COMPETITIVE EXPERIENCE REPLAY

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning has achieved remarkable successes in solving challenging reinforcement learning (RL) problems. However, it still often suffers from the need to engineer a reward function that not only reflects the task but is also carefully shaped. This limits the applicability of RL in the real world. It is therefore of great practical importance to develop algorithms which can learn from unshaped, sparse reward signals, e.g. a binary signal indicating successful task completion. We propose a novel method called competitive experience replay, which efficiently supplements a sparse reward by placing learning in the context of an exploration competition between a pair of agents. Our method complements the recently proposed hindsight experience replay (HER) by inducing an automatic exploratory curriculum. We evaluate our approach on the tasks of reaching various goal locations in an ant maze and manipulating objects with a robotic arm. Each task provides only binary rewards indicating whether or not the goal is completed. Our method asymmetrically augments these sparse rewards for a pair of agents each learning the same task, creating a competitive game designed to drive exploration. Extensive experiments demonstrate that this method leads to faster converge and improved task performance.

1 INTRODUCTION

Reinforcement learning (RL) combined with neural networks has recently led to a wide range of successes in learning policies for sequential decision-making problems. This includes simulated environments, such as playing video games (Mnih et al., 2013; 2016), defeating the best human player at the game of Go (Silver et al., 2016; 2017b;a), high dimensional continuous control (Schulman et al., 2016; 2017; 2015), as well as robotic control (Chebotar et al., 2017; Levine et al., 2016; Andrychowicz et al., 2018).

Despite these successes, a key challenge, especially for robotics, is the need to engineer a reward function that not only reflects the task at hand but is also carefully shaped to guide the policy optimization (see e.g. Popov et al. 2017, Ng et al. 2006). For example, Popov et al. (2017) use a cost function consisting of five terms which need to be carefully weighted in order to train a policy for stacking a brick on top of another one. The necessity of cost engineering limits the applicability of RL in the real world because it requires both RL expertise and domain-specific knowledge. Moreover, it is not applicable in situations where we do not know what admissible behaviour may look like. Therefore, it is of great practical importance to develop algorithms which can learn from unshaped reward signals, e.g. a binary signal indicating successful task completion. In environments where reward is sparse, only a small fraction of the agents' experiences are directly used to update the policy, leading to substantial inefficiencies during training.

In the domain of goal-directed RL, the recently proposed hindsight experience replay (HER) (Andrychowicz et al., 2017) addresses the challenge of learning from sparse rewards by re-labelling visited states as goal states during training. However, this technique continues to suffer from sample inefficiency, ostensibly due to difficulties related to exploration. In this work, we address these limitations by introducing a method called *Competitive Experience Replay (CER)*. This technique attempts to emphasize exploration by introducing a

competition between two agents attempting to learn the same task. Intuitively, agent A (the agent ultimately used for evaluation) receives a penalty for visiting states that the competitor agent (B) also visits; and B is rewarded for visiting states found by A. Our approach maintains the reward from the original task such that exploration is biased towards the behaviors best suited to accomplishing the task goals. Whereas HER re-labels samples based on an agent's individual rollout, our method re-labels samples based on intra-agent behavior; as such, the two methods are orthogonal and easily combined during training. We show that this competition between agents can automatically generate a curriculum of exploration and shape otherwise sparse reward. We jointly train both agents' policies by adopting methods from multi-agent RL. In addition, we propose two versions of CER, *independent CER*, and *interact CER*, which differ in the state initialization of agent B: whether it is sampled from the initial state distribution or sampled from off-policy samples of agent A, respectively.

We evaluate our method both with and without HER on a variety of reinforcement learning tasks, including navigating an ant agent to reach a goal position and manipulating objects with a robotic arm. For each such task the default reward is sparse, corresponding to a binary indicator of goal completion. Ablation studies show that our method is important for achieving a high success rate and often demonstrates faster convergence.

2 BACKGROUND

Here, we provide an introduction to the relevant concepts for reinforcement learning with sparse reward (Section 2.1), Deep Deterministic Policy Gradient, the backbone algorithm we build off of, (Section 2.2), and Hindsight Experience Replay (Section 2.3).

2.1 SPARSE REWARD REINFORCEMENT LEARNING

Reinforcement learning considers the problem of finding an optimal policy for an agent that interacts with an uncertain environment and collects reward per action. The goal of the agent is to maximize its cumulative reward. Formally, this problem can be viewed as a Markov decision process over the environment states $s \in S$ and agent actions $a \in A$, with the (unknown) environment dynamics defined by the transition probability $T(s'|s, a)$ and reward function $r(s_t, a_t)$, which yields a reward immediately following the action a_t performed in state s_t.

We consider goal-conditioned reinforcement learning from sparse rewards. This constitutes a modification to the reward function such that it depends on a goal $g \in G$, such that $r_g : S \times A \times G \to R$. Every episode starts with sampling a state-goal pair from some distribution $p(s_0, g)$. Unlike the state, the goal stays fixed for the whole episode. At every time step, an action is chosen according to some policy π, which is expressed as a function of the state and the goal, $\pi : S \times G \to A$. For generality, our only restriction on G is that it is a subset of S. In other words, the goal describes a target state and the task of the agent is to reach that state. Therefore, we apply the following sparse reward function:

$$r_t = r_g(s_t, a_t, g) = \begin{cases} 0, \text{ if } |s_t - g| < \delta \\ -1, \text{ otherwise} \end{cases} \tag{1}$$

where g is a goal, $|s_t - g|$ is a distance measure, and δ is a predefined threshold that controls when the goal is considered completed.

Following policy gradient methods, we model the policy as a conditional probability distribution over states, $\pi_\theta(a|[s, g])$, where $[s, g]$ denotes concatenation of state s and goal g, and θ are the learnable parameters. Our objective is to optimize θ with respect to the expected cumulative reward, given by:

$$J(\theta) = \mathbb{E}_{s \sim \rho_\pi, a \sim \pi(a|s,g), g \sim G} \left[r_g(s, a, g) \right],$$

where $\rho_\pi(s) = \sum_{t=1}^{\infty} \gamma^{t-1} \Pr(s_t = s)$ is the normalized discounted state visitation distribution with discount factor $\gamma \in [0, 1)$. To simplify the notation, we denote $\mathbb{E}_{s\sim\rho_\pi, a\sim\pi(a|s,g), g\sim G}[\cdot]$ by simply $\mathbb{E}_\pi[\cdot]$ in the rest of paper. According to the policy gradient theorem (Sutton et al., 1998), the gradient of $J(\theta)$ can be written as

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi(a|s, g) Q^\pi(s, a, g) \right], \tag{2}$$

where $Q^\pi(s, a, g) = \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_g(s_t, a_t, g) | s_1 = s, a_1 = a \right]$, called the critic, denotes the expected return under policy π after taking an action a in state s, with goal g.

2.2 DDPG ALGORITHM

Here, we introduce Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015), a model-free RL algorithm for continuous action spaces that serves as our backbone algorithm. Our proposed modifications need not be restricted to DDPG; however, we leave experimentation with other continuous control algorithms to future work. In DDPG, we maintain a deterministic target policy $\mu(s, g)$ and a critic $Q(a, s, g)$, both implemented as deep neural networks (note: we modify the standard notation to accommodate goal-conditioned tasks). To train these networks, episodes are generated by sampling actions from the policy plus some noise, $a \sim \mu(s, g) + N(0, 1)$. The transition tuple associated with each action $(s_t, a_t, g_t, r_t, s_{t+1})$ is stored in the so-called replay buffer. During training, transition tuples are sampled from the buffer to perform mini-batch gradient descent on the loss L which encourages the approximated Q-function to satisfy the Bellman equation $L = \mathbb{E}(Q(a_t, s_t, g_t) - y_t)^2$, where $y_t = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}, g_t), g_t)$. Similarly, the actor can be updated by training with mini-batch gradient descent on the loss $J(\theta) = -\mathbb{E}_s Q(s, \mu(s, g), g)$ through the deterministic policy gradient algorithm (Silver et al., 2014),

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \mu(s, g) \nabla_a Q(s, a)|_{a=\mu(s,g)}]. \tag{3}$$

To make training more stable, the targets y_t are typically computed using a separate target network, whose weights are periodically updated to the current weights of the main network (Lillicrap et al., 2015; Mnih et al., 2013; 2015).

2.3 HINDSIGHT EXPERIENCE REPLAY

Despite numerous advances in the application of deep learning to RL challenges, learning in the presence of sparse rewards remains a major challenge. Intuitively, these algorithms depend on sufficient variability within the encountered rewards and, in many cases, random exploration is unlikely to uncover this variability if goals are difficult to reach. Recently, Andrychowicz et al. (2017) proposed Hindsight Experience Replay (HER) as a technique to address this challenge. The key insight of HER is that failed rollouts (where no task reward was obtained) can be treated as successful by assuming that a visited state was the actual goal. Basically, HER amounts to a relabelling strategy. For every episode the agent experiences, it gets stored in the replay buffer twice: once with the original goal pursued in the episode and once with the goal replaced with a future state achieved in the episode, as if the agent were instructed to reach this state from the beginning. Formally, HER randomly samples a mini-batch of episodes in buffer, for each episode $(\{s^i\}_{i=1}^T, \{g^i\}_{i=1}^T, \{a^i\}_{i=1}^T, \{r^i\}_{i=1}^T, \{s'^i\}_{i=1}^T)$, for each state s_t, where $1 \le t \le T-1$ in an episode, we randomly choose s_k where $t+1 \le k \le T$ and relabel transition $(s_t, a_t, g_t, r_t, s_{t+1})$ to $(s_t, a_t, s_k, r'_t, s_{t+1})$ and recalculate reward r'_t,

$$r'_t = r_g(s_t, s_k) = \begin{cases} 0, \text{ if } |s_t - s_k| < \delta, \\ -1, \text{ otherwise.} \end{cases} \tag{4}$$

3 METHOD

In this section, we present Competitive Experience Replay (CER) for policy gradient methods (Section 3.1) and describe the application of multi-agent DDPG to enable this technique (Section 3.2).

3.1 COMPETITIVE EXPERIENCE REPLAY

While the re-labelling strategy introduced by HER provides useful rewards for training a goal-conditioned policy, it assumes that learning from arbitrary goals will generalize to the actual task goals. As such, exploration remains a fundamental challenge for goal-directed RL with sparse reward. We propose a re-labelling strategy designed to overcome this challenge. Our method is partly inspired by the success of self-play in learning to play competitive games, where sparse rewards (i.e. win or lose) are common. Rather than train a single agent, we train a pair of agents on the same task and apply an asymmetric reward re-labelling strategy to induce a competition designed to encourage exploration. We call this strategy *Competitive Experience Replay* (CER).

To implement CER, we learn a policy for each agent, π_A and π_B, as well as a multi-agent critic (see below), taking advantage of methods for decentralized execution and centralized training.

During decentralized execution, π_A and π_B collect DDPG episode rollouts in parallel. Each agent effectively plays a single-player game; however, to take advantage of multi-agent training methods, we arbitrarily pair the rollout from π_A with that from π_B and store them as a single, multi-agent rollout in the replay buffer $\mathcal{D}$. When training on a mini-batch of off policy samples, we first randomly sample a mini-batch of episodes in $\mathcal{D}$ and then randomly sample transitions in each episode. We denote the resulting mini-batch of transitions as $\{(s^i_A, a^i_A, g^i_A, r^i_A, {s'}^i_A), (s^i_B, a^i_B, g^i_B, r^i_B, {s'}^i_B)\}_{i=1}^m$, where m is the size of the mini-batch.

Reward re-labelling in CER attempts to create an implicit exploration curriculum by punishing agent A for visiting states that agent B also visited, and, for those same states, rewarding agent B. For each A state s^i_A in a mini-batch of transitions, we check if any B state s^j_B in the mini-batch satisfies $|s^i_A - s^j_B| < \delta$ and, if so, re-label r^i_A with $r^i_A - 1$. Conversely, each time such a nearby state pair is found, we increment the associated reward for agent B, r^j_B, by $+1$. Each transition for agent A can therefore be penalized only once, whereas no restriction is placed on the extra reward given to a transition for agent B. Following training both agents with the re-labelled rewards, we retain the policy π_A for evaluation.

We focus on two variations of CER that satisfy the multi-agent self-play requirements: first, the policy π_B receives its initial state from the task's initial state distribution; second, although more restricted to re-settable environments, π_B receives its initial state from a random off-policy sample of π_A. We refer to the above methods as *independent-CER* and *interact-CER*, respectively, in the following sections.

Importantly, CER re-labels rewards based on intra-agent behavior, whereas HER re-labels rewards based on each individual agent's behavior. As a result, the two methods can be easily combined. In fact, as our experiments demonstrate, CER and HER are complementary and likely reflect distinct challenges that are both addressed through reward re-labelling.

3.2 GOAL CONDITIONED MULTI-AGENT LEARNING

We extend multi-agent DDPG (MADDPG), proposed by Lowe et al. (2017), for training using CER. MADDPG attempts to learn a different policy per agent and a single, centralized critic that has access to the combined states, actions, and goals of all agents.

More precisely, consider a game with N agents with policies parameterized by $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_N\}$, and let $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_N\}$ be the set of all agent policies. $\mathbf{g} = [g_1, \ldots, g_N]$ represents the concatenation of each agent's goal, $\mathbf{s} = [s_1, \ldots, s_N]$ the concatenated states, and $\mathbf{a} = [a_1, \ldots, a_N]$ the concatenated actions. With this notation, we can write the gradient of the expected return for agent i, $J(\theta_i) = \mathbb{E}[R_i]$ as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{\pi}}[\nabla_{\theta_i} \log \pi_i(a_i|s_i, g_i) Q^{\boldsymbol{\pi}}_i(\mathbf{s}, \mathbf{a}, \mathbf{g})]. \tag{5}$$

With deterministic policies $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_N\}$, the gradient becomes:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{\mu}}[\nabla_{\theta_i} \mu_i(s_i, g_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_i(\mathbf{s}, \mathbf{a}, \mathbf{g})|_{a_i = \mu_i(s_i, g_i)}], \tag{6}$$

The centralized action-value function $Q_i^{\boldsymbol{\mu}}$, which estimates the expected return for agent i, is updated as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{r},\mathbf{s}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{s},\mathbf{a},\mathbf{g}) - y)^2], \quad y = r_i + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{s}', a_1', \ldots, a_N', \mathbf{g})\big|_{a_j'=\boldsymbol{\mu}_j'(s_j)}, \tag{7}$$

where $\boldsymbol{\mu}' = \{\boldsymbol{\mu}_{\theta_1'}, ..., \boldsymbol{\mu}_{\theta_N'}\}$ is the set of target policies with delayed parameters θ_i'. In practice people usually soft update it as $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$, where τ is a Polyak coefficient.

During training, we collect paired rollouts as described above, apply any re-labelling strategies (such as CER or HER) and use the MADDPG algorithm to train both agent policies and the centralized critic, concatenating states, actions, and goals where appropriate. Putting everything together, we summarize the full method in Algorithm 1.

4 EXPERIMENT

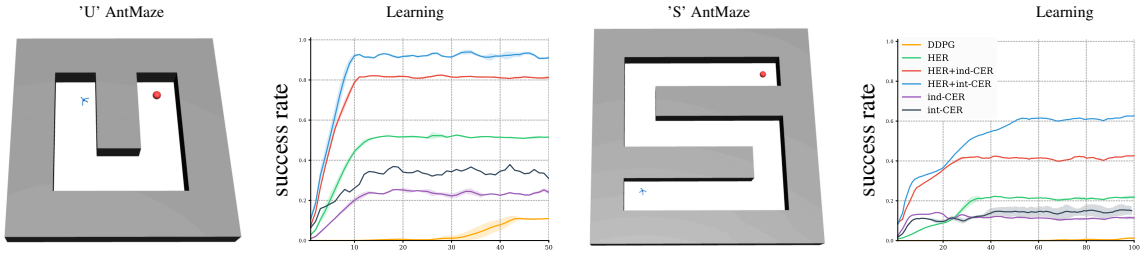

Figure 1: Goal conditioned 'U' shaped maze and 'S' shaped maze. Left two plots illustrate a rendering of the 'U' shaped maze and the associated success rate; the right two plots illustrate the same for the 'S' shaped maze. Each line shows results averaged over 5 random initializations. All metrics based on CER reference the performance of agent A.

We first study the effect of CER in an ant maze task (Section 4.1): we compare *ind-CER* and *int-CER*, with or without HER on different ant navigation maze tasks. We also examine how the performance of the adversarial agent (B) influences the outcome of CER (Section 4.2). Finally, we provide results from an extensive set of multi-goal sparse reward tasks (Plappert et al., 2018) from OpenAI gym (Brockman et al., 2016b) using the MuJoCo physics simulator (Todorov et al., 2012) in Section 4.3. Detailed settings and hyperparameters are provided in Appendix C.

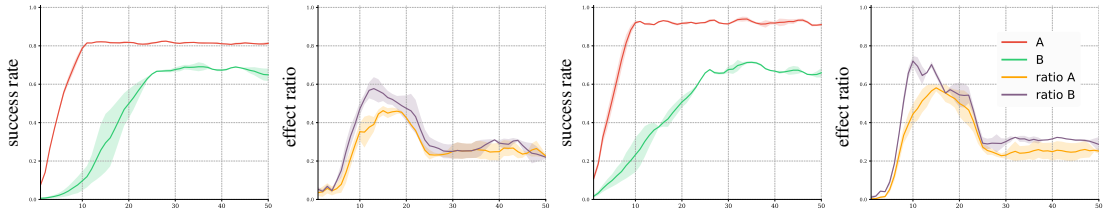

Figure 2: Illustration of the automatically generated curriculum between A and B on 'U' shaped AntMaze task. From left to right are success rate of *ind-CER*, effect ratio of *ind-CER*, success rate of *int-CER*, and effect ratio of *int-CER*. Each line shows results averaged over 5 random initializations.

4.1 COMBINING COMPETITIVE EXPERIENCE REPLAY WITH EXISTING METHODS

We start by asking whether CER improves performance and sample efficiency in a sparse reward task. To this end, we constructed two different mazes, an easier 'U' shaped maze and a more difficult 'S' shaped

maze (Figure 1). The goal of the ant agent is to reach the target mark by red sphere ball. The agent is always initialized at position $(0, 0)$. At each episode, the x- and y-axis location of the target sphere ball is sampled uniformly from $[5, 20]$, $[5, 20]$, respectively. At each step, the agent obtains a reward of 0 if the goal has been achieved and -1 otherwise. Additional details of the ant maze environments are found in Appendix B. An advantage of this environment is that it can be reset to any given state, facilitating comparison between our two proposed variants of CER.

We compare agents trained with HER, both variants of CER, and both variants of CER with HER. Since each uses DDPG as a backbone algorithm, we also include results from a policy trained using DDPG alone. The results for each maze are shown in Figure 1. DDPG by itself performs quite poorly, likely due to the sparsity of the reward in this task set up. Adding CER partially overcomes this limitation in terms of final success rate, and, notably, reaches this stronger result with many fewer examples. A similar result is seen when adding HER to DDPG. Importantly, adding both HER and CER dramatically improves the final success rate without requiring any observable increase in the number of episodes, compared to each of the other baselines. These results support our hypothesis that existing state-of-the-art methods do not sufficiently address the exploration challenge intrinsic to sparse reward environments. Furthermore, these results show that CER, when combined with existing methods, improves both the quality and efficiency of learning in such challenging settings. We find that this is especially true for *int-CER*, which we generally observe to be the stronger variant.

Figure 2 illustrates the success rates of agents A and B as well as the 'effect ratio,' which is the fraction of mini-batch samples whose reward is changed during training by CER, calculated as $\phi = \frac{N}{M}$ where N is the number of samples changed by CER and M is the number of samples in mini-batch. We observe a strong correspondence between the success rate and effect ratio, likely reflecting the influence of CER on the learning dynamics. While a deeper analysis would be required to concretely understand the interplay of these two terms, we point out that CER re-labels a substantial fraction of rewards in the mini-batch. Interestingly, even the relatively small effect ratio observed in the first few epochs is enough support rapid learning. We speculate that the sampling strategy used in *int-CER* provides a more targeted re-labelling, leading to the more rapid increase in success rate for agent A.

4.2 COMPETITIVE EXPERIENCE REPLAY WITH DIFFERENT ADVERSARIAL AGENTS

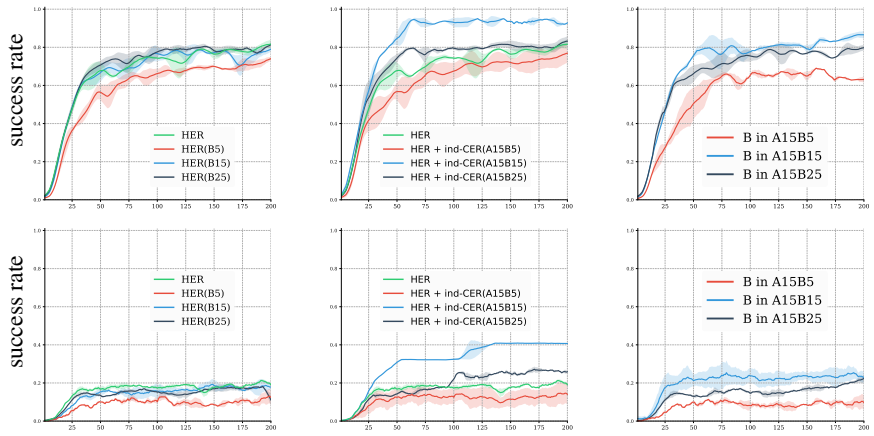

Figure 3: Performance on FetchSlide (top) and HandManipulatePenFull (bottom) for different batch-size multipliers when training using HER (left columns) or HER+CER (middle, right columns). Middle plots show the performance of agent A and, in the right plots, performance of agent B is shown for reference.

We next investigate how the performance of agent B impacts the final performance of agent A on FetchSlide and HandManipulatePenFull. Results are presented in Figure 3. As Andrychowicz et al. (2017) and

Andrychowicz et al. (2018) observe (and we also observe), performance depends on the batch size. We leverage this observation to tune the relative strengths of A and B by separately manipulating the batch sizes used for updating each.

For simplicity, we control the batch size by changing the number of MPI workers devoted to a particular update (see Appendix for details). Each MPI worker computes the gradients for a batch size of 256; averaging the gradients from each worker results in an effective batch size of $N * 256$. For our single-agent baselines, we choose $N = 30$ workers, and, when using CER, a default of $N = 15$ for each A and B In the following, $AxBy$ denotes, for agent A, $N = x$ and, for agent B, $N = y$. These results suggest that, while a sufficiently large batch size is important for achieving the best performance, the optimal configuration requires balancing the batch sizes used for the two agents. Interestingly, we observe that imbalance adversely effects both agents trained during CER.

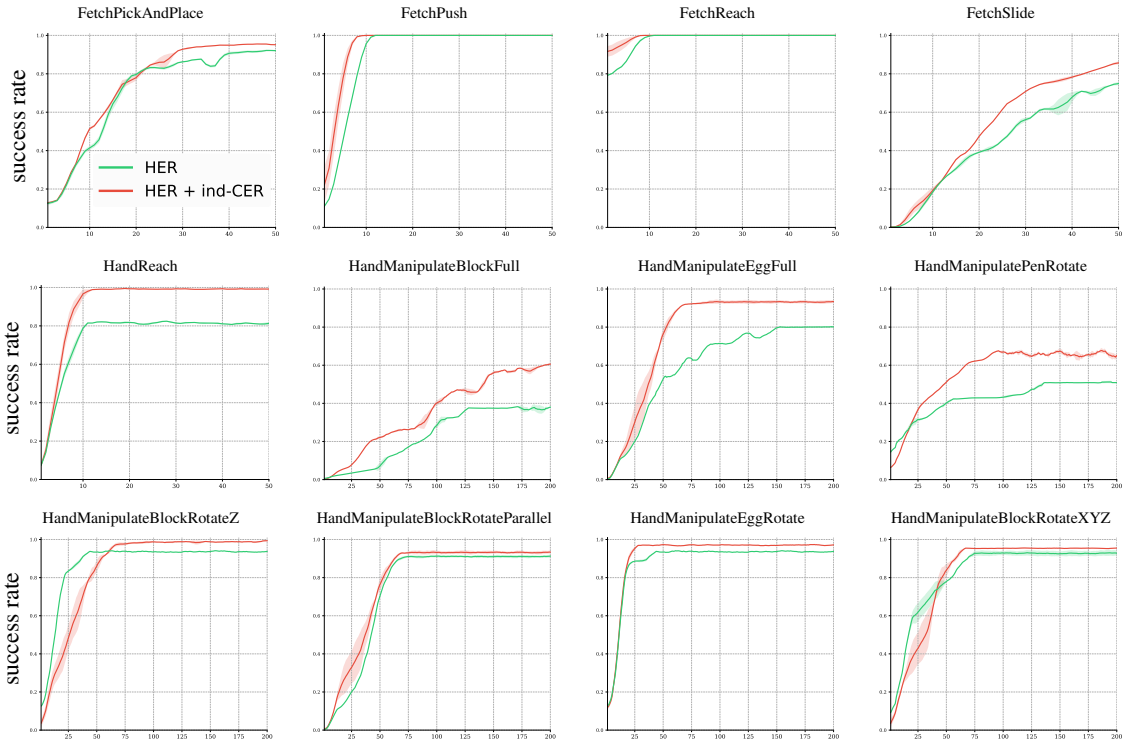

Figure 4: Evaluation of HER with *ind-CER* across different robotic control environments. Each line shows results averaged over 5 random initializations.

4.3 Comparison with state-of-the-art

To examine the efficacy of our method on a broader range of tasks, we evaluate HER with and without *ind-CER* on the challenging multi-goal sparse reward environments introduced in (Plappert et al., 2018). (Note: we would prefer to examine *int-CER* but are prevented by techincal issues related to the environment.) Results for each of the 12 tasks we trained on are illustrated in Figure 4. Our method, when used on top of HER, improves performance wherever it is not already saturated. This is especially true on harder tasks, where HER alone achieves only modest success (e.g., HandManipulateEggFull and handManipulatePenRotate). These results further support the conclusion that existing methods often fail to achieve sufficient exploration.

Our method, which provides a targeted solution to this challenge, naturally complements HER. The combined re-labelling strategy offers a simple yet effective way to learn from sparse rewards without resorting to hand-crafted goal curricula or reward shaping.

5 RELATED WORK

Self-play has a long history in this domain of research (Samuel, 1959; Tesauro, 1995; Heess et al., 2017). Silver et al. (2017a) use self-play with deep reinforcement learning techniques to master the game of Go; and self-play has even been applied in Dota 5v5 (OpenAI, 2018).

Foerster et al. (2016) introduce actor-critic methods for centralized training with decentralized execution in multi-agent domains. Similarly, Foerster et al. (2017) consider the use of a central critic for DDPG. Lowe et al. (2017) adopts the framework to let the critic observe all agents' actions to get a stationary environment.

Experience replay has been introduced in (Lin, 1992) and later was a crucial ingredient in learning to master Atari games (Mnih et al., 2013; 2015). Wang et al. (2016) propose truncation with bias correction to reduce variance from using off-policy data in buffer and achieves good performance on continuous and discrete tasks. Schaul et al. (2015) improve experience replay by assigning priorities to transitions in the buffer to efficiently utilize samples. Horgan et al. (2018) further improve experience replay by proposing a distributed RL system in which experiences are shared between parallel workers and accumulated into a central replay memory and prioritized replay is used to update the policy based on the diverse accumulated experiences.

Curriculum learning is widely used for training neural networks (see e.g., Bengio et al., 2009; Graves et al., 2017). However, the curriculum is almost always hand-crafted. Florensa et al. (2017); Held et al. (2017) propose to automatically adjust task difficulty during training Recently, Sukhbaatar et al. (2017) suggest to use self-play between two agents where reward depends on the time of the other agent to complete to enable implicit curriculum. Our work is similar to theirs, but we propose to use sample-based competitive experience replay, which is not only more readily scalable to high-dimension control but also integrates easily with Hindsight Experience Replay (Andrychowicz et al., 2017). The method of initializing based on a state not sampled from the initial state distribution has been explored in other works. For example, Ivanovic et al. (2018) propose to create a backwards curriculum for continuous control tasks through learning a dynamics model. Resnick et al. (2018b) and Salimans & Chen (2018) propose to train policies on Pommerman (Resnick et al., 2018a) and the Atari game 'Montezumas Revenge' by starting each episode from a different point along a demonstration. Recently, Goyal et al. (2018) and Edwards et al. (2018) propose a learned backtracking model to generate traces that lead to high value states in order to obtain higher sample efficiency.

6 CONCLUSION

We introduce Competitive Experience Replay, a new and general method for encouraging exploration through implicit curriculum learning in sparse reward settings. We demonstrate an empirical advantage of our technique when combined with existing methods in several challenging RL tasks. In future work, we aim to investigate richer ways to re-label rewards based on intra-agent samples to further harness multi-agent competition. We hope that this will facilitate the application of our method to more open-end environments with even more challenging task structures. In addition, future work will explore integrating our method into approaches more closely related to model-based learning, where adequate exposure to the dynamics of the environment is often crucial.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *arXiv:1808.00177*, 2018.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48. ACM, 2009.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016a.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016b.

Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3381–3388. IEEE, 2017.

Ashley D Edwards, Laura Downs, and James C Davidson. Forward-backward reinforcement learning. *arXiv preprint arXiv:1803.10227*, 2018.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.

Anirudh Goyal, Philemon Brakel, William Fedus, Timothy Lillicrap, Sergey Levine, Hugo Larochelle, and Yoshua Bengio. Recall traces: Backtracking models for efficient reinforcement learning. *arXiv preprint arXiv:1804.00379*, 2018.

Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003*, 2017.

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. *arXiv preprint arXiv:1705.06366*, 2017.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Boris Ivanovic, James Harrison, Apoorva Sharma, Mo Chen, and Marco Pavone. Barc: Backward reachability curriculum for robotic reinforcement learning. *arXiv preprint arXiv:1806.06161*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pp. 6379–6390, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pp. 363–372. Springer, 2006.

OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*, 2018a.

Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alex Peysakhovich, Kyunghyun Cho, and Joan Bruna. Backplay:" man muss immer umkehren". *arXiv preprint arXiv:1807.06919*, 2018b.

Tim Salimans and Richard Chen. Learning montezumas revenge from a single demonstration. `https://blog.openai.com/learning-montezumas-revenge-from-a-single-demonstration/`, 2018.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. In *International Conference on Machine Learning*, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference of Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *Advances in Neural Information Processing Systems*, 2017.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017b. ISSN 0028-0836.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

A ALGORITHM

We summarize the algorithm for HER with CER in Algorithm 1.

Algorithm 1 HER with CER

Initialize a random process $\mathcal{N}$ for action exploration, max episode length to L
Set CER to *ind-CER* or *int-CER*
for episode $= 1$ to M **do**
  Receive initial state s_A
  Initialize episode buffer $\textit{buffer}_A$
  for $t = 1$ to L **do**
    select action $a_A = \boldsymbol{\mu}_{\theta_i}(s_A) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions a_A and observe reward r_A and new state s'_A
    Store $(s_A, a_A, g_A, r_A, {s_A}')$ in $\textit{buffer}_A$
    $s_A \leftarrow s'_A$
  end for
  Receive initial state s_B or Receive initial state s_B, where s_B is a state sampled from $\textit{buffer}_A$
  Initialize episode buffer $\textit{buffer}_B$
  for $t = 1$ to L **do**
    select action $a_B = \boldsymbol{\mu}_{\theta_i}(s_B) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions a_B and observe reward r_B and new state s'_B
    Store $(s_B, a_B, g_B, r_B, {s_B}')$ in $\textit{buffer}_B$
    $s_B \leftarrow s'_B$
  end for
  Concatenate $\textit{buffer}_A$ and $\textit{buffer}_B$ and store $(\{\mathbf{s^i}\}_{i=1}^T, \{\mathbf{a^i}\}_{i=1}^T, \{\mathbf{g^i}\}_{i=1}^T, \{\mathbf{r^i}\}_{i=1}^T, \{\mathbf{s'^i}\}_{i=1}^T)$ in replay buffer $\mathcal{D}$
  // Optimization based on off-policy samples
  for k $= 1$ to K **do**
    // Relabelling off-policy samples
    Sample a random minibatch of S samples $(\mathbf{s}^j, \mathbf{a}^j, \mathbf{g}^j, \mathbf{r}^j, \mathbf{s}'^j)$ from $\mathcal{D}$
    Apply HER strategy on samples
    Apply *ind-CER* or *int-CER* strategy on samples
    for agent $i = A, B$ **do**
      Do one step optimization based on Eq (6) and Eq (7), and update target networks.
    end for
  end for
end for

B ENVIRONMENT DETAILS

Robotic control environments The robotic control environments shown in Figure 5 are part of challenging continuous robotics control suite (Plappert et al., 2018) integrated in OpenAI Gym (Brockman et al., 2016a) and based on Mujoco simulation enginge (Todorov et al., 2012). The Fetch environments are based on the 7-DoF Fetch robotics arm, which has a two-fingered parallel gripper. The Hand environments are based on the Shadow Dexterous Hand, which is an anthropomorphic robotic hand with 24 degrees of freedom. In all tasks, rewards are sparse and binary: the agent obtains a reward of 0 if the goal has been achieved and -1 otherwise. For more details please refer to Plappert et al. (2018).

Ant maze environments We also construct two maze environments with the *re-settable* property, rendered in Figure 1, in which walls are placed to construct "U" or "S" shaped maze. At the beginning of each episode, the agent is always initialized at position $(0, 0)$, and can move in any direction as long as it is not obstructed by a wall. The x- and y-axis locations of the target ball are sampled uniformly from $[-520]$, $[5, 20]$, respectively.

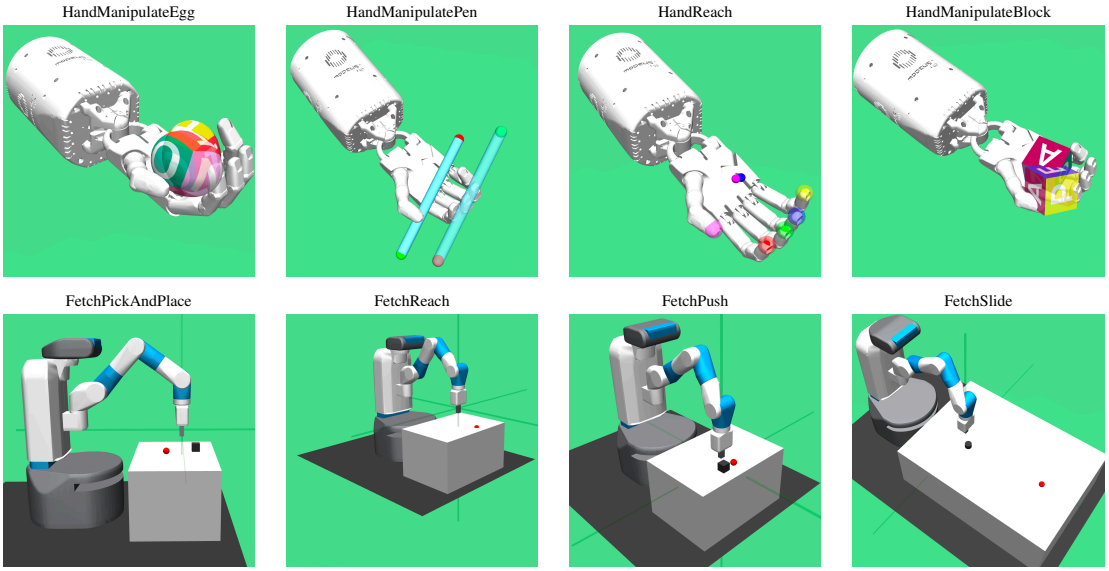

Figure 5: Robotics control environments used in our experiments

C IMPLEMENTATION DETAILS

In the code for HER (Andrychowicz et al., 2017), the authors use MPI to increase the batch size. MPI is used here to run rollouts in parallel and average gradients over all MPI workers. We found that MPI is crucial for good performance, since training for longer time with a smaller batch size gives sub-optimal performance. This is consistent with the authors' findings in their code that having a much larger batch size helps a lot. The batch size scales linearly with number of CPUs N in implementation since each MPI worker computes the gradient for a batch size of 256 and then average those gradient together, which making the effective batch size is $N * 256$. In our robotic control experiments, we set the batch size of a MPI worker to 256. For the ant maze task, we set the batch size of each MPI worker to 128, and we choose to increase N from the original 19 in HER to 30 for baseline method. For fairness, we choose $N = 15$ for both agent A and B in both *ind-self-play* and *int-self-play* methods. We perform an analysis of the impact of N on our method in Section 4.2.

In our implementation[1], we choose to create separate MPI groups for agent A and B,. Each group has a replay buffer for saving off-policy samples. The two replay buffers are communicated between the worker groups before optimization. In this way, the two groups of MPI workers interact only through the buffer, where we apply HER strategy and CER strategy. We do N number of multiple rollouts in parallel for each agent and have separate optimizers for each agent.

We found that periodically resetting the parameters of agent B in early stage of training helps agent A more consistently reach a high level of performance. This resetting helps to strike an optimal balance between the influences of HER and CER in training agent A. We also add L2 regularization, following the practice of Andrychowicz et al. (2017).

For the experiments with neural networks, all parameters are randomly initialized from $\mathcal{N}(0, 0.2)$. We use networks with three layers of hidden layer size 256. We use Adam (Kingma & Ba, 2014) for optimization. We average results over 5 random seeds; although, we found that the random seed does not have much impact on the performance of baselines and our methods in our implementations.

We summarize the hyperparameters in Table 1.

Hyperparameter name	'U' AntMaze	'S' AntMaze	Fetch Control	Hand Control
Buffer size	$1E5$	$1E6$	$1E6$	$1E6$
Batch size	128	128	256	256
Max steps of episode	50	100	50	100
Reset epochs	2	2	5	5
Max reset epochs	10	10	20	30
Total epochs	50	100	100/50	200
Actor Learning rate	0.0004	0.0004	0.001	0.001
Critic Learning rate	0.0004	0.0004	0.001	0.001
Action L2 regularization	0.01	0.01	1.00	1.00
Polyak	0.95	0.95	0.95	0.95

Table 1: Hyperparameter values used in experiments.

[1]Source code for the experiments and videos of results will be released later at `https://sites.google.com/view/cerresults`