Experience Replay Methods in Soft Actor-Critic

Lucas Kabela
Department of Computer Science
University of Texas at Austin
Email: lucaskabela@utexas.edu

***Abstract*—Soft Actor-Critic (SAC) is a state of the art off-policy algorithm for deep reinforcement learning; however, little exploration into a critical component of the algorithm, experience replay, has taken place since its inception. Meanwhile, surveys over experience replay strategies and hyperparameter settings in deterministic algorithms such as DQN and DDPG have lead to considerable improvements in algorithmic performance. Due to the entropy in the objective of SAC, we believe some strategies and parameters may perform differently than in these previous studies. Therefore, in this work we explore hyperparameter settings and strategies for experience replay in SAC on 8 environments, which include complex robotic manipulation environments and understudied areas such as discrete action spaces. We cover four of the most popular experience replay strategies - vanilla experience replay (ER), prioritized experience replay (PER), hindsight experience replay (HER), and a combination of priority and hindsight experience replay (PHER), and analyze the common hyperparameter settings of batch sizes ranging from 64 to 512 and buffer sizes ranging from 50,000 to 1,000,000. We compare the number of updates needed to solve environments, the maximum average return, and the maximum success rate achieved by our strategies and settings. Our results indicate the best performing hyperparameters are larger batches of 512 and smaller buffers of 50,000. For strategies, adding priority (PER/PHER) does not perform significantly better in any environment while HER is successful in goal environments where sub-goals are easily achieved.**

I. Introduction

Deep reinforcement learning has seen an explosion of interest in recent years thanks to its success in games such as Atari (Mnih et al. [13]) and Go (Silver et al. [19]). These successes have largely been driven by model free methods. One particular model free deep reinforcement learning algorithm which has excelled in robotics applications is Soft Actor-Critic (SAC) (Haarnoja et al. [8]; Haarnoja et al. [9]), which has seen success in a number of continuous domains, including the MuJoCo OpenAi Gym task suite. This success is largely driven by the sample efficiency of SAC, as it is an off policy algorithm, and the entropy term, which encourages exploration and prevents convergence on sub-optimal policies.

Despite the success of SAC, there are several limitations to the algorithm in practical robotic applications. First, discrete action spaces, which are nontrivial due to high variance and instability, often arise in robotics in the form of choices of control modes, gear switching or digital outputs (Neunert et al. [15]), and cannot be handled directly by the SAC algorithm proposed in the original work. While there are existing efforts to apply SAC to discrete domains such as Christodoulou [3], little work has been done studying SAC under discrete action settings outside of the Atari game suite.

Further, despite the improved sample efficiency over other algorithms, SAC is still quite costly in deployment on robotics and can fail to converge to solutions, especially in domains with sparse rewards. To address this sample efficiency, many modifications to experience replay have been proposed, such as Prioritized Experience Replay (PER). This strategy, introduced in Schaul et al. [18], uses TD-error to focus learning on salient experiences. Another wildly popular alternative uses an algorithm called Hindsight Experience Replay (HER) (Andrychowicz et al. [1]) to relabel and learn from trajectories which otherwise provide no useful learning experience due to sparse rewards.

There has been some work such as Wan and Xu [21] which investigates the comparative efficacy of these experience replay approaches; however, little work has investigated robotic manipulation tasks, and to our knowledge no work comparing experience replay methods has been applied to SAC. SAC differs from prior algorithms in its maximum entropy formulation; consequently, optimal replay strategies and hyperparameters may differ from prior studies and could provide immediate gains to algorithmic performance.

Hence, in this work we contribute 1) Analysis on experience replay under the widest variety of environments to date. Our study also includes novel goal formulations for classic control environments of Mountain Car and two Robosuite environments. 2) Comparison of the performance for four of the most common experience replay strategies - vanilla experience replay (ER), prioritized experience replay (PER), hindsight experience replay (HER), and a combination of priority and hindsight experience replay (PHER) in SAC and maximum entropy reinforcement learning. We observe priority methods (PER/PHER) are hindered by entropy, not performing significantly better in any environment, while HER is only successful in the sparse environments where a variety of goals are easy to obtain. 3) Comparison of the learning speed, maximum return, and maximum success rate of hyperparameters for experience replay, with batch sizes ranging from 64 to 512 and buffer sizes ranging from 50,000 to 1,000,000. Our results demonstrate that, counter to common literature settings, the best performing hyperparameters are larger batches of 512 and smaller buffers of 50,000. While our results offer immediate recommendations for robotic applications, they also suggest more research into intrinsic motivation, exploration, and prioritization are required to improve model-free reinforcement learning algorithms.

II. Literature Review

A. Deep Reinforcement Learning and Soft Actor-Critic

Model free deep reinforcement learning has seen success in a number of ground breaking applications including Atari (Mnih et al. [13]), Go (Silver et al. [19]), and Traffic Control (Gao et al. [7]). In robotics, SAC (Haarnoja et al. [9]) has emerged as a state of the art algorithm due to its effectiveness on the Mujoco test suite and real robotics; hence, in this work, we propose to study this algorithm. However, SAC cannot be applied to discrete settings as a result of the policy update rule. To address this shortcoming, there are two approaches: alter the policy update rules, as in Christodoulou [3], or modify the discrete probability distribution from a categorical to a gumbel softmax (Jang et al. [10]; Maddison et al. [12]), which is the approach we use due to the compatibility with the continuous setting's update rules.

While quite efficient compared to other model free approaches, SAC is still sample inefficient compared to model based approaches. There have been a number of attempts to improve the efficiency of SAC (Raffin and Stulp [17];Wang and Ross [22]). Ours falls under the class of improving experience replay instead of the network itself. Unfortunately, we do not examine experience replay approaches directly engineered for SAC to enable comparison across other surveys and due to time constraints.

B. Experience Replay

Since its introduction in literature, experience replay (Lin [11]) has been an important aspect of model-free learning algorithms, as explained in Mnih et al. [14]. While experience replay enables models to learn from prior experiences, it is incredibly naive and samples experience randomly. This led to an insight in Schaul et al. [18] which introduced prioritized experience replay (PER). The key insight was sampling should prioritize experiences the model has the highest error on to correct for the most egregious states. Another work, hindsight experience replay (HER) (Andrychowicz et al. [1]) observed prior experiences which result in no information about the goal could be re-framed to provide information about the sub-goal that was achieved instead. There are a number of other experience replay modifications and expansions such as experience replay optimization (ERO) (Zha et al. [23]), combined experience replay (CER) (Zhang and Sutton [24]), and many others (Foerster et al. [6]; Tampuu et al. [20]; Pellegrini et al. [16]), but HER and PER are by far the most popular in literature; consequently we select them for study.

C. Comparative works

Due to the explosion of experience replay strategies, comparative work is vital. Zhang and Sutton [24] explored how experience replay is affected by hyperparameter settings related to buffer size. Other works compares strategies directly such as Wan and Xu [21] which compares learning speed of CER, PER, and HER on DDPG and DQN. Fedus et al. [5] elaborate by discussing variability in experience replay over different reinforcement learning approaches, and focus there efforts on analyzing Rainbow. However, no such work to our knowledge compares as wide of a variety of strategies, settings, and environments with experience replay, or on stochastic algorithms, making our work novel in this regard.

III. Environments and Data

In this section, we discuss the data and environments which we use, listed in Table I. Environments are from OpenAI Gym (Brockman et al. [2]) with the exception of the Panda environments, which are from Robosuite (Zhu et al. [25]). Thus, all robotic environments run on the Mujoco simulator. We offer more discussion behind the use of different simulated environments in Section III-C. We note for results in Section V-A, we were only able to run one trail per strategy and hyperparameter setting due to resource constraints. For results in Section V-B, we ran three trails per strategy and environment, reporting the mean and standard deviation in graphs. We selected learning steps based on how many episodes were required to reach convergence or until learning stalled for 100,000 steps. For the remainder of this section, we provide a brief introduction of each environment we use, as well as any special formulations[1] and why the environments were included in this study.

A. Classic Control

We begin with 3 classic control environments to investigate simple relationships and validate our models. We also investigate the performance differences in discrete and continuous action SAC in this setting, as to our knowledge there are no widely available discrete action robotic environment available.

1) CartPole-v1: CartPole-v1 is a task of balancing a pole on a cart, with a reward of 1 for each step the pole is balanced. Once the cart or pole deviates from a specified range too much, the environment is considered failed and the episode ends. Since there is no target state at any given timestep, we do not formulate a goal version. This environment is the simplest of our environments, and serves to demonstrate that SAC can solve discrete environments with the Gubmel-Softmax. This environment has also been used in other comparative works (Wan and Xu [21]).

2) MountainCar-v0: MountainCar-v0 is a task of moving a car from the bottom of two hills to a flag at the top of one of the hills. This environment is incredibly difficult, as the agent must act suboptimally in order to build momentum and reach the flag position. A reward of -1 is given for each timestep the goal is not reached in this one dimensional setup, and 0 when the flag is reached.

We modify this environment for HER by creating a "goal" version. This goal version has a single goal, G where G[0] is the flag position and G[1] is the goal velocity for which the car's velocity must be greater than or equal to for success. G is set following the values in the OpenAI Gym environment. We assign a reward of -1 for each step the environment is not solved, and 500 when the environment is solved successfully

[1]Novel goal environments available at https://github.com/Lucaskabela/robot-learning-replay

	# Observations	# Controls	Control Type	Solved Score	Horizon	Learning Steps	Goal Compatible	Goal Dim
CartPole-v1	4	2	Discrete	475	500	100,000	No	–
MountainCar-v0	2	3	Discrete	-110	200	200,000	Yes	2
MountainCarContinuous-v0	2	1	Continuous	90	999	200,000	Yes	2
HalfCheetah-v2	17	6	Continuous	4800	1000	1,000,000	No	–
FetchPush-v1	25	4	Continuous	–	50	1,000,000	Yes	3
FetchPickAndPlace-v1	25	4	Continuous	–	50	1,000,000	Yes	3
PandaPickAndPlaceCan	46	7	Continuous	–	50	250,000	Yes	3
PandaDoor	46	7	Continuous	–	500	250,000	Yes	1

TABLE I: Environment Details

- this value is much greater than 0 to encourage the agent to remember the optimal trajectory.

3) MountainCarContinuous-v0: MountainCarContinuous-v0 is the continuous analog of MountainCar-v0; however, actions are the amount of energy to apply in a given direction and thus the reward in this environment is $-|a|^2$ or 100 for solving the environment by default.

We again formulate a "goal" version, which is identical to the discrete version mentioned above. We use a slightly smaller reward scale of 200 for success in the continuous setting, since the time horizon is 999 steps instead of 200.

We include the mountain car environments to compare discrete and continuous action spaces, as this is one of the only environments in OpenAI gym which has a discrete and continuous analog.

B. Mujoco (Locomotion)

1) HalfCheetah-v2: HalfCheetah-v2 is our first environment using Mujoco, and is tasked with learning a locomotion policy for the HalfCheetah. The reward is once more related to the energy expended for locomotion. We include this environment as it has been heavily studied in the SAC literature, but we do not formulate a goal version, as there is not a target state the robot needs to achieve to learn a policy for walking, which is similar to the constraints preventing formulation in CartPole-v1.

C. Mujoco Robotic Manipulation

Due to the lack of robotics environments in comparative experience replay literature, we also evaluate on 4 robotic manipulation environments. We begin with two environments from OpenAi Gym which have been heavily studied in HER literature and use the Fetch robot arm. We further include two more rigorous formulations using the Panda robot arm from Robosuite to stress test our algorithms.

1) FetchPush-v1: FetchPush-v1 requires the Fetch arm to push a puck to a designated point on a table, which is a 3D point and serves as the goal. There is a reward of -1 for each timestep the goal is not reached, and 0 once it has been. This environment is more difficult than the naive FetchReach-v1, so allows for comparison between techniques, but is still easy enough to learn policies for naive ER. Hence, this environment serves as our robotic manipulation baseline.

2) FetchPickAndPlace-v1: FetchPickAndPlace-v1 requires the Fetch arm to pick up a block and move it to a designated point in 3D space, which serves as the goal. The environment rewards -1 for each timestep the goal is not reached, and 0 when it is reached. We choose this environment to have an analog to our PandaPickAndPlaceCan environment in Robosuite and compare across robots and simulators.

3) PandaPickAndPlaceCan: PandaPickAndPlaceCan is adapted from PickPlaceCan in Robosuite, and requires the panda robot arm to grasp a can and drop it into a designated bin. Thus it is not directly comparable to FetchPickAndPlace-v1; however, similar skills must be learned to solve the task. We use the controller, environment settings, and number of learning updates from the benchmarking suite. We modify the Robosuite gym wrapper by adding a goal, which is the 3D coordinates of the box which the robot must drop the can into. To better match FetchPickAndPlace, we limit the time horizon to 50 steps, and use the sparse rewards from Robosuite.

4) PandaDoor: PandaDoor uses the Door environment from Robosuite, and requires the panda robot arm to open a randomly initialized door. We use the dense and sparse rewards specified by Robosuite to compare performance under the density of rewards. Unlike the other manipulation environments, we use a 500 timestep horizon. The environment settings and number of learning updates are consistent with Robosuite benchmarking. For our goal formulation, we only specify the hinge degree of the door, as this is how the reward was computed in Robosuite. Note, as we observe in Section V-B, this goal does not take into account the position of the door and is thus a naive representation.

IV. Architecture and Experience Replay

A. Model Architecture: SAC

1) Continuous Settings: Our implementation of SAC follows Haarnoja et al. [9]. We chose to study this algorithm as opposed to other maximum entropy algorithms due to its strong results in robotic applications and its popularity. We re-implemented the algorithm from the paper, and compared to the implementation from RLKit [2] noting there was not a significant difference in results. We thus used RLKit's implementation to provide a standard implementation for future works. We further hold the hyperparameters of the network constant across environments, as reported in the supplementary materials. These hyperparameters were selected due to the frequency with which we observed them in related works and to allow generalization to the variety in difficulty of our

[2]https://github.com/vitchyr/rlkit

environments . For background on the reinforcement learning setup of SAC, see Haarnoja et al. [9] and supplementary materials.

2) Discrete Settings: For SAC in discrete action spaces, we implement the Gumbel Softmax as opposed to the policy update modifications in Christodoulou [3]. This is done to keep the learning algorithms similar between discrete and continuous settings, as only the distribution sampled from needs to change. Following the approach in Jang et al. [10] and Maddison et al. [12], we extend the RelaxedOneHotCategorical distribution in Pytorch. We reproduce the formulation here for completeness:

Suppose we have action probabilities: $\pi_1, \pi_2, ..., \pi_k$. We can then draw samples z:

$$z = \text{one\_hot}(\text{arg\_max}_i[g_i + \log(\pi_i)]) \tag{1}$$

where $g_1, ..., g_k$ are i.i.d samples drawn from Gumbel(0, 1). To provide a differentiable, continuous approximation, we use softmax in place of argmax and generate k sample vectors, y where

$$y_i = \frac{\exp((g_i + \log(\pi_i))/\tau)}{\sum_{j=1}^{k} \exp((g_j + \log(\pi_j))/\tau} \tag{2}$$

where τ is a temperature parameter, chosen to control the flatness of the gumbel softmax. Putting this all together, the density function is

$$\mathrm{p}_{\pi,\tau}(y_1, ..., y_k) = \Gamma(k)\tau^{k-1}(\sum_{i=1}^{k} \pi_i / y_i^{\tau})^{-k} \Pi_{i=1}^{k}(\pi_i / y_i^{\tau+1}) \tag{3}$$

B. Experience Replay

For our strategies, we use four of the most common experience replay strategies. While we acknowledge there are a wealth of strategies, some specially engineered for the maximum entropy frameworks such as Wang and Ross [22], we do not have the resources to exhaustively test such strategies. We further use common hyperparameter values from literature, as we do not have resources to perform tuning and such values have been shown to perform well.

1) Experience Replay: Experience replay (ER) serves as a baseline and samples uniformly from all experience, (s, a, s', r, d) in the buffer, D. This approach is naive in that it treats all experience equally, but it still provides strong results in many environments.

2) Prioritized Experience Replay: Our implementation of PER follows from Schaul et al. [18], and is based off OpenAI baselines (Dhariwal et al. [4]) with minor modifications. Suppose the buffer has N elements. Using a segment tree to store our priorities all operations are $O(log(N))$, which is the best complexity found for this approach. Under SAC, the TD-Error of a sample, $|\delta|$, is defined as the average TD-Error per Q network, or

$$|\delta| = \frac{1}{2} \sum_{t=1}^{2} |r + \gamma V_{\phi_{targ}}(s') - Q_{\theta,t}(s, a)| \tag{4}$$

When initially added to a buffer, we assign the priority, p_i, for experience i to be the maximum priority currently in the buffer. After an update step on the other hand, p_i is assigned to $|\delta|$. The probability for which experience i is sampled is then

$$P(i) = \frac{p_i^{\alpha}}{\sum_{j=0}^{N} p_j^{\alpha}} \tag{5}$$

However, this leads to bias which needs to be corrected with importance sampling. The correction term we use for sample i is:

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^{\beta} \tag{6}$$

Prior work such as Wan and Xu [21] has found PER to outperform ER in most settings. However, PER has to our knowledge primarily been applied to deterministic algorithms, and even then does not have strong results. As such, we theorize that PER will not provide much improvements over ER, but may still offer some advantages to learning, especially in discrete settings where the strategy was initially studied.

3) Hindsight Experience Replay: Our implementation of HER follows closely from Andrychowicz et al. [1], and is based off the implementation in RLKit. A hard requirement of HER is that environments be goal environments. Goal environments, in addition to the observation, return an achieved goal and desired goal as part of the state. Thus, an episode can be captured as

$$(S_0, G, a_0, r_0, S_1), ..., (S_n, G, a_n, r_n, S') \tag{7}$$

Where G is the desired goal, and S' is the achieved goal. Under our implementation of HER, with probability $1/k$ we replace the desired goal, G, with the achieved goal of a trajectory, S' when sampling from D. Thus, the trajectory will be relabeled, with probability $1/k$ and where $c()$ is function to compute rewards from a desired and achieved goal:

$$(S_0, S', a_0, c(S', S_1), S_1), ..., (S_n, S', a_n, c(S', S'), S') \tag{8}$$

In this way, HER serves as a form of implicit curriculum learning. HER is included as it has shown to achieve remarkable success in sparse reward robotics environments, but has mostly been studied under deterministic networks in well formulated environments. In this study, we stretch the limits of HER by experimenting with a stochastic algorithm, SAC, and with the ability of the environment to make partial progress through achieved goals.

4) Prioritized Hindsight Experience Replay: This approach combines the previous two strategies by assigning priority to trajectories using the sum of TD-Errors. We include this strategy to evaluate how well combinations of the most popular strategies can do.

V. Experimental Results

We now introduce our experimental results, starting with our hyperparameter ablation study, as it motivates the settings of hyperparameters for our main experiments. Average values are reported over 100 steps unless noted otherwise.

A. Replay Buffer Settings

To begin, we evaluate the common hyperparameters, which are buffer size and batch size, on representative environments from each category of classic, locomotion, and manipulation environments. For environments where it is computationally feasible, we vary buffer size on an exponential scale from 2^6 to 2^9, and buffer size over {50,000, 100,00, 500,000, 1,000,000}. For environments which took more than 4 hours to run (HalfCheetah and FetchPush-v1), we only evaluate the extreme values of these ranges. The limited number of environments and settings is an unfortunate restriction of our computational resources.

Here we provide the results on FetchPush-v1, as this environment was complex enough to offer differences in settings and justifies our hyperparameter values for robotic manipulation environments. More results are provided in the supplementary material.

	64	512
50,000	-39.5‖-39.8‖-21.4‖-22.9	**-38.2‖-38.8‖-17.3‖-18.3**
1,000,000	-40.6‖-41.3‖-24.1‖-23.4	-39.9‖-39.1‖-18.0‖-18.5

(a) Max average return

	64	512
50,000	.13‖.12‖.57‖.55	**.14‖.13‖.69‖.67**
1,000,000	.11‖.11‖.52‖.53	.13‖.13‖.67‖.66

(b) Max average success rate

	64	512
50,000	369‖**216**‖256‖198	**366**‖281‖**152**‖**124**
1,000,000	372‖259‖325‖223	428‖315‖277‖235

(c) Updates (in thousands) to solve >.1 average success rate

TABLE II: Buffer vs batch sizes on FetchPush-v1 over 1 million learning updates [ER‖PER‖HER‖PHER]

We observe the best results for most settings occurs with a batch size of 512 and a buffer size of 50,000. Under larger batch sizes, variance is reduced, leading to more stable learning, while computation is only marginally slower. On the other hand, the small buffer size keeps the experiences sampled relevant to the current policy, as they are more recent, thus more likely to be encountered. Even with a smaller size of 50,000, the buffer can hold 50 episodes or more for our longest horizon environment. Of all our environments, CartPole-v1 was the only setting we found this trend did not hold, with no discernible values performing best. We believe this is a result of the simplicity of the environment and thus influence of noise in results as there was not significant differences in the performance of any strategy.

B. Results on Robotic Environments

In this section, we visualize the mean and standard deviation of three trails of our best performing hyperparameters on each strategy.

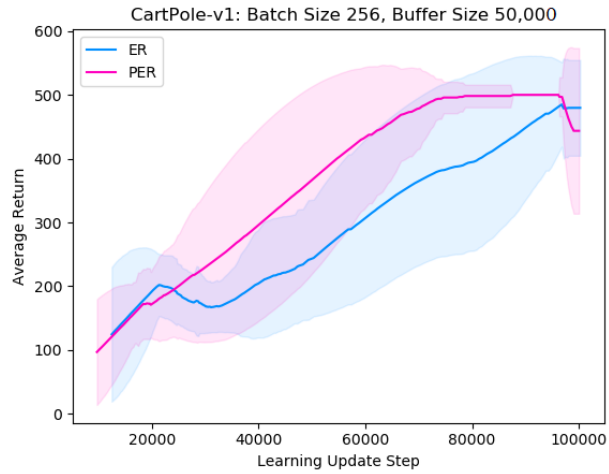
1) Non-Goal Environments: Beginning with the non-goal environments, in Figure 1, we notice PER initially learns faster than ER, as predicted. However, counter to our expectations, as we increased the number of learning steps, PER begins to perform worse than ER. We believe this to be a result of the interaction of priority and entropy - the exploration encouraged by entropy adds many states to the buffer which do not maximize rewards, especially compared to deterministic algorithms. Therefore, priority based on TD-Error may over-incentivize updating sub-optimal states. These results suggest for maximum entropy reinforcement learning algorithms such as SAC, priority is too costly - adding $O(log(N))$ complexity - for little performance gain.

2) Discrete vs. Continuous: MountainCar: Next, we compare the performance on MountainCar environments in Figure 2. Interestingly, the success rate skyrockets initially but plateaus to 0 after a moderate amount of learning. Examining our sampled goals, we discovered the velocity term, G[1] was the cause of poor learning, as the cart needed to have velocity greater than this term to be considered successful under the `compute_reward` (*c()*) function used for learning. We recommend future versions of this goal environment drop the velocity term, as it is 0 in the non-goal Gym environment. Our results also show continuous settings have a sharper peak, or better performance than discrete action spaces, which we attribute to the gumbel-softmax reshaping the categorical through approximation. Thus, while SAC is shown to perform better in continuous environments, we believe this comparison demonstrates SAC is capable of solving non-trivial discrete action spaces.

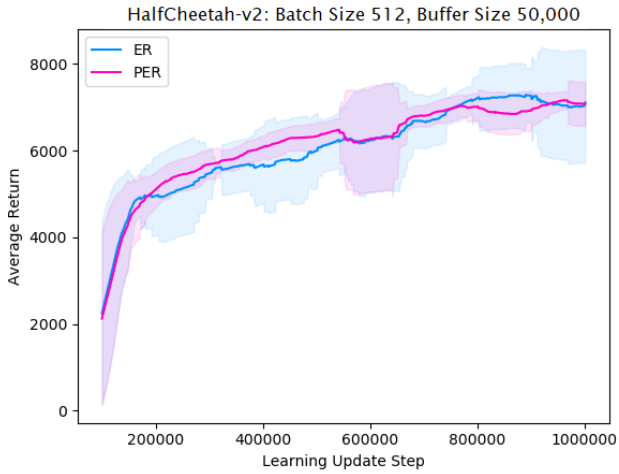
3) Robotic Manipulation Environments: We finally investigated our robotic manipulation settings in Figure 3. Beginning with FetchPush-v1, we found that ER and PER were unable to make much progress on the environment, but due to its simplicity, they were able to solve about 10% of episodes. Adding hindsight rocketed success rates to near 60%, indicating the strength of the method in solving simple goal environments. FetchPickAndPlace-v1 had similar results, with a slightly lower success rate as the environment is harder.

On the other hand, PandaPickAndPlaceCan had much worse performance, and none of our strategies were able the solve the environment. After examining the learned policy, it was apparent that the signal was too sparse for learning - the hand never was able to learn the sequence of picking up the object and dropping it, and it frequently stalled before the gripper picked up the can. This resulted in no goals other than the zero state achieved, leading to relabeling being ineffective which reduced HER to ER.

PandaDoor with sparse rewards struggled with similar issues. In the sparse setting, no learning took place, as the door hinge was never opened by any amount, despite the gripper sometimes making contact with the handle as opening the door involved a complex sequence of finding the handle, torque, and forward force. However, the dense rewards, which captured this sequence, were able to lead to strong results for all methods. HER and PHER were actually out performed by ER and PER in this dense setting, which we believe to be a result of the dense rewards being designed for the environment, and thus a better source of learning than relabeling.

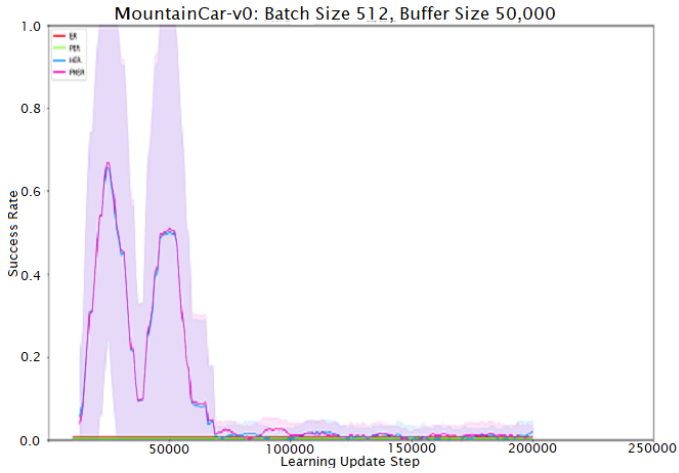


(a) Performance of CartPole-v1 over 100,000 learning steps

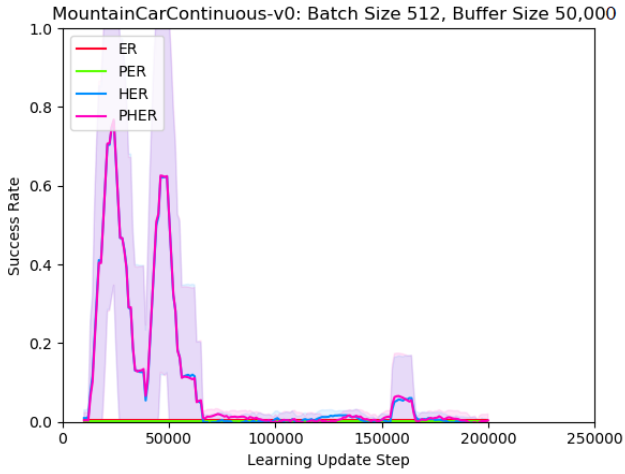


(b) Performance of HalfCheetah-v2 over 1,000,000 learning steps

Fig. 1: Evaluating Average Returns of ER, PER on environments without goals



(a) Performance of MountainCar-v0 (+ Goal) over 200,000 learning steps



(b) Performance of MountainCarContinuous-v0 (+ Goal) over 200,000 learning steps

Fig. 2: Evaluating Average Success Rate of ER, PER, HER, and PHER strategies on MountainCar environments. ER/PER achieved a success rate of 0 regardless of which formulation was used.

VI. Discussion and Limitations

A. Comparison to prior surveys

In Table III we provide the results of Wan and Xu [21], which surveyed replay strategies on DQN[3]. The results indicate SAC-D is much better at solving discrete action environments, as we are able to solve CartPole-v1 in around 60,000 learning steps or 200 episodes[4] and reached good performance on MountainCar-v0 in around 250 episodes, although our policy did not converge. Also worth noting is our results are somewhat echoed here - HER learns much more efficiently than ER in goal environments, while PER does not outperform ER significantly. Our work however differs by investigating more complex environments, continuous spaces, and SAC.

Strategy	CartPole-v0	MountainCar-v0
ER	4000	33000
PER	4000	–
HER	–	15500
PHER	–	–

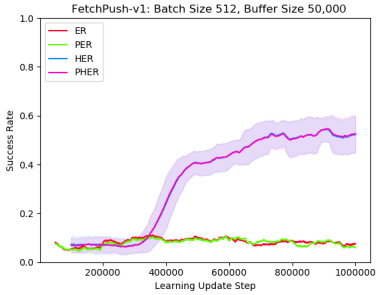
TABLE III: Episodes to Converge from Wan and Xu [21]

B. Computation Resources

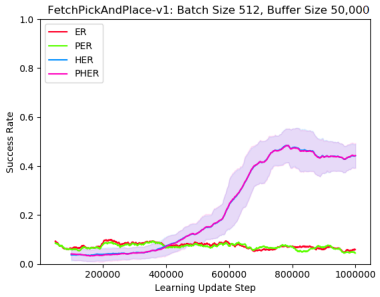
In terms of computation, we were limited to running on 2 CPUs due to Mujoco keys. This limited us to one run for

[3]We do not report learning speed in episodes as episodes are variable lengths in environments such as CartPole

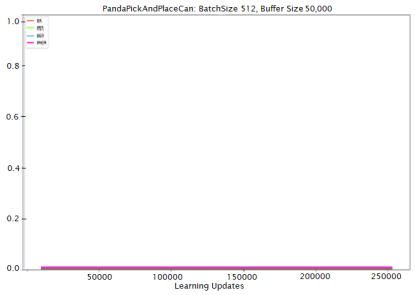
[4]From dividing number of learning steps by maximum episode length



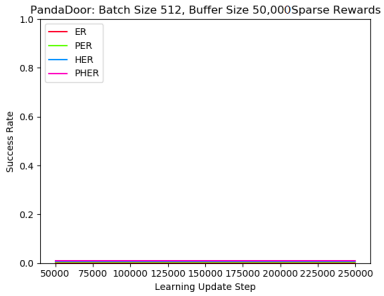
(a) Average Success Rate of FetchPush-v1 over 1,000,000 learning steps



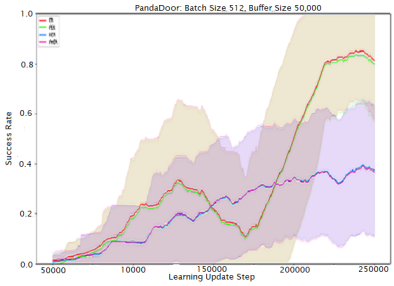
(b) Average Success Rate of FetchPickAndPlace-v1 over 1,000,000 learning steps



(c) Average Success Rate of PandaPickAndPlaceCan over 250,000 learning steps



(d) Average Success Rate of PandaDoor(Sparse) over 250,000 learning steps



(e) Average Success Rate of PandaDoor(Dense) over 250,000 learning steps

Fig. 3: Evaluating Average Success Rates of ER, PER, HER, and PHER on goal environments

each trail of our hyperparameter study, so noise could have contributed to our results. Another issue is our number of learning steps were limited from resources, which means we did not perform extremely long trails. Prior works (Zhang and Sutton [24]) have indicated the performance of smaller buffer sizes falls off as the number of learning update steps increases, but we did not observe this for the number of updates we investigated.

Further, we did not experiment with SAC specific strategies, which is a significant shortcoming as such strategies are likely to achieve higher success rates. We finally did not perform hyperparameter tuning, instead opting to select static parameters which were common in literature. Future work should take care to first search for optimal hyperparameters for the network and strategies, then proceed with investigating performance and ablating over select parameters.

C. Failure modes and Future Work

We believe that PER performed poorly with SAC due to the entropy objective, as SAC takes many sub-optimal actions due to entropy with potentially large TD-Errors. This could partially explain the slowed learning of PER as steps increased, as high error states affected convergence. We also noted several environments HER did not learn in. Error analysis revealed in such cases HER never reached a goal other than the zero state, reducing HER to ER and indicating a need for better goal formulation. Finally, our tuning did not drastically positively affect runtime or learning. This, in conjunction with our two failure modes, lead us to a recommendation for stronger forms of prioritization, curriculum, and intrinsic exploration, as it is evident in some environments, weakly formulated goals and TD-Error priority may not provide enough signal to improve learning or cause convergence.

VII. Conclusion

Inspired by research in deterministic settings such as DQN and DDPG which has demonstrated that varying buffer hyperparameters and strategies improves learning, we perform a similar survey with SAC in 8 environments. Our findings show larger batch sizes of 512 and smaller buffer sizes of 50,000 perform best in learning speed and maximum return. We then ran our best settings with each strategy in our environments and observe that continuous settings with dense rewards are easiest to learn, while environments with difficult to achieve goals and sparse rewards can lead to a failure to learn, even for HER. Further, we find priority is ineffective in SAC as a whole. We hope future research can use our strategies and parameters recommended here, and that this work motivates greater exploration into prioritization, implicit curriculum learning, and intrinsic motivation to drastically improve experience replay.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017. URL http://arxiv.org/abs/1707.01495.

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

[3] Petros Christodoulou. Soft actor-critic for discrete action settings, 2019.

[4] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[5] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay, 2020.

[6] Jakob N. Foerster, Nantas Nardelli, Gregory Farquhar, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *CoRR*, abs/1702.08887, 2017. URL http://arxiv.org/abs/1702.08887.

[7] Juntao Gao, Yulong Shen, Jia Liu, Minoru Ito, and Norio Shiratori. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. *CoRR*, abs/1705.02755, 2017. URL http://arxiv.org/abs/1705.02755.

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

[9] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL http://arxiv.org/abs/1812.05905.

[10] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017.

[11] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8(3–4):293–321, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992699. URL https://doi.org/10.1007/BF00992699.

[12] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712, 2016. URL http://arxiv.org/abs/1611.00712.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529—533, February 2015. ISSN 0028-0836. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

[15] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Jost Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics, 2020.

[16] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning, 2020.

[17] Antonin Raffin and Freek Stulp. Generalized state-dependent exploration for deep reinforcement learning in robotics, 2020.

[18] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

[19] D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[20] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4): e0172395, 2017.

[21] Tracy Wan and Neil Xu. Advances in experience replay, 2018.

[22] Che Wang and Keith Ross. Boosting soft actor-critic: Emphasizing recent experience without forgetting the past. *CoRR*, abs/1906.04009, 2019. URL http://arxiv.org/abs/1906.04009.

[23] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. Experience replay optimization. *CoRR*, abs/1906.08387, 2019. URL http://arxiv.org/abs/1906.08387.

[24] Shangtong Zhang and Richard S. Sutton. A deeper look at experience replay. *CoRR*, abs/1712.01275, 2017. URL http://arxiv.org/abs/1712.01275.

[25] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.