Continual Reinforcement Learning via Autoencoder-Driven Task and New Environment Recognition

Zeki Doruk Erden
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
zeki.erden@epfl.ch

Donia Gasmi
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
donia.gasmi@epfl.ch

Boi Faltings
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
boi.faltings@epfl.ch

ABSTRACT

Continual learning for reinforcement learning agents remains a significant challenge, particularly in preserving and leveraging existing information without an external signal to indicate changes in tasks or environments. In this study, we explore the effectiveness of autoencoders in detecting new tasks and matching observed environments to previously encountered ones. Our approach integrates policy optimization with familiarity autoencoders within an end-to-end continual learning system. This system can recognize and learn new tasks or environments while preserving knowledge from earlier experiences and can selectively retrieve relevant knowledge when re-encountering a known environment. Initial results demonstrate successful continual learning without external signals to indicate task changes or reencounters, showing promise for this methodology.

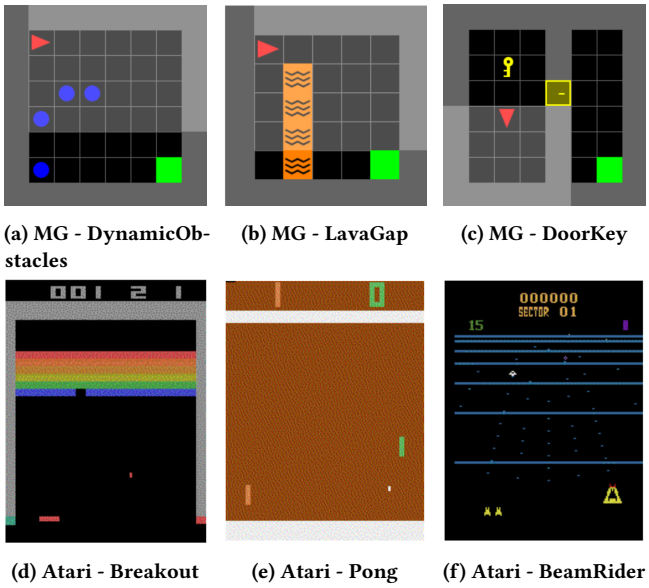
KEYWORDS

Continual learning, Reinforcement learning, Autoencoders

1 INTRODUCTION

Deep reinforcement learning [15] has successfully tackled numerous challenges once deemed among the most difficult for intelligent agents, including tasks like intuition-based gaming [24], real-world robotics [10], and multi-agent systems [28]. However, a major limitation of current systems is their inability to engage in continual learning—learning across dynamic environments without succumbing to destructive adaptation, where previously learned knowledge is lost. This stands in contrast to the abilities of human intelligence. For example, when children learn to ride a bicycle, they acquire the balancing skills necessary to maintain stability. These foundational skills are retained even as they go on to learn a variety of other tasks that also require balance, such as skateboarding or skiing, which do not destructively interfere with the skill of being balanced on a bike. Unfortunately, most approaches proposed to address the issue of destructive adaptation in continual learning in artificial agents rely on constraints within the problem domain, often requiring explicit storage and re-exposure to past data or assuming the presence of external task boundaries or signals for new tasks. These assumptions limit the systems to a constrained, partial form of continual learning.

In this work, we explore the use of autoencoders to enable continual learning in deep reinforcement learning agents, without relying on the limiting assumptions of past data replay or external task-boundary information. To achieve this, we employ an incrementally growing system design where the agent creates and utilizes a new neural network for each distinct environment, ensuring that learning in one environment does not interfere with others. Autoencoders are employed as a mechanism for detecting new environments or matching incoming observations to previously encountered ones. Our approach avoids explicit storage or replay of past samples and does not assume any designer-specified signals to inform the agent about environment changes, exposure to new environments, or which past environment it is currently facing.



(a) MG - DynamicObstacles (b) MG - LavaGap (c) MG - DoorKey

(d) Atari - Breakout (e) Atari - Pong (f) Atari - BeamRider

Figure 1: Our two experimental domains: Minigrid environment subtypes [8] (a-c) and Atari games [25] (d-f) to be learned incrementally.

We assess the efficacy of this approach in progressively learning multiple tasks across two experimental settings (Figure 1). The first setting is Minigrid [8], a straightforward grid environment where an agent learns several tasks within a shared base environment. The second setting involves the Atari benchmarks [25], where the agent is tasked with successively mastering different games without forgetting its ability to play the earlier ones. These are well-known benchmarks for agent reinforcement learning [6, 17]. In these domains, distinct environments and tasks necessitate a diverse set of

Proc. of the Adaptive and Learning Agents Workshop (ALA 2025), Avalos, Aydeniz, Müller, Mohammedalamen (eds.), May 19 – 20, 2025, Detroit, Michigan, USA, ala-workshop.github.io. 2025.

skills, each with varying degrees of relevance depending on the task at hand, making sequential learning a natural approach. In both cases, we illustrate that a conventional reinforcement learning agent experiences a total erosion of prior knowledge, whereas our method enables continual learning without compromising performance.

2 RELATED WORK

Continual learning, also known as incremental or lifelong learning, refers to the ability of AI systems to learn sequentially from an ongoing stream of tasks [11, 27]. A key challenge in continual learning is "catastrophic forgetting"—or, as we prefer to call it, "destructive adaptation"[1]. When new examples differ significantly from prior ones, they can overwrite previously learned knowledge in the network—a problem for which no reliable solution currently exists [11, 18, 30].

Fixed-capacity systems are inherently inadequate for addressing this problem: since neural networks encode information in a distributed fashion, the entire capacity is utilized by previous tasks. As a result, existing knowledge is eventually (and often immediately) overwritten when new tasks significantly diverge from earlier ones. On the other hand, methods that expand capacity face their own limitations. These systems cannot autonomously decide when to increase capacity, how to allocate new components to different tasks, or how to select the appropriate component when revisiting past tasks. For instance, in [22], external signals are required both to recognize a new task and to specify which past task is being revisited. This limitation also affects methods that explicitly store information about previous task solutions, such as [14]. Some extensions attempt to address these issues but still rely on task labels during adaptation and lack mechanisms for detecting new tasks, as seen in [12]. Consequently, these methods fail to provide systems with fully autonomous continual adaptation capabilities.

Other approaches eliminate the need for explicit "task boundaries" by relying on the storage and replay of past data [4, 7], as noted in Table 1 of [5]. However, this strategy is impractical in many real-world settings, not only because it does away with one of the primary motivators of continual learning, but also due to the extensive memory demands over a system's lifetime, as well as concerns related to confidentiality and data protection.

The challenges of continual learning and destructive adaptation extend from the domain of supervised learning to reinforcement learning models as well [1, 2, 13]. Similarly, the solutions face comparable limitations, with reinforcement learning methods often relying on either the storage or replay of past data [21, 26], or the use of externally signaled task boundaries [22, 23].

Our approach draws inspiration from the use of autoencoders for novelty detection [3, 9, 12, 20], as well as the application of autoencoders in continual learning for supervised tasks by matching observed samples to previously learned "tasks" without needing external specification of the current task [3, 12] (but still requiring external signification of an incoming new task). While novelty detection has been applied to reinforcement learning in some cases (e.g., [30]), to our knowledge, it has not yet been used in conjunction with autoencoders to achieve full continual learning capabilities. This includes new task detection and prior task recognition, without the need for external signals or data replay, and without simplifying assumptions.

[1]We use the term "destructive adaptation" to avoid the anthropomorphic connotation of "catastrophic forgetting," which inaccurately suggests a gradual, human-like forgetting process. Instead, the phenomenon involves the active overwriting of past information, an issue that affects all adaptive systems.

3 AUTOENCODER-DRIVEN TASK AND NEW ENVIRONMENT RECOGNITION

3.1 Motivation and basis

Our goal is to facilitate continual learning without destructive interference, avoiding explicit storage or replay of past samples, and without relying on external cues to indicate which task the agent is encountering or whether the current task is new or previously seen. In essence, we aim for the agent to preserve past knowledge intact, retrieve it as needed based on the demands of the current environment, and recognize when it should be acquiring new knowledge rather than reusing what it has already learned.

As outlined in Section 2, any reliable method that ensures no loss of knowledge over an indefinite period and across an arbitrary number of tasks must incorporate mechanisms for expanding learning capacity. Accordingly, our design utilizes a system of multiple neural networks, each functioning as a policy network, added incrementally. Each policy network is tailored to a specific task or environment[2]. This approach aligns with established methods in the literature, where different tasks are associated with distinct, mostly or entirely independent neural networks [12, 22], facilitating the preservation of previously acquired knowledge.

3.2 Task recognition

Building on the foundation of multiple task-specific policy networks, our next goal is to implement a mechanism that can (1) assign the appropriate policy network based on the environment the agent encounters (retrieval) and (2) detect when the agent faces a previously unseen environment (new environment recognition), all without relying on external signals. To achieve this, we propose learning the features of the encountered environment during training, employing undercomplete autoencoders as a mainstream, efficient approach to do so [16]. We leverage the ability of the autoencoder to reconstruct these features as an indicator of whether an incoming observation matches a previously learned environment, and as a quantification of the degree of match.[3] Specifically, an undercomplete autoencoder (with a bottleneck layer smaller than the input/output dimensionality) learns to encode the key features necessary for reconstructing the original observation (e.g., an image) in a lower-dimensional space. This encoding is task-dependent, emphasizing features prominent to each training task. As a result, incoming data outside the distribution the autoencoder

[2]We assume that a task change corresponds to an observable change in the environment. The generalization of our approach to handle changes in the environment's reward structure without observable changes is straightforward and covered in Section 6

[3]While simpler statistical methods, such as using averages and standard deviations per observation, can suffice in domains where tasks exhibit clear and broad differences, they fail in settings where most of the environment remains identical across tasks, with only subtle, sometimes conditional changes. The Minigrid [8] domain we experiment on is an example of this. The most general way to distinguish between environments, even in the presence of subtle variations, is to aim for a full reconstruction, capturing all conditional relationships.

was trained on will manifest as imperfect reconstructions, signaling a new environment.

Our complete design is as follows (see Figure 2 for a summary, which is described in detail below):

For each new environment, a policy network and a corresponding autoencoder are initialized. As the agent interacts with the environment and learns the policy, it simultaneously gathers observations. After the policy training is completed, these observations are used to train an environment-specific autoencoder. This autoencoder learns to capture the key features of the environment by minimizing the reconstruction error between the input observations and the reconstructed output.

Assumption: We assume that a specific environment type remains available throughout training until the policy converges, or alternatively, that past training data is stored until the policy on a given environment reaches convergence (not to be confused with the storage of past data indefinitely for replay purposes, which we specifically avoid). This assumption does not impact the theoretical capabilities of our method; it merely defines what constitutes "one distinct environment/task." This assumption can be relaxed if we accept potential overfitting to different variations of the "same" environment (e.g., different room layouts in Minigrid Multiroom). In such cases, the agent would learn distinct policies for each unique variation (e.g., in Minigrid Multiroom, rooms with opposing doors versus neighboring doors would be treated as different environments). Alternatively, if the agent encounters frequently changing subtypes of an environment before convergence, all such variations would be grouped under the same policy network, and the associated autoencoder would adapt to encompass all subtypes (for instance, one policy could learn both Multiroom and Lava subtypes in Minigrid if these variants are repeatedly observed). These variations do not degrade performance or continual learning; they only affect the final configuration of the policy and autoencoder networks.

Once the autoencoder is trained, an estimation of a *reconstruction error threshold* is computed based on the distribution of reconstruction errors during training, *per task*. We do this by using a batch-wise averaging approach. First, the reconstruction errors are calculated for batches of validation observations. A Gaussian distribution is then fitted to the batch-wise average reconstruction errors. The threshold is determined by calculating the value that corresponds to a specified confidence level (e.g. 99%) using the cumulative distribution function (CDF) of the fitted distribution. This threshold ensures that reconstruction errors exceeding it are flagged as indicative of a novel environment. This approach enables us to dynamically determine the expected reconstruction performance for different environment types—an important consideration, as our preliminary experiments revealed that typical reconstruction errors vary significantly across tasks. As a result, a fixed reconstruction threshold cannot be applied uniformly.

To recognize novelty, the collected observations are passed through a set of trained autoencoders from previously encountered environments. If all the autoencoders produce reconstruction errors above their respective estimated thresholds (see above), the environment is classified as novel, prompting the training of a new autoencoder and policy network for the new environment. However, if one or more autoencoders reconstruct the observations with an error below their thresholds, the environment is recognized as familiar. In such cases, the autoencoder with the smallest reconstruction error is identified as the best match, and its corresponding policy network is used to continue interacting with the environment.

3.3 Summary and illustrative example

Figure 2 illustrates the agent's novelty recognition process using autoencoders, exemplified with some reconstructions from our experiments in Minigrid environment variants. Here, the agent's knowledge includes three environments: LavaGap, DoorKey, and DynamicObstacles. The agent's current observation (from the LavaGap environment) is passed through all three autoencoders for the three environment types. In the first alternative outcome (Option 1 in Figure 2), Autoencoder 3 (trained on LavaGap) yields the lowest reconstruction error, below its corresponding threshold, allowing the agent to identify the environment as the one that its policy, Policy 3, was trained on (again, LavaGap); hence it retrieves this policy model for interaction with the environment. If, on the other hand, none of the autoencoders match (i.e. all reconstruction errors are above their thresholds - Option 2 in Figure 2), the agent recognizes the environment as novel. For example, if a fourth environment is introduced, the agent would start training a new PPO model and a new autoencoder, and estimate a threshold for the new task, adding it to its knowledge for future encounters. In that manner, the agent can learn an arbitrary number of future tasks, without destroying knowledge about any of the past tasks.

Finally, we would like to note that while we used this framework of associated autoencoders in conjunction with standard neural networks as basis, the method has no conflict with and can just as well be used with other continual learning frameworks as basis instead - particularly some established methods that require task boundaries provide good candidates for such an integration [14, 22].

4 EXPERIMENTAL SETUP

4.1 Implementational details of the system

Policy: For the implementation of Proximal Policy Optimization (PPO), we used Stable Baselines3. The default settings were kept. We used CnnPolicy due to the image-based nature of the environments. The optimizer used is Adam, with a learning rate of $3e-4$.

Autoencoder: We trained a convolutional autoencoder to reconstruct environment observations. The convolutional autoencoder for MiniGrid has input images of shape (7, 7, 3), uses two Conv2D layers in the encoder (16 and 8 filters, 3x3 kernels, ReLU activation), each followed by 2x2 MaxPooling. The decoder employs two Conv2DTranspose layers for upsampling, followed by a final Conv2D layer (3 filters, sigmoid activation) to reconstruct the original input. For Atari input images have shape (84, 84, 1) (downscaled and grayscaled from 210x160x3), uses two Conv2D layers in the encoder (16 and 8 filters, 3x3 kernels, ReLU activation), each followed by 2x2 MaxPooling to downsample. The decoder uses two Conv2DTranspose layers for upsampling, followed by a final Conv2D layer (1 filter, sigmoid activation) to reconstruct the grayscale input. The models were optimized using the Adam optimizer and a binary cross-entropy loss function, with early stopping based on validation loss to prevent overfitting. The autoencoders

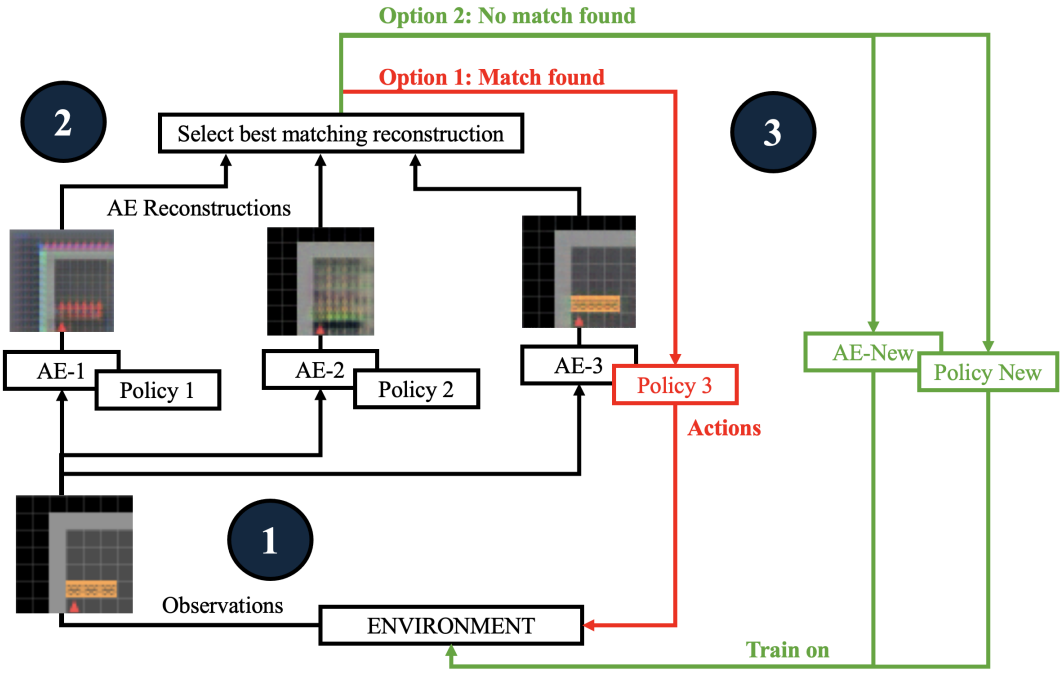


Figure 2: Overview of our system design. The system at any time is composed of a number of policy networks (three in this example) and an autoencoder (AE) associated with each of them. (1) The agent obtains observations from the environment (as images in our implementation). (2) The observations are passed as input to all autoencoders of all policy networks available for them to attempt reconstruction. The error on each autoencoder's reconstruction is computed. (3) If there are autoencoders whose reconstruction errors are below automatically estimated reconstruction thresholds (see main text), then the policy network associated with the autoencoder with the lowest reconstruction error is chosen (in this figure, that's Policy 3, meaning that AE-3 provided the lowest reconstruction error). If all reconstructions have errors above the threshold, this is interpreted as the observation of an unrecognized environment. A new policy network and a new associated autoencoder are created for training on this new task. (The illustrative reconstructions on this figure are actual in-operation outputs by our system, for tasks corresponding to Minigrid's Dynamic-Obstacles, Key-Door and Lava-Gap environments.)

were trained for up to 100 epochs with a batch size of 64, using 20% of the data for validation. Training and validation loss were monitored and plotted to assess model performance. For the desired confidence level to determine autoencoder reconstruction thresholds, we used 90% in Minigrid experiments and 99% in Atari experiments (we used a stricter threshold for Atari since they are visually more similar).

We classify the observations (or identify them as new) and select a policy network only once per episode, at the outset. This approach assumes that there is no change in the environment during a single episode, but only between episodes. However, this assumption can be relaxed without altering our system design if the environment is non-episodic.

4.2 Experiment details

In our experiments, we use Minigrid [8] and Atari environments provided by the OpenAI/Gym toolkit [25]. We test with three environment subtypes (i.e. tasks) from each domain: In Minigrid, we use the variants Dynamic-Obstacles-8x8-v0 (Task T1), LavaGapS7-v0 (Task T2), and DoorKey-8x8-v0 (Task T3). In Atari, we test with games Breakout (Task T1), Pong (Task T2), and BeamRider (Task T3), all v5. All our results with Minigrid are averages of 8 runs, and with Atari they are averages of 3 runs (the lower number of runs was due to the computational cost of learning Atari environments due to their high dimensionality). In all our experiments, the agent gets observations as images (RGB for Minigrid, grayscale for Atari). Since reward magnitudes differ greatly across tasks, reported performances are normalized to the range of $[0, 1]$, with normalization limits determined by the highest and lowest rewards obtained by the agent in the corresponding environment.

We conduct our experiments with an agent implementing our design, referred to as the "AE-CL Agent" (short for "autoencoder continual learning"), alongside a vanilla agent that utilizes a single policy network for comparison. We do not include any additional baseline comparisons, as we are unaware of any comparable continual reinforcement learning methods that can achieve continual learning without the external specification of task IDs or new task information, or without replaying past samples. Comparing our approach to methods that rely on these constraints would not provide meaningful insights, as our primary objective is to showcase the capability of our design to function without such

constraints. Moreover, we do not present this method as an alternative to existing techniques that operate under these assumptions (e.g. [14, 22]); rather, it has the potential to work in conjunction with them—especially those requiring task boundaries without automatic detection. Nothing in our design prevents an integration with such existing approaches in place of standard neural networks as policy basis as we use them, as discussed in detail on Sections 3 and 6.

4.2.1 Learning flow 1: Retrospective Performance. In this learning flow, we specifically demonstrate the agent's knowledge preservation and retrieval performance on previous tasks. The agent is sequentially exposed to each task, and after learning one task, it is tested for average performance across all the tasks that it was exposed to until that point. Specifically, the training and testing process unfolds as follows:

(1) The agent is trained on the first task (T1). After learning T1, the agent was tested on 30 episodes of T1 only.
(2) The agent is trained on the second task (T2). After learning T2, the agent was evaluated on 60 episodes, split equally between T1 and T2. The transitions between episodes were random, with the environment selection (T1 or T2) chosen randomly for each episode.
(3) The agent is trained on the second task (T3), and following that the agent was tested across 90 episodes where the environments (T1, T2, T3) were randomly selected. One-third of the episodes were from each task (T1, T2, and T3).

This flow is visualized on Figure 3. We would like to reiterate that at no point during the training process do we inform the agent of a task change, the introduction of a new task, or which of the previous tasks it is currently encountering.

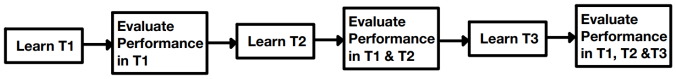


Figure 3: Learning Flow 1 (Retrospective performance)

4.2.2 Learning flow 2: Ongoing Performance. In this learning flow (visualized in Figure 4), we test the AE-CL agent's performance in a more natural, ongoing operation. We provide the agent with a distinct environment at each step, and we track the net performance of the agent across all these steps, without distinction across tasks; arguably simulating a more natural scenario compared to the retrospective evaluations in the first training flow. As congruent with our assumption (Section 3), we assume that an environment remains accessible until the convergence of the agent on that environment. To test continual learning performance, both AE-CL and Vanilla agents are trained on a given task only at their first exposure to this task (note that AE-CL does this automatically, while for Vanilla agent this was done manually).

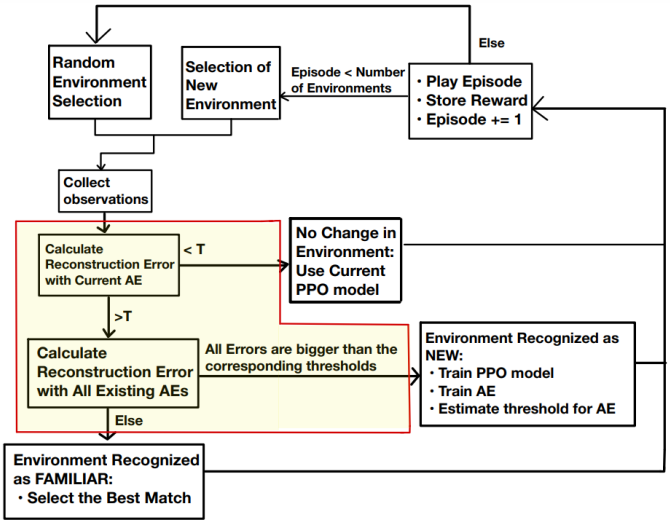


Figure 4: Learning Flow 2 (Ongoing performance)

Agent	Task 1	Task 2	Task 3
AE-CL	0.901 (0.013)	0.887 (0.007)	0.907 (0.004)
Vanilla	0.907 (0.013)	0.430 (0.003)	0.324 (0.001)

Table 1: Retrospective performances of AE-CL and Vanilla RL agents on Minigrid. Displayed values under Task X are average rewards (normalized) across all tasks up to and including Task X (e.g. the performance under Task 2 is the retrospective performance on Task 1 and 2 combined). Tasks 1, 2 and 3 are DynamicObstacles, LavaGap and DoorKey environments respectively. Results are averaged over 8 independent runs, inside parentheses are standard deviations.

5 RESULTS AND DISCUSSION

5.1 Number of distinct poliy-autoencoder pairs learned

In all our experiments we saw that the agent learned precisely three policy-autoencoder pairs, without any tasks missed or without any multiple unnecessary policy-autoencoder pairs for a single task.

5.2 Retrospective performance

Tables 1 and 2 present the retrospective performance evaluations for the Minigrid and Atari environments, respectively. In all instances, the AE-CL agent (our proposed design) demonstrates consistent average performance across all tasks introduced up to that point. For example, in the Atari environment (Table 2), the agent achieves an average normalized reward of 0.945 across Tasks 1, 2, and 3 combined (after being trained on Task 3), which is nearly identical to its original single-task performance of 0.951 on Task 1, despite not being retrained on Tasks 1 and 2. In contrast, the Vanilla agent experiences destructive adaptation, losing all knowledge of previous tasks upon the introduction of a new one, resulting in a normalized reward of approximately 1/X after task index X (i.e. achieving a reward close to 1.0 on the latest task but around 0 for all prior tasks).

Agent	Task 1	Task 2	Task 3
AE-CL	0.951 (0.026)	0.939 (0.010)	0.945 (0.007)
Vanilla	0.947 (0.026)	0.469 (0.006)	0.310 (0.002)

Table 2: Retrospective performances of AE-CL and Vanilla RL agents on Atari. Displayed values under Task X are average rewards (normalized) across all tasks up to and including Task X. Tasks 1, 2 and 3 are Breakout, Pong and Beamrider respectively. Results are averaged over 3 independent runs, inside parentheses are standard deviations.

Figure 5 gives a different view on destructive adaptation of the Vanilla Agent, where the performance on any prior task is seen to have decayed to 0 after being trained on a new task. In contrast, the AE-CL agent (Figure 6) retains original performance in prior environments after being trained on new ones, showing no sign of destructive adaptation.

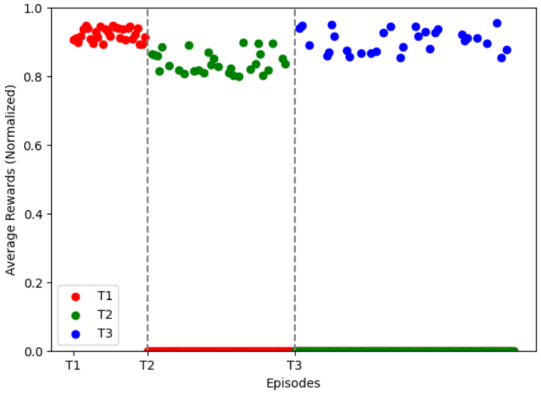


Figure 5: Normalized rewards obtained by Vanilla agent during retrospective performance evaluation at the three tasks. Labels T1, T2 and T3 on x-axis signify the time of training on the corresponding task, followed by subsequent evaluation on tasks up to that point. The plot shows clear destructive adaptation as performance in prior tasks are not retained.

The results on retrospective performances learning flow shows that the AE-CL agent can accurately recognize new tasks and correctly assign observations from previously-encountered tasks to their corresponding policy-autoencoder pairs, hence realizing continual learning without any external task change or new task signals.

5.3 Ongoing performance

Table 3 presents the average performance of the AE-CL agent during ongoing performance evaluations across three tasks, demonstrating high average performance across trials after learning three policy-autoencoder pairs during its initial encounters with each environment (as discussed at the beginning of this section). In contrast, Vanilla agent once again shows a performance close to 1/3rd of maximum performance, demonstrating that the knowledge of all tasks but the latest-trained one are lost. Figures 7 and 8 illustrate

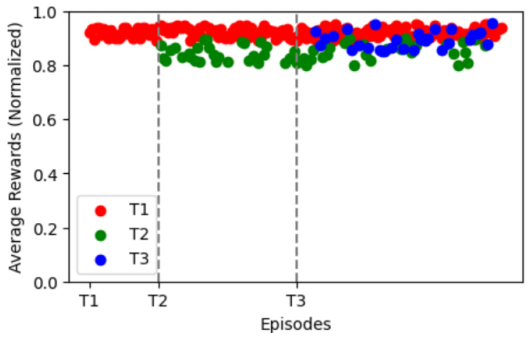


Figure 6: Normalized rewards obtained by AE-CL agent during retrospective performance evaluation at the three tasks. Labels T1, T2 and T3 on x-axis signify the time of training on the corresponding task, followed by subsequent evaluation on tasks up to that point. Performance retained without any loss upon the introduction of new tasks.

Agent	AE-CL	Vanilla
Minigrid	0.895 (0.005)	0.283 (0.080)
Atari	0.942 (0.008)	0.319 (0.080)

Table 3: Net average rewards (normalized) of AE-CL and Vanilla agents during ongoing evaluation. Results are averaged over 8 independent runs for Minigrid and 3 runs for Atari, inside parentheses are standard deviations.

this for 50 successive episodes randomly selected from our Minigrid experiments. The Vanilla agent (Figure 7) performs well in approximately 1/3rd of episodes while completely failing at the rest. AE-CL agent (Figure 8), on the other hand, achieves near-optimal performance across all episodes except for one instance (episode 45), where the environment subtype was correctly identified as LavaGap, but the agent failed due to an incorrect action taken by the policy network (which is not unlikely in LavaGap environment, since falling into lava cells by mistake results in immediate episode termination). This indicates that the AE-CL agent is capable of accurately detecting and classifying the environments it encounters in an ongoing manner, retrieving relevant past knowledge, and performing effectively based on that retrieval.

6 CONCLUSIONS

In this study, we proposed a training flow for continual learning based on dynamically growing system capacity to avoid destructive adaptation, combined with autoencoders for task assignment and new environment detection. We showed that this simple design can successfully detect new environments and accurately assign observations to previously-encountered environments if they provide a match, hence enabling continual learning without requiring external signals for neither of these tasks, and without requiring storage of past training data. Our system design can be used either as it is presented in this paper (with vanilla neural networks) or it can be used in conjunction with other methods that typically

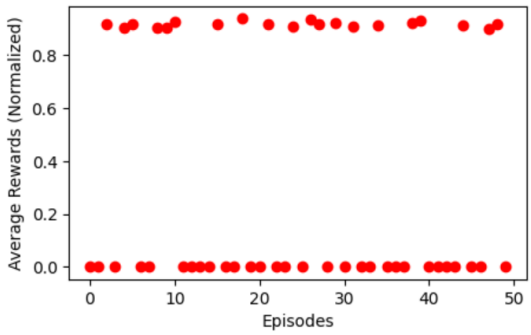


Figure 7: Normalized rewards obtained by the Vanilla agent on Minigrid across 50 episodes during ongoing performance evaluation. The environment subtype is chosen randomly at each episode.

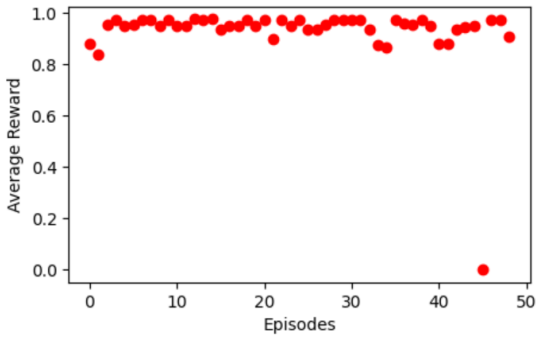


Figure 8: Normalized rewards obtained by the AE-CL agent on Minigrid across 50 episodes during ongoing performance evaluation. The environment subtype is chosen randomly at each episode.

assume the existence of task boundaries, such as Progressive Neural Networks [22] or Elastic Weight Consolidation [14].

6.1 Limitations

Our current design is tailored for detecting changes in the environment, signaled by shifts in the observation distribution, which affects the autoencoder's ability to accurately reconstruct the original observation. The key constraint here, with respect to a fully generalized continual learning setup, is the assumption that each new learning task is linked to a novel environment. While this assumption holds for many problem scenarios, in principle, a "task change" could also be defined by a shift in the reward structure within the same environment. Our framework can easily be adapted to accommodate such cases by extending the autoencoder's inputs and outputs to include the rewards obtained by the agent. This would allow the system to detect changes in both the reward structure and the underlying observations, thereby generalizing its continual learning capabilities.

While our method enables the learning of an arbitrary number of tasks without destructive adaptation due to its ability to grow capacity as needed, this also implies that memory requirements may become substantial for very long-lived agents. Each environment/task is represented by a distinct policy-autoencoder network, which presents a shared limitation across continual learning methods that increase capacity [12, 22]. Approaches that do not expand capacity (e.g., [14]) circumvent this limitation; however, as discussed, they cannot continue learning indefinitely long sequences of tasks due to their finite capacity. We believe that an effective solution to this limitation lies in mechanisms that can incrementally add partial capacity (such as network components, neurons, or layers, rather than entire neural networks), which would help control the increase in complexity, ideally following a logarithmic trend as the capacity required decreases with the agent's expanding experience. This concept also ties into our discussion of transfer learning in the subsequent subsection.

6.2 Future work

A key motivation for continual learning is knowledge transfer, where knowledge gained from past tasks is used to enhance performance on subsequent ones [29]. Our current system design and experiments do not yet incorporate this possibility, as the agent begins learning a new policy network and a new autoencoder with each new task it encounters. However, the design can be extended to integrate existing transfer learning techniques, allowing it to leverage prior knowledge from previous environments. One straightforward approach would be to initialize the policy network for a new task using the network associated with the autoencoder that best matched the new observation during reconstruction. We conducted preliminary experiments with this transfer method but did not report the results, as we observed no significant impact on performance or training progression. This may be due to a lack of similarity across environments in our experimental domain, where it is more efficient to learn a new network from scratch than to reuse an existing one. In scenarios with greater overlap between tasks or environments, this transfer strategy could have a more pronounced effect. Alternatively, it is possible that such a simple transfer scheme is inherently ineffective, and more advanced transfer learning techniques may be needed to effectively harness prior knowledge and further enhance training - our method, currently built on standard neural networks, can be easily extended to incorporate such alternative transfer learning approaches.

While our primary emphasis throughout this paper has been on the quantitative performance of continual learning, it is important to recognize that our method not only facilitates continual learning without destructive adaptation but also decouples and represents different tasks (currently represented as distinct environment subtypes, but not necessarily limited to that, as previously discussed) as isolated behavioral subunits. This aspect may be of particular interest to researchers engaged in Hierarchical Reinforcement Learning [19], which, among other objectives, seeks to represent distinct components of an overall behavioral pattern as separate entities. Although we did not explore this dimension of our system in this study, we believe it presents a promising avenue for future research for those interested in this field.

REFERENCES

[1] Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. 2023. Loss of plasticity in continual deep reinforcement learning. In *Conference*

on Lifelong Learning Agents. PMLR, 620–636.

[2] David Abel, André Barreto, Benjamin Van Roy, Doina Precup, Hado P van Hasselt, and Satinder Singh. 2024. A definition of continual reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2024).

[3] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. 2017. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3366–3375.

[4] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. 2019. Task-free continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11254–11263.

[5] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. 2020. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems* 33 (2020), 15920–15930.

[6] Michael Chang, Sid Kaushik, S Matthew Weinberg, Tom Griffiths, and Sergey Levine. 2020. Decentralized reinforcement learning: Global decision-making via local economic transactions. In *International Conference on Machine Learning*. PMLR, 1437–1447.

[7] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2018. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420* (2018).

[8] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. 2023. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. *CoRR* abs/2306.13831 (2023).

[9] Francesco Del Buono, Francesca Calabrese, Andrea Baraldi, Matteo Paganelli, and Francesco Guerra. 2022. Novelty detection with autoencoders for system health monitoring in industrial environments. *Applied Sciences* 12, 10 (2022), 4931.

[10] Luíza Caetano Garaffa, Maik Basso, Andréa Aparecida Konzen, and Edison Pignaton de Freitas. 2021. Reinforcement learning for mobile robotics exploration: A survey. *IEEE Transactions on Neural Networks and Learning Systems* 34, 8 (2021), 3796–3810.

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

[12] Maxwell J Jacobson, Case Q Wright, Nan Jiang, Gustavo Rodriguez-Rivera, and Yexiang Xue. 2022. Task Detection in Continual Learning via Familiarity Autoencoders. In *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 1–8.

[13] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. 2022. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research* 75 (2022), 1401–1476.

[14] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

[15] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[16] Umberto Michelucci. 2022. An introduction to autoencoders. *arXiv preprint arXiv:2201.03898* (2022).

[17] Volodymyr Mnih. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[18] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural networks* 113 (2019), 54–71.

[19] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. 2021. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–35.

[20] Stanislav Pidhorskyi, Ranya Almohsen, and Gianfranco Doretto. 2018. Generative probabilistic novelty detection with adversarial autoencoders. *Advances in neural information processing systems* 31 (2018).

[21] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. 2019. Experience replay for continual learning. *Advances in neural information processing systems* 32 (2019).

[22] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).

[23] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. 2018. Progress & compress: A scalable framework for continual learning. In *International conference on machine learning*. PMLR, 4528–4537.

[24] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[25] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. 2024. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032* (2024).

[26] René Traoré, Hugo Caselles-Dupré, Timothée Lesort, Te Sun, Guanghang Cai, Natalia Díaz-Rodríguez, and David Filliat. 2019. Discorl: Continual reinforcement learning via policy distillation. *arXiv preprint arXiv:1907.05855* (2019).

[27] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024. A comprehensive survey of continual learning: theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[28] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2021. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control* (2021), 321–384.

[29] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2020), 43–76.

[30] Geigh Zollicoffer, Kenneth Eaton, Jonathan C Balloch, Julia Kim, Mark Riedl, and Robert Wright. [n.d.]. Novelty Detection in Reinforcement Learning with World Models. In *First Reinforcement Learning Safety Workshop*.