GA-DRL: Genetic Algorithm-Based Function Optimizer in Deep Reinforcement Learning for Robotic Manipulation Tasks

Adarsh Sehgal, Nicholas Ward, Hung Manh La, Christos Papachristos, Sushil Louis

***Abstract*— 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 parameters can significantly impact the overall learning process. In this paper, we proposed a Genetic Algorithm-based Deep Deterministic Policy Gradient and Hindsight Experience Replay method (called GA-DRL) to find near-optimal values of learning parameters. We used the proposed GA-DRL method on fetch-reach, slide, push, pick and place, and door opening in robotic manipulation tasks. With some modifications, our proposed GA-DRL method was also applied to the aubo-reach environment. Our experimental evaluation shows that our method leads to significantly better performance, faster than the original algorithm. Also, we provide evidence that GA-DRL performs better than the existing methods.**

I. Introduction

Recently, Reinforcement Learning (RL) [1] has been put to significant uses such as robotic table tennis [2], surgical robot planning [3], rapid motion planning in bimanual regrasping for suture needles [4] and Aquatic Navigation [5]. Each of these applications make use of RL as an encouraging substitute to automating manual effort.

In this paper, we are specifically using DDPG [6] combined with HER [7] to train Deep Reinforcement Learning (DRL) policies. DDPG + HER suffer from an efficiency problem. The performance of many robotic manipulation tasks can be improved by using a better set of DDPG and HER parameters. This performance can be measured based on the number of epochs it takes for the learning agent to learn a given robotic task. Optimization algorithms such as Genetic Algorithms (GA) can help search for near-optimal parameter values and thus assume a greater role in significantly increasing the performance of an existing system.

Some of the closely related work includes [8], [9] and [10]. The results from these papers provide more evidence that when a GA is used to automatically tune the hyper-parameters for DRL, efficiency can be largely improved. The difference can greatly impact the time it takes for a learning agent to learn.

In this work, we build a novel automatic parameter tuning algorithm, which is applied to DRL from [11]. 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 parameter tuning process is extremely important, and does decrease the learning time by as much as 57%. Lastly, we compare GA-DRL with 4 methods, applied on FetchReach. GA-DRL 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:

- Novel automatic parameter tuning algorithm.
- Algorithm 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-DRL found parameters, the efficiency of DRL is evaluated over 10 runs in both simulated and real manipulation tasks.
- Compared GA-DRL with existing methods.

II. Genetic Algorithm Optimization for Deep Reinforcement Learning

A. DDPG + HER and GA

In this section, we present the primary contribution of our paper: the genetic algorithm searches through the space of parameter values used in DDPG + HER for values that maximize task performance and minimize the number of training epochs. We target the following parameters: discounting factor γ; polyak-averaging coefficient τ [12]; 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 parameters is 0-1, which can be justified using the equations following in this section.

Our experiments show that adjusting the values of parameters 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 parameters.

Adarsh Sehgal, Nicholas Ward, and Dr. Hung La are with the Advanced Robotics and Automation (ARA) Laboratory. Drs. La, Louis and Papachristos are professors of the Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557, USA. Corresponding author: Hung La, email: hla@unr.edu.

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.

Since GAs were designed for such poorly understood problems, we use our GA to optimize these parameter 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):

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'},$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}. \quad (1)$$

Algorithm 1 Proposed GA-DRL Algorithm

Choose population of n chromosomes
Set the values of parameters 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

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.

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

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

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.

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

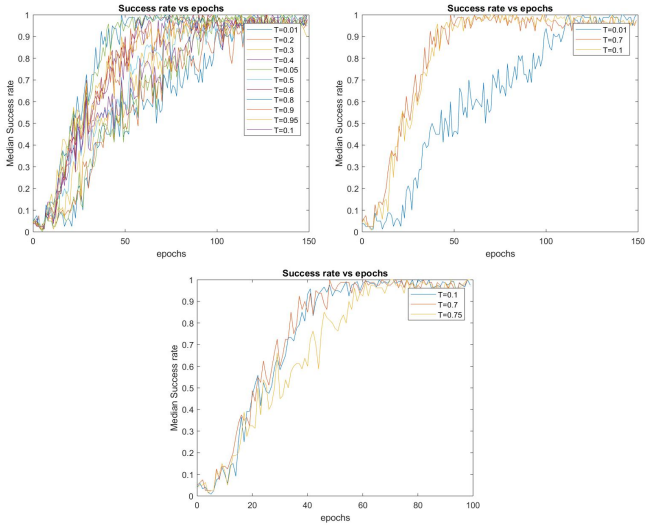


Fig. 1: Success rate vs. epochs for various τ for *FetchPick&Place-v1* task.

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 parameter. 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* [13] 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* [14]. We are also using *flip mutation* [15] with a probability of mutation to be 0.1. We use a binary chromosome to encode each parameter and concatenate the bits to form a chromosome for the GA. The six parameters are arranged in the order: τ; γ; α_{critic}; α_{actor}; ϵ and η. Since each parameter requires 11 bits to be represented to three decimal places, we need 66 bits for 6 parameters. These string chromosomes then enable domain-independent crossover and mutation string operators

to generate new parameter values. We consider parameter values up to three decimal places because small changes in values of parameters cause considerable change in success rate. For example, a step size of 0.001 is considered the best fit for our problem.

The fitness for each chromosome (set of parameter 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.

III. Experimental Results

A. Experimental setup

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

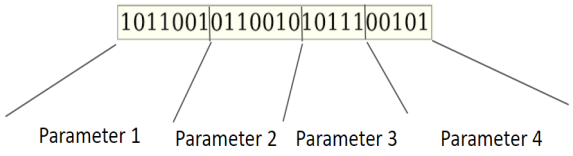
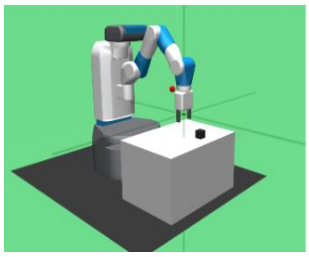


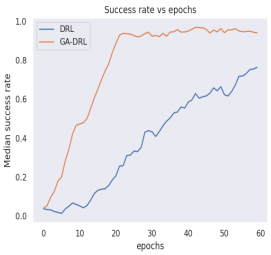
Fig. 2: Chromosome representation for the GA.

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 [16]. *DoorOpening* and *AuboReach* are custom-built gym environments, developed by us. The details of the 6 tasks are described below:

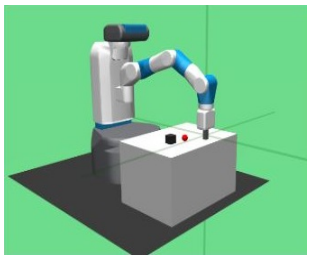
- **FetchPick&Place**: The agent picks up the box from a table and moves to the goal position, which may be anywhere on the table or the area above it.
- **FetchPush**: A box is kept in front of the agent. It pushes or rolls the box to the goal position on the table. The agent is not allowed to pick 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.



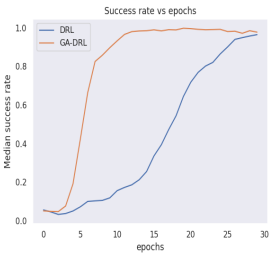
(a) FetchPick&Place environment



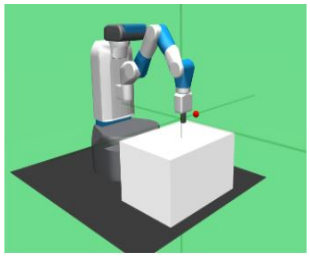
(f) FetchPick&Place plot



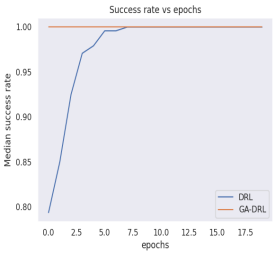
(b) FetchPush environment



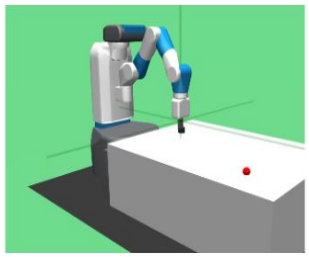
(g) FetchPush plot



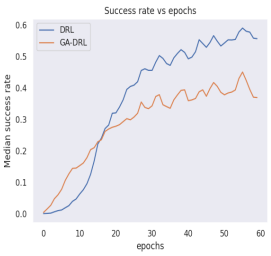
(c) FetchReach environment



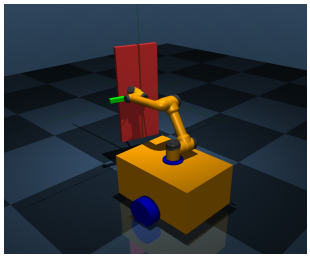
(h) FetchReach plot



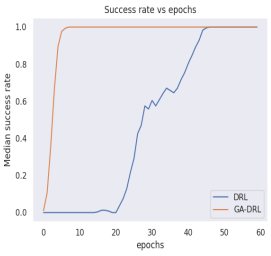
(d) FetchSlide environment



(i) FetchSlide plot



(e) Door Opening environment



(j) DoorOpening plot

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

- **AuboReach**: A simulated/real Aubo i5 manipulator learns to reach a goal joint configuration and pick up the object using a gripper.

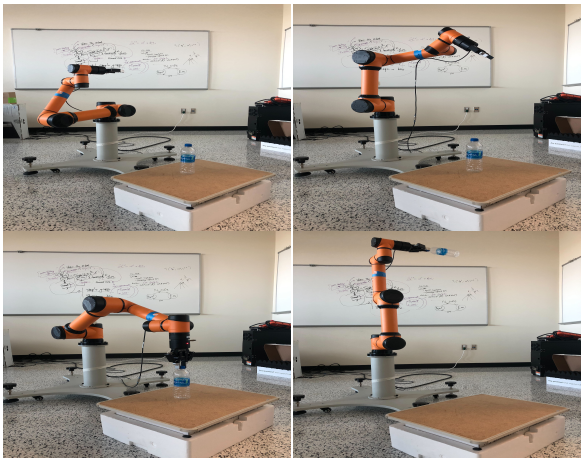


Fig. 4: AuboReach environment performing a task in a real experiment, using most accurate policy learned from GA-DRL.

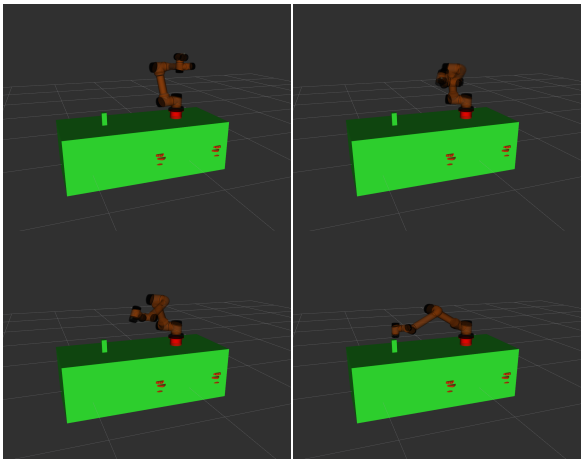
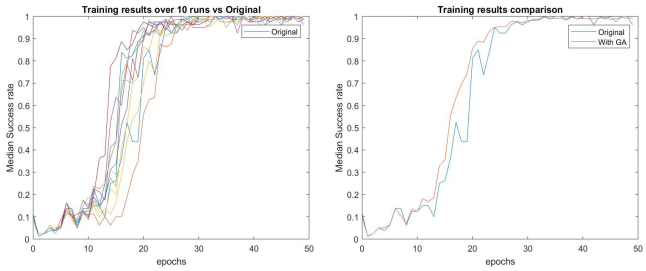


Fig. 5: AuboReach environment performing a task in a simulated experiment, using best policy learned using GA-DRL.

B. 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 parameters. 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 parameters τ and γ. Since the GA is probabilistic, we show results from ten runs of the GA and the results show that the optimized parameters 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 parameter set and the average learning over these ten different runs of the GA. The results shown in figure 6 show changes when only two parameters 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 parameters justify this optimism and are described next.



(a) GA-DRL over 10 runs, vs. Original

(b) GA-DRL 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 parameters and these results were plotted in Figure 3 for all the tasks. Table I compares the GA found parameters with the original parameters used in the RL algorithm. Though the learning rates α_{actor} and α_{critic} are the same as their original values, the other four parameters have different values than the original. The plots in figure 3 show that the GA found parameters outperformed the original parameters, 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
Parameters	DRL	GA-DRL	GA-DRL	GA-DRL
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 I: DRL vs. GA-DRL values of parameters.

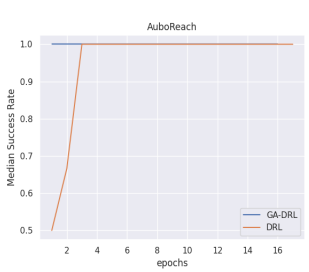
GA-DRL parameters (as in table I) were also applied on a custom-built gym environment for Aubo-i5 robotic manipulator, as shown in figures 4 and 5. This environment uses *MOVEit* package to control the motors, however DRL acts as a brain for its movement. Initially, the results were not as expected. 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 DRL parameters. 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.

C. Modifications required for AuboReach

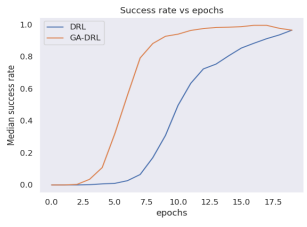
The GA-DRL parameters were then applied on AuboReach environment, but only with the values of actions. This means that the robot was not running to perform the action in both simulated or real experiments. This is reliable because each action decided by the DRL algorithm is reachable by the robot. We 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-DRL parameters (table I) were applied again. These parameters did not outperform the performance of the original parameters. 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 parameters search, in addition to some tweaks to the environment, the GA-DRL 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 DRL-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 parameters, as in Table I. The difference between success rates of DRL and GA-DRL during training is demonstrated by figure 7a. Clearly, the GA-DRL performs better than DRL.



(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-DRL have been found the optimal parameters 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, DRL and GA-DRL, 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 parameters.

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 parameters found by the GA is as listed in the table I. The plot of GA-DRL is still better than DRL, 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.

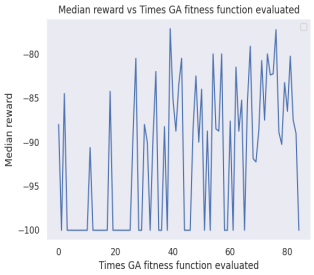
Applying GA-DRL on AuboReach environment also became an instance of automatic parameter tuning for DRL, and hence increased the performance of the algorithm.

D. Training evaluation

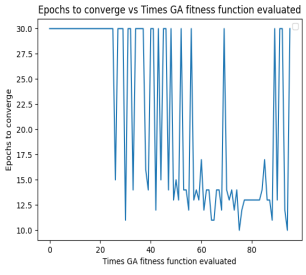
It is significantly important to monitor the progress of a GA as it is running to optimize the performance of the system. From the way GA's work, some of the chromosomes will outperform the others. Hence, it is expected that the graph of performance will not be a smooth increase curve. Some of the fitness 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 parameters. Figures 8, 9, and 10 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 parameter 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.

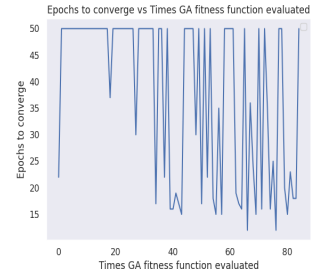
Now that the GA performed as expected, next, we will evaluate the efficiency of the system using the GA found parameters.



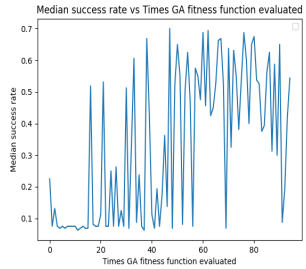
(a) FetchPick&Place - Median reward vs Times GA fitness function evaluated



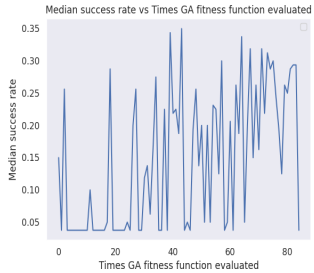
(e) FetchPush - Epochs vs Times GA fitness function evaluated



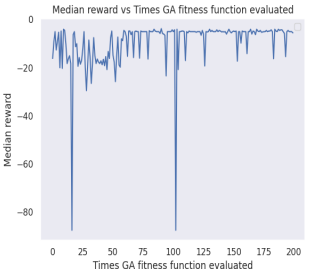
(b) FetchPick&Place - Epochs vs Times GA fitness function evaluated



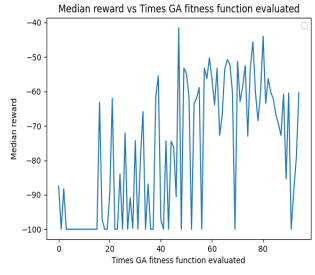
(f) FetchPush - Median success rate vs Times GA fitness function evaluated



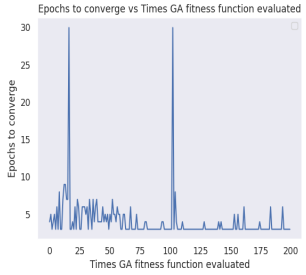
(c) FetchPick&Place - Median success rate vs Times GA fitness function evaluated



(g) FetchReach - Median reward vs Times GA fitness function evaluated

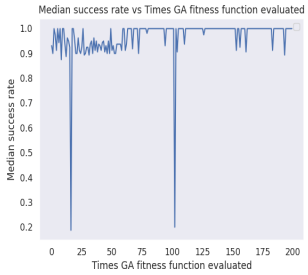


(d) FetchPush - Median reward vs Times GA fitness function evaluated

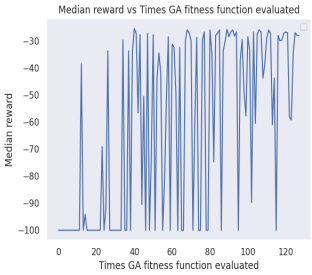


(h) FetchReach - Epochs vs Times GA fitness function evaluated

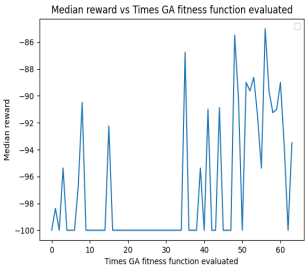
Fig. 8: GA-DRL training evaluation plots, when all the 6 parameters are found by GA. This is the result of one GA run.



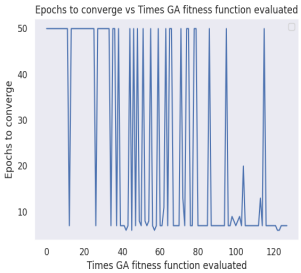
(a) FetchReach - Median success rate vs Times GA fitness function evaluated



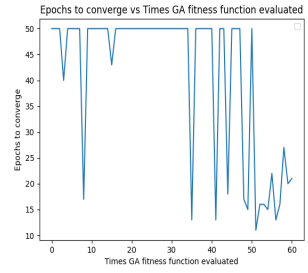
(e) DoorOpening - Median reward vs Times GA fitness function evaluated



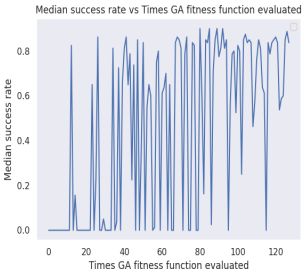
(b) FetchSlide - Median reward vs Times GA fitness function evaluated



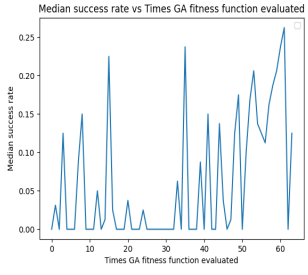
(f) DoorOpening - Epochs vs Times GA fitness function evaluated



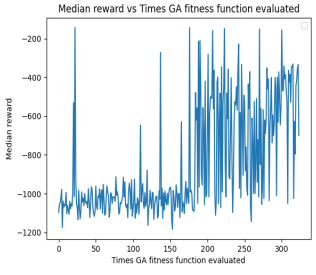
(c) FetchSlide - Epochs vs Times GA fitness function evaluated



(g) DoorOpening - Median success rate vs Times GA fitness function evaluated

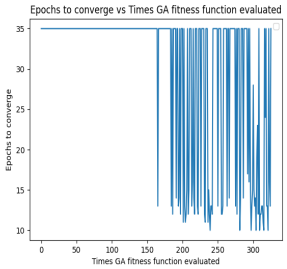


(d) FetchSlide - Median success rate vs Times GA fitness function evaluated

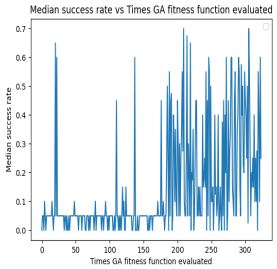


(h) AuboReach - Median reward vs Times GA fitness function evaluated

Fig. 9: GA-DRL training evaluation plots, when all the six parameters are found by GA. This is the result of one GA run.



(a) AuboReach - Epochs vs Times GA fitness function evaluated



(b) AuboReach - Median success rate vs Times GA fitness function evaluated

Fig. 10: GA-DRL training evaluation plots, when all the 6 parameters are found by GA. This is the result from one GA run.

E. Efficiency evaluation

To evaluate and compare the efficiency of the GA-DRL algorithm for training the agent to do a task, we have generated data for various parameters. These parameters are good indicators of the efficiency of the algorithm. Figure 11 shows that the total reward has been significantly improved for most of the training tasks. Higher reward notably increase the efficiency of the DRL 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-DRL. 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 parameter.

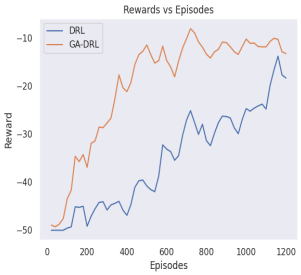
Further, we generate more data to evaluate the episodes, running time (s), steps, and epochs for an agent to learn the desired goal. This data is shown in tables II-V. The data in the tables are averaged over ten runs. Table II compares the number of episodes taken by an agent to reach the goal. The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

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

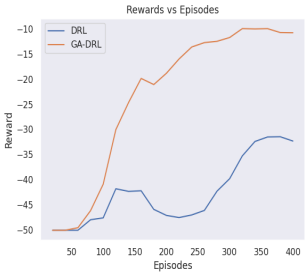
TABLE II: Efficiency evaluation: Average (over ten runs) episodes comparison to reach the goal, for all the tasks.

Running time is yet another factor to be considered to evaluate the efficiency of a DRL algorithm. Time is counted in seconds. The lesser time it takes to learn the task, the better the algorithm. Table III exhibits that the running time was less for GA-DRL algorithm for most of the environments. FetchPush, for example, takes about 57.004% less time with GA-DRL algorithm.

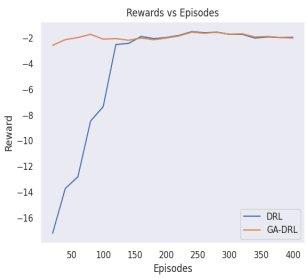
Average steps to reach the goal is yet another factor in assessing and studying the effectiveness of GA-DRL algorithm. Table IV reveals the average number of steps taken by an agent in each environment. Most of the environments outmatch the DRL performance, with the exception of



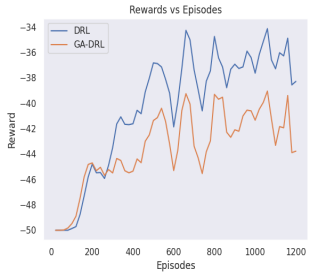
(a) FetchPick&Place



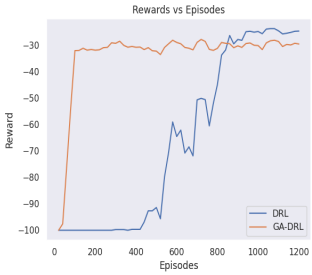
(b) FetchPush



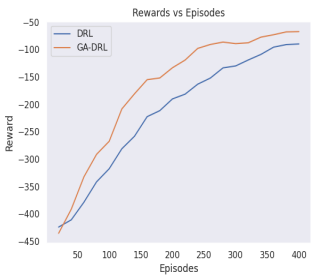
(c) FetchReach



(d) FetchSlide



(e) DoorOpening



(f) AuboReach

Fig. 11: DRL vs. GA-DRL efficiency evaluation plots (Total reward vs episodes) when all the six parameters are found by GA. All plots are averaged over ten runs.

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

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

FetchSlide environment. FetchPush, for example, takes about 54.35% less number of steps with GA-DRL algorithm.

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

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

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

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

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

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

F. Analysis

In the previous sub-sections, we presented several results and the mechanism to judge the efficaciousness of GA-DRL compared with DRL. Overall, GA-DRL performed better than DRL, 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 DRL with more than a 50% increase in efficiency, FetchSlide performed worse than DRL. 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-DRL was evaluated using multiple parameters and that too by taking an average of over ten runs. This provides enough evidence that GA-DRL performed better than DRL. 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-DRL is used. We also compare FetchReach results with four other methods in figure 12. GA-DRL beats the performance of all of these methods.

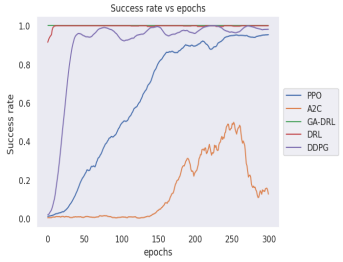


Fig. 12: GA-DRL comparison with PPO [17], A2C [18], DRL (DDPG [6] + HER [7]) and DDPG [6], on FetchReach environment. All plots are averaged over 2 runs.

IV. Conclusion and Future Work

This paper showed initial results that demonstrated that a genetic algorithm can tune reinforcement learning algorithm parameters to achieve better performance, illustrated by faster learning rates at six manipulation tasks. We discussed existing work in reinforcement learning in robotics, presented the GA-DRL 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 parameter optimization and our results on the six manipulation tasks show that the GA can find parameter values that lead to faster learning and better (or equal) performance at our chosen tasks. We compared GA-DRL 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 parameters during the process of running the system. This may point towards online parameter tuning, which will help any system have better performance, irrespective of the domain or type of testing environment.

References

[1] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the third international conference on Genetic algorithms*. Morgan Kaufmann Publishers, 1989, pp. 2–9.

[15] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[18] 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.