Multi-Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay

Adarsh Sehgal[1,2(✉)], Muskan Sehgal[1,2], and Hung Manh La[1,2(✉)]

[1] Advanced Robotics and Automation (ARA) Laboratory, Reno, USA
adarsh23.sehgal@gmail.com, hla@unr.edu
[2] Department of Computer Science and Engineering, University of Nevada, Reno 89557, NV, USA

Abstract. Actor learning and critic learning are two components of the outstanding and mostly used Deep Deterministic Policy Gradient (DDPG) reinforcement learning method. Although such a method plays a significant role in the overall robot's learning, the performance of the DDPG approach is relatively sensitive and unstable. To further enhance the performance and stability of DDPG, this paper introduces a multi-actor-critic DDPG for reliable actor-critic learning, which will be then used to create a new deep learning framework called AACHER and integrated with Hindsight Experience Replay (HER). The AACHER uses the average value of multiple actors or critics to substitute the single actor or critic in DDPG in order to increase resistance when one actor or critic performs poorly. Using numerous independent actors and critics is expected to gain knowledge from the environment more broadly. The developed AACHER is validated with goal-based environments, including *AuboReach*, *FetchReach-v1*, *FetchPush-v1*, *FetchSlide-v1*, and *FetchPickAndPlace-v1*. Various instances of actor/critic combinations are used to experimentally validate the new approach. Results reveal that AACHER outperforms the traditional algorithm (DDPG+HER) in all aspects of the actor/critic number combinations used for evaluation. When combined with *FetchPickAndPlace-v1*, the performance boost for A20C20 (20 actors and 20 critics) is as high as roughly 3.8 times the success rate in DDPG+HER.

Keywords: AACHER · Reinforcement Learning · Actor · Critic

1 Introduction

Deep Learning, a component of machine learning, employs hierarchical designs to extract high-level abstractions from data. It has been applied in various fields like transfer learning [31], the medical field [24], and more [21,25]. Deep Reinforcement Learning (DRL) [6] shows promise in robotics but faces challenges like slower learning and high sample requirements for training [16]. Autonomous

G. Bebis et al. (Eds.): ISVC 2024, LNCS 15046, pp. 31–43, 2025.
https://doi.org/10.1007/978-3-031-77392-1_3

robots use Q-learning for various tasks, with studies on both continuous and discrete action spaces [7,23]. Reinforcement learning (RL) [30] has been applied to tasks like locomotion [12], manipulation [18,19,32], and autonomous vehicle control [1]. A notable RL application is robotic hands adapting to environmental uncertainty, with soft body robots enhancing movement over soft materials [20]. Similar applications are discussed in [21–23,26–28].

RL algorithms are classified as actor-only, critic-only, and actor-critic methods [13]. The Deep Deterministic Policy Gradient (DDPG) algorithm [14] has performed well in simulated continuous control problems, playing a crucial role in the actor-critic method. Experience Replay (ER) [15] and Hindsight Experience Replay (HER) [3] enhance DDPG by improving sample efficiency and learning from sparse rewards.

Our research contrasts DDPG with HER [26]. DDPG's performance relies on both actor and critic learning. Our previous work [26] used a genetic algorithm to adjust hyperparameters, showing promising results with the Genetic Algorithm (GA).

A study [33] suggested that multi-critic learning (MC-DDPG) and Double Experience Replay (DER) could further enhance DDPG's stability and performance. Our research introduces a novel algorithm, Assorted Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay (AACHER), based on [5]. AACHER addresses deep reinforcement learning challenges, including efficiency, reproducibility, learning transfer, and real-world application. It employs the state-of-the-art DDPG algorithm with multiple actor-critic schemes, tested in various scenarios. The algorithm is applied to five goal-based gym environments with robotic manipulators: *AuboReach*, *FetchReach-v1*, *FetchPush-v1*, *FetchSlide-v1*, and *FetchPickAndPlace-v1*. Results validate that increasing the number of actors and critics enhances the learning process.

Open source code is available at https://github.com/aralab-unr/multi-actor-critic-ddpg-with-aubo.

- Developed a novel algorithm, AACHER (Assorted Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay), by advancing DDPG combined with HER.
- Utilized various combinations of actors and critics to create independent instances for multiple actors, critics, or both.
- Computed the loss and actions using the average of actor and critic networks, applying the updated parameters to target networks.
- Created Aubo-i5 custom environments (*AuboReach*) in both simulated and real-world settings to analyze AACHER.
- Implemented four of OpenAI's gym environments: *FetchReach-v1*, *Fetch Push-v1*, *FetchSlide-v1*, and *FetchPickAndPlace-v1* to further test and validate AACHER.
- Examined AACHER's effectiveness using various combinations of actors and critics in both simulated and real manipulation tasks.
- Compared AACHER's results to DDPG+HER.
- Demonstrated AACHER's superior performance over DDPG+HER, highlighting the benefits of multiple actors and critics.

By introducing multiple actors and critics into the DRL framework, AACHER offers a flexible and promising architecture. Recent extensions of DDPG include [2,9,10]. To our knowledge, no prior research has used DDPG with multiple actor and critic networks simultaneously, making AACHER novel. AACHER is easy to implement, reproduce, understand, and integrate with various RL state-action value-related approaches.

The remaining of this study is separated into several sections: Section II explains the proposed AACHER algorithm. Section III explains the experiment settings, results, and discussions. Section V explains the conclusion and future work.

2 Assorted Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay

Algorithm 1. AACHER

Create D Actor Neural networks (Policy Networks)
Create P Critic Neural networks (Q-Networks)
Initialize losses as an average of Actor and Critic neural networks
Initialize replay buffer $R \leftarrow \phi$ as an empty set
for episodes in range (M) **do**
    Sample a goal and initial state
    $g, s_0 \leftarrow$`sample_goal_and_initial_state()`
    for t in range (T) **do** t=0, T-1
        Sample an action a_t using behavioral policy generated by taking an average of policy neural (actor) networks
        $a_t \leftarrow$ sample_action (actor_networks)
        Execute the action a_t and observe a new state s_{t+1}
    end for
    for t in range(T) **do** t=0, T-1
        Compute reward for the current state-action pair and goal
        $r_t := r(s_t, a_t, g)$
        Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in R
        Sample a set of additional goals for replay $G := S($**current episode**$)$
        for $g' \in G$ **do**
            $r' := r(s_t, a_t, g')$
            Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in R
        end for
    end for
    for iteration in range(N) **do** t=1,N
        Sample a minibatch B from the replay buffer R
        Utilizing actor and critic networks for the optimization process
        update_actor_and_critic(actor_networks, critic_networks, minibatch=B)
    end for
end for

DDPG has demonstrated good performance in several areas, although there are still areas for performance enhancement and stability [17]. The training of the DDPG technique is particularly sensitive to the efficiency of the actor/critic learning process because the actor's learning process largely depends on the critic. By using the DDPG method and an average actor/critic, it is possible to obtain an evident improvement in stability and performance. The details of the AACHER approach are presented in this section (Fig. 1).

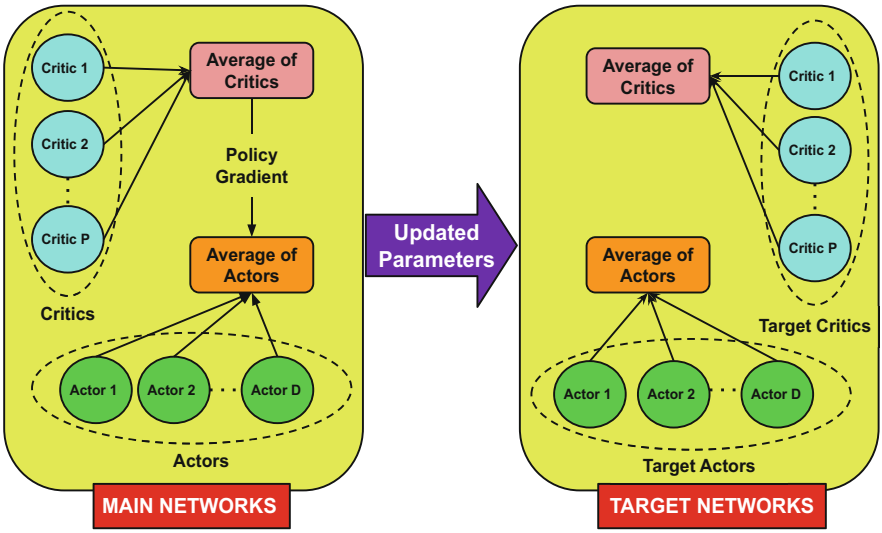


Fig. 1. AACHER using multiple actors and critics.

Because AACHER employs D actors and P critics in its actor-critic architecture, the actor-network is represented by the average of D actor values, and the state-action value function is estimated by the average of P critic values:

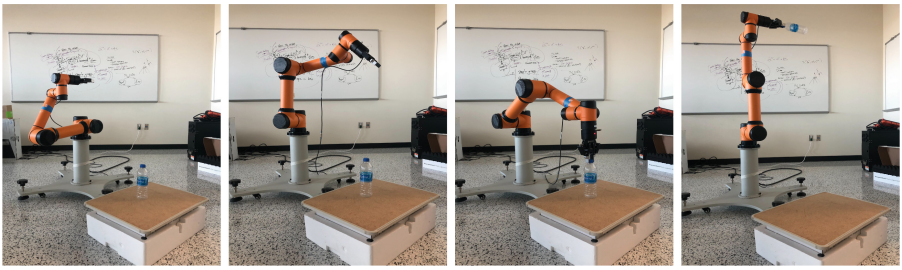
$$\mu_{avg}(s,\omega) = \frac{1}{D}\sum_{i=1}^{D}\mu_i(s,\omega_i),$$
$$Q_{avg}(s,a,\theta) = \frac{1}{P}\sum_{i=1}^{P}Q_i(s,a,\theta_i), \tag{1}$$

where $\omega_i \in \omega$ and $\theta_i \in \theta$ represent the i-th actor and critic parameters, respectively. In contrast to the actions/Q-values that were previously learned, the AACHER technique builds D/P independent actor/critic networks. As a result, when one actor or critic gives a poor performance, the average of all performers or actors/critics will somewhat mitigate the negative impact. Additionally, numerous independent actors and critics can gain a broader understanding of the environment. The actor networks are made up of a parameterized group of

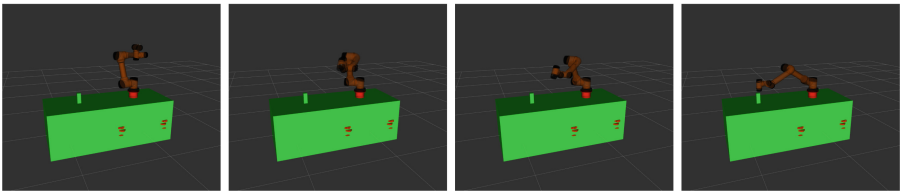
policies that are typically updated by a policy gradient, while the critic networks are updated by Temporal difference (TD) errors [29].

Training of critics uses TD errors between the average of critics and target critics:

$$L_{avg}(\theta) = \left(r(s,a) + \gamma Q^{tar}_{avg}(s,a,\theta^{-}) - Q_{avg}(s,a,\theta)\right)^2. \tag{2}$$



(a) Using the most accurate policy learned via AACHER, the *AuboReach* environment performs a task in a real experiment.



(b) In a simulated experiment, the *AuboReach* environment performs a task using the best policy learned via AACHER.

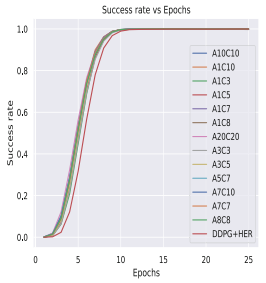
Fig. 2. Comparison of real and simulated experiments in the *AuboReach* environment.

The AACHER approach is described in Algorithm 1. The procedure of initialization starts the algorithm's initial phase. Actor (D-networks) and critic (P-networks) creation are the first and second lines of the algorithm, respectively. While the critic network assesses the Q-values (anticipated cumulative rewards) linked to state-action pairs, the actor network parameterizes the policy in charge of choosing actions. The training losses are then calculated as an average of the losses suffered by actors and critics. Additionally, an empty replay buffer, $R \leftarrow \phi$ is made, which will be used to store training experiences. The algorithm's training loop is introduced and described in the fifth line. We first sample a goal (g) and an initial state (s_0) for each episode (goal-reaching task). We then engage with the environment for a predetermined number of time steps (T) by sampling actions (a_t) from the Actor networks and carrying them out to see what new states (s_{t+1}) emerge. Goals and state-action pairs are used to calculate rewards, which are then saved in the replay buffer. Additional goals $g' \in G$ are sampled as well for replay, and transitions with these goals are kept

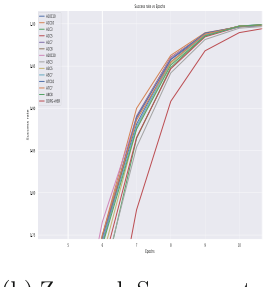
in the buffer. We perform optimization for a predetermined number of iterations (N) after gathering experience. To improve policy and Q-value estimations during optimization, we sample a minibatch of transitions from the replay buffer and update the Actor and Critic networks.

The actor-critic algorithm uses separate actor and critic networks. The actor explores based on the state and the critic evaluates actions. A replay buffer stores past experiences. Networks are updated to improve policy and value estimation. This loop refines the agent's decision-making in reinforcement learning.

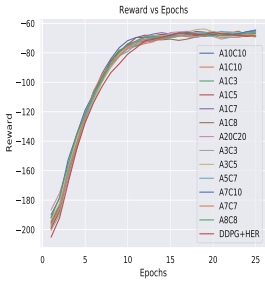
The experiments are referred to by the acronym *ADCP*, where A stands for an actor, D for the number of actors, C for the critic, and P for the number of critics. As an illustration, A2C3 denotes the usage of 2 actors and 3 critics in each main and target network, respectively, which are then averaged for training.



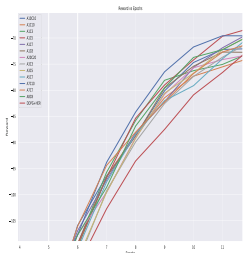
(a) Success rate vs Epochs



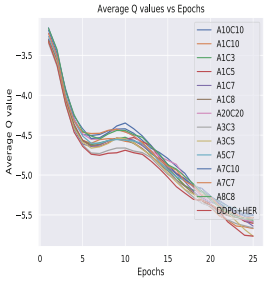
(b) Zoomed: Success rate vs Epochs



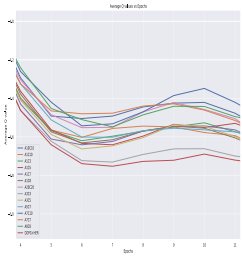
(c) Reward vs Epochs



(d) Zoomed - Reward vs Epochs



(e) Average Q value vs Epochs



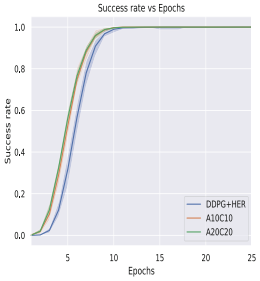
(f) Zoomed - Average Q value vs Epochs

Fig. 3. We present plots comparing the performance of AACHER and DDPG+HER on the *AuboReach* environment. Success rate, reward, and average Q-value are plotted across epochs, averaged over 20 runs. Each plot includes a zoomed-in view for clarity. (AACHER experiments are denoted according to the naming convention.)

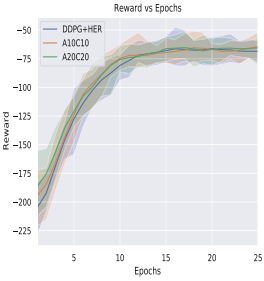
3 Experiments

3.1 Simulated Environment

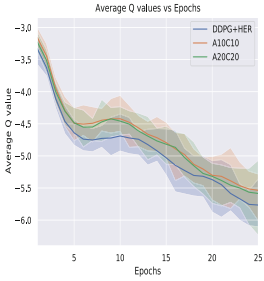
We are using four Open AI gym environments: *FetchReach-v1*, *FetchPush-v1*, *FetchSlide-v1*, and *FetchPickAndPlace-v1* [4]. In addition to these four environments, we are using the *AuboReach* environment, a custom-built gym environment that we developed for our research. In *AuboReach*, the environment comprises an industrial Aubo i5 manipulator that moves from the starting joint state configuration to the target joint state configuration by following the actions (robot joint states). The configurations of the initial and goal states are arbitrary. Four joints are used in this environment for training and testing (instead of six). The joints used are the *shoulder*, *forearm*, *upper arm*, and *wrist1*. This was done to make sure that the learning could be completed promptly. The range of each joint is −1.7 to 1.7 rad. Figures 2a and 2b display real and simulated *AuboReach* settings.



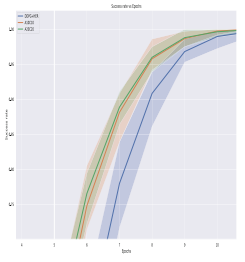
(a) Success rate vs Epochs



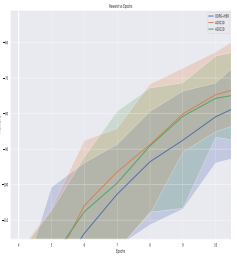
(c) Reward vs Epochs



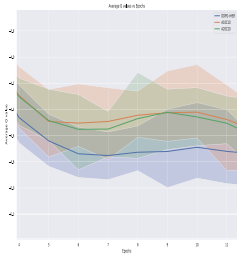
(e) Average Q value vs Epochs



(b) Zoomed - Success rate vs Epochs



(d) Zoomed - Reward vs Epochs



(f) Zoomed - Average Q value vs Epochs

Fig. 4. We compare A10C10 and A20C20 (top performers) against DDPG+HER in *AuboReach*. Plots (success rate, reward, average Q) with error shading (20 runs) and zoomed insets are shown for epochs.

3.2 Experiment Settings

The specific settings for each experiment are covered in this subsection. Every experiment has certain predetermined parameters. Actor and critic learning rates are set at 0.001. Target networks' update rates are 0.01, and the discount factor is 0.98. A zero-mean Gaussian noise with a variance of 0.2 is added to the action during the exploration. The training method for each experiment consists of 25 epochs, each of which comprises 15 cycles. Every cycle, the robot rolls out 100 steps, followed by 20 times robot training. The default batch size is 256. The experience replay buffer is designed as a 10^6 length circular queue. All experience tuples (state, action, reward, and next state) are saved during each trajectory roll-out and kept in a replay buffer of finite size. When we update the value and policy networks, we sample random mini-batches of experience from the replay buffer. The number of additional goals used for the replay is k=4, which means that the replay buffer will immediately retain 20% of normal transitions with the original goal. When neural network weights grow incredibly big, L2 regularization is added to the loss. The robot's observations are also normalized. Adam [11], a momentum-based method, is utilized to optimize the loss function during training. We use Ubuntu 16.04 and a GeForce GTX 1080 Ti Graphic card for our experiment.

In each experiment, the actor and critic networks have three hidden layers with 256 units each. This structure was chosen because testing showed that three layers improved convergence time, beneficial for our research. AACHER tests various combinations of actor and critic networks with similar shapes. To ensure accuracy, DDPG+HER and AACHER were each tested 20 times.

The training was done in a simulated *AuboReach* environment to prevent the robot from colliding with objects. The robot's motors were disabled during simulation, assuming successful transitions to any action selected by the algorithms. The *MOVEit* package [8] managed the planning and execution of internal joint movements, preventing collisions.

AuboReach considers joint states successful if the deviation from the target is less than 0.1 rad. The training objective joint states were [−0.503, 0.605, −1.676, 1.391]. Random initial and target state configurations were tested with the policy generated by training in *AuboReach*, and the robot successfully transitioned between these configurations.

3.3 Experiment Results and Discussions

This subsection presents tests for the DDPG+HER and AACHER methods, conducted in the specified simulated environments with impartial assessments.

Experiments were conducted in the *AuboReach*, *FetchReach-v1*, *FetchPush-v1*, *FetchSlide-v1*, and *FetchPickAndPlace-v1* environments. The approach was tested in various settings to validate it experimentally. Specifically, for *AuboRe-*

ach, training was done in a virtual environment to prevent unexpected manipulator movements. The trained policy was then tested on a real *AuboReach* manipulator, confirming its efficacy despite the lack of real setup measurements.

The DDPG+HER and AACHER methods were assessed using average Q values, success rate, and reward, plotted against epochs and averaged across 20 runs. Both methods were given equal chances to perform. In the *AuboReach* environment, Fig. 3 shows AACHER outperforming DDPG+HER. Variants of AACHER, even with fewer actors or critics, performed better than DDPG+HER, as indicated by higher success rates, rewards, and average Q values, validating AACHER's stability and efficacy. The best-performing A10C10 and A20C20 were further compared to DDPG+HER in Fig. 4. Shaded regions denote the range of values. AACHER converged over 11 epochs, similar to DDPG+HER, but achieved higher rewards and Q-values during training.

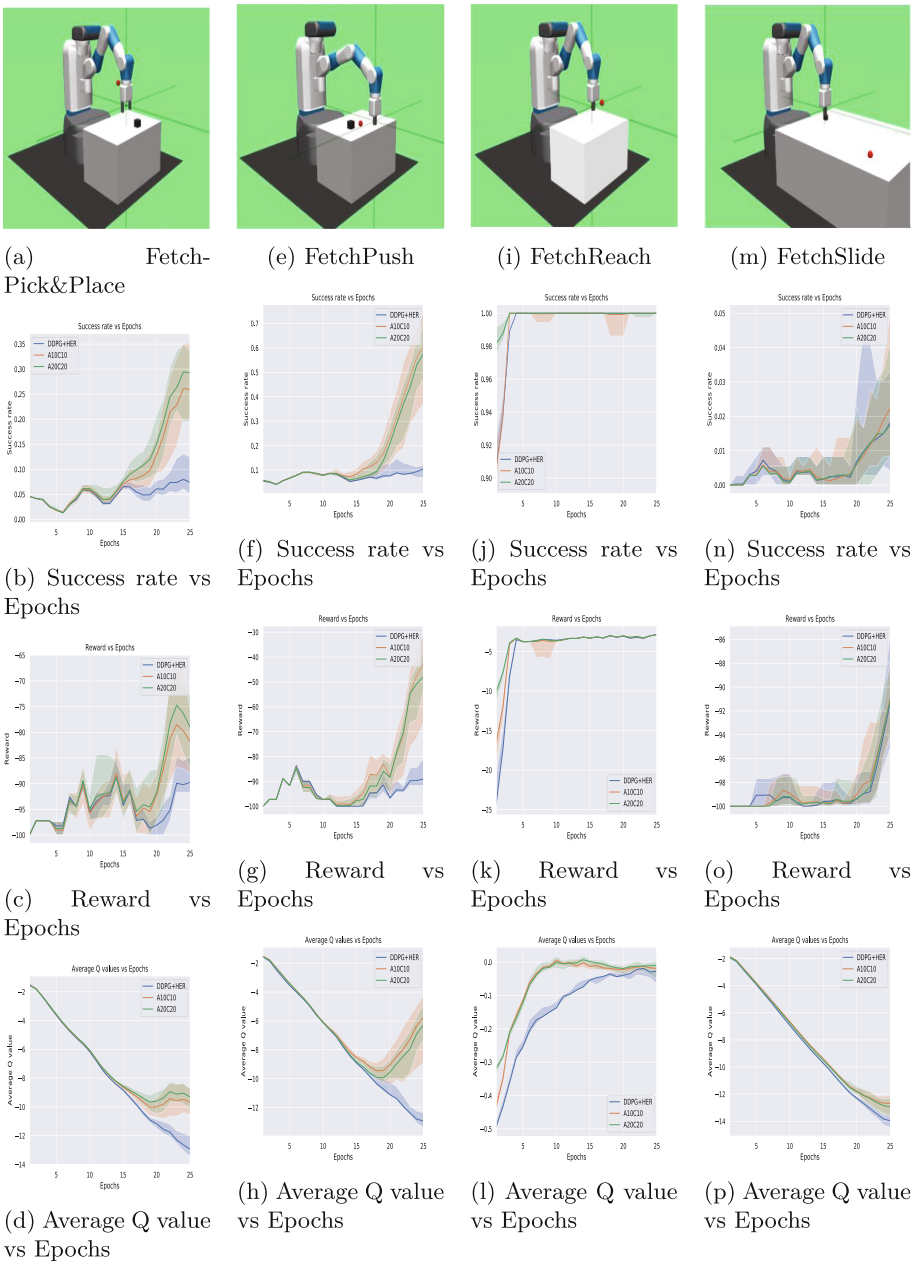
Further evaluations were done in *FetchReach-v1*, *FetchPush-v1*, *FetchSlide-v1*, and *FetchPickAndPlace-v1* environments using the top-performing A10C10 and A20C20 from *AuboReach*. Figure 5 shows the plots for these environments.

Figure 5 demonstrates AACHER's overall superior performance compared to DDPG+HER in OpenAI's gym environments. In *FetchPickAndPlace-v1*, A20C20 performed best in all metrics. While DDPG+HER's performance did not significantly improve, A10C10 was comparable to A20C20. In *FetchPush-v1*, A10C10 performed best, with A20C20 close behind, while DDPG+HER fell short. In *FetchReach-v1*, all methods had similar success rates and rewards, but A10C10 and A20C20 had higher average Q values. In *FetchSlide-v1*, A10C10 surpassed all others across all metrics, even though all methods performed similarly, indicating this is a challenging task.

Table 1 summarizes the experiment's findings, showing the average success rate, reward, and average Q values over 20 runs at the 25th epoch. Bold numbers indicate the best outcomes for each metric in a given setting. Some environments show a success rate of 1, while others like *FetchPush-v1*, *FetchSlide-v1*, and *FetchPickAndPlace-v1* are still below this maximum.

A10C10 has the highest success rate in *FetchPush-v1* and *FetchSlide-v1*, while A20C20 leads in *FetchPickAndPlace-v1*. In *FetchPush-v1*, A10C10's success rate at the 25th epoch is nearly 3.8 times higher than DDPG+HER's, but only 0.4 times higher in *FetchSlide-v1* due to task complexity. A20C20's success rate improvement in *FetchPickAndPlace-v1* is about 3.3 times larger than DDPG+HER's. A10C10's rewards in *FetchPush-v1* are 50% higher than DDPG+HER's, and A20C20's average Q value in *FetchReach-v1* is 76% higher, demonstrating AACHER's superior efficiency over DDPG+HER.

Overall, AACHER surpasses traditional DDPG+HER in success rate, reward, and average Q value across all tests.



(a) FetchPick&Place (e) FetchPush (i) FetchReach (m) FetchSlide

(b) Success rate vs Epochs (f) Success rate vs Epochs (j) Success rate vs Epochs (n) Success rate vs Epochs

(c) Reward vs Epochs (g) Reward vs Epochs (k) Reward vs Epochs (o) Reward vs Epochs

(d) Average Q value vs Epochs (h) Average Q value vs Epochs (l) Average Q value vs Epochs (p) Average Q value vs Epochs

Fig. 5. We compare the two best performers, A10C10 and A20C20, with DDPG+HER across all settings. Success rate, reward, and average Q-values are plotted for each habitat (20 runs averaged, the shaded area shows range).

Table 1. For each of the five environments, the success rate, reward, and average Q value are shown in the table. The average of all the values across 20 runs is for the 25^{th} epoch.

	Setting	Aubo Reach	Fetch Reach-v1	Fetch Push-v1	Fetch Slide-v1	Fetch Pick And Place-v1
Success rate	**DDPG+ HER**	1	1	0.13	0.016	0.08
	A10C10	1	1	**0.652**	**0.023**	0.309
	A20C20	1	1	0.647	0.013	**0.339**
Rewar -d	**DDPG+ HER**	−69.3	−2.5	−89.6	−90.81	−96.9
	A10C10	**−63.00**	−2.5	**−45.00**	**−90.70**	−83.4
	A20C20	−64.60	−2.5	−51.83	−90.8	**−81.04**
Avera -ge Q value	**DDPG+ HER**	−5.8	−0.04	−12.84	−14.16	−12.9
	A10C10	**−5.49**	−0.02	**−5.016**	**−12.74**	−9.19
	A20C20	−5.66	**−0.01**	−5.19	−13.06	**−8.808**

4 Conclusion and Future Work

In this paper, we proposed AACHER (Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay), an algorithm using multiple independent actors and critics to enhance DDPG's performance by mitigating the impact of poorly performing actors or critics. AACHER combines HER and DDPG, creating a more stable and reliable training environment with improved real-world performance. We evaluated AACHER in five simulated environments: *AuboReach*, *FetchReach-v1*, *FetchPush-v1*, *FetchSlide-v1*, and *FetchPickAndPlace-v1*. The configurations A10C10 and A20C20 showed the best results, consistently outperforming traditional DDPG+HER. AACHER is versatile, allowing for different actor-critic configurations based on available processing power and specific task demands, potentially exceeding the performance of DDPG+HER with more than 20 instances. Future work will focus on making the loss function parameters trainable and further improving the experience replay mechanism in HER.

Acknowledgement. This work was partially funded by the U.S. National Science Foundation (NSF) under grants NSF-CAREER: 1846513 and NSF-PFI-TT: 1919127. The views, opinions, findings, and conclusions reflected in this publication are solely those of the authors and do not represent the official policy or position of the NSF.

References

1. Abbeel, P., Coates, A., Quigley, M., Ng, A.: An application of reinforcement learning to aerobatic helicopter flight. Advances in neural information processing systems **19** (2006)
2. Ale, L., King, S.A., Zhang, N., Sattar, A.R., Skandaraniyam, J.: D3pg: Dirichlet ddpg for task partitioning and offloading with constrained hybrid action space in mobile-edge computing. IEEE Internet Things J. **9**(19), 19260–19272 (2022)
3. Andrychowicz, M., et al.: Hindsight experience replay. Advances in neural information processing systems **30** (2017)
4. Brockman, G., et al.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
5. Dhariwal, P., et al.: Openai baselines (2017). https://github.com/openai/baselines
6. Dong, H., Dong, H., Ding, Z., Zhang, S., Chang: deep reinforcement learning. Springer (2020)

7. Gu, S., Holly, E., Lillicrap, T., Levine, S.: Deep reinforcement learning for robotic manipulation. arXiv preprint arXiv:1610.00633 **1** (2016)
8. Hernandez-Mendez, S., Maldonado-Mendez, C., Marin-Hernandez, A., Rios-Figueroa, H.V., Vazquez-Leal, H., Palacios-Hernandez, E.R.: Design and implementation of a robotic arm using ros and moveit! In: 2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC). pp. 1–6. IEEE (2017)
9. Jin, X., Ma, H., Tang, J., Kang, Y.: A self-adaptive vibration reduction method based on deep deterministic policy gradient (ddpg) reinforcement learning algorithm. Appl. Sci. **12**(19), 9703 (2022)
10. Khalid, J., Ramli, M.A., Khan, M.S., Hidayat, T.: Efficient load frequency control of renewable integrated power system: A twin delayed ddpg-based deep reinforcement learning approach. IEEE Access **10**, 51561–51574 (2022)
11. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
12. Kohl, N., Stone, P.: Policy gradient reinforcement learning for fast quadrupedal locomotion. In: IEEE International Conference on Robotics and Automation, vol. 3, pp. 2619–2624 (2004)
13. Konda, V., Tsitsiklis, J.: Actor-critic algorithms. Advances in neural information processing systems **12** (1999)
14. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
15. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach. Learn. **8**(3), 293–321 (1992)
16. Melnik, A., Lach, L., Plappert, M., Korthals, T., Haschke, R., Ritter, H.: Tactile sensing and deep reinforcement learning for in-hand manipulation tasks. In: IROS Workshop on Autonomous Object Manipulation (2019)
17. Peng, Y., Chen, G., Zhang, M., Pang, S.: A sandpile model for reliable actor-critic reinforcement learning. In: Inter. Joint Conf. on Neural Networks (IJCNN), pp. 4014–4021. IEEE (2017)
18. Peters, J., Mulling, K., Altun, Y.: Relative entropy policy search. In: Twenty-Fourth AAAI Conference on Artificial Intelligence (2010)
19. Peters, J., Schaal, S.: Reinforcement learning of motor skills with policy gradients. Neural Netw. **21**(4), 682–697 (2008)
20. Rus, D., Tolley, M.T.: Design, fabrication and control of soft robots. Nature **521**(7553), 467–475 (2015)
21. Sehgal, A.: Genetic Algorithm as Function Optimizer in Reinforcement Learning and Sensor Odometry. Master's thesis, University of Nevada, Reno (2019)
22. Sehgal, A.: Deep Reinforcement Learning for Robotic Tasks: Manipulation and Sensor Odometry. Ph.D. thesis, University of Nevada, Reno (2022)
23. Sehgal, A., La, H., Louis, S., Nguyen, H.: Deep reinforcement learning using genetic algorithm for parameter optimization. In: 2019 Third IEEE International Conference on Robotic Computing (IRC), pp. 596–601. IEEE (2019)
24. Sehgal, A., Sehgal, M., La, H.M., Bebis, G.: Deep learning hyperparameter optimization for breast mass detection in mammograms. In: International Symposium on Visual Computing. Springer (2022)
25. Sehgal, A., Singandhupe, A., La, H.M., Tavakkoli, A., Louis, S.J.: Lidar-monocular visual odometry with genetic algorithm for parameter optimization. In: International Symposium on Visual Computing, pp. 358–370. Springer (2019)

26. Sehgal, A., Ward, N., La, H., Louis, S.: Automatic parameter optimization using genetic algorithm in deep reinforcement learning for robotic manipulation tasks. arXiv preprint arXiv:2204.03656 (2022)
27. Sehgal, A., Ward, N., La, H.M., Louis, S.: Deep reinforcement learning for robotic manipulation tasks using a genetic algorithm-based function optimizer. Encyclopedia with Semantic Computing and Robotic Intelligence (2023)
28. Sehgal, A., Ward, N., La, H.M., Papachristos, C., Louis, S.: Ga-drl: genetic algorithm-based function optimizer in deep reinforcement learning for robotic manipulation tasks. arXiv preprint arXiv:2203.00141 (2022)
29. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
30. Sutton, R.S., Barto, A.G., et al.: Introduction to Reinforcement Learning, vol. 135. MIT Press, Cambridge (1998)
31. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A survey on deep transfer learning. In: International Conference on Artificial Neural Networks, pp. 270–279. Springer (2018)
32. Theodorou, E., Buchli, J., Schaal, S.: Reinforcement learning of motor skills in high dimensions: A path integral approach. In: IEEE Inter. Conf. on Robotics and Automation, pp. 2397–2403 (2010)
33. Wu, J., Wang, R., Li, R., Zhang, H., Hu, X.: Multi-critic ddpg method and double experience replay. In: 2018 IEEE Inter. Conf. on Sys., Man, and Cybernetics (SMC), pp. 165–171. IEEE (2018)