Deep Reinforcement Learning for Robotic Manipulation Tasks using a Genetic Algorithm-based Function Optimizer

Adarsh Sehgal, Nicholas Ward, Hung Manh La
Advanced Robotics and Automation (ARA) Laboratory
Department of Computer Science and Engineering,
University of Nevada, Reno, 89557, NV, USA

Sushil Louis
Department of Computer Science and Engineering,
University of Nevada, Reno, 89557, NV, USA

Reinforcement learning (RL) enables agents to make a decision based on a reward function. However, in the process of learning, the choice of values for learning algorithm hyperparameters can significantly impact the overall learning process. In this paper, we extended our previously proposed algorithm Genetic Algorithm-based Deep Deterministic Policy Gradient and Hindsight Experience Replay method (called GA+DDPG+HER) to find near-optimal values of learning hyperparameters. We used GA+DDPG+HER method on FetchReach, FetchSlide, FetchPush, FetchPick&Place, and DoorOpening in robotic manipulation tasks. With some modifications, our GA+DDPG+HER method was also applied to the AuboReach environment. In-depth background information, experimental settings, implementation information, training evaluation, and analysis of the GA+DDPG+HER application to manipulation tasks are the main objectives of this research. Our experimental evaluation shows that our method leads to significantly better performance, faster than the original algorithm. Also, we provide evidence that GA+DDPG+HER performs better than the existing methods.

Keywords: DRL; DDPG+HER; Reinforcement Learning; Genetic Algorithm; GA+DDPG+HER; DDPG; HER.

1. Introduction

In the past, deep learning has been extensively used [1–9]. Recent significant applications of Reinforcement Learning (RL) [10] include robotic table tennis [11], surgical robot planning [12], rapid motion planning in bimanual regrasping for suture needles [13], aquatic navigation [14], and standard robotic manipulation [15]. Each of these applications uses reinforcement learning (RL) as a motivating alternative to automating manual work.

In this study, we focus on training Deep Reinforcement Learning (DRL) policies utilizing DDPG [16] and HER [17]. DDPG+HER have a difficulty with efficiency. A better selection of DDPG and HER hyperparameters can enhance the performance of various robotic manipulation jobs. The number of epochs needed for the learning agent to master a certain robotic task can be used to gauge performance. The search for near-optimal hyperparameter values can be aided by optimization methods like Genetic Algorithms (GA), which play a bigger part in dramatically boosting an existing system's performance.

In our past efforts [18–20], the algorithm GA+DDPG+HER was discovered. The findings of this study are displayed in [21]. [22] is one example of work that is closely similar. More proof that efficiency can be significantly increased when a GA is employed to automatically tune the hyperparameters for DDPG+HER is provided by the results of these articles. The difference can significantly affect how long it takes a learning agent to learn

something new.

In this work, a novel automatic hyperparameter tuning algorithm [18] is applied to DDPG+HER from [23]. The algorithm is then applied to 4 existing as well as 3 custom-built robotic manipulator gym environments. Further, the whole algorithm is analyzed at various stages to check the effectiveness of the approach in improving the overall efficiency of the learning process. The final results strengthen our claim and provide enough evidence that automating the hyperparameter tuning process is extremely important, and does decrease the learning time by as much as 57%. Lastly, we compare GA+DDPG+HER with 4 methods, applied on FetchReach. GA+DDPG+HER outperforms all of them. Open source code is available at https://github.com/aralab-unr/ga-drl-aubo-ara-lab.

Our main contributions are summarized as follows:

- Algorithm (GA+DDPG+HER [18]) is applied on 6 simulated and 1 real task. We build Aubo-i5 simulated and real custom environments for analyzing the algorithm.
- Training process is analyzed using multiple factors as the GA progresses.
- Using GA+DDPG+HER found hyperparameters, the efficiency of DDPG+HER is evaluated over 10 runs in both simulated and real manipulation tasks.
- Compared GA+DDPG+HER with existing methods.

2. Genetic Algorithm optimization for Deep Reinforcement Learning

2.1. *DDPG+HER and GA*

In this section, we present the GA+DDPG+HER algorithm, where the genetic algorithm searches through the space of hyperparameter values used in DDPG + HER for values that maximize task performance and minimize the number of training epochs. We target the following hyperparameters: discounting factor γ; polyak-averaging coefficient τ [24]; learning rate for critic network α_{critic}; learning rate for actor-network α_{actor}; percent of times a random action is taken ϵ; and the standard deviation of Gaussian noise added to not completely random actions as a percentage of maximum absolute value of actions on different coordinates η. The range of all the hyperparameters is 0-1, which can be justified using the equations following in this section.

Our experiments show that adjusting the values of hyperparameters did not increase or decrease the agent's learning in a linear or easily discernible pattern. So, a simple hill climber will probably not do well in finding optimized hyperparameters. Since GAs were designed for such poorly understood problems, we use our GA to optimize these hyperparameter values.

Specifically, we use τ, the polyak-averaging coefficient to show performance non-linearity for values of τ. τ is used in the algorithm as shown in Equation (1):

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

Equation (2) shows how γ is used in the DDPG + HER algorithm, while Equation (3) describes the Q-Learning update. α denotes the learning rate. Deep neural networks are trained based on this update equation.

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}), \tag{2}$$

$$\begin{aligned} Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \\ -Q(s_t, a_t)]. \end{aligned} \tag{3}$$

Since we have two kinds of networks, we will need two learning rates, one for the actor-network (α_{actor}), another for the critic network (α_{critic}). Equation (4) explains the use of percent of times that a random action is taken, ϵ.

$$a_t = \begin{cases} a_t^* & \textit{with probability } 1-\epsilon, \\ \textit{random action} & \textit{with probability } \epsilon. \end{cases} \tag{4}$$

Algorithm 1 GA+DDPG+HER Algorithm

Choose population of n chromosomes
Set the values of hyperparameters into the chromosome
Run the DDPG + HER to get the number of epochs for which the algorithm first reaches success rate ≥ 0.85
for all chromosome values **do**
    Initialize DDPG
    Initialize replay buffer $R \leftarrow \phi$
    for episode=1, M **do**
        Sample a goal g and initial state s_0
        for t=0, T-1 **do**
            Sample an action a_t using DDPG behavioral policy
            Execute the action a_t and observe a new state s_{t+1}
        end for
        for t=0, T-1 **do**
            $r_t := r(s_t, a_t, g)$
            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 t=1,N **do**
            Sample a minibatch B from the replay buffer R
            Perform one step of optimization using A and minibatch B
        end for
    end for
    return $1/epochs$
end for
Perform Uniform Crossover
Perform Flip Mutation at a rate of 0.1
Repeat for the required number of generations to find an optimal solution

Figure 1 shows that when the value of τ is modified, there is a change in the agent's learning, further emphasizing the need to use a GA. The original (untuned) value of τ in DDPG was set to 0.95, and we are using 4 CPUs. All the values of τ are considered up to two decimal places, to see the change in success rate with change in the value of the hyperparameter. From the plots, we can tell that there is a great scope of improvement from the original success rate.

Algorithm 1 explains the integration of DDPG + HER with a GA, which uses a population size of 30 over 30 generations. We are using *ranking selection* [25] to select parents. The parents are probabilistically based on rank, which is, in turn, decided based on the relative fitness (performance). Children are then generated using *uniform crossover* [26]. We are also using *flip mutation* [27] with a probability of mutation to be 0.1. We use a binary chromosome to encode each hyperparameter and concatenate the bits to form a chromosome for the GA. The six hyperparameters are arranged in the order: τ; γ; α_{critic}; α_{actor}; ϵ and η. Since each hyperparameter requires 11 bits to be represented to three decimal places, we need 66 bits for 6 hyperparameters. These string chromosomes then enable domain-independent crossover and mutation string operators to generate new hyperparameter values. We consider hyperparameter values up to three decimal places because small changes in values of hyperparameters cause considerable change in success rate. For example, a step size of 0.001 is considered the best fit for our

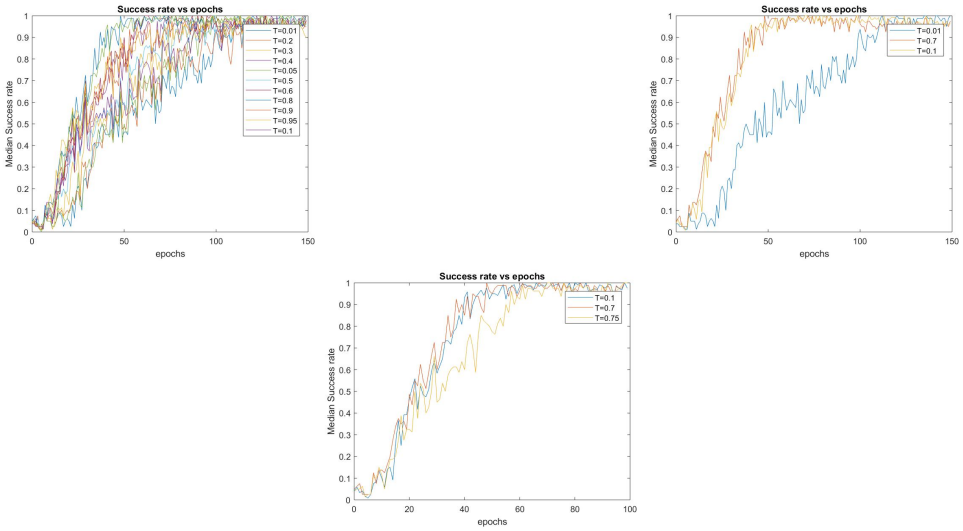


Fig. 1: Success rate vs. epochs for various τ for *FetchPick&Place-v1* task.

problem.

The fitness for each chromosome (set of hyperparameter values) is defined by the inverse of the number of epochs it takes for the learning agent to reach close to maximum success rate (≥ 0.85) for the very first time. Fitness is inverse of the number of epochs because GA always maximizes the objective function and this converts our minimization of the number of epochs to a maximization problem. Since each fitness evaluation takes significant time an exhaustive search of the 2^{66} size search space is not possible and thus we use a GA search.

3. Experimental Results

3.1. *Experimental setup*

As mentioned earlier, a chromosome is binary encoded. Each chromosome string is a combination of all the hyperparameters used in the GA. Figure 2 shows an example chromosome with 4 hyperparameters binary encoded.

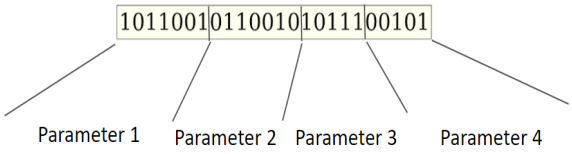
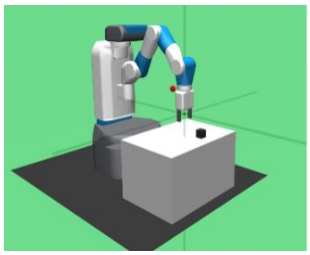


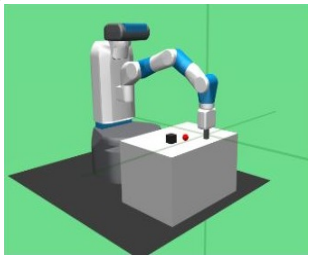
Fig. 2: Chromosome representation for the GA.

Figure 3, shows the environments used to test robot learning on five different simulation tasks: *FetchPick&Place-v1*, *FetchPush-v1*, *FetchReach-v1*, *FetchSlide-v1*, and *DoorOpening*. *AuboReach* environments are shown in figures 4 and 5 and are performed both in simulated and real experiments. We evaluate our algorithm on these 6 gym environments. The fetch environments: *FetchPick&Place*, *FetchPush*, *FetchReach*, and *FetchSlide* are from [28]. *DoorOpening* and *AuboReach* are custom-built gym environments, developed by us. The details of the 6 tasks are described below:

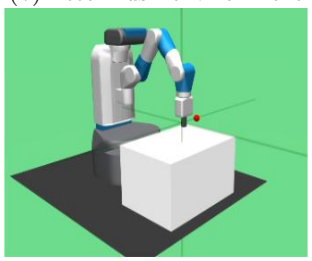
- **FetchPick&Place**: The agent picks up the box from a table and moves to the goal position, which may be anywhere on the table or the area above it.



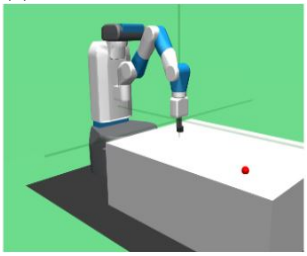
(a) FetchPick&Place environment



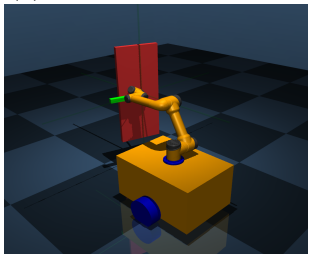
(b) FetchPush environment



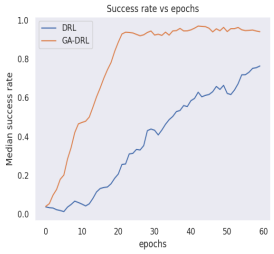
(c) FetchReach environment



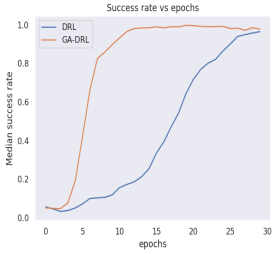
(d) FetchSlide environment



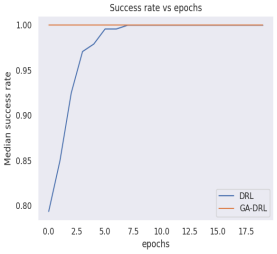
(e) Door Opening environment



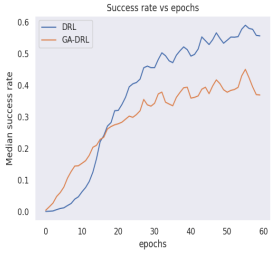
(f) FetchPick&Place plot



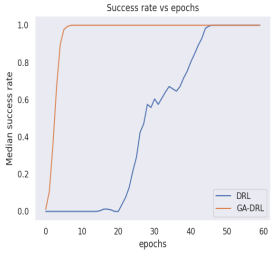
(g) FetchPush plot



(h) FetchReach plot



(i) FetchSlide plot



(j) DoorOpening plot

Fig. 3: Environments and the corresponding DDPG+HER vs GA+DDPG+HER plots, when all the 6 hyperparameters are found by GA. All plots are averaged over 10 runs. DRL stands for DDPG+HER.

- **FetchPush**: A box is kept in front of the agent. It pushes or rolls the box to the goal position on the table. The agent is not allowed to pick up the box.
- **FetchReach**: The agent has to move the end-effector to the goal position in the area around it.
- **FetchSlide**: A puck is placed on a slippery table within the reach of the agent. It has to hit the puck with such a force that it (puck) comes to rest at the goal position due to friction.
- **DoorOpening**: A simulated Aubo i5 manipulator is placed within the reach of a door, having a door handle facing the robot. The task is to push open the door by putting the force in the area of the door handle.
- **AuboReach**: A simulated/real Aubo i5 manipulator learns to reach a goal joint configuration and pick up the object using a gripper.

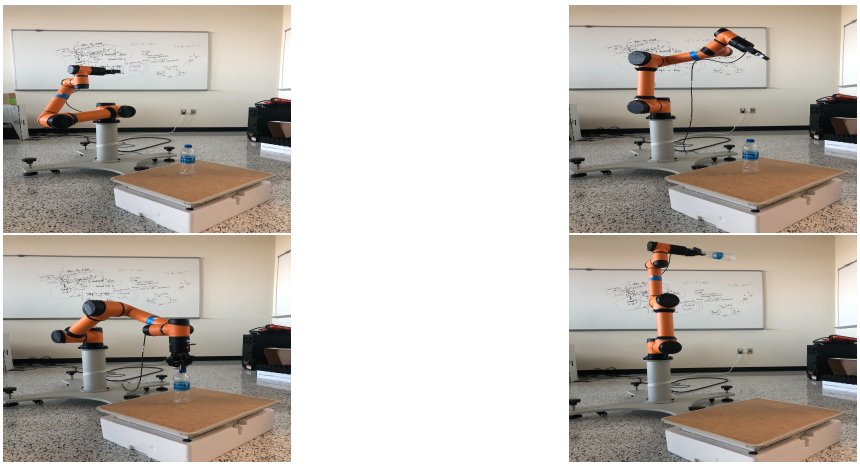


Fig. 4: AuboReach environment performing a task in a real experiment, using most accurate policy learned from GA+DDPG+HER.

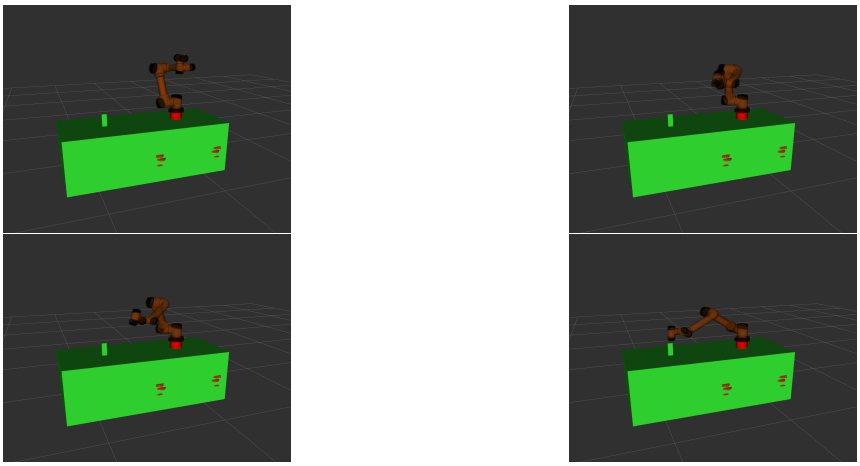
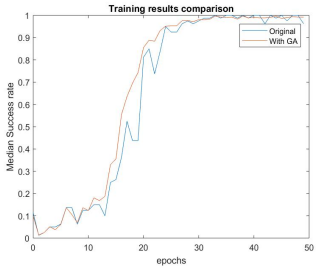


Fig. 5: AuboReach environment performing a task in a simulated experiment, using best policy learned using GA+DDPG+HER.

3.2. *Running GA*

We ran the GA separately on these environments to check the effectiveness of our algorithm and compared performance with the original values of the hyperparameters. Figure 6 (a) shows the result of our experiment with *FetchPush-v1*. We let the system run with the GA to find the best values of hyperparameters τ and γ. Since the GA is probabilistic, we show results from ten runs of the GA and the results show that the optimized hyperparameters found by the GA can lead to better performance. The learning agent can run faster and can reach the maximum success rate faster. In Figure 6 (b), we show one learning run for the original hyperparameter set and the average learning over these ten different runs of the GA. The results shown in figure 6 show changes when only two hyperparameters are being optimized as we tested and debugged the genetic algorithm, we can see the possibility for performance improvement. Our results from optimizing all five hyperparameters justify this optimism and are described next.

(a) GA+DDPG+HER over 10 runs, vs. Original



(b) GA+DDPG+HER averaged over 10 runs, vs. Original

Fig. 6: Success rate vs. epochs for *FetchPush-v1* task when τ and γ are found using the GA.

The GA was then run to optimize all hyperparameters and these results were plotted in Figure 3 for all the tasks. Table 1 compares the GA found hyperparameters with the original hyperparameters used in the RL algorithm. Though the learning rates α_{actor} and α_{critic} are the same as their original values, the other four hyperparameters have different values than the original. The plots in figure 3 show that the GA found hyperparameters outperformed the original hyperparameters, indicating that the learning agent was able to learn faster. All the plots in the above-mentioned figure are averaged over ten runs.

		All environments except Aubo-i5	Aubo-i5 - Fixed Initial and Target state	Aubo-i5 - Random Initial and Target state
Hyperparameters	DDPG+HER	GA+DDPG+HER	GA+DDPG+HER	GA+DDPG+HER
γ	0.98	0.928	0.949	0.988
τ	0.95	0.484	0.924	0.924
α_{actor}	0.001	0.001	0.001	0.001
α_{critic}	0.001	0.001	0.001	0.001
ϵ	0.3	0.1	0.584	0.912
η	0.2	0.597	0.232	0.748

Table 1: DDPG+HER vs. GA+DDPG+HER values of hyperparameters.

GA+DDPG+HER hyperparameters (as in table 1) were also applied on a custom-built gym environment for Aubo-i5 robotic manipulator, as shown in figures 4 and 5. This environment uses *MOVEit* package to control the motors, however DDPG+HER acts as a brain for its movement. Initially, the results were not as expected. Each epoch was taking several hours (> 10-15 hours) to complete. We did not run the whole learning since it

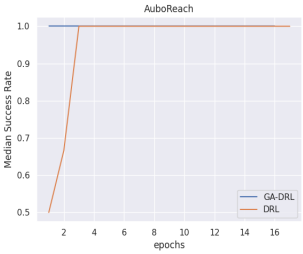
could take several weeks to complete. The same applied to DDPG+HER hyperparameters. This is primarily because both in simulation and real experiment, the movement speed of Aubo i5 robotic manipulator was kept slow to avoid any unexpected sudden movement, which in turn could cause injury. Also, planning and execution steps were involved in the successful completion of each action on AuboReach environment. Unlike other gym environments discussed in this paper, AuboReach could only work with a single CPU. This is because other environments were implemented in MuJoCo, and could easily be run with maximum possible CPUs. MuJoCo can create multiple instances for training, resulting in faster learning. AuboReach needs to perform one action at a time, which is similar to a real robot. These factors make this environment time-consuming to train.

3.3. *Modifications required for AuboReach*

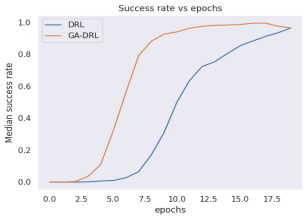
The GA+DDPG+HER hyperparameters were then applied on AuboReach environment, but only with the values of actions. This means that the robot was not running to perform the action in both simulated or real experiments. This is reliable because each action decided by the DDPG+HER algorithm is reachable by the robot. We say this because of planning and execution steps involved in the movement of the robot. This also avoids any possible collision, which could've happened, had these steps been missing. Now, each epoch took less than a minute to complete, which is significant reduction in training time, making it feasible to train in this environment.

Now that some of the environment difficulties were overcome, the GA+DDPG+HER hyperparameters (table 1) were applied again. These hyperparameters did not outperform the performance of the original hyperparameters. We believe that is because this environment is too complex and different from other environments. We considered yet more factors to make sure the environment is trainable. This environment uses four joints for training and testing (instead of six). The ones used are: *shoulder*, *forearm*, *upper-arm*, and *wrist1*. This was intended to make sure that the learning can be completed in limited time. Each of the joints can go from -1.7 to 1.7 radians. The initial and the reset state of the robot was set at an upright position, i.e., [0, 0, 0, 0].

For better learning and quick hyperparameters search, in addition to some tweaks to the environment, the GA+DDPG+HER algorithm was also modified. The success was re-defined to consider ten successful epochs. This means that the 100% success rate for ten successive epochs was considered as the success for the GA. It was experimentally found that learning never converged if α_{actor} and α_{critic} are greater than 0.001. So, α_{actor} and α_{critic} were capped at 0.001. Four CPUs could be used here since multi-threading can happen when using only action values. AuboReach considered the DDPG+HER-decided joint states as a success if the combined difference between target and the achieved joint states is less than 0.1 radians. The target joint states were set at [-0.503, 0.605, -1.676, 1.391]. With these modifications to the algorithm, we were able to find a new set of hyperparameters, as in Table 1. The difference between success rates of DDPG+HER and GA+DDPG+HER during training is demonstrated by figure 7a. Clearly, the GA+DDPG+HER performs better than DDPG+HER.



(a) Trained initial and target state



(b) Training is done with random initial and target joint states with 1 CPU

Fig. 7: Success rate vs. epochs for the *AuboReach* task. This plot is an average of over ten runs.

Once the GA+DDPG+HER have been found the optimal hyperparameters for the AuboReach environment, training was run again, using four CPUs, to find the optimal policy. This policy was then applied to the robots in simulated and real experiments. The important thing to note here is that CPU usage was updated to one

for testing. In both experiments, the robot was successfully able to go from the trained initial to target joint spaces. The environment is limited to only one possible path because randomness was not introduced in training. Since both, DDPG+HER and GA+DDPG+HER, ultimately reach a success rate of 100%, no difference was observed during testing. The main difference lies in how quickly the environment can learn using the given set of hyperparameters.

In yet another experiment, the AuboReach environment was modified to train on random joint states. Due to the modification, the robot can effectively start and reach targets at random joint states during testing. GA was run on this environment and hyperparameters found by the GA is as listed in the table 1. The plot of GA+DDPG+HER is still better than DDPG+HER, as shown in figure 7b. Figures 4 and 5 show the robot in action as it performs the task of picking the object in real and simulated experiments respectively.

Applying GA+DDPG+HER on AuboReach environment also became an instance of automatic hyperparameter tuning for DDPG+HER, and hence increased the performance of the algorithm.

3.4. *Training evaluation*

It is significantly important to monitor the progress of a GA as it is running to optimize the performance of the system. From the way GA's work, some of the chromosomes will outperform the others. Hence, it is expected that the graph of performance will not be a smooth increase curve. Some of the fitness function evaluations output a zero, suggesting that the chromosome is unfit for use. Despite having non-smooth curve, it is also expected that the overall performance of the system will increase as the GA progresses.

We generated several plots to monitor GA's progress in search of optimal hyperparameters. Figures in [21] depicts the increasing performance of the system as the GA advances. The factors considered for evaluating the training performance are: median success rate over fitness function evaluations, total reward over episodes, and epochs to reach the goal over fitness function evaluations. It can be observed that the overall performance of the system is increasing. The total reward is increasing while episodes and epochs taken by the agent to reach the goal are decreasing with fitness function evaluations. This confirms that the GA is on the right track for finding the optimal hyperparameter values. Since each GA run takes several hours to several days of run time, for plotting purposes, we have plotted results for only one run and for a limited duration of a GA run. The GA was stopped once we started seeing improvement.

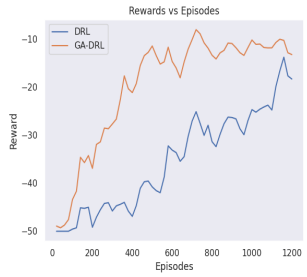
Now that the GA performed as expected, next, we will evaluate the efficiency of the system using the GA found hyperparameters.

3.5. *Efficiency evaluation*

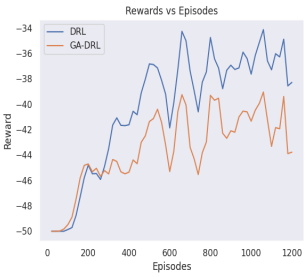
To evaluate and compare the efficiency of the GA+DDPG+HER algorithm for training the agent to do a task, we have generated data for various hyperparameters. These hyperparameters are good indicators of the efficiency of the algorithm. Figure 8 shows that the total reward has been significantly improved for most of the training tasks. Higher reward notably increase the efficiency of the DDPG+HER algorithm. The agent can learn much faster since it is guided much faster towards the desired task. We averaged these plots over ten runs to have an unbiased evaluation. FetchSlide environment performed worse with GA+DDPG+HER. We believe that this is because of the complexity of the task. For representation in the tables, the tasks that did not reach the goal while training, we considered the maximum number used for that hyperparameter.

Further, we generate more data to evaluate the episodes, running time (s), steps, and epochs for an agent to learn the desired goal. This data is shown in tables 2-5 (also shown in [21]). The data in the tables are averaged over ten runs. Table 2 compares the number of episodes taken by an agent to reach the goal. The numbers in bold indicate better performance and most of the environments perform substantially better than the DDPG+HER algorithm. FetchPush environment alone takes about 54.34% fewer episodes to learn the task.

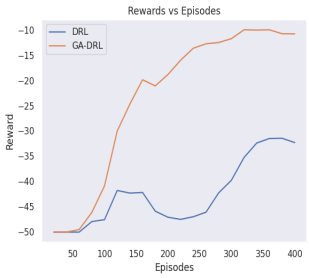
Running time is yet another factor to be considered to evaluate the efficiency of a DDPG+HER algorithm. Time is counted in seconds. The lesser time it takes to learn the task, the better the algorithm. Table 3 exhibits that the running time was less for GA+DDPG+HER algorithm for most of the environments. FetchPush, for



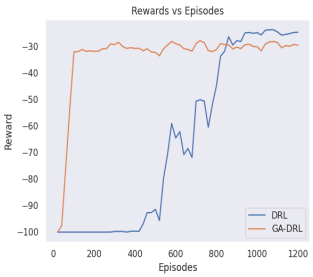
(a) FetchPick&Place



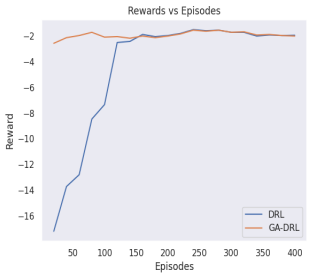
(d) FetchSlide



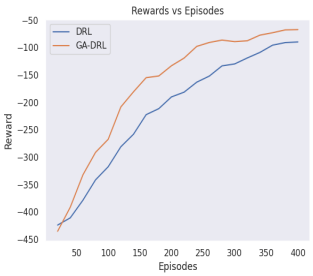
(b) FetchPush



(e) DoorOpening



(c) FetchReach



(f) AuboReach

Fig. 8: DDPG+HER vs. GA+DDPG+HER efficiency evaluation plots (Total reward vs episodes) when all the six hyperparameters are found by GA. All plots are averaged over ten runs.

Method	FetchPick&Place	FetchPush	FetchReach	FetchSlide	DoorOpening	AuboReach
DDPG+HER	6,000	2,760	100	**4,380**	960	320
GA+DDPG+HER	**2,270**	**1,260**	**60**	6,000	**180**	**228**

Table 2: Efficiency evaluation: Average (over ten runs) episodes comparison to reach the goal, for all the tasks.

example, takes about 57.004% less time with GA+DDPG+HER algorithm.

Method	FetchPick&Place	FetchPush	FetchReach	FetchSlide	DoorOpening	AuboReach
DDPG+HER	3069.981	1314.477	47.223	**2012.645**	897.816	93.258
GA+DDPG+HER	**1224.697**	**565.178**	**28.028**	3063.599	**167.883**	**66.818**

Table 3: Efficiency evaluation: Average (over 10 runs) running time (s) comparison to reach the goal, for all the tasks.

Average steps to reach the goal is yet another factor in assessing and studying the effectiveness of GA+DDPG+HER algorithm. Table 4 reveals the average number of steps taken by an agent in each environment. Most of the environments outmatch the DDPG+HER performance, with the exception of FetchSlide environment. FetchPush, for example, takes about 54.35% less number of steps with GA+DDPG+HER algorithm.

Method	FetchPick&Place	FetchPush	FetchReach	FetchSlide	DoorOpening	AuboReach
DDPG+HER	300,000	138,000	5000	**219,000**	48000	65,600
GA+DDPG+HER	**113,000**	**63,000**	**3000**	300,000	**9000**	**46,000**

Table 4: Efficiency evaluation: Average (over ten runs) steps comparison to reach the goal, for all the tasks.

The last hyperparameter used to contrast the competence of DDPG+HER and GA+DDPG+HER algorithms are the number of epochs taken by the agent to reach the goal. Table 5 presents average epochs for all the environments. Almost all the environments exceed efficacy with GA+DDPG+HER. FetchPush, for example, takes about 54.35% less number of epochs with GA+DDPG+HER.

Method	FetchPick&Place	FetchPush	FetchReach	FetchSlide	DoorOpening	AuboReach
DDPG+HER	60	27.6	5	**43.8**	47	16
GA+DDPG+HER	**22.6**	**12.6**	**3**	60	**8**	**11.4**

Table 5: Efficiency evaluation: Average (over 10 runs) epochs comparison to reach the goal, for all the tasks.

Next, we present the overall analysis of the GA+DDPG+HER algorithm compared to DDPG+HER.

3.6. *Analysis*

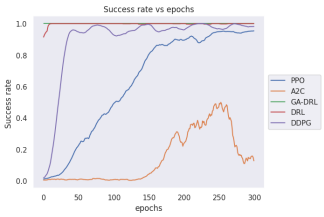


Fig. 9: GA+DDPG+HER comparison with PPO [29], A2C [30], DRL (DDPG [16] + HER [17]) and DDPG [16], on FetchReach environment. All plots are averaged over 2 runs.

In the previous sub-sections, we presented several results and the mechanism to judge the efficaciousness of GA+DDPG+HER compared with DDPG+HER. Overall, GA+DDPG+HER performed better than DDPG+HER, with an exception of the FetchSlide environment. The average comparison tables show that each environment can assume different values of the evaluation parameters. This is governed by the type of task agent is trying to learn. While most of the tasks outperformed DDPG+HER with more than a 50% increase in efficiency, FetchSlide performed worse than DDPG+HER. This performance is also attributed to the goal of the task. This task is unique in the sense that the end-effector does not physically go to the target position to place the box. GA+DDPG+HER was evaluated using multiple parameters and that too by taking an average of over ten runs. This provides enough evidence that GA+DDPG+HER performed better than DDPG+HER. Figures 3 and 7b further strengthens our claim by showing that the task in most of the environments can be learned much quicker when GA+DDPG+HER is used. We also compare FetchReach results with four other methods in figure 9. GA+DDPG+HER beats the performance of all of these methods.

4. Conclusion and Future Work

This paper showed initial results that demonstrated that a genetic algorithm can tune reinforcement learning algorithm hyperparameters to achieve better performance, illustrated by faster learning rates at six manipula-

tion tasks. We discussed existing work in reinforcement learning in robotics, presented the GA+DDPG+HER algorithm to optimize the number of epochs required to achieve maximal performance, and explained why a GA might be suitable for such optimization. Initial results bore out the assumption that GAs are a good fit for such hyperparameter optimization and our results on the six manipulation tasks show that the GA can find hyperparameter values that lead to faster learning and better (or equal) performance at our chosen tasks. We compared GA+DDPG+HER with existing methods and our method proved to the best of all.

We provided further evidence that heuristic search as performed by genetic and other similar evolutionary computing algorithms are a viable computational tool for optimizing reinforcement learning and sensor odometry performance. Adaptive Genetic Algorithms can also be deployed to have different sets of hyperparameters during the process of running the system. This may point towards online hyperparameter tuning, which will help any system have better performance, irrespective of the domain or type of testing environment.

Acknowledgments

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.

References

1. H. Ahmed, H. M. La, and K. Tran, "Rebar detection and localization for bridge deck inspection and evaluation using deep residual networks," *Automation in Construction*, vol. 120, p. 103393, 2020.
2. U. H. Billah, H. M. La, and A. Tavakkoli, "Deep learning-based feature silencing for accurate concrete crack detection," *Sensors*, vol. 20, no. 16, p. 4403, 2020.
3. A. Sehgal, M. Sehgal, H. M. La, and G. Bebis, "Deep learning hyperparameter optimization for breast mass detection in mammograms," in *International Symposium on Visual Computing*. Springer, 2022.
4. H. AHMED and H. M. LA, "Steel defect detection in bridges using deep encoder-decoder networks," *STRUCTURAL HEALTH MONITORING 2021*, 2021.
5. H.-D. Bui, H. Nguyen, H. M. La, and S. Li, "A deep learning-based autonomous robot manipulator for sorting application," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2020, pp. 298–305.
6. H. Ahmed, H. M. La, and G. Pekcan, "Rebar detection and localization for non-destructive infrastructure evaluation of bridges using deep residual networks," in *International Symposium on Visual Computing*. Springer, 2019, pp. 631–643.
7. U. H. Billah, A. Tavakkoli, and H. M. La, "Concrete crack pixel classification using an encoder decoder based deep learning architecture," in *International Symposium on Visual Computing*. Springer, 2019, pp. 593–604.
8. U. H. Billah, H. M. La, N. Gucunski, and A. Tavakkoli, "Classification of concrete crack using deep residual network," in *9th International Conference on Structural Health Monitoring of Intelligent Infrastructure: Transferring Research into Practice, SHMII 2019*. International Society for Structural Health Monitoring of Intelligent ..., 2019, pp. 1442–1447.
9. H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2019, pp. 590–595.
10. R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.
11. J. Tebbe, L. Krauch, Y. Gao, and A. Zell, "Sample-efficient reinforcement learning in robotic table tennis," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 4171–4178.
12. J. Xu, B. Li, B. Lu, Y.-H. Liu, Q. Dou, and P.-A. Heng, "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)*, 2021, pp. 1821–1828.
13. Z.-Y. Chiu, F. Richter, E. K. Funk, R. K. Orosco, and M. C. Yip, "Bimanual regrasping for suture needles using reinforcement learning for rapid motion planning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 7737–7743.

14. E. Marchesini, D. Corsi, and A. Farinelli, "Benchmarking safe deep reinforcement learning in aquatic navigation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 5590–5595.
15. A. Sehgal, M. Sehgal, and H. M. La, "Aacher: Assorted actor-critic deep reinforcement learning with hindsight experience replay," *arXiv preprint arXiv:2210.12892*, 2022.
16. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
17. M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.
18. A. Sehgal, H. La, S. Louis, and H. Nguyen, "Deep reinforcement learning using genetic algorithm for parameter optimization," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2019, pp. 596–601.
19. A. Sehgal, "Genetic algorithm as function optimizer in reinforcement learning and sensor odometry," Master's thesis, University of Nevada, Reno, 2019.
20. A. Sehgal, N. Ward, H. La, and S. Louis, "Automatic parameter optimization using genetic algorithm in deep reinforcement learning for robotic manipulation tasks," *arXiv preprint arXiv:2204.03656*, 2022.
21. A. Sehgal, N. Ward, H. La, C. Papachristos, and S. Louis, "Ga+ddpg+her: Genetic algorithm-based function optimizer in deep reinforcement learning for robotic manipulation tasks," in *2022 IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2022.
22. A. Sehgal, A. Singandhupe, H. M. La, A. Tavakkoli, and S. J. Louis, "Lidar-monocular visual odometry with genetic algorithm for parameter optimization," in *Advances in Visual Computing*, G. Bebis *et al.*, Eds. Cham: Springer International Publishing, 2019, pp. 358–370.
23. P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.
24. B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992.
25. D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 69–93.
26. G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the third international conference on Genetic algorithms*. Morgan Kaufmann Publishers, 1989, pp. 2–9.
27. D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.
28. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.
29. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
30. V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.