HAL
archives-ouvertes.fr

DISCORL: Continual reinforcement learning via policy distillation

René Traoré, Hugo Caselles-Dupré, Timothée Lesort, Te Sun, Guanghang Cai, David Filliat, Natalia Díaz-Rodríguez

To cite this version:

René Traoré, Hugo Caselles-Dupré, Timothée Lesort, Te Sun, Guanghang Cai, et al.. DISCORL: Continual reinforcement learning via policy distillation: A preprint. NeurIPS workshop on Deep Reinforcement Learning, Dec 2019, Vancouver, Canada. hal-02381494

HAL Id: hal-02381494

https://hal.archives-ouvertes.fr/hal-02381494

Submitted on 26 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

arXiv:1907.05855v1 [cs.LG] 11 Jul 2019

DisCoRL: Continual Reinforcement Learning via Policy Distillation

A Preprint

René Traoré[*,1], Hugo Caselles-Dupré[*,1,3], Timothée Lesort[*,1,2], Te Sun[1], Guanghang Cai[1], Natalia Díaz-Rodríguez[1], and David Filliat[1]

[1]Flowers Team (ENSTA Paris, Institute Polytechnique de Paris & INRIA).
[2]Thales, Theresis Laboratory.
[3]AI Lab, Softbank Robotics Europe
[*]Equal contribution.

July 15, 2019

Abstract

In multi-task reinforcement learning there are two main challenges: at training time, the ability to learn different policies with a single model; at test time, inferring which of those policies applying without an external signal. In the case of continual reinforcement learning a third challenge arises: learning tasks sequentially without forgetting the previous ones. In this paper, we tackle these challenges by proposing DisCoRL, an approach combining state representation learning and policy distillation. We experiment on a sequence of three simulated 2D navigation tasks with a 3 wheel omni-directional robot. Moreover, we tested our approach's robustness by transferring the final policy into a real life setting. The policy can solve all tasks and automatically infer which one to run.

1 Introduction

An autonomous agent should be able to learn and exploit its knowledge in any situation all along its life. In a sequence of learning experiences, it should therefore be able to build representations and skills that can be reactivated and reused later. Machine learning is a research area that addresses the problem of learning automatically from any agent experiences. Nevertheless several challenges still block the way toward a fully autonomous agent. In particular, we focus on the situations when the agent needs to learn skills sequentially into a single model and use them independently afterwards.

This challenge is partially addressed by a sub-domain of machine learning called multi-task learning. Multi-task learning [Caruana, 1997] studies how to optimize several problems *simultaneously* with a single model. However, when those problem can not be optimized at the same time and have to be learned sequentially, we identify the learning setting as a continual learning problem [Lesort et al., 2019]. In this paper, we propose to address a continual learning problem of reinforcement learning (RL). In this continual learning setting, each learning experience is called a task and a task solution is a policy.

The goal is to propose a learning setting compatible with a real autonomous agent. To this end, we propose three simulated robotics tasks and propose an approach that will solve such tasks sequentially. At each task, the agent should learn a policy based on a reward function and a RL algorithm.

RL is a popular framework to learn robot controllers that also has to face the CL challenges. In order to fit reinforcement learning into a continual setting, we use a method called policy distillation [Rusu et al., 2015] that allows to transfer several policies learned sequentially into a single model. To validate our approach, we evaluate the final results on the three simulated learning setting but also in a real life setting similar to the simulation (Figure 1). It is important to note that, at test time, the agent does not have access to a task label to determine which policy to run, and thus, it needs to figure it out by itself from its observations.

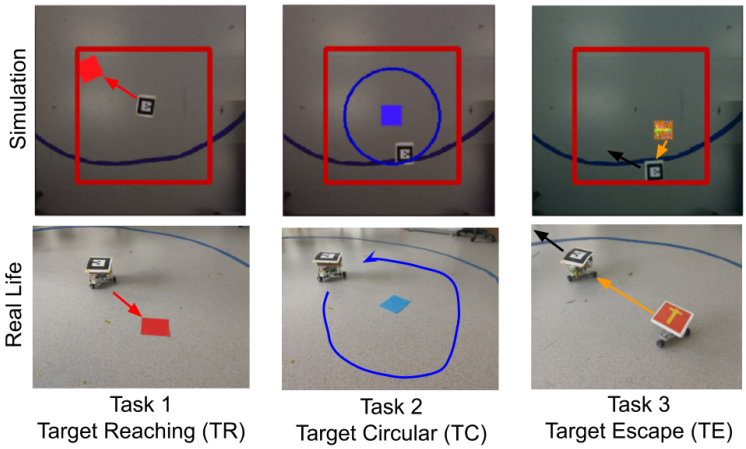

Figure 1: Image of the three tasks, in simulation (top) and in real life (bottom) sequentially experienced. Learning is performed in simulation, real life is only used at test time.

Our contribution is to propose **DisCoRL** (*Distillation for Continual Reinforcement learning*): a modular, effective and scalable pipeline for continual RL. This pipeline uses policy distillation for learning without forgetting, without access to previous environments, and without task labels. Our results show that the method is efficient and learns policies transferable into real life scenarios.

The article is structured as follows. Sec. 2 introduces related work, Sec. 3 details the methods utilized, Sec. 4 describes the robotics setting and tasks, Sec. 5 presents the experiments performed, and Sec. 6 concludes with future insights from our experiments.

2 Related work

Multi-task RL: The objective of Multi-task learning (MTL) [Caruana, 1997] is to learn several tasks simultaneously; generally by training tasks in parallel with an unique model. Therefore, multi-task RL aims at constructing one single policy that can solve a number of different tasks. Note how in classification this problem is quite simple, as data from all tasks just have to be shuffled randomly and can then be learned all together at once. However, in RL environments, data is sampled on sequences that can not be shuffled randomly with all other environments because the environments are not accessible simultaneously. Learning multiple tasks at once is thus more complicated.

Policy distillation [Rusu et al., 2015] can be used to merge different policies into one single module/network. This approach uses two models, a trained policy (the teachers) to annotate data with soft-annotations, and a model to learn from the former (the student). The student is trained in a supervised manner with the soft-labels. The soft-annotation is supposed to help the student to learn faster than the teacher did [Furlanello et al., 2018]. Policy distillation can be used then to learn several policies separately and simultaneously, and distill them into a single model as in the distral algorithm [Teh et al., 2017]. In our approach, we also use distillation but we do not keep the teacher model, we just label a set of data and then delete the teacher. Furthermore, tasks are learned sequentially, and not simultaneously. Other approaches such as SAC-X [Riedmiller et al., 2018] or HER [Andrychowicz et al., 2017] take advantage of Multi-task RL by learning auxiliary tasks in order to help learning a main task. This approach is extended in the CURIOUS algorithm [Colas et al., 2018]. It selects tasks to be learned that improve an absolute learning progress metric the most.

Continual Learning: Continual learning (CL) is the ability of a model to learn new skills without forgetting previous knowledge [Lesort et al., 2019]. Continual learning is in many ways similar to multi-task learning but past task can not be replayed. In our context, it means learning several tasks sequentially and being able to solve any of the learned tasks at the end of the sequence.

Most CL approaches can be classified into four main methods that differ in the way they handle the memory from past tasks. The first method, referred to as *rehearsal*, keeps samples from previous tasks [Rebuffi et al., 2017, Nguyen et al., 2017]. The second approach consists of applying *regularization*, either by constraining weight updates in order to maintain knowledge from previous tasks [Kirkpatrick et al., 2017, Zenke et al., 2017, Maltoni and Lomonaco, 2018], or by keeping an old model in memory and distilling knowledge [Hinton et al., 2015] into it later to remember [Li and Hoiem, 2018, Schwarz et al., 2018]. The third category of strategies, *dy-*

namic network architectures, maintains past knowledge thanks to architectural modifications while learning [Rusu et al., 2016a, Fernando et al., 2017, Li and Hoiem, 2018, Fernando et al., 2017]. The fourth method is *generative replay* [Shin et al., 2017, Lesort et al., 2018, Wu et al., 2018, Lesort et al., 2018], where a generative model is used as a memory to produce samples from previous tasks.

In the context of continual reinforcement learning, several approaches have been proposed, such as the use of *Progressive Nets* in [Rusu et al., 2016b], *EWC* [Kirkpatrick et al., 2017], *Progress And Compress* (P&C) [Schwarz et al., 2018], or *CRL-Unsup* [Lomonaco et al., 2019]. However they either need a task indicator at test time to choose which policies to run or, they have some hyper-parameter difficult to tune during a continual learning training, such as the importance of the Fisher information matrix in EWC. Our method does not add any new hyper-parameter to tune during the sequence of tasks and does not need a task label at test time.

Reinforcement learning in Robotics: Applying RL to real-life scenarios such as robotics is a major challenge that has been studied widely.

One of the major problems in this setting is that sampling data and a fortiori learning is costly. Therefore sample efficiency and stability in learning are highly valuable. One common approach to reduce training cost, is training policies in simulation and then deploying them in real-life hoping that they will successfully transfer, considering the gap in complexity between simulation and the real world. Such approaches are termed *Sim2Real* [Golemo, 2018], and have been successfully applied [Christiano et al., 2016, Matas et al., 2018] in many scenarios. One of these approaches is Domain Randomization [Tobin et al., 2017], which we use in this paper. This technique trains policies in numerous simulations that are randomly different from each other (different background, colors, etc.). Using this technique, the transfer to real life is easier.

Another method we also exploit is to first learn a state representation [Lesort et al., 2018] to compress the observation into a low dimensional embedding and secondly learn the policy on top of this representation. This method helps to improve sample efficiency and stability of RL algorithms [Raffin et al., 2019] and thus can make them directly applicable in real life.

Others have tried to train a policy directly on real robots, facing the hurdle of the lack of sample efficiency of RL algorithms. SAC-X [Riedmiller et al., 2018] is one example that takes advantage of multi-task learning to improve efficiency, by simultaneously learning the policy and a set of auxiliary tasks to explore its observation space - in search for sparse rewards of the externally defined target task.

In the literature, most approaches focus on the single-task or simultaneous multi-task scenario. In this paper, we attempt to train a policy on several tasks sequentially and deploy it in real life by combining policy distillation, training in simulation and state representation learning.

3 Methods

In this section we present our approach towards continual reinforcement learning for a sequence of vision based tasks. We assume that observations visually allow to recognize the current task from other tasks. We first explain how we learn a single task by combining SRL and RL, then how each task is incorporated in the continual learning pipeline. Finally, we present how we evaluate the full pipeline.

3.1 Learning one task

Each task i is solved by first learning a state representation encoder E_i in order to compress input images into a representation of the important underlying factor of variation. This step allows to reduce the input space for the reinforcement learning algorithm and makes it learn more efficiently [Raffin et al., 2019]. To train this encoder, as shown in Fig. 2 (left), we sample data from the environment Env_i with a random policy. We call this dataset $D_{R,i}$. $D_{R,i}$ is then used to train the SRL model composed of an *inverse model* and an *auto-encoder*. The inverse model is trained to predict the action a_t that led to transition from state s_t to s_{t+1}, both extracted from respective observations o_t and o_{t+1} by the auto-encoder using E_i. The auto-encoder is additionally trained to reconstruct the observations from the encoded states. The architecture is motivated by the results from [Raffin et al., 2019], and illustrated in the Appendix F.

Once the SRL model is trained, we use its encoder E_i to provide features as input of a policy π_i trained using RL. We also experimented to learn the policy directly in the raw pixel space but, as shown in [Raffin et al., 2019], it was less sample efficient.

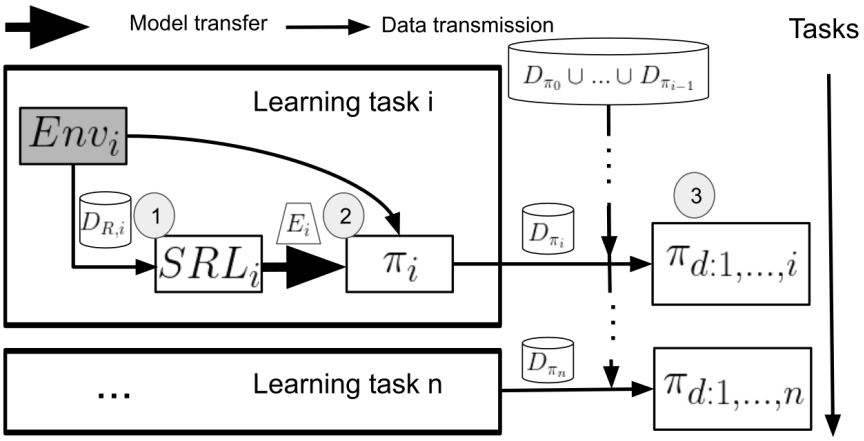

Figure 2: Overview of our full pipeline for Continual Reinforcement Learning. White cylinders are for datasets, gray squares for environments, and white squares for learning algorithms, whose name corresponds to the model trained. Each task i is learned sequentially and independently by first generating a dataset $D_{R,i}$ with a random policy to train a state representation with an encoder E_i with an SRL method (1), then we use E_i and the environment to learn a policy π_i in the state space (2). Once trained, π_i is used to create a distillation dataset D_{π_i} that acts as a memory of the learned behaviour. All policies are finally compressed into a single policy $\pi_{d:1,..,i}$ by merging the current dataset D_{π_i} with datasets from previous tasks $D_{\pi_1} \cup ... \cup D_{\pi_{i-1}}$ and using distillation (3).

Once π_i is learned, we use it to generate sequences of on-policy observations with associated actions, which will eventually be used for distillation (Fig. 2, right). We call this the distillation dataset D_{π_i}. We generate D_{π_i} the following way: we randomly sample a starting position and then let the agent generate a trajectory. At each step we save both the observation and associated action probabilities. We collect the shortest sequences maximizing the reward for an episode (see Section 3.3).We also experiment to generate D_{π_i} with a regular sampling and a random policy but annotated with π_i to compare results, as detailed in section 5.1.

From each task we only keep dataset D_{π_i}. As soon as we change task, $D_{R,i}$ and Env_i are not available anymore. D_{π_i} is split into a training set and a validation set.

3.2 Learning continually

To learn continually we adapt policy distillation [Rusu et al., 2015] to a continual learning setting. The distillation consists in training a student policy to imitate a teacher policy. In our case, a student model learn from a teacher policy the action probability associated to each observation. Each dataset D_{π_i} allows to distill the policy π_i (the **teacher** model) into a new network $\pi_{d:i}$ (the **student** model). In classic distillation, both data and models need to be saved, however saving just soft-annotated data is a lighter solution adapted to a continual setting.

With the aggregation of several distillation datasets D_{π_i}, we can distill several policies into the same network that can achieve all tasks (Fig.2, bottom right). By extension of the previous nomenclature, we denote $\pi_{d:1,..,n}$ a model where policies π_1 ... π_n have been distilled in. When distilling all policies into the student, we select our best models with early stopping, and test later in simulation and in real life settings.

Since we assume that observations visually allow to recognize the current task, $\pi_{d:1,..,n}$ is able to choose the right action for the current task without a task indicator.

The method, termed *DisCoRL* for *Distillation for Continual Reinforcement learning*, allows to learn continually several policies while minimizing forgetting. Regarding scalability, saving data from all past experiments may not look ideal if there is a high number of tasks. However, this solution is highly effective for remembering and letting the reinforcement learning algorithm be absolutely free to learn a new policy without regularization. It is worth mentioning that RL is the real bottleneck in the whole process: Dataset D_{π_i} contains approximately 10k samples per task, which allows to perform the distillation quickly, relative to how long and computationally expensive RL is (few minutes needed to learn $\pi_{d:i}$ while several hours are needed to learn π_i). Thus, in this context, it is better not to curb RL with regularization. Indeed, as explained in Section 5.3, we tried several regularization based approaches that were not successful.

Within the Continual Learning framework for robotics [Lesort et al., 2019], our setting falls within the category of Multi Task learning scenario (MT) with a NIC (New Instances and New Concepts) content update type for each task. Our approach can be classified into the rehearsal family of approaches where memory is saved as data points.

3.3 Evaluation

The first evaluation is the performance of the final policy on the simulated environment. This evaluation can then be compared with the performance of each teacher policy. For the second evaluation we test if the policy is robust to the reality gap and can be adapted into a real life scenario. The simulation is voluntary close the real life setting but the reality gap is notoriously problematic.

To get an insight on the evolution of the distilled model, we also save distillation datasets at different checkpoints while learning each tasks. By distilling and evaluating at several time steps, we assess catastrophic forgetting on previous task when finetuning on new a task (See Appendix D).

4 Experimental setup

We apply our approach to learn continually three 2D navigation tasks applicable in real life. The software related to our experimental setting is available online [1].

4.1 Robotic setup

The experiments consists of 2D navigation tasks using a 3 wheel omni-directional robot similar to the 2D mobile navigation in [Raffin et al., 2018]. The input image is a top-down view of the floor and the robot is identified by a black QR code. The room where the real-life robotic experiments are performed is lighted by surroundings windows and artificial illumination and is subject to illumination changes depending on the weather and time of the day. The robot uses 4 high level discrete actions (move left/right, move up/down in a cartesian plane relative to the robot) rather than motor commands.

We simulate the experiment to increase sampling and learning speed. The simulation is performed by artificially moving the robot picture inside the background image according to the chosen actions. We use domain randomization [Tobin et al., 2017] to improve the stability and facilitate transfer to the real world : during RL training, at each timestep, the color of the background is randomly changed.

4.2 Continual learning setup

Our continual learning scenario is composed of three similar environments, where the robot is rewarded according to the associated task (Fig. 1). In all environments, the robot is free to navigate for up to 250 steps, performing only discrete actions within the boundaries identified by a red line. Each task is associated to a visual target, which color depends on the task. This way, the controller can automatically infer which policy it needs to run and thus, does not need task labels at test time.

Task 1. The task of environment 1 is named Target Reaching (TR). The robot gets at each timestep t a positive reward $+1$ for reaching the target (red square), a negative reward -1 for bumping into the boundaries, and no reward otherwise.

Task 2. The task of environment 2 is named Target Circling (TC). The robot gets at each timestep t a reward R_t defined in Eq. 1 (where z_t is the 2D coordinate position with respect to the center of the circle) designed for agents to learn the task of circling around a central blue tag. This reward is highest when the agent is both on the circle (red (first) square in Eq. 1), and has been moving for the previous k steps (blue, second square). An additional penalty term of -1 is added to the reward function in case of bump with the boundaries (last, green square. A coefficient $\lambda = 10$ is introduced to balance the behaviour.

$$R_t = \lambda * \boxed{(1 - \lambda(\|z_t\| - r_{circle})^2)} * \boxed{\|z_t - z_{t-k}\|_2^2} + \lambda^2 * \boxed{R_{t,bump}} \tag{1}$$

[1] https://github.com/kalifou/robotics-rl-srl

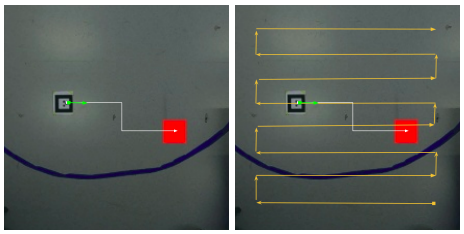

Figure 3:

Task 3. The task of environment 3 is named Target Escaping (TE). Robot A is being chased down by another robot B with an orange tag. Robot B is hard-coded to follow robot A, and robot A has to learn to escape using RL. Robot A gets at each timestep t a reward of $+1$ if it's far enough from robot B, otherwise, if it is in the range of robot B, it gets a reward of -1. Additionally, robot A gets a negative reward -1 for bumping into the boundaries.

All RL tasks are learned with PPO2 [Schulman et al., 2017a] and the same SRL model, as described in section 3.1. We select the model architecture as in [Raffin et al., 2018] for RL and SRL. All datasets size and characteristics are described in the Appendix. The input observations of all models are RGB images of size $224 * 224 * 3$.

5 Results

We first present our design choices for the distillation process: loss functions and data sampling strategies. We then use these choices to present our main result: the distillation of three tasks continually into a single policy that can achieve the three tasks both in simulation and real-life. We provide a supplementary video of this policy deployed in real-life on the robot showing the successful behaviors. We also present the different strategies we tried but that did not work in our setting.

5.1 Evaluation of distillation

Distillation strategies: Distillation is done with a loss function that minimizes the difference between the student model's output and the teacher model's output for the same input. As in the policy distilation paper [Rusu et al., 2015], we investigate variations of the loss function : Mean Squared Error loss ($\mathcal{L}_{MSE}(x, y) = \mathbb{E}[||x-y||_2^2]$), Kullback-Lieber divergence, and Kullback-Lieber divergence with temperature smoothing ($\mathcal{L}_{KL,\tau}(p|q) = \mathbb{E}[\text{softmax}(\frac{p}{\tau})ln(\frac{\text{softmax}(\frac{p}{\tau})}{\text{softmax}(q)})]$).

We run a performance comparison of the different losses by computing the mean normalized performance of a student policy trained to perform all three tasks (Tab.1). Using the Kullback-Lieber divergence loss function with temperature smoothing with $\tau = 0.01$ is best, and optimizing the temperature parameter yields a small performance boost. This result is coherent with [Rusu et al., 2015] where they reach the same conclusion.

Distillation loss	**Student performance ($\pm$ std)**
MSE	0.71 ($\pm$ 0.22)
KL ($\tau = 1$)	0.76 ($\pm$ 0.14)
KL ($\tau = 0.1$)	0.68 ($\pm$ 0.18)
KL ($\tau = 0.01$)	**0.77 ($\pm$ 0.13)**

Table 1: Mean normalized performance[3] of a student policy trained with distillation using 4 different loss functions. The student policy is trained to perform all three tasks. Kullback-Lieber divergence with $\tau = 0.01$ performs best.

Data sampling strategies: We evaluate the effect of two different sampling strategies to create D_{π_i} for policy distillation. Data sampling is a key component as the sampled dataset should be as small as possible but contain sufficient information for student model training. The strategies involved for data generation are:

- *On-policy generation (Fig.3, top)*: We start an episode from a random point, then at each timestep t, we collect an observation o_t and perform the action $a_{\pi_i,t}$ of the teacher policy. D_{π_i} is thus composed of tuples $(o_t, p(a_{\pi_i,t} \mid o_t))$, with $p(a_{\pi_i,t} \mid o_t)$ the action probability associated to the action $a_{\pi_i,t}$ taken by the teacher, i.e., a *soft label*, since we use the Kullback-Lieber divergence loss.

- *Off-policy generation from a grid walker (Fig.3, bottom)*: at each timestep t, we collect an observation o_t by performing an action $a_{grid,t}$ of a *grid walker* exhaustively exploring the space of the arena. However, for each o_t we save the probability of action $p(a_{\pi_i,t} \mid o_t)$ that would have been taken by a teacher policy. D_π is thus composed of tuples $(o_t, p(a_{\pi_i,t} \mid o_t))$. The goal of this strategy is to provide a more exhaustive sampling of the space of robot positions.

Performance of policies distilled using such strategies (see Fig. 4) show that *on-policy generation* (i.e., demonstrations) suffice to reproduce performance closed to those of teacher policies on every task individually, with reasonable stability. In particular cases, see Fig. 4 for task TC, this strategy even provides a small boost in performances in the student policy over the teacher policy.

However, using *off-policy data generation from a grid walker* for distillation results in either unstable or poorly performing policies, especially in tasks defined by a reward function requiring the agent to move actively (*TC* task, blue part of eq. 1) or anticipate the behaviour of another agent (TE task). In this case, the resulting policy reaches the performances of a lower-bound baseline obtained by distilling from trajectories of an untrained policy (see *Student on off-policy data with a random walker* in fig. 4), i.e. from a policy with random weights with input in the raw pixels' space.

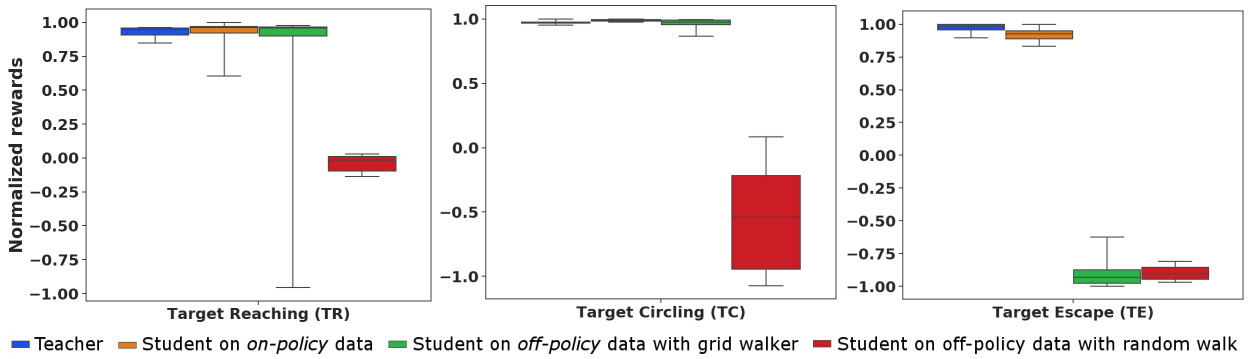

Figure 4: Efficiency (normalized rewards w.r.t the best teacher performance) of policies distilled on 8 seeds using various data generation strategies for each task separately. Each evaluated policy is distilled on 15k tuples of sampled observations and action probabilities, for 4 epochs (see criteria of stopping in section 3.2 and Appendix B).

5.2 Main result

We present our final results in Fig. 5. We used *on policy* data generation and training using KL divergence loss with $\tau = 0.01$. We show box plots over 10 episodes of reward performances for teacher policies in each task, and for the distillation of the same three teachers into a single student using DisCoRL. Each policy is evaluated in simulation and also in real-life on the robot. As a reference, we also show the performance of a random agent in each task. Our approach is effective in a continual reinforcement learning setting: the performance of teachers and student are similar.

More precisely, there are two main challenges to overcome in our setting: learning a behaviour via distillation by using only a limited number of examples, and the reality gap which can notoriously [Tobin et al., 2017] introduce variations that may lead the policy to fail. Fig. 5 demonstrates the efficiency of our approach at overcoming both of these issues: only a small fraction of performance is lost from teacher to student, and from simulation to reality. We can see that the single student distilled policy achieves close to maximum rewards in all tasks, in real-life.

5.3 Negative results

While distillation is effective for policy transfer, we also tested other alternatives worth mentioning.

Elastic Weight Consolidation (EWC) [Kirkpatrick et al., 2017] was implemented as a continual learning baseline to compare with the distillation method. EWC has the appealing advantage of not re-using any data from previous tasks. However, in all cases we found the method unsuccessful.

Tuning the λ parameter that controls the trade-off between weight protection and learning the new task showed that either λ is too low and catastrophic forgetting happens, or λ is too high and nothing new is learned (i.e., the full network is frozen). A λ value providing a proper balance in between both effects could not be found for such sequential tasks to be learned.

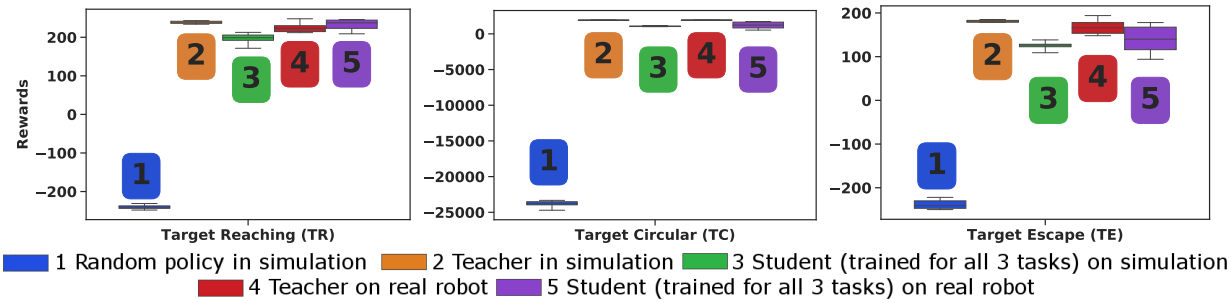

Figure 5: Main result: distillation in a continual learning setting of three teacher policies into a single student policy. The resulting policy is able to perform all three tasks both in simulation and in the real world, while minimizing forgetting.

Progress and Compress (P&C) [Schwarz et al., 2018] was tested but as EWC, we had problems with the importance factor λ and we where not able to learn three policies into a single model with this method.

Adding task labels for distillation. Even if all tasks contain a visually differentiating identifier, they remain visually similar. In cases, we found that a distilled policy trained to perform well on several tasks can mix up tasks and thus not perform adequately. Hence, either adding tasks labels directly, or adding a module in the network that predicts the task label could be a way to improve the efficiency of distillation. However, none of the approaches were successful in practice, yielding the same results with or without task labels.

Gumbel-Softmax action sampling for the student [Jang et al., 2016]. This trick allows to sample from a categorical distribution using a softmax output layer. It has proven to be useful for action sampling in policy learning [Schulman et al., 2017b]. However, in our case we saw no improvement over a simple argmax strategy for action sampling when we used it on the student policy at test time.

6 Discussion and Future Work

Continual learning is a complex field: every setting is different and expectations may vary from one algorithm to another. For example it is not easy to compare results with and without task indicator. Task labels always add information to learn or test, and thus, they often improve results. However in a realistic setting they may be lacking.

Otherwise, the scalability vs stability trade-off is a difficult question. Learning online in a single model or a dual architecture scales well to a high number of tasks. However, this solution is often unstable, in particular because if a task fails, there is high risk of forgetting everything that has been learned previously. For example, in generative replay, the generator is used as a memory. However, if at some moment it diverges while learning, all data from the past is destroyed. The approach we propose uses soft-labelled samples as a memory, similarly to rehearsal methods, which will grow the memory continually. However, this brings no risk of forgetting or destroying past knowledge.

Nevertheless, even if we believe this work proposes a stable and scalable framework for continual reinforcement learning, several possibilities for improvement exists. Our road map includes having not only a policy learned in a continual way, but also the SRL model associated. We would need to update the SRL model as new tasks are presented sequentially. One possible approach would be to use Continual SRL methods like S-TRIGGER [Caselles-Dupré et al., 2019] or VASE [Achille et al., 2018]. Moreover, we would like to optimize more the memory needed to save samples by reducing their number and their size.

Our empirically best distillation strategy currently consists on performing early stopping on the joint policy $\pi_{d:1,..,n}$. In future work, we plan to perform automatic model selection for the final policy.

Finally, training policies on real robot experiences without the use of simulation would be desirable. However, at the moment, this is more a RL challenge than a CL challenge. One promising approach would be to use model-based RL while learning the SRL model to improve sample efficiency. Though, nowadays approaches still do not offer solutions working in a reasonable amount of time.

7 Conclusion

In this paper we presented DisCoRL, an approach for continual reinforcement learning. The method consists of summarizing sequentially learned policies into a dataset to distill them into a student model. It allows to learn sequential tasks in a stable pipeline without forgetting. Some loss in performance may occur while transferring knowledge from teacher to student, or while transferring a policy from simulation to real life. Nevertheless, our experiments show promising results in simulated environments and real life settings. Future work will evaluate it in more complex tasks.

8 Acknowledgement

This work is supported by the EU H2020 DREAM project (Grant agreement No 640891).

References

[Achille et al., 2018] Achille, A., Eccles, T., Matthey, L., Burgess, C., Watters, N., Lerchner, A., and Higgins, I. (2018). Life-long disentangled representation learning with cross-domain latent homologies. In *Advances in Neural Information Processing Systems*, pages 9873–9883.

[Andrychowicz et al., 2017] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058.

[Caruana, 1997] Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1):41–75.

[Caselles-Dupré et al., 2019] Caselles-Dupré, H., Garcia-Ortiz, M., and Filliat, D. (2019). S-TRIGGER: Continual State Representation Learning via Self-Triggered Generative Replay. *arXiv preprint arXiv:1902.09434*.

[Christiano et al., 2016] Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., and Zaremba, W. (2016). Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*.

[Colas et al., 2018] Colas, C., Sigaud, O., and Oudeyer, P.-Y. (2018). CURIOUS: Intrinsically Motivated Multi-Task, Multi-Goal Reinforcement Learning. *arXiv preprint arXiv:1810.06284*.

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

[Furlanello et al., 2018] Furlanello, T., Lipton, Z. C., Tschannen, M., Itti, L., and Anandkumar, A. (2018). Born again neural networks. *arXiv preprint arXiv:1805.04770*.

[Golemo, 2018] Golemo, F. (2018). *How to Train Your Robot - New Environments for Robotic Training and New Methods for Transferring Policies from the Simulator to the Real Robot*. Theses, Université de Bordeaux.

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

[Jang et al., 2016] Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

[Kirkpatrick et al., 2017] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

[Lesort et al., 2018] Lesort, T., Caselles-Dupré, H., Garcia- Ortiz, M., Stoian, A., and Filliat, D. (2018). Generative Models from the perspective of Continual Learning. *arXiv e-prints*, page arXiv:1812.09111.

[Lesort et al., 2018] Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. (2018). State representation learning for control: An overview. *Neural Networks*.

[Lesort et al., 2018] Lesort, T., Gepperth, A., Stoian, A., and Filliat, D. (2018). Marginal Replay vs Conditional Replay for Continual Learning. *ArXiv e-prints*.

[Lesort et al., 2019] Lesort, T., Lomonaco, V., Stoian, A., Maltoni, D., Filliat, D., and Díaz-Rodríguez, N. (2019). Continual Learning for Robotics. *arXiv e-prints*, page arXiv:1907.00182.

[Li and Hoiem, 2018] Li, Z. and Hoiem, D. (2018). Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947.

[Lomonaco et al., 2019] Lomonaco, V., Desai, K., Culurciello, E., and Maltoni, D. (2019). Continual reinforcement learning in 3d non-stationary environments. *CoRR*, abs/1905.10112.

[Maltoni and Lomonaco, 2018] Maltoni, D. and Lomonaco, V. (2018). Continuous learning in single-incremental-task scenarios. *arXiv preprint arXiv:1806.08568*.

[Matas et al., 2018] Matas, J., James, S., and Davison, A. J. (2018). Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*.

[Nguyen et al., 2017] Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2017). Variational continual learning. *arXiv preprint arXiv:1710.10628*.

[Raffin et al., 2019] Raffin, A., Hill, A., Traoré, K. R., Lesort, T., Díaz-Rodríguez, N., and Filliat, D. (2019). Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics. *Workshop on "Structure and Priors in Reinforcement Learning" (SPiRL) at ICLR*.

[Raffin et al., 2018] Raffin, A., Hill, A., Traoré, R., Lesort, T., Díaz-Rodríguez, N., and Filliat, D. (2018). S-RL toolbox: Environments, datasets and evaluation metrics for state representation learning. *arXiv preprint arXiv:1809.09369*.

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

[Riedmiller et al., 2018] Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*.

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

[Rusu et al., 2016a] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016a). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

[Rusu et al., 2016b] Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2016b). Sim-to-real robot learning from pixels with progressive nets. *CoRR*, abs/1610.04286.

[Schulman et al., 2017a] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017a). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

[Schulman et al., 2017b] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Schwarz et al., 2018] Schwarz, J., Luketina, J., Czarnecki, W. M., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018). Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*.

[Shin et al., 2017] Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999.

[Teh et al., 2017] Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506.

[Tobin et al., 2017] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE.

[Wu et al., 2018] Wu, C., Herranz, L., Liu, X., Wang, Y., van de Weijer, J., and Raducanu, B. (2018). Memory replay gans: learning to generate images from new categories without forgetting. *arXiv preprint arXiv:1809.02058*.

[Zenke et al., 2017] Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3987–3995. PMLR.

A Dataset generation

The notation summarizing our data generation and distillation processes is:

- Env_i: Environment for task i.
- $\mathcal{T}_i$: The set of all encountered environments when encountering task i.
- $D_{R,i}$: Dataset generated by a random policy on environment i.
- E_i: Encoder i from SRL step from task i.
- π_i: Policy i.
- D_{π_i}: Dataset generated by and on-policy π_i.
- $\pi_{d:1,\dots,i}$: Policy distilled on environment 1 to i.

While generating on-policy datasets $D_{\pi 1}$ (see Section 5.1.2) for task 1 (TR), we allow the robot to perform a limited number of contacts with the target to reach ($N_{contacts} = 10$) in order to mainly preserve the frames associated with the correct reaching behaviour. There are no such additional constraints when recording for task 2 (TC) or 3 (TE), the limit is the standard episode length, i.e. 250 time-steps.

B Evaluation of each task separately

Before moving to a continual setup, we should test if it is possible to solve each task separately using RL.

One of the challenges in CL is not the ability to distill knowledge, but the ability to know when a policy has been distilled, i.e. properly learned by the student. Due to the hypothesis of real continual learning settings where access to previous environments is not possible, one of the challenges is finding a proxy task that can help indicate when early stopping of the policy distillation can be applied. That proxy task or signal should be different from the reward achieved in previous environments, which is no longer available. In our experiments, we found empirically that a small number of epochs, i.e $N = 4$, guarantee policy learning.

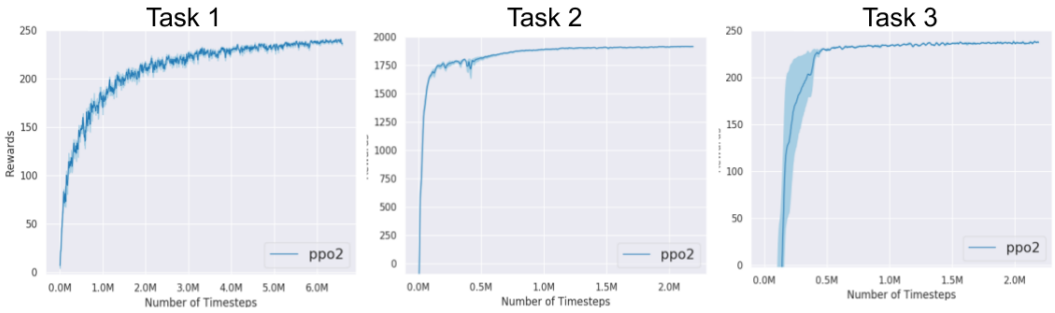

Figure 6: Mean and standard error of rewards during RL learning of each task separately. Each task is learned using the same type of SRL model (SRL Combination), trained on each environment. All three tasks are mastered within roughly 2M timesteps.

C Memory needs

The memory occupation by the different models that form part of our pipeline are the following:

- **Dataset**: $D_{\pi_{1,2,3}}$: 554.6 MB
- **SRL model + RL policy** (PPO2): 4.8 MB + 0.143 MB
- **RL model from raw pixels** : 0.143 MB
- **Distilled student policy** (replay model): 1.1 MB

D Evaluation of sequential policy learning via fine-tuning

Results on policy fine-tuning (see Fig. 7) show that catastrophic forgetting occurs on a previously learned task $k < i$ (TR) when the fine-tuned policy reaches near convergence on a new task i (TC). Moreover, it appears that the fine-tuning could be interrupted such that the resulting policy would have reasonable performances on the set of all encountered environments so far $\mathcal{T}_i = \{TC, TR\}$.

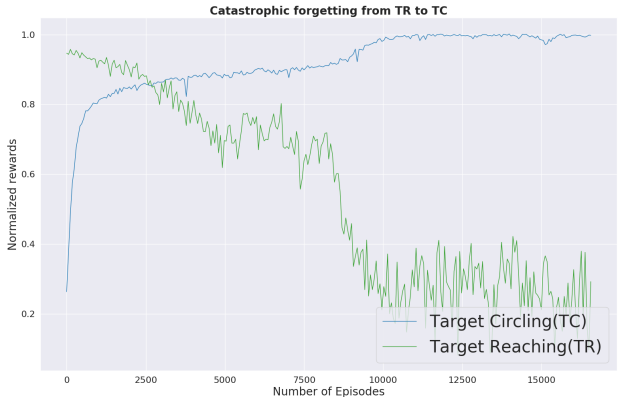

Figure 7: Demonstration of catastrophic forgetting phenomenon occurring while fine-tuning a network on a new task. The blue curve shows the progression of the mean reward of the policy currently being learned (task i: TC). The green curve represents its mean reward when evaluated on the task $i-1$ previously learned (first task: TR). Both evaluations were made with 5 random seeds.

E Evaluating distillation while learning the knowledge to distill from

We performed a more explicit evaluation of distillation in the task 2 (Target Circling (TC)). While we train a policy using RL, we save the policy every 200 episodes (50K timesteps), and distill it into a new student policy which we test.

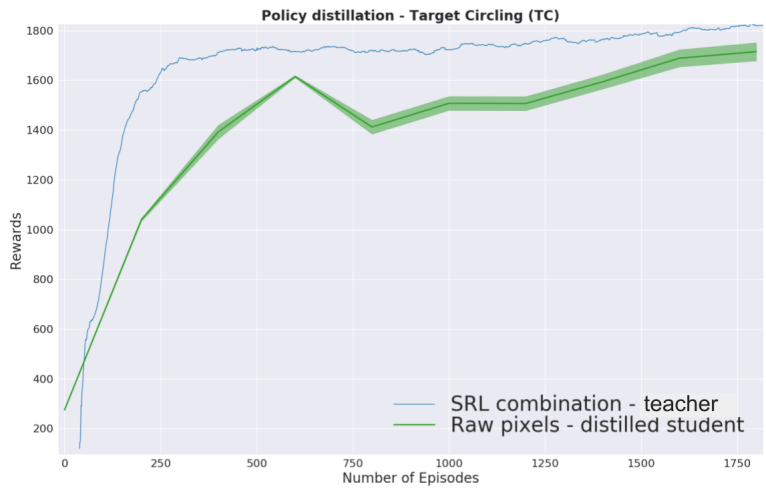

Figure 8: Demonstration of the effectiveness of distillation. Blue: RL training curve of an SRL based Policy (SRL Combination) on the target circling (TC) task. Green: Mean and standard deviation performance on 8 seeds of distilled student policy. The teacher policy in blue is distilled into a student policy every 200 episodes (1 episode = 250 timesteps).

This is illustrated in Fig. 8. Both curves are very close, which indicates that policy distillation enables to reproduce the skills of a teacher policy regardless of the teacher's state of convergence on the evaluated task. Moreover, distillation is able to transfer knowledge from teacher policy into a student using a small number of observations, i.e only 15k samples (w.r.t. the volume of samples required to learn the teacher policy, see Fig. 6).

F State Representation Learning (SRL) model

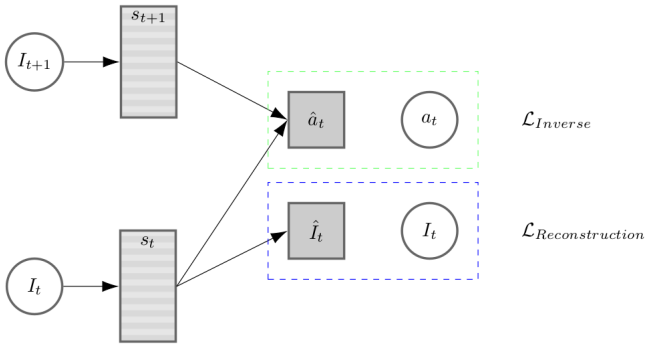

Figure 9: *SRL Combination* model: combines the prediction of an image I's reconstruction loss and an inverse dynamics model loss in a state representation s. Arrows represent inference, dashed frames represent losses computations, rectangles are state representations, circles are real observed data, and squares are model predictions; t represents the timestep.

G Distillation model architecture

Architecture available at : https://github.com/araffin/srl-zoo/blob/438a05ab625a2c5ada573b47f73469d92de82132/models/models.py#L179-L214