Reward Relabelling for combined Reinforcement and Imitation Learning on sparse-reward tasks

Extended Abstract

Jesus Bujalance
MINES ParisTech, PSL University, Center for robotics
Paris, France
jesus.bujalance_martin@mines-paristech.fr

Fabien Moutarde
MINES ParisTech, PSL University, Center for robotics
Paris, France

ABSTRACT

In the search for more sample-efficient reinforcement-learning (RL) algorithms, a promising direction is to leverage as much external off-policy data as possible. For instance, expert demonstrations. In the past, multiple ideas have been proposed to make good use of the demonstrations added to the replay buffer, such as pre-training on demonstrations only or minimizing additional cost functions. We present a new method, able to leverage both demonstrations and episodes collected online in any sparse-reward environment with any off-policy algorithm. Our method is based on a reward bonus given to demonstrations and successful episodes (via relabeling), encouraging expert imitation and self-imitation. Our experiments focus on several robotic-manipulation tasks across two different simulation environments. We show that our method based on reward relabeling improves the performance of the base algorithm (SAC) on these tasks.

KEYWORDS

Reinforcement Learning; Imitation Learning; Robotic Manipulation

ACM Reference Format:
Jesus Bujalance and Fabien Moutarde. 2023. Reward Relabelling for combined Reinforcement and Imitation Learning on sparse-reward tasks: Extended Abstract. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 3 pages.

1 INTRODUCTION

In this work we focus on off-policy RL, and present a way to leverage offline data in the form of expert demonstrations. Our method is based on the observation that, in hindsight, a successful episode of collected experience is in fact a demonstration, so it should receive the same treatment. In particular, we propose to add a reward bonus to transitions coming from both demonstrations and successful episodes. Our approach provides a simple way of tying positive rewards and desired behaviour. We instantiate our approach with Soft Actor-Critic (SAC) [4], and compare it to three other algorithms. Just like ours, these methods are task-agnostic and generic, in the sense that they can be applied to any continuous-action off-policy algorithm with some minor modifications.

SAC-fD [10] (originally DDPGfD based on DDPG) introduces transitions from demonstrations into the replay buffer, and uses a mix of 1-step and n-step return losses.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

SAC-BC [6] (has no name and originally based on DDPG) also introduces transitions from demonstrations into a separate additional replay buffer, and uses an auxiliary behaviour-cloning loss.

SAC-SAIL [3] (originally SAIL based on Q-learning algorithms) uses an additional self-imitation loss based on advantage learning. We also introduce transitions from demonstrations into the replay buffer for a fair comparison with the other methods.

2 METHOD

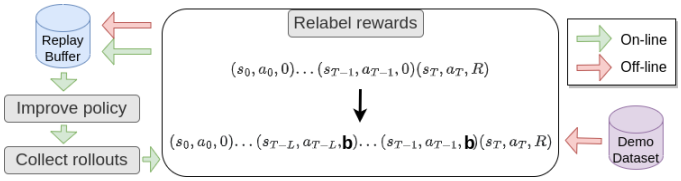

Figure 1: Reward Relabeling procedure.

We propose SAC-R^2, acronym for SAC with Reward Relabeling, a straight-forward method that could be implemented to any other off-policy RL algorithm with sparse rewards. Let R be the sparse reward from the environment, b the reward bonus of our method, and L the amount of transitions that will receive the bonus. First, we add demonstration data to the buffer: the last transition of each expert trajectory is given the sparse reward R, and the last non-final L transitions are given a reward equal to b. Then, as the agent explores the environment during training, every new successful episode is relabeled the same way: the last L transitions leading to the sparse reward are assigned a reward equal to b. Intuitively, our method helps propagate the signal of the sparse reward by explicitly turning zero rewards into positive rewards, rather than entirely relying on bootstrapping rare future rewards.

Reward bonus to demonstrations. The first part of our algorithm is most similar to SQIL [9], a pure IL algorithm where the replay buffer is initially filled with demonstrations with all rewards equal to $r = 1$, and new experiences collected by the agent are added with reward $r = 0$. Intuitively, it gives the agent an incentive to imitate the expert. One important difference is that our method learns from both the reward bonuses and the reward from the environment. Also, SQIL gives an incentive to avoid states that were not in the demonstration data, which could potentially be harmful if those states led to successful behaviour.

Reward bonus to successful episodes. The relabeling part of our algorithm tries to mitigate this issue and is most similar to self-imitation methods such as SIL [8]. In our method, the self-imitation is achieved by effectively treating successful episodes as if they were demonstrations. In order to give a reward bonus to successful episodes, we need to wait for the episodes to end first. We follow the idea presented in HER [1] to relabel past experiences and modify the transitions' rewards.

2.1 Decay and Hyper-parameters

One issue with our method so far, is that it might change the optimal policy of the problem. Intuitively, the bonus encourages a policy that requires at least L steps to reach the goal, which might not take the shortest path available. We want our method to converge to the optimal policy of the sparse reward problem.

As stated in [7], in order to still converge to the optimal policy of the original problem one can only add a reward term such as, for a given transition between two states, the term is expressible as in the difference in value of an arbitrary potential function applied to those states. Our reward bonus cannot be expressed as such since it depends on the time-step of the states. To ensure that our method converges to the optimal policy, we decide to use a decay that eventually causes the bonus to completely disappear. By doing so, our method operates in two steps: An initial RL training with reward bonuses that might not converge to the optimal policy, followed by a second RL training without any reward bonuses which should converge to the optimal policy. For our experiments, we use a simple linear decay that quickly turns the bonus to 0 when the success rate stops improving.

Our method has two hyper-parameters to tune, b and L. L represents how far the sparse reward should be propagated into the past, and is similar to other parameters in RL such as the n of the n-step look-ahead in Q-learning.

How about b? We propose to adjust the value of b during training based on the agent's recent success. This also acts as a more natural decay during the entire training process. Let's place ourselves in the episodic undiscounted scenario. Let ζ be the fraction of successful episodes over the last 100 episodes, and r the total reward obtained over a successful episode τ. We propose $r(\tau) = R + L \cdot b \cdot (1 - \zeta)$.

Intuitively, the more the agent struggles, the larger the bonus to help guide it. We can prove that this reward-shaping bonus doesn't affect the converged performance of the algorithm as long as R prevails over the bonuses:

$$\begin{aligned} &\text{undiscounted} \quad b \cdot L \leq R \\ &\text{discounted} \quad b\left(\sum_{i=0}^{L-1} \gamma^i\right) \leq R \cdot \gamma^L \end{aligned} \tag{1}$$

3 EXPERIMENTS

We test our method on two different simulation environments: RLBench [5] and Meta-World [11]. The reward is fully sparse, and is equal to +100 if the robot solves the task and 0 otherwise. The state includes the robot proprioceptive state (joint angles, joint speeds, gripper pose) and the task-related information (3D coordinates of each target object). We evaluate our method on four RLBench tasks for a 6-degrees-of-freedom robot manipulator: reaching a ball, pushing a button, flipping a switch, and sliding a block to a target square, and four Meta-World tasks for a 7-degrees-of-freedom robot manipulator: reaching a ball, pressing a button, closing a drawer, and sliding a block to a target location. The RLBench demonstrations are generated with motion planning, while the Meta-World demonstrations come from a pre-trained RL agent.

	SAC-Demo	SAC-BC	SAC-SAIL	SAC-fD	SAC-R2
RLBench					
Reach	1.0 / 95k	1.0 / 95k	1.0 / 90k	**1.0 / 70k**	**1.0 / 70k**
Push	.60 / 170k	.70 / 170k	**.95 / 140k**	**.95 / 140k**	**.95 / 140k**
Flip	.40 / 160k	.80 / 160k	.90 / 160k	**.95 / 120k**	.90 / 120k
Slide	.75 / 300k	.75 / 300k	**.80 / 300k**	**.80 / 300k**	.75 / 300k
Meta-World					
Sweep	0.0 / 50k	**.40 / 50k**	.30 / 50k	.30 / 50k	.30 / 50k
Close	1.0 / 15k	**1.0 / 10k**	**1.0 / 10k**	**1.0 / 10k**	1.0 / 15k
Press	.80 / 25k	**1.0 / 5k**	.85 / 25k	.90 / 15k	.85 / 15k
Reach	.90 / 25k	**.95 / 15k**	.90 / 25k	**.95 / 15k**	.90 / 25k

Table 1: Experimental results (averaged on three random seeds) for all 8 tasks. The values correspond to the final success rate and the amount of iterations needed to converge.

We compare our method SAC-R^2 to a simple baseline SAC-Demo, which we define as SAC with demonstrations in the buffer, as well as SAC-fD, SAC-BC and SAC-SAIL. The results are reported in Table 1 and show that our method has comparable results to the other baselines. Overall, the best methods are SAC-fD in the RLBench tasks, and SAC-BC in the Meta-World tasks.

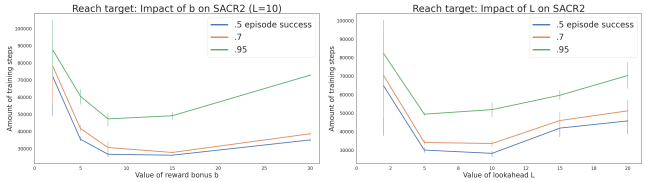

Figure 2: For a given hyper-parameter value, we plot the number of training steps required for the average episode success to reach a certain threshold.

Hyper-parameters. Figure 2 shows the impact of different values of L and b. For this RLBench task, there seems to exist an optimal value for both parameters. In the case of b, the optimal value is attained for the largest possible value that respects the condition 1.

4 DISCUSSION

We propose Reward Relabeling (R^2), a generic method that can be applied to any off-policy RL algorithm in any sparse-reward environment. It encourages two behaviours: imitate the expert demonstrations (if available), and imitate the past successful trajectories. From our experiments, our method SAC-R^2 had comparable results to previous works. In the full version of this paper [2], we propose an additional algorithm that combines elements from SAC-R^2, SAC-fD, SAC-BC and SAC-SAIL to outperform all baselines.

REFERENCES

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf

[2] Jesus Bujalance and Fabien Moutarde. 2023. STIR^2: Reward Relabelling for combined Reinforcement and Imitation Learning on sparse-reward tasks. https://doi.org/10.48550/ARXIV.2201.03834

[3] Johan Ferret, Olivier Pietquin, and Matthieu Geist. 2020. Self-Imitation Advantage Learning. *CoRR* abs/2012.11989 (2020). https://arxiv.org/abs/2012.11989

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. https://openreview.net/forum?id=HJjvxl-Cb

[5] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. 2020. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters* 5, 2 (2020), 3019–3026.

[6] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. 2018. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 6292–6299.

[7] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, Vol. 99. 278–287.

[8] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. 2018. Self-Imitation Learning. In *ICML*.

[9] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. 2020. {SQIL}: Imitation Learning via Reinforcement Learning with Sparse Rewards. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1xKd24twB

[10] Matej Vecerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin A. Riedmiller. 2017. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *CoRR* abs/1707.08817 (2017). http://arxiv.org/abs/1707.08817

[11] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. 2019. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *Conference on Robot Learning (CoRL)*. arXiv:1910.10897 [cs.LG] https://arxiv.org/abs/1910.10897