_all = random.uniform() $< k_{deact}$ ▷ Run raw-best node k_{deact} frac of time
$train\_policy = true$
for $node \in \mathcal{N}$ **do**
    raw_act, raw_unc = node.critic(s)
    if deactivate_all **then** ▷ If all nodes are deactivated, use raw act alone
        act_score = raw_act
        train_policy = false
    else if node.inactive or node.temp_inactive **then** ▷ If node is inactive, $\rightarrow f_a = 0$ & $f_{UB} = 0$
        act_score = raw_act + raw_unc
        train_policy = false
    else
        act_score = (1+node.f_a) raw_act + (1+node.f_{unc}) raw_unc
    end if
    if act_score > max_activation **then**
        max_activation,max_node = act_score, node
    end if
end for
$max\_node.collect\_data(train\_policy)$

"open the shoe box and take the shoes out"
"set the shoes down on the table"

Table 6.1: Example language examples for Task 15.

6.3 Experiments

The RLBench [70] robotics simulation environment provides 100 tasks across a wide variety of interaction tasks, using the Franka Panda Arm. Additionally, it provides the ability to generate trajectories and language descriptions for the tasks. The number of language descriptions per task ranges between 3 and 8.

Tasks were selected to leverage visual similarity in the observations, to verify the network's successful use of leverage language, and for behavioral similarity, to provide an opportunity for transfer. The selected tasks are shown in Figure 6.1.

We generated 10 demonstrations per task, and evaluated over 10 episodes with randomly seeded task settings. Tasks randomization varies by task, but generally varies the position and orientation of furniture involved (e.g. drawer, grill), and often the movement of the smaller objects (e.g. money, toilet paper). Evaluation episodes end if: 1) the agent succeeds, 2) the max steps is reached, or 3) the agent predicts an invalid action (e.g. would cause collision).

The difficulty of the task can vary by seed, so to make the results more representative of a model's true ability, we fixed the set of seeds used for evaluation for the runs presented in this paper. Results are presented over 3

model training seeds.

Observation/Action Spaces. We used the default image size of 128x128, collected across three cameras (front, overhead, wrist), as shown in Figure 6.2. The environment also provides left and right views, but we found that including these increased processing times and did not significantly impact performance. These images were processed, as in SANER, into a point cloud. We used random down-sampling to reduce the number of points to 3500. We did not include a contextual point cloud (as we did in SANER), instead utilizing language.

The action space is the same as in SANER, representing a position and orientation for the gripper, plus gripper state and an estimate of the fraction of the task completed.

Keypoints. As in SANER, we used keypoints for learning the trajectory. These were automatically extracted from the full trajectory based on gripper state change and joint acceleration, as well as the first and last timesteps. The number of keypoints for all drawer and pick-and-place tasks is between 2 and 5. For opening the box and removing/placing shoes, there are about 14 keypoints. During evaluation, the robot was given a budget of $train\_keypoints * 1.5$ timesteps in which to complete the task.

SANERv2 Details. SANERv2 uses the same architecture and hyperparameters as SANER, with the exception of the adjustments mentioned above. PRS uses a removal fraction of 0.3 (for both source and new), and the chance of evaluating activation with all modules "temporarily deactivated" is $k_{deact} = 0.25$.

Baselines. We compare against CLEAR [150], which has consistently shown the best performance across the baselines used in this work (see Chapter 2.6). All hyperparameters are the same between the two methods; CLEAR uses a replay buffer of 20,000, and SANERv2 has a maximum of 32 modules, each with a replay buffer of size 625.

We also compare performance with CLEAR trained individually on each task; this can alternatively be seen as a SANERv2 ensemble where the modules are pre-created, 1 per task, and activation occurs perfectly.

6.3.1 Experiments

RB15.

The 15 tasks selected for this experiment are shown in Figure 6.1, and include all tasks but the one in blue, which is used for the next experiment. The first several tasks involve interacting with an articulated object (a chest of

Figure 6.2: Visualization of the three views used in this set of tasks: front, top, and wrist-mounted.

drawers), the next several are pick-and-place tasks, and the last one combines both skills.

We evaluated on all of the metrics presented in Chapter 2.3. Rewards have been normalized by the maximum value observed for a task across all algorithms; this effectively scales each task by difficulty.

RB15+1: Composable modules.

In this experiment, we selected one SANERv2 agent from RB15 and trained it on a 16th task, shown in blue in Figure 6.1. We then took the module that it used at the end of training, and added it to the ensembles from all runs trained only on RB15. When SANERv2 agents are checkpointed, each module is saved separately, and there is one metadata file that associates them into the ensemble. Adding the module simply entails: copying the module folder into the new ensemble, and updating the metadata file.

We evaluated final performance on all tasks, including the new one, to evaluate how adding a new module impacts performance. In particular, this experiment was looking to evaluate whether: 1) the existing modules, which had seen no negative examples from the new task, would mis-activate, and 2) whether the training dynamics of a SANERv2 ensemble create modules that are only calibrated against each other.

We compare the SANERv2 ensembles that were augmented via composition (SANERv2-Aug) to the original performance of the ensemble from which the module was taken (SANERv2-S, 1 seed), as well as with CLEAR trained on all 16 tasks.

It is worth noting that this is more of a probing experiment than a full analysis. A more complete analysis would include composing in new modules that had not been trained on the same set of tasks as the ensemble.

6.4 Results

	Single		CLEAR					SANERv2				
i	$\mathbf{R_{i,0}}$	$\mathbf{R_i}$	$\mathbf{R_{i,\ N}}$	$\mathbf{\Delta R}$	Z_C	$-F_C$	-I	$\mathbf{R_{i,\ N}}$	$\mathbf{\Delta R}$	Z_C	$-F_C$	-I
0: Open, btm	0.00	0.62	0.90	0.71	-	0.19	0.29	0.62	0.62	-	0.00	0.00
1: Open, mid	0.00	0.70	0.70	0.41	0.00	0.30	0.00	0.30	0.67	0.00	-0.37	-0.41
2: Open, top	0.00	0.67	0.73	0.83	0.00	-0.10	0.07	0.63	0.80	0.00	-0.17	-0.03
3: Close, btm	0.10	0.83	0.80	0.27	0.40	0.03	-0.03	0.93	0.67	0.17	0.00	0.10
4: Close, mid	0.13	1.00	0.97	0.20	0.57	0.07	-0.03	0.97	0.07	0.77	0.00	-0.03
5: Close, top	0.13	1.00	1.00	0.10	0.73	0.03	0.00	0.90	0.30	0.57	-0.10	-0.10
6: Put money in safe	0.00	0.61	0.44	0.56	0.00	-0.11	-0.17	0.39	0.44	0.00	-0.06	-0.22
7: Take money from safe	0.00	0.71	0.33	0.67	0.00	-0.33	-0.38	0.67	0.43	0.00	0.24	-0.05
8: Put meat on grill	0.00	0.97	0.90	0.63	0.00	0.27	-0.07	0.93	0.90	0.00	0.03	-0.03
9: Take meat off grill	0.00	0.73	0.60	0.57	0.00	0.03	-0.13	0.87	0.83	0.00	0.03	0.13
10: Put umbrella in box	0.00	0.33	0.00	0.33	0.00	-0.33	-0.33	0.00	0.33	0.00	-0.33	-0.33
11: Take umbrella from box	0.03	0.37	0.60	0.47	0.30	-0.20	0.23	0.43	0.33	0.00	0.07	0.07
12: Put TP on stand	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.33	0.33
13: Take TP off stand	0.00	0.33	0.22	0.11	0.22	-0.11	-0.11	0.78	0.22	0.11	0.44	0.44
14: Open box & Put shoes in box	0.00	1.00	0.67	0.33	0.00	-	-0.33	0.67	0.33	0.00	-	-0.33
Avg	0.03	0.66	0.59	0.41	**0.16**	-0.02	-0.07	**0.63**	**0.46**	0.12	**0.01**	**-0.03**

Table 6.2: The per-task average results over 3 seeds, for CLEAR and SANERv2. We have indicated via shading green any values that are > 0.2 larger than the other's. In addition to the standard metrics, we provide the average return before and after training on each task.

	Drawer Tasks (0-5)			Other Tasks (6+)		
	CLEAR	SANERv2	Δ	CLEAR	SANERv2	Δ
$\mathbf{R}_{final}$	0.85	0.72	-0.13	0.42	0.56	0.14
$\mathbf{\Delta R}$	0.42	0.52	0.10	0.41	0.43	0.02
Z_C	0.34	0.30	-0.04	0.06	0.01	-0.05
$-F_C$	0.09	-0.11	-0.19	-0.10	0.10	0.19
$-I$	0.05	-0.08	-0.13	-0.14	0.00	0.14

Table 6.3: Results for the RB15 experiments, clustered over the first six tasks, and over the remainder, to see the performance gap between the methods in each case. The Δ $(S - C)$ is provided for convenience; green indicates SANERv2's results exceeded CLEAR's, and red indicates the opposite.

6.4.1 RB15

We present the results across all tasks in Table 6.2. Overall we observe that while CLEAR performs better in earlier tasks, it struggles by the end, particularly on the interactions with the toilet paper. SANERv2, on the other hand, performs less well in earlier tasks, but is capable of learning tasks as the sequence proceeds.

We can further observe the models' transfer abilities, by observing the cases where each model beats the individually trained one ($-I > 0$): CLEAR exceeds it on tasks (0, 2, 11), and SANERv2 exceeds it on (3, 9, 11, 12, 13), indicating representations previously learned aided performance. While CLEAR demonstrates zero-shot forward transfer ($\mathcal{Z}_c$) somewhat more consistently than SANERv2, it only demonstrates it on one task that SANERv2 did not: taking the umbrella from the box.

We split the task results into the first 6 tasks and the remainder of the tasks, shown in Table 6.3, to more clearly highlight the difference. We believe the effect is not incidental: CLEAR experiences positive backwards transfer ($-\mathcal{F}_C$) on the earlier tasks, as it is continuing to sample from a replay buffer that contains at least as many instances of the earlier tasks as the later ones. This likely also contributes to its poorer performance on later tasks. SANERv2, on the other hand, does not get the benefit of continuing to train on earlier tasks, but is instead more able to focus on the challenging later ones.

6.4.2 RB15 +1

Task	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SANERv2-Aug	0.81	0.41	**0.77**	**1.00**	0.97	0.93	0.33	0.62	0.87	0.80	0.00	0.40	**0.33**	0.33	0.83	**0.78**
SANERv2-Source	**0.86**	0.22	0.60	**1.00**	**1.00**	**1.00**	0.33	0.57	0.80	**1.00**	0.00	0.10	0.00	**0.50**	**1.00**	0.67
CLEAR (16 task)	0.67	**0.83**	0.73	0.90	0.87	**1.00**	**0.72**	**0.67**	**0.90**	0.70	**0.33**	**0.87**	0.00	0.33	0.17	0.11
Individual	0.62	0.79	0.67	0.83	1.00	1.00	0.61	0.71	0.97	0.73	0.33	0.37	0.00	0.50	0.50	0.11

Table 6.4: Final performance results for the 16 task sequence, where SANERv2-Aug is the average over the 3 augmented ensembles, SANERv2-Source is the single ensemble from which the node was extracted, and CLEAR (16 task) and Individual are for comparison, also over 3 seeds. In particular, note that though the ensembles in SANERv2-Aug were not trained on task 15, they still achieve high scores.

We present our results on the composition experiment in Table 6.4. We observe that the augmented ensemble maintains its performance on earlier tasks, and gains the skill present in the new module, even slightly exceeding

performance on the ensemble from which it was drawn, and significantly exceeding both CLEAR and the individually trained model.

One of the motivations for this experiment was to begin to answer the question of whether modules in the ensemble are merely calibrated to perform well against each other, but the scores estimated by the critic are not usable outside of the ensemble in which it was created. We have found evidence that this is not the case, suggesting SANERv2 is suited to being used for a composable library of skills.

6.5 Summary

In this chapter, we present SANERv2, a modification of SANER that performs well on longer sequences of tasks, showing increased performance on later tasks over both CLEAR and individual-task modules, as demonstrated on a sequence of 15 robotics tasks in simulation.

A key factor in SANERv2's success is that it is designed to partition the input space, creating more specialized modules that work well in particular domains. This can be contrasted with CLEAR, which is designed to have an even ratio of samples across all tasks. We have demonstrated experimentally that an even distribution results in a model that struggles to learn later tasks in a long sequence.

One of the key insights during development of SANERv2 is that the replay buffer should be modified to take a more active approach to ensuring it contains necessary samples. We did this in two ways: 1) by introducing Priority Reservoir Sampling, and 2) by labeling samples as "negative", the actor can continue to be trained on replay buffer samples in perpetuity.

Additionally, we experimented with transferring a module into a different ensemble, which had not been trained on the skill for which the module was proficient. We demonstrate that the augmented ensemble was capable of the new skill, and had not lost capability on any existing behaviors.

Chapter 7

Conclusion

The vision presented in this thesis is that of an adaptable home robot, capable of learning from both the needs of its users and the environment it lives in. Such a robot would need to learn in a variety of ways, including through its own experiences and through examples provided by others.

Capabilities. In this thesis, I approach this problem through the lens of continual learning, which aims to solve the difficulties that neural networks encounter when learning over time, from shifting data. An ideal continual learner will be capable of: leveraging prior experience, continuing to learn well over time, and not forgetting previous experiences. I additionally aimed to accomplish two other goals: being able to leverage language, and for agents to be able to share learned skills.

Settings. This work explores the problem specifically in two settings: continual reinforcement learning, where the agent learns from its own experiences, and continual imitation learning, where the agent learns from demonstrations. Experiments were conducted across both simulation environments, and with real robots.

Contributions: Accessibility. A number of barriers have made continual learning inaccessible, including: inefficient metrics, computationally expensive benchmarks, and closed-source baselines. In this work I have developed a set of more efficient metrics for robotics, presented in Chapter 2, as well as work establishing a faster set of benchmarks, and open source baselines (in Chapter 3.

Contributions: SANE. Leveraging these improvements to accessibility, this thesis presents a modular method, SANE, that is capable of learning specialized modules, in a dynamic ensemble. In Chapter 4, SANE was demonstrated to mitigate forgetting on several RL environments. In Chapter 5, SANER was

demonstrated to generalize to significant variation in setting, on a real robot. Finally, in Chapter 6, SANERv2 was evaluated on an extended series of 15 tasks, using language. SANERv2 proved capable of not only being able to continue to learn new skills, which the baseline struggled with, but to improve on the performance of an individually trained model, demonstrating forward transfer.

Additionally, a pilot experiment showed that a module from one ensemble could be readily transplanted into another; prior skills were maintained, and a new task, unseen to the ensemble, was successfully accomplished.

Diversity of Setting. This thesis utilizes the modes of reinforcement learning and imitation learning to advance the field of continual policy learning, but the question bears asking: how extensible is this work to other settings?

SANE can, in principle, be applied to anything where a score (i.e. return, **R**) can be defined, measuring the agent's success. This is easiest to see in settings where there is a source of truth, as the reward from the environment is in RL, or the demonstrations are in IL. However, it could also be improving an internal metric, as in self-supervised learning: for example, the return could be high if the agent correctly predicted some subsequent observation, as in [184].

There are many possibilities for this direction, and many challenges as well, but a composable skill library comprised of independent modules seems a a promising direction.

Bibliography

[1] Text embedding by text-embedding-ada-002. https://api.openai.com/v1/embeddings, 2023. 85

[2] W. C. Abraham and A. Robins. Memory retention–the synaptic stability versus plasticity dilemma. *Trends in neurosciences*, 28(2):73–78, 2005. 43

[3] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 64

[4] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018. 44

[5] R. Aljundi, E. Belilovsky, T. Tuytelaars, L. Charlin, M. Caccia, M. Lin, and L. Page-Caccia. Online continual learning with maximal interfered retrieval. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 11849–11860, 2019. 44

[6] R. Aljundi, P. Chakravarty, and T. Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3366–3375, 2017. 45

[7] R. Aljundi, K. Kelchtermans, and T. Tuytelaars. Task-free continual learning. In *Proceedings of the CVPR Workshop on Continual Learning in Computer Vision*, pages 11254–11263, 2019. 42, 44

[8] O. Anschel, N. Baram, and N. Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, pages 176–185. PMLR, 2017. 48

[9] A. Ayub and C. Fendley. Few-shot continual active learning by a robot. *arXiv preprint arXiv:2210.04137*, 2022. 63

[10] A. Ayub and A. R. Wagner. Tell me what this is: Few-shot incremental object learning by a robot. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8344–8350. IEEE, 2020. 63

[11] B. Bagus, A. Gepperth, and T. Lesort. Beyond supervised continual learning: a review, 2022. 63

[12] S. Bahl, A. Gupta, and D. Pathak. Human-to-robot imitation in the wild, 2022. 73

[13] D. Batra, A. X. Chang, S. Chernova, A. J. Davison, J. Deng, V. Koltun, S. Levine, J. Malik, I. Mordatch, R. Mottaghi, et al. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020. 19

[14] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016. 19

[15] S. Behbahani, S. R. Chhatpar, S. Zahrai, V. Duggal, and M. Sukhwani. Episodic memory model for learning robotic manipulation tasks. *CoRR*, abs/2104.10218, 2021. 63

[16] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 47:253–279, 2013. 4, 19, 20, 21

[17] E. Ben-Iwhiwhu, S. Nath, P. K. Pilly, S. Kolouri, and A. Soltoggio. Lifelong reinforcement learning with modulating masks, 2022. 64

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 18

[19] S. Brodeur, E. Perez, A. Anand, F. Golemo, L. Celotti, F. Strub, J. Rouat, H. Larochelle, and A. Courville. Home: A household multimodal environment. *arXiv preprint arXiv:1711.11017*, 2017. 19

[20] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022. 64

[21] E. Brunskill and L. Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International conference on machine learning*, pages 316–324. PMLR, 2014. 45

[22] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006. 44

[23] L. Caccia, E. Belilovsky, M. Caccia, and J. Pineau. Online learned continual compression with adaptive quantization modules. In *International Conference on Machine Learning*, pages 1240–1250. PMLR, 2020. 44

[24] H. Caselles-Dupré, M. Garcia-Ortiz, and D. Filliat. S-trigger: Continual state representation learning via self-triggered generative replay. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2021. 44

[25] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018. 2, 9, 10, 14, 44, 73

[26] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations*, 2018. 44

[27] J. Chen, T. Nguyen, D. Gorur, and A. Chaudhry. Is forgetting less a good inductive bias for forward transfer?, 2023. 11

[28] L. Chen, S. Jayanthi, R. Paleja, D. Martin, V. Zakharov, and M. Gombolay. Scalable lifelong learning from heterogeneous demonstrations. *IROS Workshop on Lifelong Robot Learning*, 2022. 63

[29] B. Cheung, A. Terekhov, Y. Chen, P. Agrawal, and B. Olshausen. Superposition of many models into one. *Advances in neural information processing systems*, 32, 2019. 45

[30] M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018. 19

[31] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020. 4, 5, 18, 19, 20, 22, 38, 43, 51

[32] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang. Adanet: Adaptive structural learning of artificial neural networks. In *International conference on machine learning*, pages 874–883. PMLR, 2017. 44

[33] M. Crawshaw. Multi-task learning with deep neural networks: A survey. *CoRR*, abs/2009.09796, 2020. 1

[34] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, page 271–278, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. 45

[35] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088, 2016. 63

[36] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000. 45

[37] T. J. Draelos, N. E. Miner, C. C. Lamb, J. A. Cox, C. M. Vineyard, K. D. Carlson, W. M. Severa, C. D. James, and J. B. Aimone. Neurogenesis deep learning: Extending deep networks to accommodate new classes. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 526–533. IEEE, 2017. 44

[38] K. Ehsani, W. Han, A. Herrasti, E. VanderBilt, L. Weihs, E. Kolve, A. Kembhavi, and R. Mottaghi. Manipulathor: A framework for visual object manipulation. In *CVPR*, 2021. 19

[39] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018. 32, 33, 50, 72

[40] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *CoRR*, abs/1711.06782, 2017. 61

[41] J. Farebrother, M. C. Machado, and M. Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018. 22

[42] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017. 45

[43] R. M. French. Pseudo-recurrent connectionist networks: An approach to the'sensitivity-stability'dilemma. *Connection science*, 9(4):353–380, 1997. 44

[44] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999. 42, 43

[45] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. 45, 48

[46] T. Furlanello, J. Zhao, A. M. Saxe, L. Itti, and B. S. Tjan. Active long term memory networks. *arXiv preprint arXiv:1606.02355*, 2016. 44

[47] C. Gan, J. Schwartz, S. Alter, D. Mrowca, M. Schrimpf, J. Traer, J. D. Freitas, J. Kubilius, A. Bhandwaldar, N. Haber, M. Sano, K. Kim, E. Wang, M. Lingelbach, A. Curtis, K. T. Feigelis, D. Bear, D. Gutfreund, D. D. Cox, A. Torralba, J. J. DiCarlo, J. B. Tenenbaum, J. Mcdermott, and D. L. Yamins. ThreeDWorld: A platform for interactive multi-modal physical simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. 19

[48] C. Gao, H. Gao, S. Guo, T. Zhang, and F. Chen. CRIL: continual robot imitation learning via generative and prediction model. *CoRR*, abs/2106.09422, 2021. 63

[49] X. Gao, R. Gong, T. Shu, X. Xie, S. Wang, and S.-C. Zhu. Vrkitchen: an interactive 3d virtual environment for task-oriented learning. *arXiv preprint arXiv:1903.05757*, 2019. 19

[50] J.-B. Gaya, T. Doan, L. Caccia, L. Soulier, L. Denoyer, and R. Raileanu. Building a subspace of policies for scalable continual learning, 2022. 63

[51] A. Gepperth and C. Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, 8(5):924–934, 2016. 44

[52] D. Ghosh, J. Rahme, A. Kumar, A. Zhang, R. P. Adams, and S. Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. *Advances in Neural Information Processing Systems*, 34, 2021. 7

[53] A. Goyal, S. Sodhani, J. Binas, X. B. Peng, S. Levine, and Y. Bengio. Reinforcement learning with competitive ensembles of information-constrained primitives. In *International Conference on Learning Representations*, 2019. 45, 46

[54] S. Grossberg. How does a brain build a cognitive code? In *Studies of mind and brain*, pages 1–52. Springer, 1982. 43

[55] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, pages 1025–1037. PMLR, 2020. 19

[56] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, et al. The minerl 2019 competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 2019. 19

[57] A. Guzman-Rivera, D. Batra, and P. Kohli. Multiple choice learning: Learning to produce multiple structured outputs. *Advances in neural information processing systems*, 25, 2012. 45

[58] A. Hallak, D. Di Castro, and S. Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015. 7

[59] H. v. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016. 45

[60] X. He and H. Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. In *International Conference on Learning Representations*, 2018. 44

[61] G. Hinton, O. Vinyals, J. Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 44

[62] Y.-C. Hsu, Y.-C. Liu, A. Ramasamy, and Z. Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018. 17

[63] A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager. "good robot!": Efficient reinforcement learning for multi-step visual tasks with sim to real transfer. *IEEE Robotics and Automation Letters*, 5(4):6724–6731, 2020. 64

[64] D. Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35:155–164, 2009. 67

[65] M. Igl, G. Farquhar, J. Luketina, W. Boehmer, and S. Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2021. 22, 37

[66] D. Isele and A. Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. 44

[67] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, pages 876–885. Association For Uncertainty in Artificial Intelligence (AUAI), 2018. 48

[68] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991. 46

[69] H. Jaeger. Using conceptors to manage neural long-term memories for temporal patterns. *The Journal of Machine Learning Research*, 18(1):387–429, 2017. 44

[70] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020. 4, 19, 78, 86

[71] S. James, K. Wada, T. Laidlow, and A. J. Davison. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13739–13748, 2022. 61, 64

[72] K. Javed and M. White. Meta-learning representations for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1820–1830, 2019. 17

[73] A. Juliani, A. Khalifa, V.-P. Berges, J. Harper, E. Teng, H. Henry, A. Crespi, J. Togelius, and D. Lange. Obstacle tower: A generalization challenge in vision, control, and planning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019. 19

[74] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *CoRR*, abs/2104.08212, 2021. 80

[75] N. Kamra, U. Gupta, and Y. Liu. Deep generative dual memory network for continual learning. *arXiv preprint arXiv:1710.10368*, 2017. 44

[76] H. Kannan, D. Hafner, C. Finn, and D. Erhan. Robodesk: A multi-task reinforcement learning benchmark. https://github.com/google-research/robodesk, 2021. 19

[77] C. Kaplanis, M. Shanahan, and C. Clopath. Policy consolidation for continual reinforcement learning. In *Proceedings of the International Conference on Machine learning (ICML)*, pages 3242–3251, 2019. 44

[78] K. Kase, C. Paxton, H. Mazhar, T. Ogata, and D. Fox. Transferable task execution from pixels through deep planning domain learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10459–10465. IEEE, 2020. 63

[79] R. Kemker and C. Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*, 2018. 44

[80] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2018. 43

[81] C. C. Kemp, A. Edsinger, H. M. Clever, and B. Matulevich. The design of stretch: A compact, lightweight mobile manipulator for indoor human environments. *CoRR*, abs/2109.10892, 2021. 4, 71

[82] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016. 19

[83] S. Kessler, J. Parker-Holder, P. Ball, S. Zohren, and S. J. Roberts. Same state, different task: Continual reinforcement learning without interference. *arXiv preprint arXiv:2106.02940*, 2021. 45

[84] K. Khetarpal, S. Sodhani, S. Chandar, and D. Precup. Environments for lifelong reinforcement learning. *arXiv preprint arXiv:1811.10732*, 2018. 18

[85] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023. 7, 64

[86] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114(13):3521–3526, 2017. 2, 4, 7, 8, 14, 15, 18, 20, 21, 22, 42, 44, 62, 63, 73

[87] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017. 18, 19, 20, 25

[88] I. Kostrikov. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018. 32

[89] H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette. TorchBeast: A PyTorch Platform for Distributed RL. *arXiv preprint arXiv:1910.03552*, 2019. 33, 36

[90] H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The nethack learning environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020. 18, 19, 20, 23

[91] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017. 45

[92] M. D. Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. G. Slabaugh, and T. Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *CoRR*, abs/1909.08383, 2019. 63

[93] K. Lee, C. Hwang, K. Park, and J. Shin. Confident multiple choice learning. In *International Conference on Machine Learning*, pages 2014–2023. PMLR, 2017. 45

[94] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang. Overcoming catastrophic forgetting by incremental moment matching. *Advances in neural information processing systems*, 30, 2017. 44

[95] S. Lee, J. Ha, D. Zhang, and G. Kim. A neural dirichlet process mixture model for task-free continual learning. In *International Conference on Learning Representations*, 2019. 43, 45

[96] S. Lee, S. Purushwalkam Shiva Prakash, M. Cogswell, V. Ranjan, D. Crandall, and D. Batra. Stochastic multiple choice learning for training diverse deep ensembles. *Advances in Neural Information Processing Systems*, 29, 2016. 45

[97] Y. Lee, E. S. Hu, and J. J. Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. 19

[98] T. Lesort, H. Caselles-Dupré, M. Garcia-Ortiz, A. Stoian, and D. Filliat. Generative models from the perspective of continual learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019. 44

[99] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020. 15, 18, 43, 61, 63

[100] S. Lewandowsky and S.-C. Li. Catastrophic interference in neural networks: Causes, solutions, and data. In *Interference and inhibition in cognition*, pages 329–361. Elsevier, 1995. 43

[101] Z. Li and D. Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017. 44

[102] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016. 48

[103] V. Lomonaco, K. Desai, E. Culurciello, and D. Maltoni. Continual reinforcement learning in 3D non-stationary environments. In *Proceedings of the CVPR Workshop on Continual Learning in Computer Vision*, pages 248–249, 2020. 18, 43

[104] V. Lomonaco and D. Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning*, pages 17–26. PMLR, 2017. 63

[105] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6467–6476, 2017. 9, 11, 12, 13, 14, 17, 44, 63

[106] N. Lucchesi, A. Carta, and V. Lomonaco. Avalanche rl: a continual reinforcement learning library. *arXiv preprint arXiv:2202.13657*, 2022. 18

[107] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. 22, 36

[108] Z. Mai, R. Li, J. Jeong, D. Quispe, H. Kim, and S. Sanner. Online continual learning in image classification: An empirical survey. *arXiv preprint arXiv:2101.10423*, 2021. 17

[109] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018. 44

[110] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018. 44

[111] H. Mania, A. Guy, and B. Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 1805–1814, 2018. 18

[112] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989. 2, 9, 42, 43, 45

[113] J. A. Mendez, H. van Seijen, and E. Eaton. Modular lifelong reinforcement learning via neural composition, 2022. 64

[114] M. Mermillod, A. Bugaiska, and P. BONIN. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, 4:504, 2013. 20, 43, 63

[115] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *Proceedings of the NIPS Workshop on Deep Learning*, 2013. 5

[116] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 48, 51

[117] K. Mo, L. J. Guibas, M. Mukadam, A. Gupta, and S. Tulsiani. Where2act: From pixels to actions for articulated 3d objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6813–6823, 2021. 61, 64

[118] M. Mundt, Y. W. Hong, I. Pliushch, and V. Ramesh. A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning. *arXiv preprint arXiv:2009.01797*, 2020. 15, 18, 43

[119] R. Newbury, M. Gu, L. Chumbley, A. Mousavian, C. Eppner, J. Leitner, J. Bohg, A. Morales, T. Asfour, D. Kragic, et al. Deep learning approaches to grasp synthesis: A review. *arXiv preprint arXiv:2207.02556*, 2022. 63

[120] F. Normandin, F. Golemo, O. Ostapenko, P. Rodriguez, M. D. Riemer, J. Hurtado, K. Khetarpal, D. Zhao, R. Lindeborg, T. Lesort, et al. Sequoia: A software framework to unify continual learning research. *arXiv preprint arXiv:2108.01005*, 2021. 18

[121] OpenAI. Robogym. https://github.com/openai/robogym, 2020. 19

[122] OpenAI. Gpt-4 technical report, 2023. 80

[123] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvári, S. Singh, B. V. Roy, R. S. Sutton, D. Silver, and H. van Hasselt. Behaviour suite for reinforcement learning. *CoRR*, abs/1908.03568, 2019. 18

[124] O. Ostapenko, P. Rodriguez, M. Caccia, and L. Charlin. Continual learning via local module composition. *Advances in Neural Information Processing Systems*, 34, 2021. 45

[125] G. Parascandolo, N. Kilbertus, M. Rojas-Carulla, and B. Schölkopf. Learning independent causal mechanisms. In *International Conference on Machine Learning*, pages 4036–4044. PMLR, 2018. 45

[126] P. Parashar, J. Vakil, S. Powers, and C. Paxton. Spatial-language attention policies for efficient robot learning, 2023. 73

[127] J. Pari, N. M. Shafiullah, S. P. Arunachalam, and L. Pinto. The surprising effectiveness of representation learning for visual imitation. *CoRR*, abs/2112.01511, 2021. 61, 73

[128] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. 15, 18, 43, 63

[129] G. I. Parisi, J. Tani, C. Weber, and S. Wermter. Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization. *Frontiers in neurorobotics*, page 78, 2018. 44

[130] D. Park, S. Hong, B. Han, and K. M. Lee. Continual learning by asymmetric loss approximation with single-side overestimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3335–3344, 2019. 44

[131] D. Pathak, D. Gandhi, and A. Gupta. Self-Supervised exploration via disagreement. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the International Conference on Machine Learning (ICML)*, 2019. 45

[132] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine. MCP: learning composable hierarchical control with multiplicative compositional policies. *CoRR*, abs/1905.09808, 2019. 63

[133] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, abs/1802.09464, 2018. 19

[134] E. A. Platanios, A. Saparov, and T. Mitchell. Jelly bean world: A testbed for never-ending learning. In *International Conference on Learning Representations*, 2020. 18

[135] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992. 48

[136] S. Powers, A. Gupta, and C. Paxton. Evaluating continual learning on a home robot, 2023. 69, 71

[137] S. Powers, E. Xing, and A. Gupta. Self-activating neural ensembles for continual reinforcement learning. In S. Chandar, R. Pascanu, and D. Precup, editors, *Proceedings of The 1st Conference on Lifelong Learning Agents*, volume 199 of *Proceedings of Machine Learning Research*, pages 683–704. PMLR, 22–24 Aug 2022. 53

[138] S. Powers, E. Xing, E. Kolve, R. Mottaghi, and A. Gupta. Cora: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. In S. Chandar, R. Pascanu, and D. Precup, editors, *Proceedings of The 1st Conference on Lifelong Learning Agents*, volume 199 of *Proceedings of Machine Learning Research*, pages 705–743. PMLR, 22–24 Aug 2022. 34, 63

[139] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018. 19

[140] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017. 61, 64, 66

[141] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. 80

[142] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018. 19

[143] R. Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308, 1990. 2, 9, 42, 43, 63

[144] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 44

[145] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2018. 44

[146] M. B. Ring. CHILD: A first step towards continual learning. In *Learning to learn*, pages 261–292. Springer, 1998. 44

[147] H. Ritter, A. Botev, and D. Barber. Online structured laplace approximations for overcoming catastrophic forgetting. *Advances in Neural Information Processing Systems*, 31, 2018. 44

[148] A. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995. 44, 63

[149] N. D. Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni. Don't forget, there is more than forgetting: new metrics for continual learning. *CoRR*, abs/1810.13166, 2018. 12

[150] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32, pages 350–360, 2019. 4, 8, 15, 17, 18, 19, 20, 21, 22, 35, 36, 43, 44, 50, 62, 73, 74, 77, 87, 113

[151] D. Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988. 48

[152] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. In *International Conference on Learning Representations (ICLR)*, 2015. 44

[153] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 42, 44, 63

[154] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. In S. Levine, V. Vanhoucke, and K. Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 262–270. PMLR, 13–15 Nov 2017. 63

[155] P. Ruvolo and E. Eaton. ELLA: An efficient lifelong learning algorithm. In *Proceedings of the International Conference on Machine learning (ICML)*, pages 507–515, 2013. 17

[156] M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Kuttler, E. Grefenstette, and T. Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. 4, 5, 18, 19, 20, 23, 30

[157] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun. Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*, 2017. 19

[158] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9339–9347, 2019. 19

[159] J. Schwarz, D. Altman, A. Dudzik, O. Vinyals, Y. W. Teh, and R. Pascanu. Towards a natural benchmark for continual learning. In *Proceedings of the NeurIPS Workshop on Continual Learning*, 2018. 18

[160] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537, 2018. 4, 8, 9, 16, 18, 19, 20, 21, 22, 35, 36, 42, 44

[161] Y. Seo, K. Lee, I. Clavera Gilaberte, T. Kurutach, J. Shin, and P. Abbeel. Trajectory-wise multiple choice learning for dynamics generalization in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:12968–12979, 2020. 45

[162] J. Serra, D. Suris, M. Miron, and A. Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557. PMLR, 2018. 44

[163] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. 46

[164] B. Shen, F. Xia, C. Li, R. Martín-Martín, L. Fan, G. Wang, S. Buch, C. D'Arpino, S. Srivastava, L. P. Tchapmi, et al. igibson, a simulation environment for interactive tasks in large realistic scenes. *arXiv preprint arXiv:2012.02924*, 2020. 19

[165] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017. 44

[166] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022. 64

[167] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. *arXiv preprint arXiv:2209.05451*, 2022. 61, 64, 72

[168] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020. 4, 5, 19, 20, 25

[169] A. Soltoggio, K. O. Stanley, and S. Risi. Born to learn: the inspiration, progress, and future of evolved plastic artificial neural networks. *Neural Networks*, 108:48–67, 2018. 44

[170] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, March 1998. 6

[171] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999. 45

[172] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, O. Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *arXiv preprint arXiv:2106.14405*, 2021. 19

[173] D. Tanneberg, K. Ploeger, E. Rueckert, and J. Peters. Skid raw: Skill discovery from raw trajectories. *IEEE Robotics and Automation Letters*, 6(3):4696–4703, 2021. 63

[174] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 18

[175] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017. 44

[176] A. V. Terekhov, G. Montone, and J. K. O'Regan. Knowledge transfer in deep block-modular neural networks. In *Conference on Biomimetic and Biohybrid Systems*, pages 268–279. Springer, 2015. 44

[177] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 1553–1561. AAAI Press, 2017. 45

[178] R. Traoré, H. Caselles-Dupré, T. Lesort, T. Sun, N. Díaz-Rodríguez, and D. Filliat. Continual reinforcement learning deployed in real-life using policy distillation and sim2real transfer. *arXiv preprint arXiv:1906.04452*, 2019. 44

[179] G. M. Van de Ven and A. S. Tolias. Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*, 2018. 44

[180] J. Veness, T. Lattimore, D. Budden, A. Bhoopchand, C. Mattern, A. Grabska-Barwinska, E. Sezener, J. Wang, P. Toth, S. Schmitt, and M. Hutter. Gated linear networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(11):10015–10023, May 2021. 43, 45

[181] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017. 19

[182] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, mar 1985. 82

[183] J. Wang, X. Wang, Y. Shang-Guan, and A. Gupta. Wanderlust: Online continual object detection in the real world. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10829–10838, 2021. 13

[184] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. *CoRR*, abs/1505.00687, 2015. 93

[185] Y. Wen, D. Tran, and J. Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020. 45

[186] M. Wolczyk, M. Zajac, R. Pascanu, L. Kucinski, and P. Milos. Continual world: A robotic benchmark for continual reinforcement learning. *CoRR*, abs/2105.10919, 2021. 11, 18, 19, 63

[187] S. Woo, J. Park, J. Lee, and I. S. Kweon. CBAM: convolutional block attention module. *CoRR*, abs/1807.06521, 2018. 67

[188] C. Wu, L. Herranz, X. Liu, J. van de Weijer, B. Raducanu, et al. Memory replay gans: Learning to generate new categories without forgetting. *Advances in Neural Information Processing Systems*, 31, 2018. 44

[189] R. Wu, Y. Zhao, K. Mo, Z. Guo, Y. Wang, T. Wu, Q. Fan, X. Chen, L. Guibas, and H. Dong. Vat-mart: Learning visual action trajectory proposals for manipulating 3d articulated objects. *arXiv preprint arXiv:2106.14440*, 2021. 61, 64

[190] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020. 19

[191] Y. Xiang, Y. Fu, P. Ji, and H. Huang. Incremental learning using conditional adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6619–6628, 2019. 44

[192] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 177–186, 2014. 44

[193] E. Xing, A. Gupta, S. Powers, and V. Dean. Kitchenshift: Evaluating zero-shot generalization of imitation-based policy learning under domain shifts. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2021. 19

[194] J. Xu and Z. Zhu. Reinforced continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 899–908, 2018. 44

[195] C. Yan, D. Misra, A. Bennnett, A. Walsman, Y. Bisk, and Y. Artzi. Chalet: Cornell house agent learning environment. *arXiv preprint arXiv:1801.07357*, 2018. 19

[196] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018. 44

[197] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020. 19

[198] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, et al. Transporter networks:

Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pages 726–747. PMLR, 2021. 61, 64

[199] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018. 61, 64

[200] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3987–3995, 2017. 44

[201] C. Zeno, I. Golan, E. Hoffer, and D. Soudry. Task agnostic continual learning using online variational bayes. *arXiv: Machine Learning*, 2018. 42

[202] K. R. Zentner, R. Julian, U. Puri, Y. Zhang, and G. S. Sukhatme. A simple approach to continual learning by transferring skill parameters. *CoRR*, abs/2110.10255, 2021. 63

[203] C. Zhang, O. Vinyals, R. Munos, and S. Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018. 7

[204] J. Zhang, H. Yu, and W. Xu. Hierarchical reinforcement learning by discovering intrinsic options. *arXiv preprint arXiv:2101.06521*, 2021. 45

[205] T. Zhang, Z. Lin, Y. Wang, D. Ye, Q. Fu, W. Yang, X. Wang, B. Liang, B. Yuan, and X. Li. Dynamics-adaptive continual reinforcement learning via progressive contextualization, 2023. 8

[206] G. Zhou, K. Sohn, and H. Lee. Online incremental feature learning with denoising autoencoders. In *Artificial intelligence and statistics*, pages 1453–1461. PMLR, 2012. 44

[207] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine. The ingredients of real-world robotic reinforcement learning. *CoRR*, abs/2004.12570, 2020. 61

[208] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020. 19, 64

Appendices

A Priority Reservoir Sampling Analysis

The replay buffer method used by SANE and SANER uses reservoir sampling [150], which has the property that all samples are maintained with equal probability regardless of when they were collected. While developing SANERv2, we realized this was not sufficient for longer task sequences. We therefore created a modified version, which we call Priority Reservoir Sampling (PRS).

Specifically, we adapted reservoir sampling by uniformly removing a fraction f of samples when drift is detected, to ensure the buffer has a ratio of at least f entries. PRS effectively partitions the buffer: new entries will only compete amongst themselves, leaving previous entries untouched, until the minimum reservoir value equalizes. Thus we still get the benefits of reservoir sampling for the new entries (more diversity than, say, a FIFO buffer), but with a bias towards newer samples.

A.1 Initial Tasks

We begin by walking through the behavior on two sequential tasks, analyzed in the simple case where the same node is activated for all tasks, and entries are removed from its replay buffer at task boundaries. We use maximum reservoir instead of minimum, to make the math slightly clearer.

We define B as the per-node size of our replay buffer, and N as the number of timesteps per task, and assume that $B < N$. Additionally, given some task t, we define m_t as the reservoir value after training on t, and $r_{t,s}$ as the ratio of samples from task s in the buffer at that point (prior to removal).

Since reservoir sampling is uniform, after the first task ($t = 0$) is complete, m is given by:

$$m_0 = \frac{B}{N}$$

On node creation, we remove fraction f of replay entries, leaving our node with $(1-f)B$ task 0 entries. During task 1, those empty entries will tend to fill up, replacing each other, until the maximum of our new entries equals the maximum of our old entries, which we'll call timestep t_1^*:

$$\frac{B}{N} = \frac{fB}{t_1^*}$$
$$t_1^* = fN$$

For the rest of training task 1, all entries have an equal chance of being replaced, as though no replacement occurred. This is represented by what we refer to as an "effective timestep", $t_{e,1} = N + (N - t_1^*)$ timesteps. It follows that:

$$m_1 = \frac{B}{t_{e,1}} = \frac{B}{(2-f)N}$$

We can now analyze the ratio of task 0 in the node's buffer, based on the known bounding minimum values and initial fraction in the buffer:

$$r_{1,0} = (1-f)\frac{m_1}{m_0} = \frac{1-f}{2-f}$$

And thus:

$$r_{1,1} = 1 - r_{1,0} = \frac{1}{2-f}$$

A.2 General Solution

We assume that T tasks have been seen. We thus know $t_T^*, t_{e,T}, m_T, \{r_{T,0}...r_{T,T}\}$ and our goal is to determine $t_{T+1}^*, m_{T+1}, \{r_{T+1,0}...r_{T+1,T}\}$.

First, we'll define the reservoir value for task T : m_T. Based on our observations above, we know that it is equivalent to a standard reservoir sampling value, only sampled at an effective time, instead of the true number of timesteps:

$$m_T = \frac{B}{t_{e,T}} \tag{1}$$

where the effective timestep, $t_{e,T}$, is defined as:

$$t_{e,T} = t_{e,T-1} + N - t_T^* \tag{2}$$

As before, after we begin training on $T+1$, we remove fraction f. New samples will equalize with the old ones at the time t^*_{T+1}:

$$m_T(t_{e,T}) = m_{T+1}(t^*_{T+1})$$
$$t^*_{T+1} = f t_{e,T}$$

We combine this with Equation 2 to obtain the effective time recurrent equation:

$$t_{e,T+1} = (1-f)t_{e,T} + N \tag{3}$$

We can solve this recurrent function to obtain the non-recurrent effective time equation, $t_{e,T}(T)$:

$$t_{e,T}(T) = t_\varnothing + (1-f)^{T+1} N \sum_{i=0}^{T} \frac{1}{(1-f)^i}$$
$$= (1-f)^{T+1} N g(T) \tag{4}$$

Where $t_\varnothing = 0$, and we define $g(T) \equiv \sum_{i=0}^{T} \frac{1}{(1-f)^i}$ for convenience. We can now compute the fraction of samples from task s present in the buffer at subsequent task boundaries:

$$r_{T+1,s} = (1-f) r_{T,s} \frac{m_{T+1}}{m_T}$$

Using Equations 1 and 4, we obtain our **replay ratio equation**:

$$r_{T,s}(T) = r_{s,s}(T) \prod_{i=s}^{T-1} \left(\frac{g(i)}{g(i+1)}\right)$$

The replay ratio equation can only be used for tasks whose samples already exist in the replay buffer, i.e. where $r_{s,s}(T)$ is known; when samples are added for a new task, they are initialized according to:

$$r_{T+1,T+1} = 1 - \sum_{i=0}^{T} r_{T+1,i}(T)$$

Essentially after the reduction in replay ratio is accounted for, for all existing tasks' samples, the new samples fill the remainder of the buffer. We assume that $r_{0,0} = 1$.

Putting all of these together, we now have a way to predict the ratios of samples from each task in our buffer.

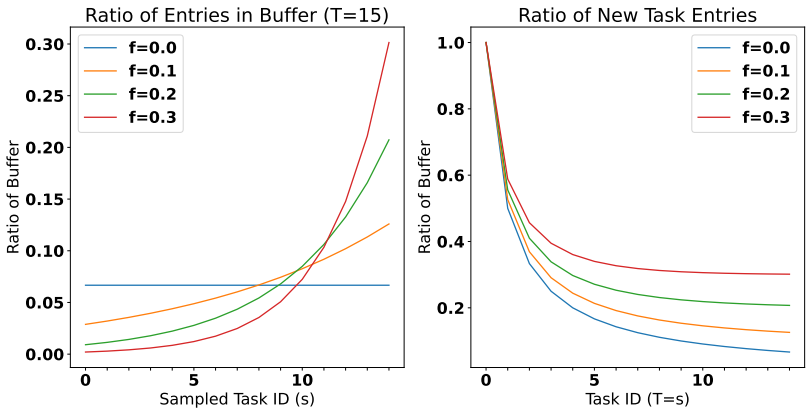


Figure A.1: Visualization of (left) the ratio of replay entries for each task at the end of training, and (right) the ratio of replay entries for the task just trained upon.

A.3 Model Properties

In Figure A.1 we can visualize two key properties: 1) the ratio in the replay buffer of each task s, at the end of training on 15 tasks ($T = 15$); and 2) the ratio for a given task, right after training on it. We can observe that PRS enables the designer to trade-off between ensuring that new tasks will always be represented with at least a ratio of f, and having more samples from old tasks.

Steady-state. Steady-state behavior occurs when the entire duration of training a single task is exactly required to match the prior reservoir value. At this point the buffer has "stabilized"; the minimum reservoir value won't be driven up significantly. However, older replay buffer ratios decay by f, which is significantly more removal than they would otherwise experience.