Project: Final Report - Group 10
CS461: Artifical Intelligence

Uğur Can Altun
Bilkent University
Computer Engineering
Ankara, Turkey
can.altun@ug.bilkent.edu.tr

Kaan Berk Kabadayı
Bilkent University
Computer Engineering
Ankara, Turkey
berk.kabadayi@ug.bilkent.edu.tr

Utku Boran Torun
Bilkent University
Computer Engineering
Ankara, Turkey
borun.torun@ug.bilkent.edu.tr

Mehmet Onur Uysal
Bilkent University
Computer Engineering
Ankara, Turkey
onur.uysal@ug.bilkent.edu.tr

Barış Tan Ünal
Bilkent University
Computer Engineering
Ankara, Turkey
tan.unal@ug.bilkent.edu.tr

***Abstract*—Robotic manipulation using Reinforcement Learning (RL) has been a complicated task for robotic agents to successfully complete, the biggest challenge being due to the environment having rewards that are sparse and binary. In this paper, we present our research comparing two well-known algorithms Soft Actor-Critic (SAC) and Twin Delayed Deep Deterministic Policy Gradient (TD3) and how well they learn in a continuous reward setting pusher-v4 environment and test how the algorithms perform when we change some of their hyperparameters. Furthermore, we show our findings when we implemented the Hindsight Experience Replay (HER) on both algorithms and discuss the results we obtained when compared to the default algorithms. We delve deeper into discussing the applicability of HER in a continuous reward setting and compare the learning accuracy we obtained by integrating the method to the algorithms. We show that integrating HER into the algorithms decrease the performance rather than increasing it when experimented on 100k timesteps.**

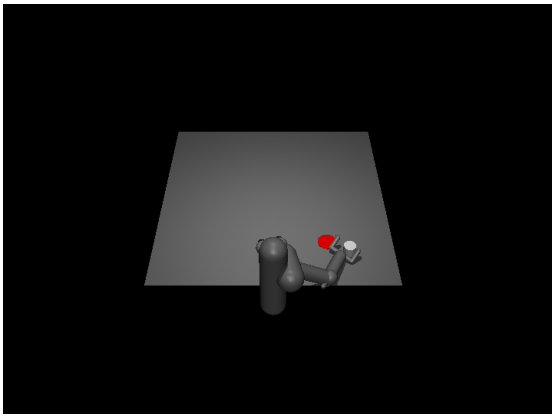

Fig. 1. Gymnasium Pusher-v4 Environment.

I. Introduction

In the realm of robotic learning, the evolution of intelligent agents navigating through environments and interacting with objects has been an interesting research area [1], [2]. Our experiment delves into the realm of robotic manipulation, specifically focusing on a robotic learning scenario within the OpenAI Gymnasium using the ”pusher-v4” environment [3]. The fundamental challenge posed in this experiment lies in training an AI agent embodied as a robot, entrusted with the singular task of pushing a cylindrical object to a goal location.

To analyze the optimal learning approach for this constrained robotic scenario, our experiment focuses on experimenting on two distinct reinforcement learning algorithms: Twin Delayed Deep Deterministic Policy Gradients (TD3) and Soft Actor-Critic (SAC). The primary objective of this research is to conduct an in-depth comparative analysis of these algorithms. By assessing their performance metrics in terms of accuracy and average returns, we aim to find the algorithm out of the two that demonstrates superior adaptability and learning efficacy within the “pusher-v4” environment [3]. Furthermore, we aim to analyze the applicability of the algorithm/method Hindsight Experience Replay (HER) which is a technique used to drastically increase the learning performance in environments where the agent faces difficulty learn due to sparse and binary rewards to our environment which has a well-defined continuous reward system. Note that we took the implementations of the algorithms from the internet and reused the codes to make some modifications

This Final Report will contain a more in-depth information on the simulated environment, details on the used algorithms TD3 and SAC including the intuition and the implementation details, introduction on the HER method and how it can be applied to our codebase, the methodology on how we conducted the experiments, the results and discussion on the data we have gathered by simulating the agent many times on the same environment with different modifications on the both algorithms and the implications we could infer from the data we have gathered.

II. Environment

The environment we selected, pusher-v4, is an environment within the MuJoCo environments framework. The environment can be seen in Fig 1. It presents an environment where a robotic entity and its freely moving mechanical arm tries to push a white, cylindrical object to a goal location marked red. The robot has an action space of 7 elements, each denoting movements like joint rotations and limb adjustments for its mechanical arm, requiring a nuanced policy to successfully push the object to the goal location. The observation space consist of 23 elements, each capturing a distinct feature of the game's state: i.e the rotations of the robot's limbs such as its shoulder, wrist, elbow, the rotational velocities of its limbs, the coordinates of its fingertips, the coordinates of the object etc. The reward system works as a linear combination of three reward values: the distance between its fingertip and the object, the distance between the object and the goal destination and a negative reward for penalizing the agent when it takes large actions. Lastly, each episode on the environment ends with either through truncation, when timestep reaches a value of 100 or through termination when the agent successfully pushes the object to the destination or drops the object. The reward is calculated as the following:

$R = \text{reward\_dist} + 0.1 \times \text{reward\_ctrl} + 0.5 \times \text{reward\_near}$ where

reward_dist: "object" - "target"
reward_ctrl: negative reward associated with large actions
reward_near: "fingertip" - "target"

Please note that reward system in this environment is not binary and sparse and actually really well designed and crafted unlike some similar other robotic pushing environments where the agent only gains a reward when it successfully pushes the object to the desired location. However, in the pusher-v4 environment, the agent learns exactly how successful it was doing the task and can learn to improve itself with the above rewarding scheme. If the reward was sparse, the agent would very rarely encounter a reward i.e. successfully push the object to the target position. Hence, would rarely get valuable information whether it did well or not. Furthermore, if the reward was binary, the agent would only learn whether it was successful or not at each episode and it would again be not very valuable in the learning process. But, in this environment, the reward system is neither binary nor sparse, which enables learning if reward engineering has been done well.

III. Related Work

A. TD3 Algorithm

Twin Delayed DDPG (TD3) is an algorithm that is built upon Deep Deterministic Policy Gradient (DDPG) [6]. DDPG is an off-policy algorithm that learns a policy and a Q-function at the same time. It utilizes the Bellman equation and off-policy data to learn the Q-function. Then, it uses the Q-function to learn the policy [4]. It can be assumed as being deep Q-learning for continuous spaces.

DDPG's approach is closely related to Q-learning and it has the same motivation as Q-learning. DDPG's motivation is that in any given state, the optimal action $a^*(s)$ can be found by swallowing the following equation if the optimal action-value function $Q^*(s,a)$ is known:

$$a^*(s) = \arg\max_a Q^*(s,a). \tag{1}$$

DDPG combines the training of an estimator for $Q^*(s,a)$ with that of an estimator for $a^*(s)$, and this approach is adjusted in a way for environments with continuous action spaces [5].

While there is no problem to compute the max over actions when there are a finite number of discrete actions, the cost of computing the max over actions when the action space is continuous is expensive. As the action space is continuous, the function $Q^*(s,a)$ can be assumed to be differentiable with respect to the action argument. This enables to set up an efficient, gradient-based learning rule for a policy $\mu(s)$ that utilizes this information. Therefore, we can approximate the $\max_a Q(s,a)$ with $\max_a Q(s,a) \approx \max_a Q(s,\mu(s))$ instead of running an expensive optimization subroutine [5].

DDPG utilizes Bellman equation to calculate the optimal action-value function $Q^*(s,a)$:

$$Q^*(s,a) = \underset{s' \sim P}{E}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right] \tag{2}$$

The Bellman equation is the starting point for learning an approximator to $Q^*(s,a)$. DDPG also relies on replay buffers and target networks for Q-learning [4].

While DDPG can sometimes achieve great results, tuning its hyperparameters and other kinds of parameters can be frequently brittle [6]. One of the problems of DDPG is that the learned Q-function starts to significantly overestimate Q-values, which results in the policy breaking. Twin Delayed DDPG, TD3, is an algorithm that addresses this issue by introducing three important tricks:

1) **Clipped Double-Q Learning:** TD3 learns two Q-functions, $Q_\Phi 1$ and $Q_\Phi 2$, instead of one Q-function (that's why it is called "twin") and the smaller of the two Q-functions is selected to form the Bellman error loss functions' targets [6].
2) **Delayed Policy Updates:** TD3 updates the policy and the target networks less frequently than the Q-function. Fujimoto et al. [6] states that one policy update should occur for every two Q-function updates.
3) **Target Policy Smoothing:** TD3 deliberately adds noise to the target action, so the policy has a harder time exploiting function errors as Q is getting smoothed out along the changes in action. It addresses a specific failure scenario that can happen in DDPG which is the scenario where the Q-function approximator calculates an incorrect sharp peak for some actions and the policy will exploit that peak and result in policy having incorrect behavior [6].

These three tricks together result in significantly improved performance over baseline DDPG [6]. It should be noted that

TD3, just like DDPG, is an off-policy algorithm that is used for environments with continuous action spaces.

B. SAC Algorithm

We have used the standard implementation of SAC as it was introduced in Haarnoja et al. in the paper "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning" which combines actor-critic methods with entropy regularization to encourage exploration [7]. Soft Actor-Critic (SAC) is an off-policy reinforcement learning algorithm that optimizes a stochastic policy and designed for continuous control tasks that involve high-dimensional action spaces [7], [8]. SAC combines the elements of actor-critic methods with maximum entropy reinforcement learning. The algorithm aims to optimize a stochastic policy that not only maximizes the expected return but also maximizes entropy, promoting exploration and robust learning. SAC achieves this by introducing an **entropy regularization term** in the objective function, balancing the trade-off between exploration and exploitation [7]. *Entropy* is a quantity that specified how random a random variable is. Let x be a random variable with probability mass or density function P. The entropy H of x is computed from its distribution P according to

$$H(P) = \underset{x\sim P}{E}[-\log P(x)] \tag{3}$$

In reinforcement learning with entropy regularization, the agent receives an additional reward at each time step proportional to the entropy of the policy at that time step. This modification transforms the RL problem to:

$$\pi^* = \arg\max_{\pi} \underset{\tau\sim\pi}{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(.|s_t)\right)\right)\right] \tag{4}$$

where $\alpha > 0$ is the trade-off coefficient. The use of off-policy learning, target networks, and a replay buffer further contributes to the stability and efficiency of SAC in complex and challenging environments [7], [8].

The SAC algorithm consists of an actor network, a critic network, and a value network. The actor generates stochastic policies, providing a mean and standard deviation for each action dimension. The critic evaluates the quality of actions by estimating the Q-value for state-action pairs, while the value network estimates the state's value. The inclusion of entropy regularization encourages the policy to be more exploratory, enabling the agent to learn a diverse and adaptable set of behaviors. SAC's off-policy learning allows it to leverage past experiences efficiently, and the use of target networks stabilizes the training process by smoothing out Q-value estimates [7]. The loss functions for the Q-networks in SAC are:

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right]} \tag{5}$$

SAC optimizes its policy according to the equation:

$$\max_{\theta} \underset{\substack{s\sim\mathcal{D}\\ \xi\sim\mathcal{D}}}{E}\left[\min_{j=1,2} Q_{\phi_j}\left(s, \widetilde{a}_\theta(s,\xi) - \alpha log \pi_\theta\left(\widetilde{a}_\theta(s,\xi)|s\right)\right)\right] \tag{6}$$

Soft Actor-Critic is particularly suitable for tasks such as a robot arm pushing, where precise control and exploration are crucial. In scenarios where a robot arm needs to interact with its environment, SAC's stochastic policy enables the robot to explore a variety of actions, leading to more versatile and adaptive behavior. The algorithm's ability to handle continuous action spaces makes it well-suited for the fine-grained control required in manipulation tasks [7], [8]. Additionally, SAC's incorporation of entropy regularization ensures that the robot explores a diverse set of actions, preventing it from converging prematurely to suboptimal solutions. This makes SAC a powerful choice for training robotic systems to perform complex and nuanced tasks, such as pushing objects with the required precision and adaptability [7], [8].

C. Hindsight Experience Replay

A sample collection method called Hindsight Experience Replay (HER) is mainly employed in Reinforcement Learning (RL) to teach agents, particularly in robotics. It was first presented by OpenAI to solve an issue in reinforcement learning (RL) called the sparse reward problem, in which an agent receives incentives only infrequently and finds it challenging to learn why it received a reward or how to replicate it [9]. The key idea of HER is to learn from both mistakes and successful outcomes. Replaying and recycling failures allows the agent to learn from them, even if it did not result in a successful outcome. This contributes to learning becoming more sample-efficient and successful, particularly in situations where reaching the intended aim is difficult or uncommon.

In robotics, HER is particularly useful due to the complex and high-dimensional nature of robotic control tasks [9]. Robots often face difficulties in learning due to sparse rewards or high-dimensional action spaces. HER helps in overcoming these challenges by allowing the agent to learn from failures and explore different strategies to achieve various goals, thus making the learning process more efficient and robust. This method contributes to more stable and efficient learning in robotics and other RL applications.

HER works as follows:

- **Goal Re-labeling:** HER re-labels the events after they happen, so that the learning is not limited to the initial goals that the environment set. If the agent fails to reach the first objective, HER reinterprets the experience as successful by treating the reached state as the new objective. The agent is able to learn from these events as successful outcomes because to this reframing, which improves the effectiveness of the learning process.
- **Experience Replay:** HER keeps track of previous attempts—both successful and unsuccessful—to accomplish various objectives in a replay buffer. The agent is therefore able to learn from a wider range of events and objectives when these encounters are replayed during training.

Upon this time we did not implement algorithms that use HER. The outcomes and findings that we found without HER are mentioned in the next section.

IV. Methodology

This study aims to identify the differences between the performance of SAC and TD3 algorithms on OpenAI Gymnasium's "pusher-v4" environment and the objective of learning to push objects to a designated point with and without HER integration to these algorithms and the affect of HER algorithm on the performance of aforementioned algorithms. We investigated the performances of TD3 and SAC algorithms and their HER-integrated variations in the context of different seeds, buffer sizes, discount rates, and batch sizes. We conducted 9 different types of experiments to compare the algorithms and their variations under different hyperparameters and circusmtances. Each type of experiment was also repeated numerous types as some experiments required the algorithms to train on different seeds and different quantities of the given hyperparameters.

We set our total time step for each experiment to 100,000. We calculated the average returns by taking the average of every 1000 time step, thus we have 95 points to show on the graphs as the algorithms do not learn at first 5000 time steps. We created alternative versions of the algorithms where we applied the HER sample collection method to the algorithms. We trained both algorithms with and without HER on 6 different seeds to have a more accurate analysis and assess the algorithms sample efficiency. We also explored how the algorithms performed under different hyperparameters we indicated above. The default values of arguments in all of the algorithms can be seen in Table I.

In the first experiment, we trained the default versions (without HER) of SAC and TD3 algorithms with six different seeds to evaluate their sample efficiency and to get a more accurate results on their average return values. At the end of the experiment, we calculated the average of the average returns of the two algorithms and compared the results. In the second experiment, we conducted the same procedure with the HER-integrated versions of SAC and TD3 algorithms. For the third experiment, we trained SAC and TD3 algorithms on different values which are: 1000000, 10000, and 100. After we trained the respective algorithms on respective buffer size values, we compared the results and assessed in terms of the attributes of the environment and algorithms. For the fourth experiment, we trained SAC and TD3 algorithms on different discount rates which are: 0.99, 0.90, and 0.80. After we plotted the results, we compared each algorithm's performance on the respective discount rates. For the fifth and last experiment, we trained the algorithms with different batch sizes, namely 128, 256, and 512, respectively. After we plotted the results, we compared the algorithm's performance on the respective batch sizes.

V. Results & Discussion

The experiments we conducted aim to present a comprehensive investigation into the performance of two algorithms, Soft Actor-Critic (SAC) and Twin Delayed DDPG (TD3), and their HER-integrated versions within the challenging context of the OpenAI Gymnasium "pusher-v4" environment. The primary objective of the experiments is to assess the learning capabilities and convergence behaviors of SAC and TD3 and their HER variations over an extended training period of 100,000 steps.

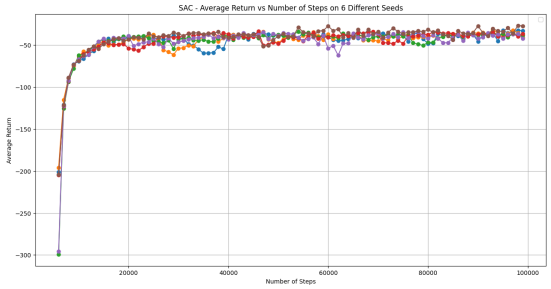

Fig. 2. Average Return of SAC on 6 Different Seeds.

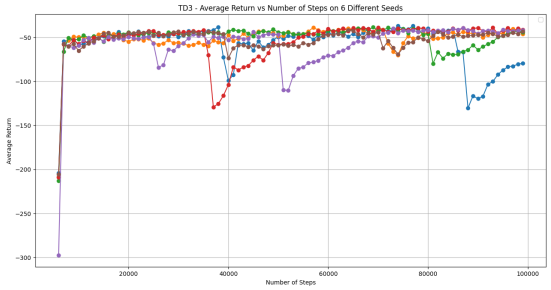

Fig. 3. Average Return of TD3 on 6 Different Seeds.

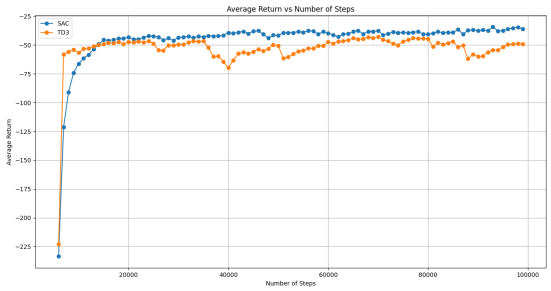

Fig. 4. Average return vs number of steps graph for both models.

A. SAC vs TD3 (Without HER)

The experiment conducted in this part aims to present a comprehensive investigation into the performance of two

TABLE I
DEFAULT VALUES OF ARGUMENTS IN ALL ALGORITHMS.

Arguments	Default Value
Total Timesteps	100,000
Buffer Size (of Replay Memory)	1,000
Gamma (Discount Factor)	0.99
Tau (Target Smoothing Coefficient)	0.005
Alpha (Entropy Regularization Coefficient)	0.2
Batch Size (of Sample from Replay Memory)	256
Q-Learning Rate (of Q-Network Optimizer)	0.001
Policy Learning Rate	0.0004
Timestamp to Start Learning	5,000

algorithms, Soft Actor-Critic (SAC) and Twin Delayed DDPG (TD3), within the challenging context of the OpenAI Gymnasium Pusher environment. The primary objective of this experiment is to assess the learning capabilities and convergence behaviors of SAC and TD3 over an extended training period of 100,000 steps.

For both of the algorithms, a discount factor (gamma) of 0.99 was chosen to weigh the importance of future rewards, emphasizing a balance between short-term and long-term decision-making. Also, the soft update parameter (tau) was set at 0.005, influencing the rate at which target networks are updated and contributing to the stability of the learning process. By setting these parameters of both experiences to the same values, a common ground to compare these two from the aspects of convergence time and performance has been created.

For the implementation of the Soft Actor-Critic (SAC) algorithm, specific hyperparameters were selected to optimize learning performance. The replay buffer size was set to 1e6, providing a balance between memory efficiency and the ability to capture diverse experiences during training.

Furthermore, the policy network optimizer learning rate is set to 3e-4 and Q-network optimizer learning rate to 1e-3. This results in a nuanced consideration for effective weight updates during training. The policy frequency, set at 2, dictates how often the policy network is updated relative to the Q-networks, introducing a strategic balance between exploration and exploitation. Simultaneously, a target network update frequency of 1 ensures a consistent and stable reference for the temporal difference error calculations.

The alpha parameter of the model was set to 0.2. The choice of alpha for this experiment was influenced by what's been suggested in previous studies, some trials, and a goal to find a sweet spot for quick learning while keeping the algorithm stable in the environment.

The training for SAC, took 83 minutes and by the time the training finished the model was able to place the object in the red dot most of the time. The training was successful with this algorithm. The model not only found and pushed the object, it improved itself further to make it faster and more efficiently.

The Figure 2 shows the SAC algorithm's average return of each 1000 steps on six different seeds. The graph in Fig 2 shows the averages of the average return of each 1000 steps in terms of its performance on these six different seeds. So, there is one point in the graph for each 1000 steps. The average return value gives an estimate of the performance of the model after the previous point. From the graph, it can be seen that the learning happens smoothly and consistently. Despite the fluctuations, the model's performance is mostly stable and increasing. The graph is similar to a logarithmic function. The return value seems to converge around -25. This convergence behavior is a sign of successful training.

On the other hand, TD3 was also trained on six different seeds as it can be seen in the Figure 3. In the implementation of the Twin Delayed DDPG (TD3) algorithm, specific hyperparameters were selected to optimize the learning process. The learning rate, set at 3e-4, governs the step size during the weight updates of the neural networks, influencing the speed and stability of learning. A buffer size of 1e6 was chosen to strike a balance between efficient memory usage and the storage of a diverse range of experiences for learning.

Additionally, a batch size of 256 was chosen for the sampling of experiences from the replay buffer during each training iteration, influencing the efficiency of the learning process. The policy noise, set at 0.2, introduces randomness to the policy updates, facilitating exploration in the action space. Simultaneously, an exploration noise of 0.1 contributes to the exploration process by adding noise to the selected actions.

The policy frequency, set at 2, dictates how often the policy network is updated relative to the Q-networks, introducing a strategic balance between exploration and exploitation. The noise clip parameter, set to 0.5, constrains the amount of noise added to the policy updates, preventing excessive perturbations that could compromise stability. By the time the training process had ended, the model could detect and move towards the object but was not able to push it most of the time. Training took almost the same amount of time as the SAC. Although the model was showing signs of intelligence where it could locate and push the object, the training was not successful. The model was not able to push the object accurately to the target.

It can be seen from the graph in Fig 5 that the learning starts fast but continues inconsistently. At some points, dramatical decreases occur which makes the return value unpredictable and unstable. The graph implies that it is stuck in this region and cannot improve any further. There are three regions

that show consistent improvement. Two of these end with a substantial decrease and the last one is unknown due to the limitation of the number of steps. Therefore, it can be argued that the TD3 algorithm was not successful to converge to a value where the model could perform well in the environment.

From the graph in Figure 4, it is safe to say that the SAC algorithm outperforms TD3 in terms of performance and stability in the Pusher environment with parameters stated above. Although the difference does not seem substantial in the graph, according to the observations made during the training, the return values that are greater than -30 can be considered to be successful while others simply fail to push the object to the target. Therefore, SAC almost always manages to push the object to target and score above -30 while TD3 usually fails and scores anywhere between -30 and -100.

TD3 fluctuates more when it is run with different seeds. It can be inferred that SAC has a better sample efficiency. TD3 generally approaches to -30 in a faster rate whereas SAC generally approaches to -30 in a slower rate. However, SAC converges to -30 in a more successful and consistent way. After 20000 steps, SAC holds its return value between -25 and -50 whereas TD3 randomly fluctuates down below -50.

- **Exploration Strategy:** SAC uses an **entropy regularization term**, which encourages the policy to discover more diverse actions and explore the environment effectively.
- **Sample Efficiency:** SAC maintains a **stochastic policy**, whereas TD3 uses a **deterministic policy**. A stochastic policy provides a more robust learning process, enhancing sample efficiency.
- **Handling Continuous Action Spaces:** The actor-critic framework enables SAC to perform better in continuous action spaces as it simultaneously learns a policy and a value function.

The OpenAI's Gymnasium "pusher-v4" environment is an environment that has a reward system that is not binary and sparse and a relatively well-designed environment in terms of reward system, the agent learns to successfully push the object to the desired location by evaluating the process of executing the task. If the environment's reward system was sparse, the agent would rarely be rewarded and if the reward system was binary, the agent would only learn whether it was successful at the end of each episode. SAC is an off-policy algorithm which means that it can learn from past experiences more effectively. This can be crucial in environments like "pusher-v4" environment where experiences are not collected sequentially. Furthermore, SAC introduces a temperature parameter that controls the amount of entropy regularization. The temperature parameter is learned during the training, so it allows the algorithm to adapt the level of exploration based on the task's complexity. This adaptability can also be beneficial in environments with well-crafted reward systems such as "pusher-v4" environment. Other than that, SAC's stochastic policy can be more advantageous in the non-binary and non-sparse reward structure of the "pusher-v4" environment due to its ability to express and learn a distribution over actions.

Fig. 5. Average Returns of SAC with HER on 6 Different Seeds.

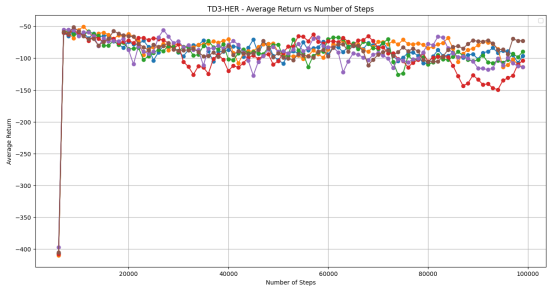

Fig. 6. Average Returns of TD3 with HER on 6 Different Seeds.

B. Effect of HER on SAC and TD3

Once we start using the Hindsight Experience Replay approach alongside with the Soft Actor Critic algorithm, we observe the graph in Figure 5, where the average returns of SAC with HER on 6 different seeds are illustrated. Likewise, Figure 6 depicts the same data of TD3 with HER. Apparently, HER increases the fluctuations and the standard deviation of the average returns. When we dig deeper into the reason for this inconsistency, there are a few possible answers. The most probable one is that HER introduces additional transitions with achieved goals that might differ from the intended goals as Andrychowicz et al. suggests, "there is a huge discrepancy between what we optimise (i.e. a shaped reward function) and the success condition" [9]. This discrepancy can lead to fluctuations in the learning process, especially when the agent is attempting to learn from experiences where the intended and achieved goals are different, which might be the case for our tests, too. Another possible reason might be related with the hyperparameters as the accuracy of HER is sensitive to hyperparameters. Finally, the dynamics of the environment is almost always a possible argument for any unexpected behaviour in the RL domain. These cases are applicable for both Figure 5 and Figure 6.

Figure 7 demonstrates the comparison of pure SAC, represented with blue dots, and SAC with HER, represented with orange dots. Similarly, Figure 8 depicts the same comparison with TD3 and TD3 with HER. From the given graphs, we can simply conclude that unlike our expectations, HER decreases the average of return values for both algorithms. When we

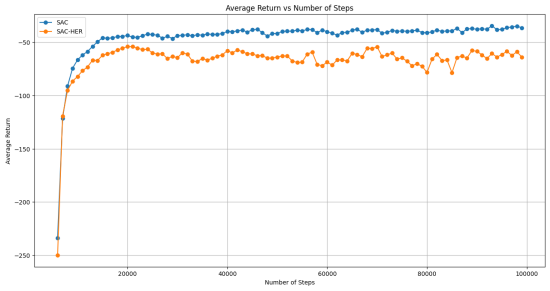

Fig. 7. Averages of Average Returns of SAC & SAC with HER.

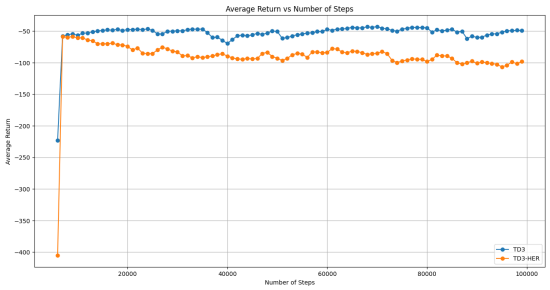

Fig. 8. Averages of Average Returns of TD3 & TD3 with HER.

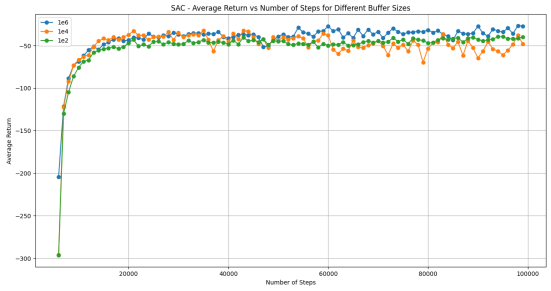

Fig. 9. Average Returns of SAC for Different Buffer Sizes (1,000,000 vs 10,000 vs 100).

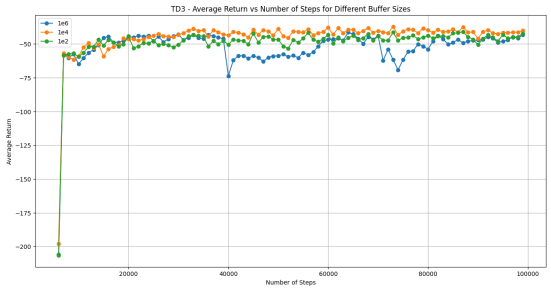

Fig. 10. Average Returns of TD3 for Different Buffer Sizes (1,000,000 vs 10,000 vs 100).

investigate the root causes of these results, our fundamental hypothesis is that our pusher environment has already been reward-engineered so that it has a continuous reward system. Another reason might be that HER "can cause the agent to learn not to touch the box [ball] at all if it can not manipulate it precisely" [9]. For example, in our case, our agent might have successfully touched the ball multiple times but in each one of these attempts, it might have pushed the ball away from the red target spot. In this way, it is probable that the agent infers a wrong causality between touching the ball and having a negative reward. This faulty causality leads the agent to think that touching the ball is bad, which hinders exploration.

Andrychowicz et al. showed that HER might not be advantageous at all times, too. Andrychowics et al. portray the failure of HER in improving the learning curve in a pushing task [9].

C. Average Returns of for Different Buffer Sizes

Environments in reinforcement learning can be classified as either on-policy or off-policy based on the characteristics and requirements of the learning algorithm. Usually, this classification has to do with how the agent interacts with the environment and how learning is accomplished using the data that is gathered. Both SAC and TD3 are originally off-policy algorithms. However, some hyper parameters such as buffer size affects whether the algorithm acts as on-policy or off-policy. While buffer size alone doesn't determine whether an algorithm is on-policy or off-policy, it influences the learning dynamics. Larger buffer sizes are often associated with off-policy learning, and smaller buffer sizes are more common in on-policy learning. Figure 9 and Figure 10 illustrate how batch sizes affect the average returns with three different buffer sizes: 100, 10,000, and 1,000,000.

Usually, we would expect a larger buffer size to perform better in the terms of learning and convergence speed since it allows the algorithm to store and reuse a more diverse set of experiences. This increased data efficiency can accelerate learning by providing a broader exploration of the state-action space while reducing the correlation among samples used for learning updates which stabilises the learning process. However, as one can see from the given graphs, there is no mutual direct correlation between buffer size and average returns for SAC and TD3. A buffer size of 10,000 performed the best for TD3 whereas a buffer size of 1,000,000 surpassed others for SAC without a significant gap. There are a few possible explanations to this inconsistency.

The most common explanation to a smaller buffer size outperforming a larger one is that in dynamic environments, experiences stored in a larger buffer may become outdated or less relevant over time. If the environment undergoes significant changes, using old experiences might hinder learning, and a smaller buffer that prioritises recent experiences could adapt faster. However, this cannot be the case for our experiments since the environment is static. The most probable reason seems to be the fact that the effectiveness of a buffer size is interconnected with other hyperparameters. Suboptimal

choices for learning rates, exploration strategies, or target update frequencies can impact the performance of a larger buffer. Most probably, there is an optimal buffer size regarding other hyperparameters and the closer the buffer size gets to it, the faster the agent learns and converges.

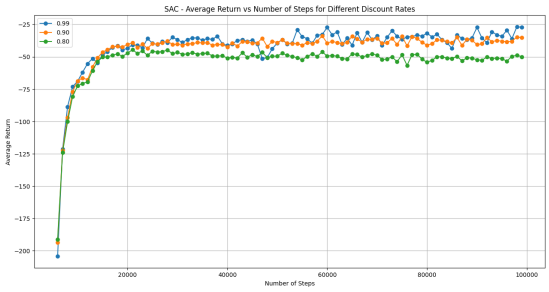

Fig. 11. Average Returns of SAC for Different Discount Rates (0.99 vs 0.90 vs 0.80).

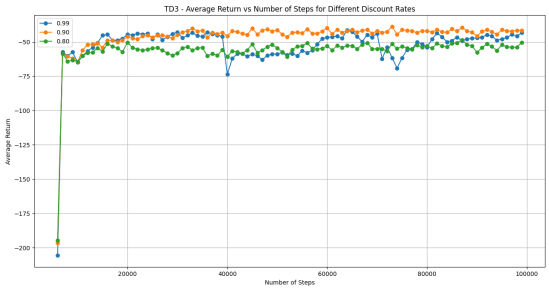

Fig. 12. Average Returns of TD3 for Different Discount Rates (0.99 vs 0.90 vs 0.80).

D. Average Returns for Different Discount Rates

In Figure 11 and Figure 12, it can be seen that three different gamma (discount rate) are 0.99, 0.90, and 0.80. 0.99 is the default gamma in our experiments. In the graphs, as the discount rate approaches 1, the average return increases. This shows that when no discount rate is present, no penalty is applied for latency while achieving rewards, as expected. Another reason may be that a higher discount factor (0.99) emphasizes long-term rewards, aiding in better overall performance by enabling effective planning for cumulative returns over extended trajectories. Conversely, a lower discount factor (0.80) prioritizes short-term rewards, potentially hindering the agent's ability to optimize performance by undervaluing long-term beneficial actions. The intermediate performance observed with the medium discount rate (0.90) likely results from balancing short and long-term rewards, impacting the agent's learning and decision-making capabilities in the given context. Environmental dynamics and reward structures could further influence the optimal discount factor for effective learning and adaptation.

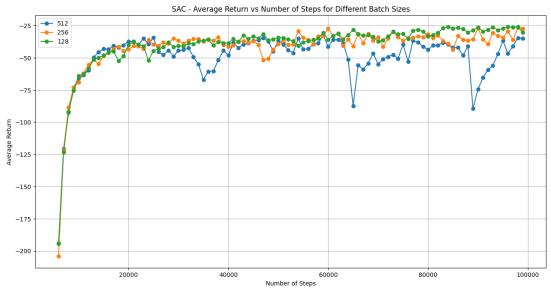

Fig. 13. Average Returns of SAC for Different Batch Sizes (128, 256, 512).

E. Average Returns for Different Batch Sizes

We tried the SAC algorithm with 3 different batch sizes which are 128, 256 and 512. Our observations in Figure 13 clearly show that the trials with the smaller batch sizes performed better and had less fluctuations than the one with 512 as the batch size. The smaller ones performed similar, however the trial with 128 batch size usually resulted in higher rewards. This result can be expected because of the diverse action space of our environment. The agent explores a wide range of possibilities, therefore the sampled batch consists of a mix of highly diverse actions some of which are extreme or noisy samples that the agent learns from. Increasing the batch size, therefore, could have increased the diversity and not resulted in a focused learning curve as the ones with smaller batch sizes. Before the experiment, it was expected that the higher batch sizes would be more efficient in terms of computational performance. However, we did not experience any significant change in efficiency.

VI. Conclusion

In conclusion, we have discussed our experiments when we deployed the two RL algorithms SAC and TD3 and observed that SAC have achieved quite a high average return and can effectively push the object to the desired location, while TD3 struggles to push the object to the goal in some episodes, though it also knows where the object is located.

We observed that HER did not enable/increase learning in our choice of environment, even though our literature review show that it enables learning in environments where the reward is sparse and binary and other choices algorithms show success percentage of 0. We link this result to the fact that the pusher-v4 environment has a very well defined continuous reward system and the agent can learn very well how it performed in the episode, while the HER technique's introduction of replaying goals is not necessary and observed to be slowing down the learning process.

References

[1] M. Q. Mohammed, K. L. Chung, and C. S. Chyi, "Pick and Place Objects in a Cluttered Scene Using Deep Reinforcement Learning," International Journal of Mechanical & Mechatronics Engineering (IJMME) - IJENS, vol. 20, no. 04, pp. 50, Aug. 2020, [Accessed: October 30, 2023].

[2] T. Tanaka, T. Kaneko, M. Sekine, V. Tangkaratt, and M. Sugiyama, "Simultaneous Planning for Item Picking and Placing by Deep Reinforcement Learning," in Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA (Virtual), Oct. 25-29, 2020, [Accessed: October 30, 2023].
[3] Farama Gymnasium, "Pusher Environment - MuJoCo," Farama Gymnasium, https://gymnasium.farama.org/environments/mujoco/pusher/, [Accessed: October 30, 2023].
[4] T. P. Lillicrap et al., 'Continuous control with deep reinforcement learning', arXiv [cs.LG]. 2019.
[5] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, 'Deterministic Policy Gradient Algorithms', 31st International Conference on Machine Learning, ICML 2014, vol. 1, 06 2014.
[6] S. Fujimoto, H. van Hoof, and D. Meger, 'Addressing Function Approximation Error in Actor-Critic Methods', arXiv [cs.AI]. 2018.
[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, 'Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor', arXiv [cs.LG]. 2018.
[8] T. Haarnoja et al., 'Soft Actor-Critic Algorithms and Applications', arXiv [cs.LG]. 2019.
[9] M. Andrychowicz et al., 'Hindsight Experience Replay', in Advances in Neural Information Processing Systems, 2017, vol. 30, [Accessed: November 25, 2023].