Deep Reinforcement Learning using Genetic Algorithm for Parameter Optimization

Adarsh Sehgal, Hung Manh La, Sushil J. Louis, Hai Nguyen

Abstract—**Reinforcement learning (RL) enables agents to take 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 use a genetic algorithm (GA) to find the values of parameters used in Deep Deterministic Policy Gradient (DDPG) combined with Hindsight Experience Replay (HER), to help speed up the learning agent. We used this method on fetch-reach, slide, push, and pick and place in robotic manipulation tasks. Our experimental evaluation shows that our method leads to better performance, faster than the original algorithm.**

I. INTRODUCTION

Q-learning methods have been applied on a variety of tasks by autonomous robots, and much research has been done in this field starting many years ago [1], with some work specific to continuous action spaces [2]–[5] and others on discrete action spaces [6]. Reinforcement Learning (RL) has been applied to locomotion [7] [8] and also to manipulation [9], [10].

Much work specific to robotic manipulators also exists [11], [12]. Some of this work used fuzzy wavelet networks [13], others used neural networks to accomplish their tasks [14] [15]. Off-policy algorithms such as the Deep Deterministic Policy Gradient algorithm (DDPG) [16] and Normalized Advantage Function algorithm (NAF) [17] are helpful for real robot systems. A complete review of recent deep reinforcement learning methods for robot manipulation is given in [18]. We are specifically using DDPG combined with Hindsight Experience Replay (HER) [19] for our experiments. Recent work on using experience ranking to improve the learning speed of DDPG + HER was reported in [20].

The main contribution of this paper is a demonstration of better final performance at several manipulation tasks using a Genetic Algorithm (GA) to find DDPG and HER parameter values that lead more quickly to better performance at these tasks. Our experiments revealed that learning algorithm parameters are non-linearly related to task performance and learning speed. Rather, success rate can vary significantly based on the values of the parameters used in RL. In the following sections, we describe the manipulation tasks, the DDPG + HER algorithms, and the parameters that affect performance for these algorithms. Initial experimental results showing performance and speed gains when using a GA to search for good parameter values then provide evidence that GAs find good parameter values leading to better task performance, faster.

The paper is organized as follows: In Section 2, we present related work. Section 3 describes the DDPG + HER algorithms. In Section 4, we describe the GA being used to find the values of parameters. Section 5 then describes our learning tasks and experiments and our experimental results. The last section provides conclusions and possible future research.

II. RELATED WORK

RL has been widely used in training/teaching both a single robot [21], [22] and a multi-robot system [23]–[26]. Previous work has also been done on both model-based and model-free learning algorithms. Applying model-based learning algorithms to real world scenarios, rely significantly on a model-based teacher to train deep network policies.

Similarly, there is also much work in GA's [27] [28] and the GA operators of crossover and mutation [29], applied to a variety of problem. GA has been specifically applied to variety of RL problems [29]–[32].

In this paper, we use model-free RL with continuous action spaces and deep neural network. Our work is built on existing work using the same techniques applied to robotic manipulator [16] [19]. Specifically, we use a GA to search for good DDPG + HER algorithm parameters and compare it with original values of parameters [33], and hence the success rates. DDPG + HER, a RL algorithm using deep neural networks in continuous action spaces has been successfully used for robotic manipulation tasks, and our GA improves on this work by finding learning algorithm parameters that needs fewer epochs (one epoch is a single pass through full training set) to learn better task performance.

III. BACKGROUND

A. Reinforcement Learning

Consider a standard RL setup consisting of a learning agent, which interacts with an environment. An environment can be described by a set of variables where S is the set of states, A is the set of actions, $p(s_0)$ is a distribution of initial states, $r : S \times A \rightarrow R$, $p(s_{t+1}|s_t, a_t)$ are transition probabilities and $\gamma \in [0, 1]$ is a discount factor.

Adarsh Sehgal, Hai Nguyen and Dr. Hung La are with the Advanced Robotics and Automation (ARA) Laboratory. Dr. Sushil Louis is professor of the Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557, USA. Corresponding author: Hung La, email: hla@unr.edu

This material is based upon work supported by the National Aeronautics and Space Administration (NASA) Grant No. NNX15AI02H issued through the NVSGC-RI program under sub-award No. 19-21, and the RID program under sub-award No. 19-29, and the NVSGC-CD program under sub-award No. 18-54. This work is also partially supported by the Office of Naval Research under Grant N00014-17-1-2558.

A deterministic policy maps from states to actions: $\pi : S \rightarrow A$. The beginning of every episode is marked by sampling an initial state s_0. For each timestep t, the agent performs an action based on the current state: $a_t = \pi(s_t)$. The performed action gets a reward $r_t = r(s_t, a_t)$, and the distribution $p(.|s_t, a_t)$ helps to sample the environments new state. The total return is: $R_t = \sum_{i=T}^{\infty} \gamma^{i-t} r_i$. The agents goal is to try to maximize its expected return $E[R_t|s_t, a_t]$ and an optimal policy denoted by π^* can be defined as any policy π^* , such that $Q^{\pi^*}(s, a) \geq Q^{\pi}(s, a)$ for every $s \in S, a \in A$ and any policy π. The optimal policy, which has the same Q-function, is called an optimal Q-function, Q^* , which satisfies the *Bellman* equation:

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

B. Deep Q-Networks(DQN)

A *Deep Q-Networks (DQN)* [34] is defined as a model free reinforcement learner, designed for discrete action spaces. In a DQN, a neural network Q is maintained, which approximates Q^*. $\pi_Q(s) = argmax_{a \in A} Q(s, a)$ denotes a greedy policy w.r.t. Q. A - greedy policy takes a random action with probability ϵ and action $\pi_Q(s)$ with probability $1 - \epsilon$.

Episodes are generated during training using a ϵ-greedy policy. A *Replay buffer* stores transition tuples (s_t, a_t, r_t, s_{t+1}) experienced during training. The neural network training is interlaced by generation of new episodes. A Loss $\mathcal{L}$ defined by $\mathcal{L} = E(Q(s_t, a_t) - y_t)^2$ where $y_t = r_t + \gamma max_{a' \in A} Q(s_{t+1}, a')$ and tuples (s_t, a_t, r_t, s_{t+1}) are being sampled from the replay buffer.

The *target network* changes at a slower pace than the main network, which is used to measure targets y_t. The weights of the target networks can be set to the current weights of the main network [34]. Polyak-averaged parameters [35] can also be used.

C. Deep Deterministic Policy Gradients (DDPG)

In *Deep Deterministic Policy Gradients (DDPG)*, there are two neural networks: an Actor and a Critic. The actor neural network is a target policy $\pi : S \rightarrow A$, and critic neural network is an action-value function approximator $Q : S \times A \rightarrow R$. The critic network $Q(s, a|\theta^Q)$ and actor network $\mu(s|\theta^{\mu})$ are randomly initialized with weights θ^Q and θ^{μ}.

A behavioral policy is used to generate episodes, which is a noisy variant of target policy, $\pi_b(s) = \pi(s) + \mathcal{N}(0, 1)$. The training of a critic neural network is done like the Q-function in DQN but where the target y_t is computed as $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$, where γ is the discounting factor. The loss $\mathcal{L}_a = -E_a Q(s, \pi(s))$ is used to train the actor network.

D. Hindsight Experience Replay (HER)

Hindsight Experience Reply (HER) tries to mimic human behavior to learn from failures. The agent learns from all episodes, even when it does not reach the original goal. Whatever state the agent reaches, HER considers that as the modified goal. Standard experience replay only stores the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ with original goal g. HER tends to store the transition $(s_t||g', a_t, r'_t, s_{t+1}||g')$ to modified goal g' as well. HER does great with extremely sparse rewards and is also significantly better for sparse rewards than shaped ones.

E. Genetic Algorithm (GA)

Genetic Algorithms (GAs) [27], [36], [37] were designed to search poorly-understood spaces, where exhaustive search may not be feasible, and where other search approaches perform poorly. When used as function optimizers, GAs try to maximize a fitness tied to the optimization objective. Evolutionary computing algorithms in general and GAs specifically have had much empirical success on a variety of difficult design and optimization problems. They start with a randomly initialized population of candidate solution typically encoded in a string (chromosome). A selection operator focuses search on promising areas of the search space while crossover and mutation operators generate new candidate solutions. We explain our specific GA in the next section.

IV. 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 [35]; learning rate for critic network α_{critic}; learning rate for actor network α_{actor}; percent of times a random action is taken ϵ; and 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 agents learning in a linear or easily discernible pattern. So, a simple hill climber will probably not do well in finding optimized parameters. 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 the performance non-linearity for values of τ . τ is used in the algorithm as show in Equation (2):

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^{Q} + (1 - \tau)\theta^{Q'}, \\ \theta^{\mu'} &\leftarrow \tau \theta^{\mu} + (1 - \tau)\theta^{\mu'}. \end{aligned} \quad (2)$$

Equation (3) shows how γ is used in the DDPG + HER algorithm, while Equation (4) describes the Q-Learning update. denotes the learning rate. Networks are trained based on this update equation.

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}), \quad (3)$$

$$\begin{aligned} Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \\ -Q(s_t, a_t)]. \end{aligned} \quad (4)$$

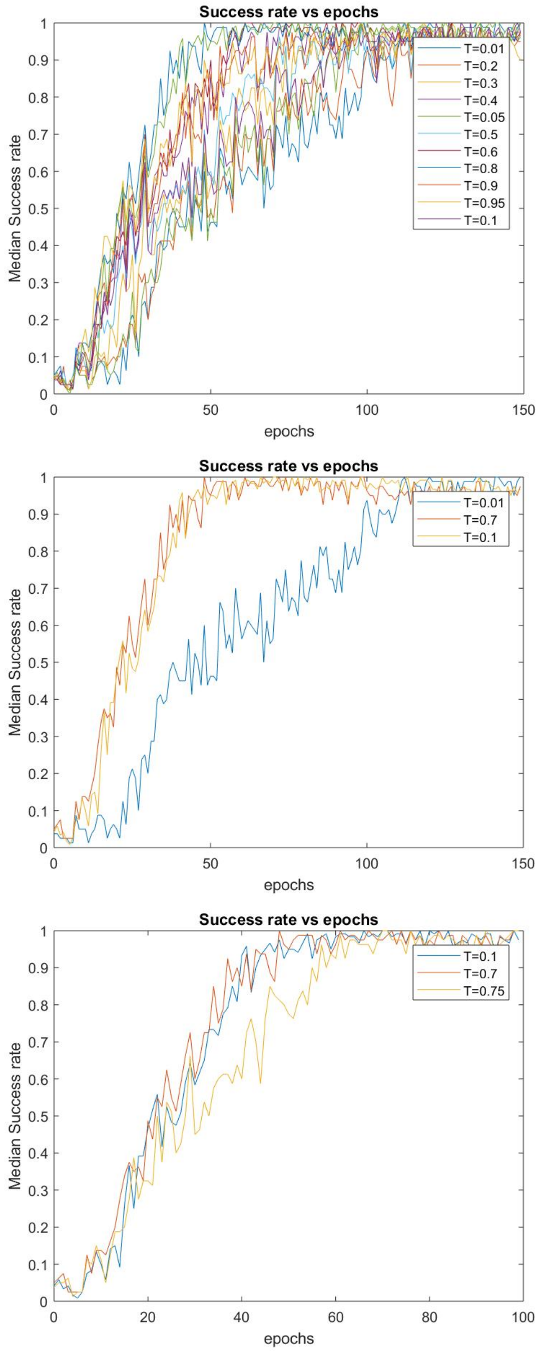
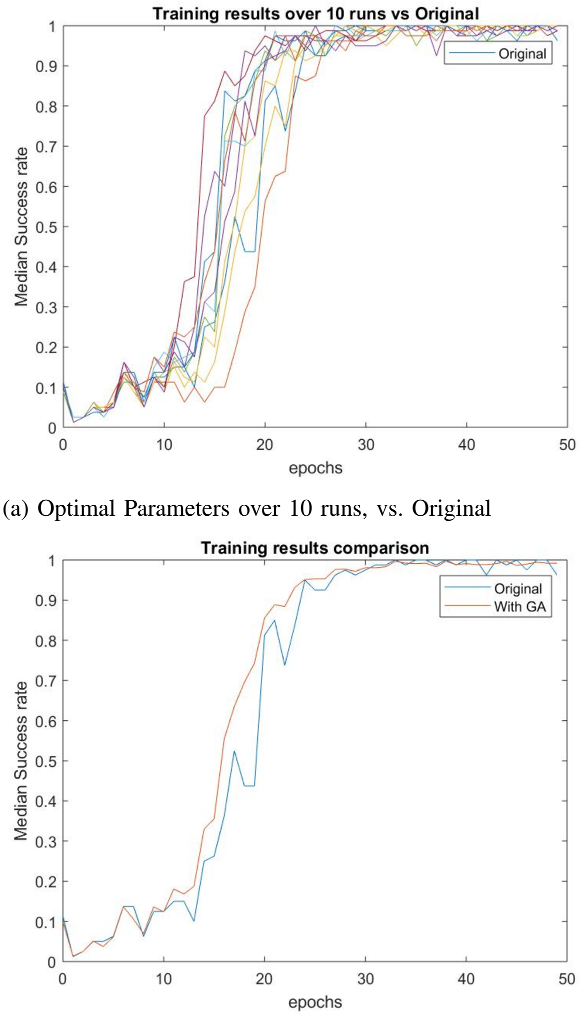


Fig. 1: Success rate vs. epochs for various τ for *FetchPick&Place-v1* task.

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 (5) explains the use of percent of times that a random action is taken, ϵ.

$$a_t = \begin{cases} a_t^* & with\ probability\ 1-\epsilon, \\ random\ action & with\ probability\ \epsilon. \end{cases} \tag{5}$$



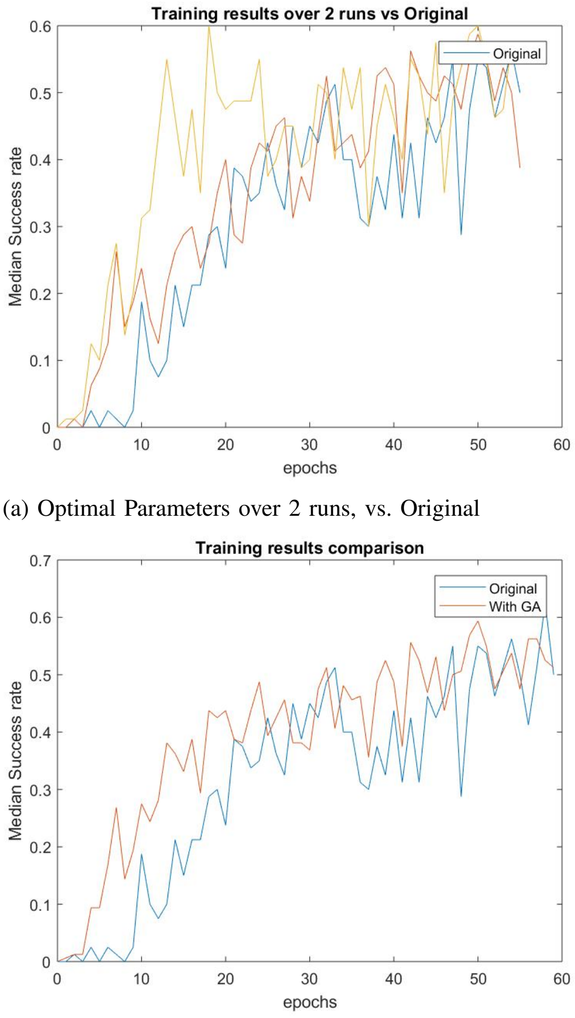
(a) Optimal Parameters over 10 runs, vs. Original

(b) Optimal Parameters averaged over 10 runs, vs. Original

Fig. 2: Success rate vs. epochs for *FetchPush-v1* task when τ and γ are found using the GA.

Figure 1 shows that when the value of τ is modified, there is a change in the agents 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, in order to see the change in success rate with change in value of the parameter. From the plots, we can clearly 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* [38] 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* [39]. We are also using *flip mutation* [37] with 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: polyak-averaging coefficient; discounting factor; learning rate for critic network;



(a) Optimal Parameters over 2 runs, vs. Original

(b) Optimal Parameters averaged over 2 runs, vs. Original

Fig. 3: Success rate vs. epochs for *FetchSlide-v1* task when τ and γ are found using the GA.

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 maximum absolute value of actions on different coordinates. 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 causes considerable change in success rate. For example, a step size of 0.001 is considered as the best fit for our problem.

The fitness for each chromosome (set of parameter values) is defined by the inverse of 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 the inverse of number of epochs because GA always maximizes the objective function and this converts our minimization of 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 we thus use GA search.

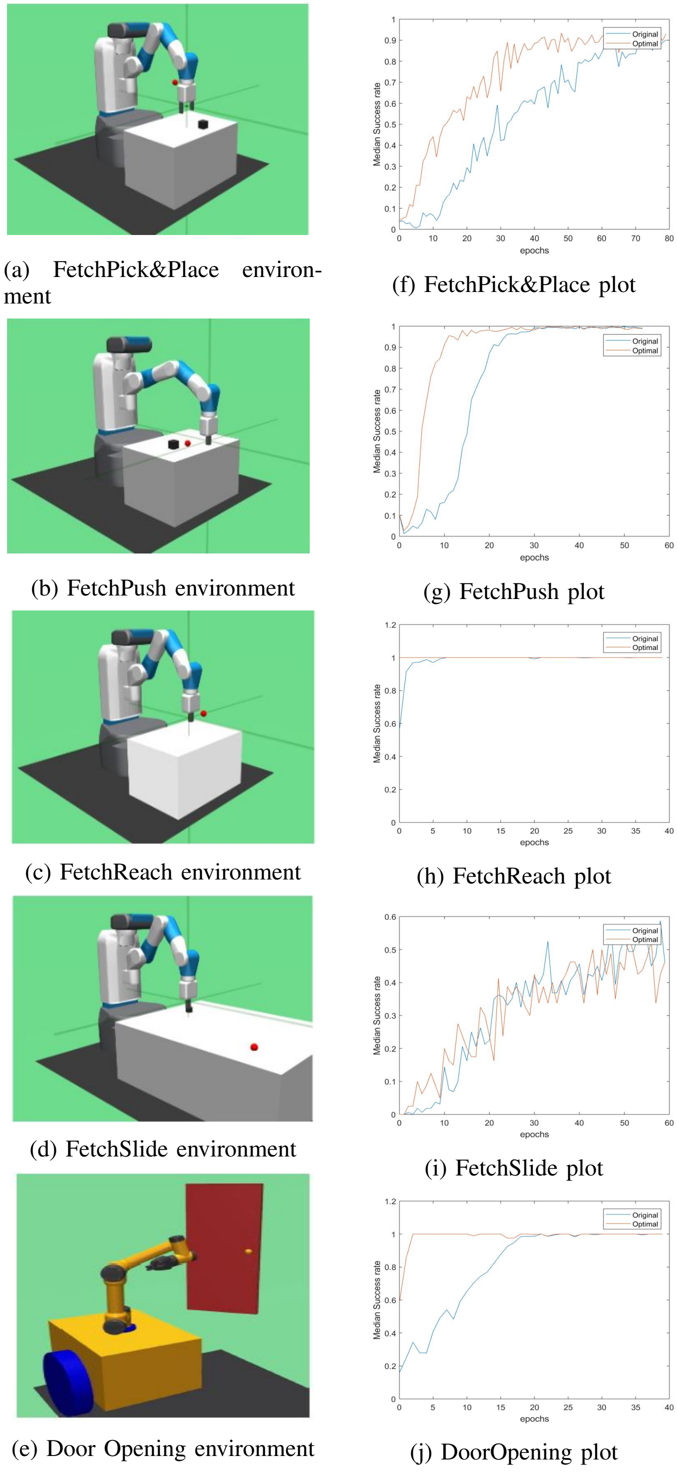
Algorithm 1 DDPG + HER and GA

```
Choose population of $n$ chromosomes
Set the values of parameters into the chromosome
Run the DDPG + HER to get number of epochs for which the algorithm first reaches success rate $\geq 0.85$
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 rate 0.1
Repeat for required number of generations to find optimal solution
```

V. Experiment and Results

Figure 4, shows the environments used to test the learning of a robot in four different tasks: *FetchPick&Place-v1*, *FetchPush-v1*, *FetchReach-v1*, and *FetchSlide-v1* . We ran GA separately on these environments to check the effectiveness of our algorithm and compared it with the original values of the parameters.

Figure 4, shows the environments used to test robot learning on four different tasks: *FetchPick&Place-v1*, *FetchPush-v1*, *FetchReach-v1*, and *FetchSlide-v1* . We ran the GA separately



(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. 4: Environments and the corresponding Original vs Optimal plots, when all the 6 parameters are found by GA

on these environments to check the effectiveness of our algorithm and compared performance with the original values of the parameters. Figure 2 (a) shows the result of our experiment with *FetchPush-v1*, while Figure 3 (a) shows the results with *FetchSlide-v1*. We let the system run with GA to find the optimal parameters and . Since the GA is probabilistic, we show results from 10 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 2 (b), we show one learning run for the original parameter set and the average learning over these 10 different runs of the GA.

Parameters	Original	Optimal
γ	0.98	0.88
τ	0.95	0.184
α_{actor}	0.001	0.001
α_{critic}	0.001	0.001
ϵ	0.3	0.055
η	0.2	0.774

TABLE I: Original vs Optimal values of parameters

Figure 3 (b) compares one run for original with averaged 2 runs for optimizing parameters τ and γ. For this task, we have run it for only 2 runs because these tasks can take a few hours for one run. The results shown in Figures 2 and 3 show changes when only two parameters are being optimized as we tested and debugged the genetic algorithm be we can see the possibility for performance improvement. Our results from optimizing all five parameters justify this optimism and are described next.

The GA was then run to optimize all parameters and these results were plotted in Figure 4 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 same as their original values, the other four parameters have different values than original. The plots in the figure 4 shows that the GA found parameters outperformed the original parameters, indicating that the learning agent was able to learn faster. All the plots in this figure are averaged over 10 runs.

VI. Discussion and Future Work

In this paper, we showed initial results that demonstrated that a genetic algorithm can tune reinforcement learning algorithm parameters to achieve better performance, faster at six manipulation tasks. We discussed existing work in reinforcement learning in robotics, presented an algorithm, which integrates DDPG + HER with GA 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 thus provide further evidence that heuristic search as performed by genetic and other similar evolutionary computing algorithms are a viable computational tool for optimizing reinforcement learning performance in multiple domains.

APPENDIX

We have the code for this paper on github: *https://github.com/aralab-unr/ReinforcementLearningWithGA*. The parameters used in this paper can be found in *baselines.her.experiment.config* module. The parameters are: discounting factor; polyak-averaging coefficient; 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 maximum absolute value of actions on different coordinates, corresponds to $gamma$; $polyak$; Q_lr; pi_lr; $random\_eps$, $noise\_eps$, respectively in the code.

REFERENCES

[1] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[2] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-learning in continuous state and action spaces," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 1999, pp. 417–428.

[3] K. Doya, "Reinforcement learning in continuous time and space," *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[4] H. V. Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," 2007.

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

[6] Q. Wei, F. L. Lewis, Q. Sun, P. Yan, and R. Song, "Discrete-time deterministic *q*-learning: A novel convergence analysis," *IEEE transactions on cybernetics*, vol. 47, no. 5, pp. 1224–1237, 2017.

[7] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3. IEEE, 2004, pp. 2619–2624.

[8] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 213–228, 2008.

[9] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search." in *AAAI*. Atlanta, 2010, pp. 1607–1612.

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

[11] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," 2011.

[12] L. Jin, S. Li, H. M. La, and X. Luo, "Manipulability optimization of redundant manipulators using dynamic neural networks," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 6, pp. 4710–4720, June 2017.

[13] C.-K. Lin, "H reinforcement learning control of robot manipulators using fuzzy wavelet networks," *Fuzzy Sets and Systems*, vol. 160, no. 12, pp. 1765–1786, 2009.

[14] Z. Miljković, M. Mitić, M. Lazarević, and B. Babić, "Neural network reinforcement learning for visual control of robot manipulators," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1721–1736, 2013.

[15] M. Duguleana, F. G. Barbuceanu, A. Teirelbar, and G. Mogan, "Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 2, pp. 132–146, 2012.

[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] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[18] H. Nguyen and H. M. La, "Review of deep reinforcement learning for robot manipulation," in *The Third IEEE International Conference on Robotic Computing (IRC2019) (Submitted)*, 2018, pp. 1–6.

[19] 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.

[20] H. Nguyen, H. M. La, and M. Deans, "Deep learning with experience ranking convolutional neural network for robot manipulator," *arXiv:1809.05819, cs.RO*, 2018.

[21] H. X. Pham, H. M. La, D. Feil-Seifer, and L. V. Nguyen, "Autonomous uav navigation using reinforcement learning," *arXiv:1801.05086, cs.RO*, 2018.

[22] ——, "Reinforcement learning for autonomous uav navigation using function approximation," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Aug 2018, pp. 1–6.

[23] H. M. La, R. S. Lim, W. Sheng, and J. Chen, "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*, May 2013, pp. 337–342.

[24] H. M. La, R. Lim, and W. Sheng, "Multirobot cooperative learning for predator avoidance," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 52–63, Jan 2015.

[25] H. X. Pham, H. M. La, D. Feil-Seifer, and A. Nefian, "Cooperative and distributed reinforcement learning of drones for field coverage," *arXiv:1803.07250, cs.RO*, 2018.

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

[27] L. Davis, "Handbook of genetic algorithms," 1991.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[29] P. W. Poon and J. N. Carter, "Genetic algorithm crossover operators for ordering applications," *Computers & Operations Research*, vol. 22, no. 1, pp. 135–147, 1995.

[30] F. Liu and G. Zeng, "Study of genetic algorithm with reinforcement learning to solve the tsp," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6995–7001, 2009.

[31] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 11, pp. 241–276, 1999.

[32] S. Mikami and Y. Kakazu, "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*. IEEE, 1994, pp. 223–228.

[33] 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.

[34] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[35] 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.

[36] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

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

[38] 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.

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