Automatic Parameter Optimization Using Genetic Algorithm in Deep Reinforcement Learning for Robotic Manipulation Tasks

Adarsh Sehgal\textsuperscript{1,2}, Nicholas Ward\textsuperscript{1,2}, Hung La\textsuperscript{1,2*} and Sushil Louis\textsuperscript{2}

\textsuperscript{1} Advanced Robotics and Automation (ARA) Laboratory. \\
\textsuperscript{2} Department of Computer Science and Engineering, University of Nevada, Reno, 89557, NV, USA.

*Corresponding author(s). E-mail(s): hla@unr.edu; asehgal@nevada.unr.edu; nhward@nevada.unr.edu; sushil@cse.unr.edu;

Abstract

Learning agents can make use of Reinforcement Learning (RL) to decide their actions by using a reward function. However, the learning process is greatly influenced by the elect of values of the parameters used in the learning algorithm. This work proposed a Deep Deterministic Policy Gradient (DDPG) and Hindsight Experience Replay (HER) based method, which makes use of the Genetic Algorithm (GA) to fine-tune the parameters' values. This method (GA-DRL) experimented on six robotic manipulation tasks: fetch-reach; fetch-slide; fetch-push; fetch-pick and place; door-opening; and aubo-reach. Analysis of these results demonstrated a significant increase in performance and a decrease in learning time. Also, we compare and provide evidence that GA-DRL is better than the existing methods.

Keywords: DRL, Reinforcement Learning, Genetic Algorithm, GA-DRL, DDPG, HER
1 Introduction

Reinforcement Learning (RL) [1] has recently been applied to a variety of applications, including robotic table tennis [2], surgical robot planning [3], rapid motion planning in bimanual suture needle regrasping [4], and Aquatic Navigation [5]. Each one of these applications employs RL as a motivating alternative to automating manual labor.

We use DDPG [6] in conjunction with HER [7] to learn Deep Reinforcement Learning (DRL) policies in this paper. The efficiency of DDPG + HER is a concern. A better set of DDPG and HER parameters can improve the performance of various robotic manipulation tasks. The number of epochs it takes the learning agent to learn a given robotic task can be used to assess its performance. Genetic Algorithms (GA) and other optimization algorithms can aid in the search for near-optimal parameter values, allowing them to play a larger role in considerably improving the performance of an existing system. [8], [9], [10], and [11] are some of the closely related works. These publications’ findings add to the growing body of data that using a GA to automatically modify the hyper-parameters for DRL can greatly enhance efficiency. The discrepancy can have a notable impact on how long it takes a learning agent to learn.

We develop a novel automatic parameter tweaking approach in this paper, which we apply to DRL from [12]. The algorithm is then applied to four existing robotic manipulator gym environments as well as three custom-built robotic manipulator gym environments. Furthermore, the entire algorithm is examined at various stages to determine whether the technique is effective in increasing the overall efficiency of the learning process. The final results support our claim and provide sufficient evidence that automating the parameter tuning procedure is critical since it reduces learning time by up to 57%. Finally, we compare GA-DRL to four different approaches used on FetchReach. GA-DRL outperforms them all. Open source code is available at https://github.com/aralab-unr/ga-drl-aubo-ara-lab.

The following is a list of our major contributions:

- A novel algorithm for automatic parameter tweaking.
- Six simulated and one real task were used to test the algorithm. To analyze the algorithm, we created Aubo-i5 simulated and actual custom environments.
- As the GA develops, the training process is evaluated based on a variety of parameters.
- The efficiency of DRL was tested over ten runs using GA-DRL discovered parameters in both simulated and actual manipulation tasks.
- GA-DRL was compared to other approaches.
2 Background

2.1 Genetic Algorithm (GA)

Genetic Algorithms (GAs) [13–15] were created to explore poorly-understood areas [16], where an exhaustive search is impossible and other search methods perform poorly. GAs, when employed as function optimizers, aim to maximize fitness that is linked to the optimization goal. On a range of tough design and optimization issues, evolutionary computing techniques in general, and GAs in particular, have had a lot of empirical success. They begin with a population of candidate solutions that have been randomly initialized and are commonly encoded in a string (chromosome). A selection operator narrows the search space to the most promising locations, whereas crossover and mutation operators provide new potential solutions.

To choose parents for crossover and mutation, we employed ranking selection [17]. Higher-ranked (fitter) individuals are probabilistically selected through rank selection. Unlike fitness proportionate selection, ranking selection is concerned with the existence of a fitness difference rather than its magnitude. Uniform crossover [18] is used to create children, who are then altered by flip mutation [15]. Binary coding with concatenated parameters is used to encode chromosomes. [19] shows one such example of GA paired with Lidar-monocular visual odometry (LIMO).

2.2 Deep Reinforcement Learning (DRL)

Q-learning [20] approaches have been used by autonomous robots to do a range of tasks [21], and significant study has been done in this field since its inception [20], with some work focused on continuous action spaces [22–25] and others on discrete action spaces [26]. Reinforcement Learning (RL) [1] has been used to improve locomotion [27] [28] and manipulation [29, 30].

There is also a lot of work on robotic manipulators [31, 32]. Some of this work relied on fuzzy wavelet networks [33], while others relied on neural networks [34] [35] to complete their goals. For real robot systems, off-policy methods [36] such as the Deep Deterministic Policy Gradient algorithm (DDPG) [6] and the Normalized Advantage Function algorithm (NAF) [37] are useful. [38] provides a comprehensive overview of modern deep reinforcement learning (DRL) algorithms for robot handling. For our trials, we are employing DDPG in conjunction with Hindsight Experience Replay (HER) [7]. [39] describes recent work on applying experience ranking to increase the learning pace of DDPG + HER.

Both a single robot [40, 41] and a multi-robot system [42–46] have been extensively trained/taught using RL. Both model-based and model-free learning algorithms have been studied previously. Model-based learning algorithms are heavily reliant on a model-based teacher to train deep network policies in real-world circumstances.
Similarly, there has been a lot of work on GA’s [13] [47] and the GA operators of crossover and mutation [48], which have been applied to a broad variety of problems. GA has been used to solve a wide range of RL problems [48–51].

2.3 GA on DRL

GA can be used to solve a variety of optimization problems as a function optimizer. This study concentrates on the DRL, which was briefly discussed earlier in this chapter. Based on their fitness values, GAs can be utilized to optimize the parameters in the system. GA tries to achieve the highest level of the fitness function. Various mathematical formulas can be used to convert an objective function to a fitness function.

Existing DRL algorithms have a set of parameters that cannot be changed. When GA is applied to DRL, it discovers a better set of parameters, allowing the learning agent to learn more quickly. The fitness value for this problem is the inverse of the number of epochs. GA appears to be a potential technique for improving the system’s efficiency.

3 Genetic Algorithm optimization for Deep Reinforcement Learning

3.1 Reinforcement Learning

Consider a typical RL system, which consists of a learning agent interacting with the environment. $S$ is the set of states, $A$ is the set of actions, $p(s_0)$ is a distribution of initial states, $r : S \times A \rightarrow R$ is a reward function, $p(s_{t+1}|s_t, a_t)$ are transition probabilities, and $\gamma \in [0, 1]$ is a discount factor that can be used to represent an environment. A deterministic policy depicts the relationship between states and actions as follows: $\pi : S \rightarrow A$. Every episode begins with the sampling of the initial state $s_0$. The agent acts at for each timestep $t$ based on the current state $s_t$: $a_t = \pi(s_t)$. The accomplished action is rewarded with $r_t = r(s_t, a_t)$, and the distribution $p(.|s_t, a_t)$ aids in sampling the new state of the environment. $R_t = \sum_{i=T}^{\infty} \gamma^{i-t} r_i$ is the discounted sum of future rewards. The purpose of the agent is to maximize its expected return $E[R_t|s_t, a_t]$, and an optimal policy can be defined as any policy $\pi^*$, such that $Q^*(s,a) \geq Q^*(s,a)$ for each $s \in S, a \in A$, and any policy $\pi$. An optimal Q-function, $Q^*$, is a policy that has the same Q-function as the best policy and fulfills the Bellman equation:

$$Q^*(s,a) = E_{s'}[p(.|s,a))[r(s,a) + \gamma \max_{a' \in A} Q^*(s',a')]]. \quad (1)$$

3.2 Deep Q-Networks (DQN)

A Deep Q-Network (DQN) [52] is a model-free reinforcement learner [53] that is optimized for discrete action spaces. A neural network $Q$ is maintained in a
DQN to approximate $Q^*$. $\pi_Q(s) = \arg\max_{a \in A} Q(s,a)$ denotes a greedy policy w.r.t. $Q$. A $\epsilon$-greedy policy takes a random action with probability $\epsilon$ and action $\pi_Q(s)$ action with probability $1 - \epsilon$.

A $\epsilon$-greedy policy is used to produce episodes during the training. During training, a Replay buffer contains transition tuples $(s_t, a_t, r_t, s_{t+1})$. A generation of fresh episodes is interspersed with neural network training. Tuples $(s_t, a_t, r_t, s_{t+1})$ and a Loss $L$ defined by $L = E(Q(s_t, a_t) - y_t)^2$ where $y_t = r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a')$ are sampled from the replay buffer.

The target network, which is used to measure targets $y_t$, changes at a slower rate than the main network. The target networks’ weights can be set to the main network’s current weights [52]. It is also possible to employ polyak-averaged parameters [54].

3.3 Deep Deterministic Policy Gradients (DDPG)

There are two neural networks in Deep Deterministic Policy Gradients (DDPG): an Actor and a Critic. The critic neural network is an action-value function approximator $Q : S \times A \rightarrow R$ and the actor neural network is a target policy $\pi : S \rightarrow A$. With weights $\theta_Q$ and $\theta_\mu$, the critic network $Q(s,a|\theta_Q)$ and actor network $\mu(s|\theta_\mu)$ are randomly initialized.

To produce episodes, a behavioral policy is used, which is a noisy variation of the target policy, $\pi_b(s) = \pi(s) + N(0,1)$. A critic neural network is trained similarly to a DQN, except that the target $y_t$ is computed as $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$, where $\gamma$ is the discounting factor. The actor network is trained using the loss $L_a = -E_a Q(s, \pi(s))$.

3.4 Hindsight Experience Replay (HER)

Hindsight Experience Reply (HER) aims to learn from setbacks by imitating human behavior. Even if the agent does not achieve the intended goal, the agent learns from each experience. HER considers the modified goal to be whatever condition the agent reaches. Only the transition $(s_t, a_t, r_t, s_{t+1})$ with the original aim $g$ is saved in standard experience replay. The transition $(s_t g', a_t, r_t', s_{t+1} g')$ to modified goal $g'$ is also stored by HER. HER performs admirably with extremely scarce incentives and is far superior to shaped rewards in this regard.

3.5 Open Problem Discussion

The efficiency of DDPG + HER is a concern. Using a better set of parameters in the algorithm can increase the performance of the majority of robotic activities. The number of epochs for the learning agent to learn specific robotic tasks can be used to assess performance. The remaining sections of this section demonstrate how changing the values of various parameters has a major impact on the agent’s learning rate. Later in this section, the solution to this problem is described, and the accompanying experimental findings show that the proposed approach outperforms the existing reinforcement learning technique.
3.6 DDPG + HER and GA

The main contribution of our paper is presented in this section: the genetic algorithm explores the space of parameter values utilized in DDPG + HER for values that maximize task performance while reducing the number of training epochs. The following parameters are targeted: discounting factor $\gamma$; polyak-averaging coefficient $\tau$ [54]; learning rate for critic network $\alpha_{\text{critic}}$; learning rate for actor-network $\alpha_{\text{actor}}$; percent of times a random action is taken $\epsilon$; and standard deviation of Gaussian noise added to not completely random actions as a percentage of the maximum absolute value of actions on different coordinates $\eta$. All of the parameters have a range of 0-1, which may be justified using the equations in this section.

Adjusting the values of parameters did neither boost nor reduce the agent’s learning in a linear or immediately visible pattern, according to our experiments. As a result, a basic hill climber is unlikely to identify optimal parameters. We use our GA to optimize these parameter values because GAs were created for such poorly understood problems.

We use $\tau$, the polyak-averaging coefficient to show the performance non-linearity for different values of $\tau$. As seen in Equation (2), $\tau$ is employed in the algorithm:

$$
\begin{align*}
\theta^Q' &\leftarrow \tau \theta^Q + (1 - \tau) \theta^Q', \\
\theta^\mu' &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^\mu'.
\end{align*}
$$

Equation (3) demonstrates how is employed in the DDPG + HER method, whereas Equation (4) explains the Q-Learning update. $\alpha$ denotes the rate at which you are learning. This update equation is used to train networks.

$$
y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1})|\theta^Q'),
$$

$$
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)].
$$

We will need two learning rates, one for the actor-network ($\alpha_{\text{actor}}$) and the other for the critic network ($\alpha_{\text{critic}}$) because we have two types of networks. The use of the percent of times that a random action is executed $\epsilon$ is explained by equation (5).

$$
a_t = \begin{cases} 
a^*_t & \text{with probability } 1 - \epsilon, \\
\text{random action} & \text{with probability } \epsilon.
\end{cases}
$$
Figure 1 indicates that changing the value of $\tau$ causes a change in the agent’s learning, underscoring the importance of using a GA. We are using four CPUs and the original (untuned) value of $\tau$ in DDPG was 0.95. All values are taken into account up to two decimal places to examine how the success rate changes as the parameter value change. We can see from the plots that there is a lot of room for improvement from the original success rate.

Algorithm 1 Proposed GA-DRL Algorithm

1: Choose population of $n$ chromosomes  
2: Set the values of parameters into the chromosome  
3: Run the DDPG + HER to get the number of epochs for which the algorithm first reaches success rate $\geq 0.85$  
4: for all chromosome values do  
5: Initialize DDPG  
6: Initialize replay buffer $R \leftarrow \phi$  
7: for episode=1, M do  
8: Sample a goal $g$ and initial state $s_0$  
9: for $t=0$, T-1 do  
10: Sample an action $a_t$ using DDPG behavioral policy  
11: Execute the action $a_t$ and observe a new state $s_{t+1}$  
12: end for  
13: for $t=0$, T-1 do  
14: $r_t := r(s_t, a_t, g)$  
15: Store the transition $(s_t g, a_t, r_t, s_{t+1} g)$ in $R$  
16: Sample a set of additional goals for replay $G := S$(current episode)  
17: for $g' \in G$ do  
18: $r' := r(s_t, a_t, g')$  
19: Store the transition $(s_t g', a_t, r', s_{t+1} g')$ in $R$  
20: end for  
21: end for  
22: for $t=1,N$ do  
23: Sample a minibatch $B$ from the replay buffer $R$  
24: Perform one step of optimization using $A$ and minibatch $B$  
25: end for  
26: end for  
27: return $1/epochs$  
28: end for  
29: Perform Uniform Crossover  
30: Perform Flip Mutation at a rate of 0.1  
31: Repeat for the required number of generations to find an optimal solution

The integration of DDPG + HER with a GA, which uses a population size of 30 across 30 generations, is explained in Algorithm 1. To choose parents, we
Fig. 1: Success rate vs. epochs for various $\tau$ for FetchPick&Place-v1 task.
used ranking selection [17]. The parents are determined probabilistically based on rank, which is determined by relative fitness (performance). Then, using uniform crossover [18], children are created. We are also utilizing flip mutation [15] with a 0.1 mutation frequency. Each parameter is encoded using a binary chromosome, and the bits are concatenated to generate a chromosome for the GA. Polyak-averaging coefficient, discounting factor, learning rate for critic network, learning rate for actor-network, percent of times a random action is taken, and standard deviation of Gaussian noise added to not completely random actions as a percentage of the maximum absolute value of actions on different coordinates are the six parameters in order. We need 66 bits for six parameters since each parameter requires 11 bits to be expressed to three decimal places. Domain-independent crossover and mutation string operators can then generate new parameter values using these string chromosomes. Because tiny changes in parameter values produce significant variations in success rates, we examined parameter values up to three decimal places. A step size of 0.001 is, for example, considered the greatest fit for our problem.

The inverse of the number of epochs it takes for the learning agent to achieve close to maximum success rate ($\geq 0.85$) for the first time determines the fitness of each chromosome (set of parameter values). Because GA always maximizes the objective function, fitness is inversely proportional to the number of epochs, converting our minimization problem into a maximization problem. We utilize a GA search since an exhaustive search of the $2^{66}$-size search area is not practicable due to the length of each fitness evaluation.

4 Experimental Results

4.1 Experimental setup

A chromosome is binary encoded, as previously stated. Each chromosomal string is the result of combining all of the GA’s arguments. Figure 2 depicts an example chromosome with four binary encoded parameters.

FetchPick&Place-v1, FetchPush-v1, FetchReach-v1, FetchSlide-v1, and DoorOpening are the environments used to test robot learning on five different simulations tasks (see Figures 3 and 4). Figures 5 and 6 depict AuboReach habitats, which are used in both simulated and real-world research. On these six gym situations, we test our algorithm. FetchPick&Place, FetchPush, FetchReach, and FetchSlide are fetch environments from [55]. We created
Automatic Parameter Optimization Using Genetic Algorithm in Deep Reinforcement Learning for Robotic Manipulation Tasks

Fig. 3: When all six parameters are identified by GA, the matching DRL versus GA-DRL charts are generated. Over ten runs, all graphs are averaged. DDPG+HER is abbreviated as DRL.

DoorOpening and AuboReach, which are custom-built gym environments. The following are the details of the six tasks:

- **FetchPick&Place**: The agent picks up the box from a table and moves to the goal position, which may be anywhere on the table or the area above it.
Automatic Parameter Optimization Using Genetic Algorithm in Deep Reinforcement Learning for Robotic Manipulation Tasks

Fig. 4: When all six parameters are found by GA, the environments and related DRL versus GA-DRL charts are shown. Over ten runs, the graphs are averaged. DDPG+HER is known as DRL.

- **FetchPush**: In front of the agent, a box is stored. It pushes or rolls the box to the table’s goal location. The agent is not permitted to take possession of the box.
- **FetchReach**: In the area around the end-effector, the agent must move it to the goal position.
- **FetchSlide**: The puck is placed on a slippery surface within reach of the agent. It must strike the puck with sufficient force that it (the puck) comes to a halt in front of the goal owing to friction.
- **DoorOpening**: A simulated Aubo i5 manipulator is positioned within reach of a door, with the door handle pointing in the direction of the robot. The goal is to force the door open by applying force to the area of the door handle.
- **AuboReach**: An Aubo i5 manipulator, simulated or real, learns to achieve a desired joint configuration and pick up an object with a gripper.
Automatic Parameter Optimization Using Genetic Algorithm in Deep Reinforcement Learning for Robotic Manipulation Tasks

Fig. 5: Using the most accurate policy learned via GA-DRL, the AuboReach environment performs a task in a real experiment.

Fig. 6: In a simulated experiment, the AuboReach environment performs a task using the best policy learned via GA-DRL.

4.2 Running GA

We ran the GA independently on each of these scenarios to test the effectiveness of our method and compared results to the parameters’ original values. Figure 7a depicts the outcome of our FetchPush-v1 experiment. We used the GA to run the system and discover the best values for the parameters $\tau$ and $\gamma$. We display results from ten GA runs because the GA is probabilistic, and
the results show that optimized parameters determined by the GA can lead to greater performance. The learning agent can learn faster and achieve a higher success rate. Figure 7b depicts one learning run for the initial parameter set, as well as the average learning for the ten GA iterations. As we evaluated and debugged the genetic algorithm, the results displayed in figure 7 demonstrate changes when only two parameters are tuned. We can see the potential for performance improvement. Our findings from optimizing all five parameters back up our optimism, and they are detailed below.

Figure 8 (b) shows a comparison of one original experiment with two averaged runs for optimizing parameters $\tau$ and $\gamma$. We only ran this task twice because these tasks can take several hours to complete in a single run. As we evaluated and debugged the genetic algorithm, the results displayed in Figures 7 and 8 demonstrate changes when only two parameters are tuned. We can see the potential for performance improvement. Our findings from optimizing all five parameters back up our optimism, and they are detailed below.

After that, GA was used to optimize all parameters, with the results presented in Figures 3 and 4 for each task. Table 1 compares the GA-discovered parameters to the RL algorithm’s initial parameters. The learning rates, $\alpha_{actor}$ and $\alpha_{critic}$ are the same as they were in the beginning, while the other
Fig. 8: Success rate vs. epochs for FetchSlide-v1 task when $\tau$ and $\gamma$ are found using the GA.

Four parameters have different values. Figures 3 and 4 indicate that the GA-discovered parameters outperformed the original parameters, suggesting that the learning agent was able to learn more quickly. All of the plots in the previous figure have been averaged over ten runs.

Figures 5 and 6 show how GA-DRL settings (as listed in table 1) were applied to a custom-built gym environment for the Aubo-i5 robotic manipulator. The motors in this environment are controlled by the MOVEit package,
but the DRL works as a brain for movement. The results were at first unexpected. Each epoch took several hours to complete (> 10-15 hours). We did not complete the entire curriculum because it could take several weeks. The same holds for DRL settings. This is because the movement speed of the Aubo i5 robotic manipulator was kept sluggish in both simulation and real-world studies to avoid any unexpected sudden movements, which could result in harm. In the AuboReach setting, there were also planning and execution processes involved in the successful completion of each action. AuboReach, unlike the other gym environments covered in this study, could only run on a single CPU. This is because other environments were introduced in MuJoCo and could easily run with the maximum number of CPUs available. MuJoCo can create several instances for training, which allows for faster learning. AuboReach must only perform one action at a time, much like a real robot. Because of these characteristics, training in this setting takes a long period.

### 4.3 Modifications required for AuboReach

The GA-DRL settings were then applied to the AuboReach environment, but only to action values. This means that in both simulated and real-world studies, the robot did not run to do the action. This is accurate since the robot can do each action determined by the DRL algorithm. Because of the planning and execution processes involved in the robot’s movement, we say this. This also eliminates the possibility of a collision, which may have occurred if these procedures had been skipped. Each epoch now took less than a minute to complete, a considerable reduction in training time that allows this environment to be used for training.

The GA-DRL settings (table 1) were re-applied now that some of the environmental challenges had been overcome. These parameters did not outperform the original parameters in any way. We believe this is due to the fact that this environment is far more intricate and unique than others. We took into account even more aspects to ensure that the environment is trainable. For training and testing, this environment employs four joints (instead of six). shoulder, forearm, upper-arm, and wrist1 are the ones used. This was done to ensure that the learning could be finished in a reasonable amount of time.

| Parameters | DRL | GA-DRL | GA-DRL | GA-DRL |
|------------|-----|--------|--------|--------|
| $\gamma$  | 0.98| 0.928  | 0.949  | 0.988  |
| $\tau$    | 0.95| 0.484  | 0.924  | 0.924  |
| $\alpha_{\text{actor}}$ | 0.001 | 0.001 | 0.001 | 0.001 |
| $\alpha_{\text{critic}}$ | 0.001 | 0.001 | 0.001 | 0.001 |
| $\epsilon$ | 0.3 | 0.1  | 0.584 | 0.912 |
| $\eta$    | 0.2 | 0.597  | 0.232  | 0.748  |

Table 1: DRL vs. GA-DRL values of parameters.
Automatic Parameter Optimization Using Genetic Algorithm in Deep Reinforcement Learning for Robotic Manipulation Tasks

Fig. 9: The AuboReach task’s success rate against epochs. This graph represents the average of over ten runs.

Each joint has a range of -1.7 to 1.7 radians. The robot’s initial and reset states were both set to upright, i.e., [0, 0, 0, 0].

The GA-DRL method was tweaked for better learning and faster parameter search, in addition to minor adjustments to the environment. The definition of success was changed to include ten successful epochs. This means that the GA was regarded successful if it had a 100% success rate for ten consecutive epochs. When \( \alpha_{\text{actor}} \) and \( \alpha_{\text{critic}} \) are both greater than 0.001, learning never converges, according to experiments. As a result, the values of \( \alpha_{\text{actor}} \) and \( \alpha_{\text{critic}} \) were set to 0.001. Multi-threading can occur when only action values are employed; hence four CPUs could be utilized. If the cumulative discrepancy between the target and the achieved joint states is less than 0.1 radians, AuboReach considers the DRL-determined joint states to be a success. [−0.503, 0.605, -1.676, 1.391] were chosen as the objective joint states. We were able to find a new set of parameters using these changes to the algorithm, as shown in Table 1. Figure 9a shows the difference in success rates between DRL and GA-DRL during training. Without a doubt, the GA-DRL outperforms the DRL.

After the GA-DRL had determined the best parameters for the AuboReach environment, the training was repeated using four CPUs to determine
the best policy. The robots were then subjected to this policy in both simulated and real-world testing. The crucial point to remember here is that for testing purposes, CPU consumption was reduced to one. In both studies, the robot was able to transition from the training’s beginning joint space to the goal joint space. Because unpredictability was not added during training, the environment was constrained to only one possible path. Since both DRL and GA-DRL eventually achieved a 100% success rate, there was no discernible difference throughout testing. The main distinction is the speed with which the environment may learn given a set of parameters.

The AuboReach environment was updated in another experiment to train on random joint states. The robot may now start and reach objectives in various joint states during testing thanks to the update. The GA was run on this environment, and the parameters discovered by the GA are shown in table 1. Figure 9b shows that the plot of GA-DRL is still better than DRL. Figures 5 and 6 show the robot in action as it accomplishes the task of picking the object in real and simulated tests.

The use of GA-DRL in the AuboReach environment resulted in automatic DRL parameter adjustment, which improved the algorithm’s performance.

4.4 Training evaluation

Monitoring the status of a GA while it is operating is critical for optimizing the system’s performance. Because of the way GAs work, certain chromosomes will perform better than others. As a result, the performance graph is unlikely to be a smooth rise curve. Some fitness function assessments result in a zero, implying that the chromosome is unsuited for use. Despite the non-smooth slope, it is believed that the system’s overall performance would improve as GA advances.

We created numerous charts to track GA’s progress while we looked for the best parameters. Figures 10, 11, and 12 show how the system’s performance improves as GA progresses. Median success rate across fitness function evaluations, total reward across episodes, and epochs to accomplish the target throughout fitness function evaluations are all parameters taken into account when evaluating training performance. It can be seen that the system’s overall performance is improving. With fitness function evaluations, the overall reward increases as the agent’s episodes and epochs to accomplish the goal decrease. This indicates that the GA is on the right road in terms of determining the best parameter values. We plotted data for only one GA run and limited the duration of a GA run because each GA run requires several hours to many days of run time. When we started to see results, we decided to terminate the GA.

Now that the GA has behaved as intended, we will evaluate the system’s efficiency using the GA’s discovered parameters.
(a) FetchPick&Place - Median reward vs Times GA fitness function evaluated

(b) FetchPick&Place - Epochs vs Times GA fitness function evaluated

(c) FetchPick&Place - Median success rate vs Times GA fitness function evaluated

(d) FetchPush - Median reward vs Times GA fitness function evaluated

(e) FetchPush - Epochs vs Times GA fitness function evaluated

(f) FetchPush - Median success rate vs Times GA fitness function evaluated

**Fig. 10:** When GA finds all six parameters, the GA-DRL training evaluation charts. This is the outcome of a single GA run.

### 4.5 Efficiency evaluation

We generated data for various parameters to evaluate and compare the efficiency of the GA-DRL algorithm for training the agent to accomplish a
Automatic Parameter Optimization Using Genetic Algorithm in Deep Reinforcement Learning for Robotic Manipulation Tasks

Fig. 11: When all six parameters are identified by GA, the GA-DRL training evaluation plots. One GA run yielded this result.

(a) FetchSlide - Median reward vs Times GA fitness function evaluated
(b) FetchSlide - Epochs vs Times GA fitness function evaluated
(c) FetchSlide - Median success rate vs Times GA fitness function evaluated
(d) DoorOpening - Median reward vs Times GA fitness function evaluated
(e) DoorOpening - Epochs vs Times GA fitness function evaluated
(f) DoorOpening - Median success rate vs Times GA fitness function evaluated

Task. These parameters are good indicators of the algorithm’s efficiency. For the majority of the training tasks, as shown in Figure 13, the total reward has increased significantly. The DRL algorithm’s efficiency was significantly
Fig. 12: When GA finds all six parameters, the GA-DRL training evaluation charts. This is the outcome of a single GA run.

improved when rewards were raised. Because it is steered considerably faster towards the desired task, the agent can learn a lot faster. To get an unbiased review, we averaged these plots over ten runs. With GA-DRL, the FetchSlide
environment performed worse. We believe that this is due to the task’s complexity. The maximum number used for that parameter was used to represent in the tables the tasks that did not attain the target during training.

We also generate more data to evaluate the episodes, running time (s), steps, and epochs that an agent must learn to achieve the desired goal. Tables 2-5 present this information. The information in the tables is an average of ten runs. Table 2 compares the number of episodes an agent needs to achieve a goal. The bolded values imply superior performance, and the majority of environments outperform the rest of the algorithms. The task is learned in 54.34% fewer episodes in the FetchPush environment than in DRL.

| Method   | Fetch Pick & Place | Fetch Push | Fetch Reach | Fetch Slide | Door Opening | Aubo Reach |
|----------|--------------------|------------|-------------|-------------|--------------|------------|
| DRL      | 6,000              | 2,760      | 100         | 4,380       | 960          | 320        |
| GA-DRL   | 2,270              | 1,260      | 60          | 6,000       | 180          | 228        |
| PPO      | 2,900              | 2,900      | 1,711       | 4,880       | 1,500        | 2,900      |
| A2C      | 119,999            | 119,999    | 119,999     | 119,999     | 119,999      | 32,512.3   |
| DDPG     | 10,000             | 2,000      | 423         | 10,000      | 706          | 1,000      |

**Table 2:** Efficiency evaluation: For all activities, compare average (over ten runs) episodes to accomplish the target.

Another factor to consider while evaluating an algorithm’s efficiency is its running time. Seconds are used to measure time. The algorithm is better if it takes less time to learn the task. In the majority of the environments, the GA-DRL algorithm had the lowest running time, as shown in Table 3. For example, When compared to the DRL algorithm, Fetch-push with GA-DRL takes about 57.004% less time.

| Method   | Fetch Pick & Place | Fetch Push | Fetch Reach | Fetch Slide | Door Opening | Aubo Reach |
|----------|--------------------|------------|-------------|-------------|--------------|------------|
| DRL      | 3,069.981          | 1,314.477  | 47.223      | 2,012.645   | 897.816      | 93.258     |
| GA-DRL   | 1,224.697          | 565.178    | 28.028      | 3,063.599   | 167.883      | 66.818     |
| PPO      | 1,964.411          | 2,154.052  | 776.512     | 2,379.393   | 997.779      | 710.576    |
| A2C      | 2,025.344          | 2,082.807  | 2,061.807   | 2,268.114   | 2,718.769    | 214.075    |
| DDPG     | 5,294.984          | 1,000.586  | 236.4       | 5,346.516   | 438.7        | 1,721.992  |

**Table 3:** Efficiency evaluation: For all activities, compare the average (over ten runs) running time (s) to attain the target.

Another factor to consider while evaluating and researching the GA-DRL algorithm’s performance is the average number of steps required to attain the goal. The average number of steps taken by an agent in each environment is
Fig. 13: When all six parameters are determined using GA, the DRL vs. GA-DRL efficiency evaluation plots (Total reward vs episodes). Over ten runs, all graphs are averaged.

shown in Table 4. Except for the FetchSlide environment, most of the environments, when employed with GA-DRL, outperform all other algorithms. When compared to the DRL algorithm, Fetch-Push with GA-DRL takes about 54.35% fewer steps.
Automatic Parameter Optimization Using Genetic Algorithm in Deep Reinforcement Learning for Robotic Manipulation Tasks

| Method | Fetch Pick & Place | Fetch Push | Fetch Reach | Fetch Slide | Door Opening | Aubo Reach |
|--------|--------------------|------------|-------------|-------------|--------------|------------|
| DRL    | 300,000            | 138,000    | 5,000       | 219,000     | 48,000       | 65,600     |
| GA-DRL | **113,000**        | **63,000** | **3,000**   | **199,000** | **9,000**    | **46,000** |
| PPO    | 595,968            | 595,968    | 324,961     | 1,000,000   | 301,056      | 595,968    |
| A2C    | 600,000            | 600,000    | 600,000     | 600,000     | 600,000      | 162,566.5  |
| DDPG   | 500,000            | 100,000    | 21,150      | 500,000     | 35,300       | 200,000    |

Table 4: Efficiency evaluation: For all activities, compare the average (over ten runs) steps taken to attain the target.

The number of epochs taken by the agent to attain the goal is the final parameter used to compare the competency of GA-DRL with four other algorithms. The average epochs for all of the environments are shown in Table 5. Almost all environments outperform GA-DRL in terms of efficacy. FetchPush, for example, uses 54.35% fewer epochs with GA-DRL than it does with DRL. Following that, we give a comparison of the GA-DRL algorithm to the other algorithms.

| Method   | Fetch Pick & Place | Fetch Push | Fetch Reach | Fetch Slide | Door Opening | Aubo Reach |
|----------|--------------------|------------|-------------|-------------|--------------|------------|
| DRL      | 60                 | 27.6       | 5           | 43.8        | 47           | 16         |
| GA-DRL   | **22.6**           | **12.6**   | **3**       | **8**       | **8**        | **11.4**   |
| PPO      | 290                | 290        | 171.1       | 488         | 150          | 290        |
| A2C      | 1,200              | 1,200      | 1,200       | 1,200       | 1,200        | 325.2      |
| DDPG     | 1,000              | 100        | 42.3        | 1,000       | 70.6         | 100        |

Table 5: Efficiency evaluation: Average (over ten runs) epochs comparison to reach the goal, for all the tasks.

Next, we present the overall analysis of the GA-DRL algorithm compared to the other algorithms.

4.6 Analysis

We provided various findings and the mechanism for judging the efficacy of GA-DRL versus DRL, PPO, A2C, and DDPG algorithms in the previous subsections. Overall, GA-DRL works best, with one exception of the FetchSlide environment. The average comparison tables illustrate that different values of the assessment parameters can be assumed by each environment. This is determined by the type of task that the agent is attempting to learn. While the majority of the tasks outperformed DRL with more than a 50% increase in efficiency, FetchSlide underperformed DRL. The task’s goal is also credited with this performance. The end-effector does not physically go to the desired position to place the box, which makes this task unique. GA-DRL was tested
using a variety of parameters and an average of over ten runs. This is sufficient proof that GA-DRL outperformed several algorithms. Figures 3, 4 and 9b support our claim by demonstrating that when GA-DRL is utilized, the task may be learned substantially faster in most environments. Finally, in Figure 14, we compare the results of five different algorithms. It can be concluded that GA-DRL outperforms all of these techniques.

Fig. 14: GA-DRL comparison with PPO [56], A2C [57], DRL (DDPG[6] + HER[7]) and DDPG [6]. All plots are averaged over ten runs.
5 Conclusion and Future Work

This research presented preliminary findings that indicated how a genetic algorithm may optimize reinforcement learning algorithm parameters to improve performance, as seen by faster learning rates on six manipulation tasks. We reviewed previous work in reinforcement learning in robotics, presented the GA-DRL algorithm for reducing the number of epochs required to reach maximum performance, and described why a GA would be appropriate for such optimization. Our results on the six manipulation tasks show that the GA can identify parameter values that lead to faster learning and higher (or equal) performance at our chosen tasks, confirming our hypothesis that GAs are a suitable fit for such parameter optimization. We compared GA-DRL to other approaches and found ours as the most effective.

We also showed that heuristic search, as implemented by genetic and other evolutionary computing techniques, is a feasible computational tool for improving reinforcement learning and sensor odometry performance. Adaptive genetic algorithms can also be implemented to take different sets of parameters during system execution. This could indicate online parameter adjustment, which can help any system perform better, regardless of the domain or testing environment type.

Declarations

Funding

This work is supported by the U.S. National Science Foundation (NSF) under grants NSF-CAREER: 1846513 and NSF-PFI-TT: 1919127, and the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (USDOT/OST-R) under Grant No. 69A3551747126 through INSPIRE University Transportation Center. 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 and USDOT/OST-R.

Conflicts of interest/Competing interests

The authors declare that there is no conflict of interest.

Code/Data availability

Open source code, and data used in the study is available at https://github.com/aralab-unr/ga-drl-aubo-ara-lab.

Authors’ contributions

Adarsh Sehgal is the first author and primary contributor to this paper. Adarsh did the research and made this manuscript. Nicholas Ward assisted Adarsh in code execution and gathering results. Drs. Hung La and Sushil Louis advised and overlooked this study.
Ethics approval
Not applicable

Consent to participate
All authors are full-informed about the content of this paper, and the consent was given for participation.

Consent for publication
All authors agreed on the content of this paper for publication.

References

[1] Sutton, R.S., Barto, A.G., et al.: Introduction to Reinforcement Learning vol. 135. MIT press, Cambridge (1998)

[2] Tebbe, J., Krauch, L., Gao, Y., Zell, A.: Sample-efficient reinforcement learning in robotic table tennis. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 4171–4178 (2021). https://doi.org/10.1109/ICRA48506.2021.9560764

[3] Xu, J., Li, B., Lu, B., Liu, Y.-H., Dou, Q., Heng, P.-A.: Surrol: An open-source reinforcement learning centered and dvrk compatible platform for surgical robot learning. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1821–1828 (2021). https://doi.org/10.1109/IROS51168.2021.9635867

[4] Chiu, Z.-Y., Richter, F., Funk, E.K., Orosco, R.K., Yip, M.C.: Bimanual regrasping for suture needles using reinforcement learning for rapid motion planning. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 7737–7743 (2021). https://doi.org/10.1109/ICRA48506.2021.9561673

[5] Marchesini, E., Corsi, D., Farinelli, A.: Benchmarking safe deep reinforcement learning in aquatic navigation. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5590–5595 (2021). https://doi.org/10.1109/IROS51168.2021.9635925

[6] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)

[7] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O.P., Zaremba, W.: Hindsight experience replay. In: Advances in Neural Information Processing Systems, pp. 5048–5058 (2017)
[8] 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 (2019). IEEE

[9] Sehgal, A.: Genetic algorithm as function optimizer in reinforcement learning and sensor odometry. Master’s thesis, University of Nevada, Reno (2019)

[10] 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)

[11] Sehgal, A., Singandhupe, A., La, H.M., Tavakkoli, A., Louis, S.J.: Lidar-monocular visual odometry with genetic algorithm for parameter optimization. In: Bebis, G., et al. (eds.) Advances in Visual Computing, pp. 358–370. Springer, Cham (2019)

[12] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Zhokhov, P.: OpenAI Baselines. GitHub (2017)

[13] Davis, L.: Handbook of genetic algorithms (1991)

[14] Holland, J.H.: Genetic algorithms. Scientific american 267(1), 66–73 (1992)

[15] Goldberg, D.E., Holland, J.H.: Genetic algorithms and machine learning. Machine learning 3(2), 95–99 (1988)

[16] De Jong, K.: Learning with genetic algorithms: An overview. Machine learning 3(2-3), 121–138 (1988)

[17] Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of Genetic Algorithms vol. 1, pp. 69–93. Elsevier, Netherlands (1991)

[18] Syswerda, G.: Uniform crossover in genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms, pp. 2–9 (1989). Morgan Kaufmann Publishers

[19] 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 (2019). Springer
[20] Watkins, C.J., Dayan, P.: Q-learning. Machine learning 8(3-4), 279–292 (1992)

[21] La, H.M., Lim, R., Sheng, W.: Multirobot cooperative learning for predator avoidance. IEEE Transactions on Control Systems Technology 23(1), 52–63 (2015). https://doi.org/10.1109/TCST.2014.2312392

[22] Gaskett, C., Wettergreen, D., Zelinsky, A.: Q-learning in continuous state and action spaces. In: Australasian Joint Conference on Artificial Intelligence, pp. 417–428 (1999). Springer

[23] Doya, K.: Reinforcement learning in continuous time and space. Neural computation 12(1), 219–245 (2000)

[24] Hasselt, H.V., Wiering, M.A.: Reinforcement learning in continuous action spaces (2007)

[25] Baird, L.C.: Reinforcement learning in continuous time: Advantage updating. In: Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference On, vol. 4, pp. 2448–2453 (1994). IEEE

[26] Wei, Q., Lewis, F.L., Sun, Q., Yan, P., Song, R.: Discrete-time deterministic q-learning: A novel convergence analysis. IEEE transactions on cybernetics 47(5), 1224–1237 (2017)

[27] Kohl, N., Stone, P.: Policy gradient reinforcement learning for fast quadrupedal locomotion. In: Robotics and Automation, 2004. Proceedings. ICRA’04. 2004 IEEE International Conference On, vol. 3, pp. 2619–2624 (2004). IEEE

[28] Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., Cheng, G.: Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot. The International Journal of Robotics Research 27(2), 213–228 (2008)

[29] Peters, J., Mülling, K., Altun, Y.: Relative entropy policy search. In: AAAI, pp. 1607–1612 (2010). Atlanta

[30] Kalakrishnan, M., Righetti, L., Pastor, P., Schaal, S.: Learning force control policies for compliant manipulation. In: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference On, pp. 4639–4644 (2011). IEEE

[31] Deisenroth, M.P., Rasmussen, C.E., Fox, D.: Learning to control a low-cost manipulator using data-efficient reinforcement learning (2011)
[32] Jin, L., Li, S., La, H.M., Luo, X.: Manipulability optimization of redundant manipulators using dynamic neural networks. IEEE Transactions on Industrial Electronics 64(6), 4710–4720 (2017). https://doi.org/10.1109/TIE.2017.2674624

[33] Lin, C.-K.: $H_\infty$ reinforcement learning control of robot manipulators using fuzzy wavelet networks. Fuzzy Sets and Systems 160(12), 1765–1786 (2009)

[34] Miljković, Z., Mitić, M., Lazarević, M., Babić, B.: Neural network reinforcement learning for visual control of robot manipulators. Expert Systems with Applications 40(5), 1721–1736 (2013)

[35] Duguleana, M., Barbuceanu, F.G., Teirelbar, A., Mogan, G.: Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning. Robotics and Computer-Integrated Manufacturing 28(2), 132–146 (2012)

[36] Munos, R., Stepleton, T., Harutyunyan, A., Bellemare, M.: Safe and efficient off-policy reinforcement learning. In: Advances in Neural Information Processing Systems, pp. 1054–1062 (2016)

[37] Gu, S., Lillicrap, T., Sutskever, I., Levine, S.: Continuous deep q-learning with model-based acceleration. In: International Conference on Machine Learning, pp. 2829–2838 (2016)

[38] Nguyen, H., La, H.M.: Review of deep reinforcement learning for robot manipulation. In: 2019 Third IEEE International Conference on Robotic Computing (IRC), pp. 590–595 (2019). IEEE

[39] Nguyen, H., La, H.M., Deans, M.: Deep learning with experience ranking convolutional neural network for robot manipulator. arXiv:1809.05819, cs.RO (2018) https://arxiv.org/abs/1809.05819

[40] Pham, H.X., La, H.M., Feil-Seifer, D., Nguyen, L.V.: Autonomous uav navigation using reinforcement learning. arXiv:1801.05086, cs.RO (2018) https://arxiv.org/abs/1801.05086

[41] Pham, H.X., La, H.M., Feil-Seifer, D., Nguyen, L.V.: Reinforcement learning for autonomous uav navigation using function approximation. In: 2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), pp. 1–6 (2018). https://doi.org/10.1109/SSRR.2018.8468611

[42] La, H.M., Lim, R.S., Sheng, W., Chen, J.: Cooperative flocking and learning in multi-robot systems for predator avoidance. In: 2013 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, pp. 337–342 (2013). https://doi.org/10.1109/
[43] La, H.M., Sheng, W., Chen, J.: Cooperative and active sensing in mobile sensor networks for scalar field mapping. IEEE Transactions on Systems, Man, and Cybernetics: Systems 45(1), 1–12 (2015). https://doi.org/10.1109/TSMC.2014.2318282

[44] Pham, H.X., La, H.M., Feil-Seifer, D., Nefian, A.: Cooperative and distributed reinforcement learning of drones for field coverage. arXiv:1803.07250, cs.RO (2018) https://arxiv.org/abs/1803.07250

[45] Dang, A.D., La, H.M., Horn, J.: Distributed formation control for autonomous robots following desired shapes in noisy environment. In: 2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), pp. 285–290 (2016). https://doi.org/10.1109/MFI.2016.7849502

[46] Rahimi, M., Gibb, S., Shen, Y., La, H.M.: A comparison of various approaches to reinforcement learning algorithms for multi-robot box pushing. In: International Conference on Engineering Research and Applications, pp. 16–30 (2018). Springer

[47] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multi-objective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation 6(2), 182–197 (2002)

[48] Poon, P.W., Carter, J.N.: Genetic algorithm crossover operators for ordering applications. Computers & Operations Research 22(1), 135–147 (1995)

[49] Liu, F., Zeng, G.: Study of genetic algorithm with reinforcement learning to solve the tsp. Expert Systems with Applications 36(3), 6995–7001 (2009)

[50] Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. Journal of Artificial Intelligence Research 11, 241–276 (1999)

[51] Mikami, S., Kakazu, Y.: Genetic reinforcement learning for cooperative traffic signal control. In: Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference On, pp. 223–228 (1994). IEEE

[52] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N.: Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581 (2015)
[53] Degris, T., Pilarski, P.M., Sutton, R.S.: Model-free reinforcement learning with continuous action in practice. In: 2012 American Control Conference (ACC), pp. 2177–2182 (2012). IEEE

[54] Polyak, B.T., Juditsky, A.B.: Acceleration of stochastic approximation by averaging. SIAM Journal on Control and Optimization 30(4), 838–855 (1992)

[55] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)

[56] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[57] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937 (2016). PMLR