

# Hindsight Experience Replay With Experience Ranking

Hai Nguyen, Hung Manh La  
*Advanced Robotics and Automation Lab*  
*Department of Computer Science and Engineering*  
*University of Nevada, Reno, NV 89557, USA*  
hla@unr.edu

Matthew Deans  
*Intelligent Robotics Group*  
*NASA Ames Research Center*  
*Moffett Field, CA 94035, USA*

**Abstract**—Reinforcement Learning (RL) algorithms face difficulties when dealing with robotic tasks in sparse reward settings and as a result, they often require millions of interactions with the environment to learn successfully. A recent algorithm Hindsight Experience Replay (HER) was introduced to tackle this difficulty by adding virtual goals and therefore increase significantly the sample-efficiency by learning in transitions when the robot does not achieve the original goal. However, these additional goals are sampled randomly from each episode batch of transitions, which might have no relationship with the original goal. This might make learning with the original goal slower due to the bad influence of irrelevant virtual goals. In this paper, we address this issue by applying experience ranking (ER) to these additional goals. We first compare each sampled virtual goal and the original goal and then compare the difference with a threshold. Transitions in which the robot achieves a virtual goal that is not close to the original goal are filtered out, and the remaining are used for training the policy. The improvement in learning performance is validated in four simulated robotic tasks. The experiment results show significant improvement in terms of the learning speed and robustness.

**Index Terms**—reinforcement learning, hindsight experience replay, experience ranking.

## I. INTRODUCTION

Reinforcement Learning (RL) methods have been applied to a variety of real-world robotics problems ranging from multi-robot systems, locomotion [1] to manipulation [2] [3]. However, one of the main challenges that limit RL's full potential for robotics applications is how to learn efficiently from sparse and binary rewards, which is very common in many robotic tasks. It is challenging as in most learning attempts, by exploration, the robot will mostly end up learning nothing from zero rewards, and they are clueless about what to improve next time. One of the best methods that we have had so far to tackle the problem is using Hindsight Experience Replay (HER) [4] in combination with an off-policy RL method such as Deep Deterministic Policy Gradient (DDPG) [5]. By modifying the original goal, HER allows the robot to even learn from unsuccessful experiences [4].

One problem with HER is that it uses a hard-coded strategy [6] for randomly choosing additional goals from each episode of transitions to replay. This lack of an evaluation scheme makes it possible that the chosen transitions are poor experiences, which might actually slow the learning process [7].

To solve this problem and respond to OpenAI's request for research for an automatic hindsight goal generation for HER mentioned in [6], we propose a novel idea to rank training experiences and therefore filter out a majority amount of low-ranked experiences. Those remaining high-scored experiences will then be used for learning. We also verify that the proposed experience ranking (ER) scheme can improve significantly the learning curve in four different tasks in robotic tasks within Gym [8].

## II. RELATED WORK

In biological systems, experience replay has played an important role in learning behaviors. Various studies show that there is frequent experience replay in the hippocampus of rodents of both awake and sleeping animals. Research also shows that disrupting experience replay might lead to impairing spatial memory [9] [10], and some show that experience replay can contribute to better memory consolidation and retrieval [11], both are instrumental in boosting learning ability.

In machine learning, the idea of using experience replay was first introduced in [12] when it is used to speed up learning by repeating past experiences. Experience replay helps break the temporal correlation by mixing old and new experiences, and frequent experiences, therefore, might be replayed more often. Since then, experience replay has been widely adopted, and it becomes the norm in many success in RL research [13] [14].

However, the current way to use experience replay is to uniformly sample from first-in-first-out buffers of experiences [5] [14]. By doing this way, the importance of each transition is neglected, and we might replay experience that is not very helpful for learning at all. When faced with the design choice of which experiences to restore, neuroscience research shows that surprising [15] and rewarding experiences are more preferred [16].

In RL, attempts to prioritizing experiences have been made in [17] where temporal difference (TD) error is used as a way to measure the priority. Our approach is goal-favored and more direct. We will look at the transition of the training episode and use the current achievement of the robot in this transition

and decide whether the associated experience is worth being sampled and stored for future learning.

### III. BACKGROUND

#### A. Reinforcement Learning Formalism

Considering the standard RL setting when a robot interacts with an environment, it continually makes decisions, and the environment responds to the chosen actions and brings the robot to new states. At each step, the robot receives the current environment's state  $S_t \in \mathcal{S}$  and based on the current policy  $\pi$  to select an action  $a_t \in \mathcal{A}$ . In the next time step, the robot receives a numerical reward  $R_{t+1} \in \mathbb{R}$  generated from a reward function  $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  and itself being in another state  $S_{t+1}$ . The goal of the robot is to maximize the reward over time through learning policy, which can help to choose actions that lead to maximizing the accumulated reward. The dynamics of the environment is defined by the state-transition probability  $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ .

A deterministic policy will map states to actions:  $\pi : \mathcal{S} \rightarrow \mathcal{A}$ . Every episode starts with sampling an initial state  $s_0$ , and at every time step  $t$  the robot selects an action based on the current policy and its current state:  $a_t = \pi(s_t)$ . Then it receives the reward  $r_t = r(s_t, a_t)$  from the environment, and the environment's new state is sampled from the distribution  $p(\cdot|s_t, a_t)$ .

A discounted sum of future rewards is  $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$  with a discount rate  $\gamma \in [0, 1]$ , and the robot's goal is to maximize its expected return starting from the beginning  $\mathbb{E}_{s_0}[R_0|s_0]$ .

The action-value function describing the value of taking action  $a$  in state  $s$  under the policy  $\pi$  is defined as  $Q^\pi(s, a) = \mathbb{E}[R_t|s_t = s, a_t = a]$ . An optimal policy  $\pi^*$  i.e. any policy  $\pi^*$  s.t.  $Q^{\pi^*}(s, a) \geq Q^\pi(s, a)$  for every  $s \in \mathcal{S}, a \in \mathcal{A}$ , and any policy  $\pi^*$  share the same *optimal Q-function* denoted as  $Q^*$ . It satisfies the *Bellman* equation:

$$Q^*(s, a) = \mathbb{E}\left[R(s, a) + \gamma \max_{a'} Q^*(s', a')\right].$$

#### B. Deep Q-Networks (DQN)

DQN [14] is a model-free RL algorithm for discrete action spaces. In DQN, a neural networks is used to approximate the action-value function  $Q(s, a)$ . Two important features making the success of DQN are *experience replay* and *target network*.

*Experience replay* is used to store previous transitions, which are defined as tuples  $(s_t, a_t, r_t, s_{t+1})$  where  $s_{t+1}$  is the next state. These tuples will then be added to a *replay buffer* from which batch of transitions will be sampled for training the policy. The usage of experience replay provides uncorrelated samples for training and also improves data efficiency [18].

The network is then trained using mini-batch gradient descent on the loss  $\mathcal{L} = \mathbb{E}(Q(s_t, a_t) - y_t)^2$ , where  $y_t = r_t + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a')$ , with the tuple  $(s_t, a_t, r_t, s_{t+1})$  sampled from the replay buffer.

In order to make this optimization procedure stabler, the target  $y_t$  is usually computed using a separate *target network*, which changes at a slower pace than the main network. After

that, the weights of the *target network* will be periodically updated to the main network to increase stability.

#### C. Deep Deterministic Policy Gradient - DDPG

Although DQN can achieve performance in higher dimensional problems such as Atari games [13], it only works with discrete action spaces. DDPG [5] is developed based on DQN, and it is a model-free off-policy RL algorithm for continuous action spaces.

DDPG belongs to the actor-critic class where there are two neural networks: an *actor*  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  and a *critic*  $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ . The critic's job is to evaluate the current policy by estimating the Q-value from the current state and the action outputted from the actor.

#### D. Hindsight Experience Replay - HER

The idea behind HER is to mimic the human ability to learn from failures. HER allows learning from all episodes, even if in those episodes the robot did not achieve the original goal. Instead, HER considers some of the states that the robot finally reached to be modified goals. While in the standard experience replay, only the transition  $(s_t||g, a_t, r_t, s_{t+1}||g)$  with the original goal  $g$  is stored, HER also keeps the transition  $(s_t||g', a_t, r', s_{t+1}||g')$  with the modified goal  $g'$ . By adding these additional goals, the RL algorithm can still have a learning outcome because it can learn to achieve a goal even that goal is not the original target.

In [4], different strategies are experimented for choosing additional goals. The chosen *future* strategy, a replay with  $k$  random states, which come from the same episode as the transition being replayed and was observed *after* it. However, both strategies neglect the significance of each transition to the original goal.

### IV. DDPG + HER AND EXPERIENCE RANKING

#### A. Motivation

We are motivated by humans when we try to perform a new task such a child learns to ride a bike for the first time. At first, it is very likely that he will fall down many times but he gets better as he keeps learning from failures. The process of learning from failures is adopted in HER by adding additional goals besides the original goal. However, when trying to learn from failures, humans do not treat all failures equally. We often start with the most successful attempts in which we achieved the most. The child would try to modify his way of riding the bike from those past attempts when he could ride a long way without falling. In other words, humans improve from the experiences in which we were closer to the target task. In this process, additional goals are sampled not in a random way as presented in HER but instead are sampled depending on how close we were to the main target task. In our proposed approach, the ER is added to the original HER, giving rise to ER-HER.

---

**Algorithm 1** DDPG + HER modified with ER
 

---

```

1: Initialize DDPG
2: Initialize replay buffer  $R$ 
3: for episode = 1,  $M$  do
4:   Sample a goal  $g$  and an initial state  $s_0$ 
5:   for  $t = 0, T-1$  do
6:     Sample an action  $a_t$  using DDPG behavioral policy
7:     Execute the action  $a_t$  and observe a new state  $s_{t+1}$ 
8:   end for
9:   for  $t = 0, T - 1$  do
10:     $r_t := r(s_t, a_t, g)$ 
11:    Store the transition  $(s_t || g, a_t, r_t, s_{t+1})$  in  $R$  ▷  $||$ denotes concatenation
12:    Sample a set of additional goals for replay  $G := \mathbb{S}$  (current episode)
13:    for  $g' \in G' \subset G$  do ▷  $G'$  contains  $k/(1+k)$  the total number of elements from  $G$ 
14:      Calculate the distance to the original goal  $d = d(g', g)$  ▷ Rank the experience
15:      if  $d \leq \text{threshold}$  then
16:         $r' := r(s_t, a_t, g')$  ▷ Recalculate reward with the new goal
17:        Store the transition  $(s_t || g', a_t, r', s_{t+1})$  in  $R$  ▷ HER
18:      end if
19:    end for
20:  end for
21:  for  $t = 1, N$  do
22:    Sample a mini-batch from the replay buffer  $R$ 
23:    Perform one step of optimization
24:  end for
25: end for

```

---

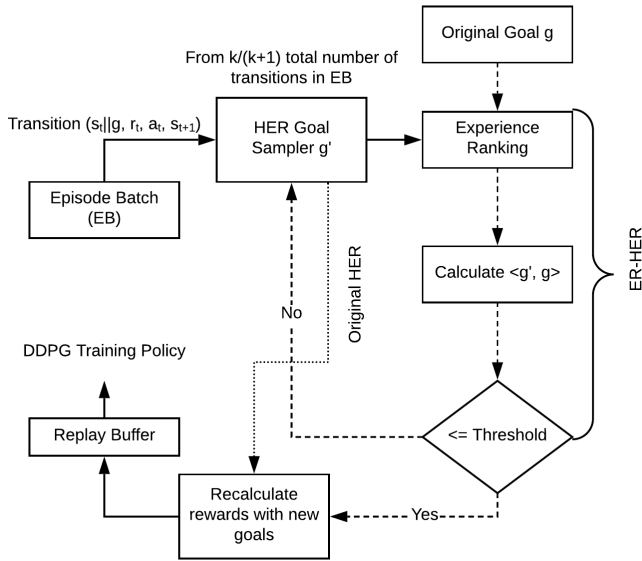


Fig. 1: Proposed approach with ER.

### B. Experience Ranking

1) *How to rank experiences*: In HER, binary rewards in the form  $r(s, a, g) = -(|g_{\text{achieved}} - g| \leq \epsilon)$  are considered. In this equation,  $g$  is the current desired goal, and  $g_{\text{achieved}}$

is the goal that the robot has achieved. Remember that the function  $r(s, a, g)$  is a 0/1 reward with the value determined by comparing what the robot has achieved with the desired achievement. As shown in Figure 1, after a robot finishes an episode of training (the duration from an initial state until a terminal state), an episode batch will contain multiple transitions in the form of the tuple  $(s_t || g, a_t, r_t, s_{t+1})$ . Original HER (densely dotted arrow) algorithm will then sample randomly additional goals  $g'$  from this episode batch. These goals are random states that the robot reached during the episode. Rewards with these new additional goals will then be calculated to create new transitions associated with new goals, which will be stored in the replay buffer for training the policy. However, with ER (less dense dotted arrows), the distance between  $g'$  and  $g$  will be computed first to measure how close the robot was to the original goal. For our validated robotic tasks, as the goal  $g$  and  $g'$  are vectors (of 3D positions or joint states), we use the Euclidean distance to measure their distance. The additional goal  $g'$  will only be selected if the difference is smaller than a threshold. The selection indicates that during the transition, the robot has moved towards the original goal within the threshold distance. This additional goal will then be processed similarly as described earlier.

2) *Overcome over-exploitation*: As ER favors experiences in which we more closely reach the original goal, it might overly favor exploitation over exploration, leading to sub-

optimal policies. Fortunately, HER already possesses an internal mechanism to balance exploitation and exploration. For choosing additional goals, HER employs the so-called *future* mechanism. In this strategy, additional goals selected will be  $k$  random states, which come from the same episode as the transition being replayed. This hyper-parameter controls the ratio between HER replays and regular replays. For instance, for the recommended value of  $k = 4$ , 20% of normal transitions with the original goal will be stored right away at the replay buffer. ER is deployed in the majority 80% of the remaining transitions. The output of HER sampler with ER will then include both ranked experiences and normal ones, allowing the algorithm to have a certain degree of exploration.

### C. Algorithm

We chose the combination of DDPG + HER mentioned in [4] even HER can be used together with any off-policy RL algorithm. The procedure would be the same with other RL algorithms. The way that we apply ER for the combination of DDPG + HER is described in Algorithm 1. Start in line 13, additional goals are sampled from  $G' \subset G$ . The hyper-parameter  $k$  as described earlier controls the ratio between the total number of elements in  $G'$  and  $G$ . For instance, when  $k = 4$ , all elements of  $G'$  will be taken randomly from 80% the total number of elements of  $G$ . ER is performed in the next line when the difference between an additional goal  $g'$  (randomly sampled from  $G'$ ), and the original goal  $g$  is calculated. We only keep  $g'$  when the difference is smaller than a certain threshold. When  $g'$  is kept, a new reward will be re-calculated with this new goal, and the transition is stored in the replay buffer. If  $g'$  does not satisfy the chosen condition, the loop continues when a next goal is sampled from  $G'$  until we loop through every element in the set.

## V. EXPERIMENTS AND RESULTS

### A. Environments

We analyze the effect of ER with four simulated robotic tasks from Gym with MuJoCo [19] as a physics simulator. We also use the DDPG and HER implementation from OpenAI's baselines source code. We experiment on three Fetch robotic tasks (FetchReach-v1, FetchPush-v1, and FetchPickAndPlace-v1) and one ShadowHand task (HandReach-v0) as illustrated in Figure 2.

- *FetchReach-v1*: A goal position is randomly chosen in 3D space. Control Fetch's end-effector to reach that goal as quickly as possible.
- *FetchPush-v1*: A goal position is randomly chosen on the table surface. Control Fetch's end-effector to push the block towards that position.
- *FetchPickAndPlace-v1*: A goal is randomly chosen in 3D space. Control Fetch's end-effector to grasp and lift the block up to reach that goal.
- *HandReach-v0*: A goal hand pose is randomly chosen in 3D space. Control the ShadowHand actuators to reach this position for all the five fingers.

The Fetch tasks are based on a 7-DoF Fetch robot, which has a two-fingered parallel gripper. In these tasks:

- *States*: The states include angles and velocities of all joints and the positions, rotations, and velocities (linear and angular) of all objects.
- *Goals*: Goals describe the desired position of the object with some fixed tolerance of  $\epsilon$  i.e.  $G = \mathbb{R}^3$  and  $f_g(s) = (|g - s_{object}| \leq \epsilon)$ . In this formula,  $s_{object}$  is the position of the object in the state  $s$ .
- *Rewards*: Rewards are binary values, calculated by evaluating  $r(s, a, g) = -(f_g(s') == 0)$  where  $s'$  is the state after the robot takes the action  $a$  in the state  $s$ .
- *Observations*: The observation vector includes the absolute position of the gripper, the relative position between the object and the target, and the distance between the 2 fingers. The linear velocity of the gripper and fingers, as well as relative linear and angular velocity of the object, are also given.
- *Actions*: Gripper rotation is kept fixed as it does not need for our tasks. Action space is four-dimensional including three for the desired relative gripper position at the next time step and one dimension specifies the distance between the 2 fingers of the gripper.

In the ShadowHand task, there is a robotic hand with 24 degrees of freedom with 20 of them can be controlled independently.

- *Actions*: Actions are 20-dimensional, indicating independent position control for 20 non-coupled joints of the hand.
- *Observations*: The vector includes the 24 positions and velocities of the hand's joints. Cartesian positions of all 5 fingertips are also included.

### B. Does ER Improve Learning?

We first compute the ranges of the Euclidean distance between HER's normal additional goals with the original goal as shown in Table I. We first run a small number of initial episodes to gather data to calculate these ranges, which are listed in the column *DistanceRange*. In these runs, the robot just starts learning with actions taken mostly from a random policy.

Task	Distance Range	Tested Thresholds
FetchReach-v1	[-0.43 0]	[- <b>0.3</b> -0.2 -0.1]
FetchPush-v1	[-0.35 0]	[-0.3 - <b>0.2</b> -0.1]
FetchPickAndPlace-v1	[-0.75 0]	[-0.5 -0.4 - <b>0.3</b> ]
HandReach-v0	[-0.18 0]	[-0.15 -0.1 - <b>0.05</b> ]

TABLE I: Distance ranges and tested thresholds. Threshold values that perform best are emphasized in bold.

If we keep running for a longer duration, the distance will become close to zero, indicating that the robot is closer to successfully learn to perform the task. After we obtained these ranges, we choose different threshold values within the ranges and run the learning process again but now with ER. The results are shown in Figure 3. Note that in this figure and the

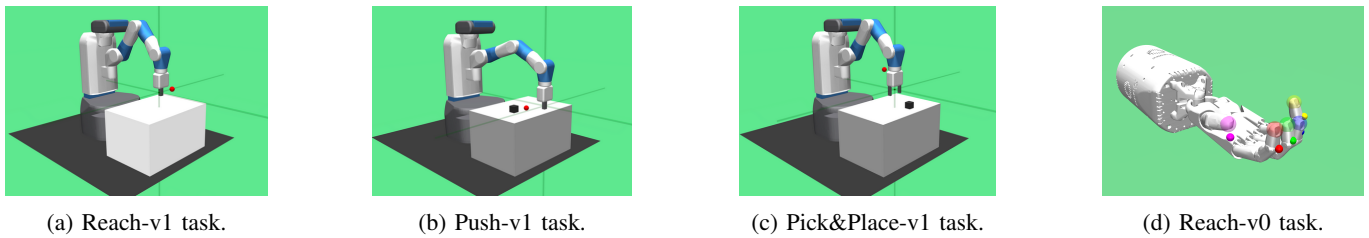


Fig. 2: Three Fetch and one ShadowHand tasks used for validation.

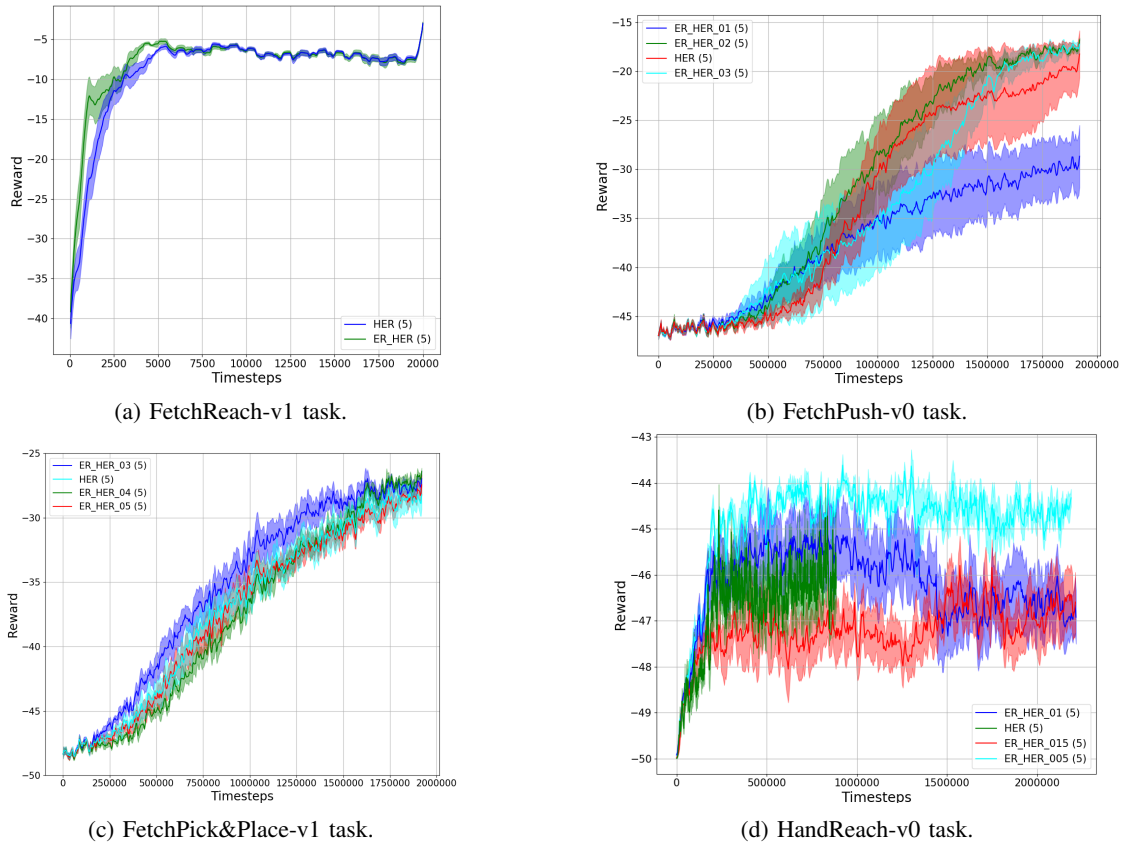


Fig. 3: Learning curves with ER. Multiple runs with five different random seeds are plotted with variances. *HER* denotes the original DDPG + HER algorithm. *ER\_HER\_0x* denotes ER-HER that uses the threshold value of  $-0.x$ .

next, the legend *ER\_HER\_0x* denotes ER-HER that uses the threshold value of  $-0.x$ . In all four tasks, there is always a threshold making the learning more efficient. These threshold values that perform the best are put in bold as shown in Table I. Compared to the original performance of DDPG + HER, with ER and best values of the threshold, the robot actually learns faster with a smaller variance in the learning curve. The smaller variance can be more clearly seen towards the end of the learning curves when they are converging as shown in Figure 4 with the best threshold in the FetchPush task. In this case, the robot achieves more reward and the variance between runs of different random seeds is smaller compared to the original DDPG + HER. This might be explained when being near the convergence, ER still strictly removes those

transitions in which the robot achieved less. On the other hand, the original DDPG + HER still allows a large degree of transitions in which the robot is far from the given target to be stored in the replay buffer. As a result, when drawing samples from the buffer, the policy might actually improve from these bad transitions, making the learning vary more. Also, from the learning curves of the FetchPush-v1 task, we can also see that there will be values of threshold slowing the learning process. In this case, the experience ranker is so strict that they only allow a small number of transitions to pass through. As a sequence, the robot would not have enough good samples to learn from and as a result, the learning process is actually slowed down.

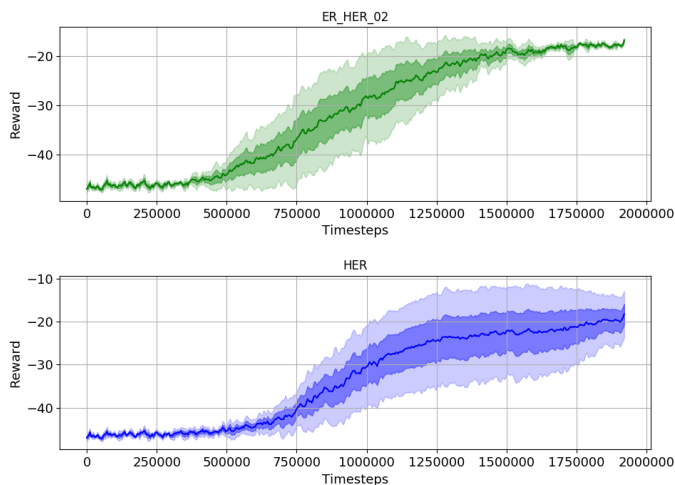


Fig. 4: Learning variances of the FetchPush-v1 task.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new method to improve the performance of the existing DDPG + HER combination by introducing ER to decide whether a transition should be stored and replayed for future learning. We also validated the performance with four simulated robotic tasks. By limiting the lowest rank experience to be used for training, the results show that the learning performance outperformed the original DDPG + HER in all tasks. Future development is to perform ER with increasingly reducing threshold instead of fixed values in a  $\epsilon$ -greedy style. In this case, the experience ranker will become increasingly stricter when only allowing experiences when the robot nearly achieves the original goal to pass. This is the same when we use the  $\epsilon$ -greedy exploration mechanism when the degree of exploration is reduced when the robot is closer to a good policy. Finally, perhaps to learn faster, we should focus more on the quality of training data rather than the quantity [20].

## ACKNOWLEDGMENT

This material is based upon work supported by the National Aeronautics and Space Administration (NASA) Grant No. NNX15AI02H issued through the NVSGC-RI program under sub-award No. 19-21, and the RID program under sub-award No. 19-29, and the NVSGC-CD program under sub-award No. 18-54. This work is also partially supported by the Office of Naval Research under Grant N00014-17-1-2558.

## APPENDIX

The source code is available at <https://github.com/aralab-unr/ExperienceRanking>. In this repository, we store our modified version of OpenAI's baseline. We modified the baseline's implementation of HER sampler (the `_sample_her_transitions` function) in `her_sampler.py`. We run training of all tasks on an Alienware Aurora R7 computer with 12 physical cores and a 16 GB NVIDIA GeForce GTX 1080 GPU card.

## REFERENCES

- [1] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, April 2004, pp. 2619–2624 Vol.3.
- [2] E. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning of motor skills in high dimensions: A path integral approach," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 2397–2403.
- [3] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant manipulation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 4639–4644.
- [4] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.
- [5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.
- [6] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," *CoRR*, vol. abs/1802.09464, 2018.
- [7] A. Sehgal, H. La, S. Louis, and H. Nguyen, "Deep reinforcement learning using genetic algorithm for parameter optimization," in *IEEE International Conference on Robotic Computing (IRC)*. Italy, 2019, pp. 596–601.
- [8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
- [9] G. Girardeau, K. Benchenane, S. I. Wiener, G. Buzsáki, and M. B. Zugaro, "Selective suppression of hippocampal ripples impairs spatial memory," *Nature Neuroscience*, vol. 12, pp. 1222 EP –, Sep 2009.
- [10] V. Ego-Stengel and M. A. Wilson, "Disruption of ripple-associated hippocampal activity during rest impairs spatial learning in the rat," *Hippocampus*, vol. 20, no. 1, pp. 1–10, Jan 2010, 19816984[pmid].
- [11] M. F. Carr, S. P. Jadhav, and L. M. Frank, "Hippocampal replay in the awake state: a potential substrate for memory consolidation and retrieval," *Nature Neuroscience*, vol. 14, pp. 147 EP –, Jan 2011, review Article.
- [12] L. J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.
- [13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.
- [14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
- [15] S. Cheng and L. M. Frank, "New experiences enhance coordinated neural activity in the hippocampus," *Neuron*, vol. 57, no. 2, pp. 303 – 313, 2008.
- [16] L. A. Atherton, D. Dupret, and J. R. Mellor, "Memory trace replay: the shaping of memory consolidation by neuromodulation," *Trends in Neurosciences*, vol. 38, no. 9, pp. 560 – 570, 2015.
- [17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations*, Puerto Rico, 2016.
- [18] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *CoRR*, vol. abs/1611.01224, 2016.
- [19] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.
- [20] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, Feb 2019, pp. 590–595.