[mytitlenote]Fully documented templates are available in the elsarticle package on CTAN.
Radarweg 29, Amsterdam [myfootnote]The authors are from University of Nevada, Reno.
mymainaddress,mysecondaryaddress]Elsevier Inc [url]https://ara.cse.unr.edu/
mysecondaryaddress]Global Customer Servicemycorrespondingauthor [mycorrespondingauthor]Corresponding author hla@unr.edu
[mymainaddress]1600 John F Kennedy Boulevard, Philadelphia [mysecondaryaddress]360 Park Avenue South, New York

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 journal, we use a genetic algorithm (GA) to find the values of parameters used in the Deep Deterministic Policy Gradient (DDPG) combined with Hindsight Experience Replay (HER) algorithm, to help speed up the learning agent. We used this method on fetch-reach, slide, push, pick and place, and door opening in robotic manipulation tasks. With some modifications, this algorithm was also applied on aubo-reach environment. Our experimental evaluation shows that our method leads to significantly better performance, faster than the original algorithm.

1 Introduction

1.1 Motivation

Accuracy and efficiency play an influential part in developing as well as existing technologies. One of the many areas of robotics where efficiency matters, deals with intelligent robots [1]. Robots have long been able to function teleoperated. The world is already moving towards adding artificial intelligence (AI) to robots. Many technologies help robots make decisions. The measure of how well the robot learns is a matter of efficiency. Let's say a robot takes a few months just to learn how to open a door, which is definitely not efficient because of the amount of time this learning process would take. Hence, there's a need for efficiency in self-learning robots. The robots should be able to learn faster, which in turn save resources and time. We specifically deal with Reinforcement Learning (RL) [2] in this journal to increase the efficiency of learning robots (agents).

Hence, in order to increase the efficiency and accuracy of above-said systems, optimization algorithms such as Genetic Algorithms (GA) can assume a greater role.

1.2 Background on Genetic Algorithm (GA)

Genetic Algorithms (GAs) [3, 4, 5] were designed to search poorly-understood spaces [6], 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 solutions 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 used ranking selection [7] to select parents for crossover and mutation. Rank selection probabilistically selects higher ranked (higher fitness) individuals. Unlike fitness proportional selection, ranking selection pays attention to the existence of a fitness difference rather than to the magnitude of fitness difference. Children are generated using uniform crossover [8], which are then mutated using flip mutation [5]. Chromosomes are binary encoded with concatenated parameters. One such implementation of GA combined with Lidar-monocular visual odometry (LIMO) exists in [9].

1.3 Background on Deep Reinforcement Learning (DRL)

Q-learning [10] methods have been applied to a variety of tasks by autonomous robots [11], and much research has been done in this field starting many years ago [10], with some work specific to continuous action spaces [12, 13, 14, 15] and others to discrete action spaces [16]. Reinforcement Learning (RL) [2] has been applied to locomotion

[17] [18] and also to manipulation [19, 20].

A lot of work specific to robotic manipulators also exists [21, 22]. Some of this work used fuzzy wavelet networks [23], while others used neural networks to accomplish their tasks [24] [25]. Off-policy algorithms [26] such as the Deep Deterministic Policy Gradient algorithm (DDPG) [27] and Normalized Advantage Function algorithm (NAF) [28] are helpful for real robot systems. A complete review of recent deep reinforcement learning (DRL) methods for robot manipulation is given in [29]. We are specifically using DDPG combined with Hindsight Experience Replay (HER) [30] for our experiments. Recent work on using experience ranking to improve the learning speed of DDPG + HER was reported in [31].

RL has been widely used in training/teaching both a single robot [32, 33] and a multi-robot system [34, 35, 36, 37, 38]. Previous work has also been done on both model-based and model-free learning algorithms. Applying model-based learning algorithms to real world scenarios relies significantly on a model-based teacher to train deep network policies.

Similarly, there is also much work in GA's [3] [39] and the GA operators of crossover and mutation [40], applied to a variety of problems. GA has been specifically applied to variety of RL problems [41, 42, 43, 40].

1.4 GA on DRL

GA as a function optimizer can be used with various optimization problems. This journal focuses on the DRL, background on which was presented earlier in this chapter. GAs can be used to optimize the parameters used in the system based on their fitness values. GA tries to maximize the fitness function. An objective function can be converted to a fitness function using various mathematical formulations.

Existing DRL algorithms use a fixed set of parameters. GA when applied to DRL, finds better set of parameters, which helps the learning agent to learn faster. The inverse of the number of epochs serves as the fitness value to this problem. GA offers a promising way to increase the efficiency of the system.

1.5 Content

The following chapters of this journal are as follows: Chapter 2 introduces DRL algorithms, discusses the open problem, proposes an algorithm to solve the problem and the experimental results. Lastly, the conclusion and future work are provided in the last chapter of this journal.

2 Genetic Algorithm optimization for Deep Reinforcement Learning

2.1 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, $r : S \times A \rightarrow R$ is a reward function, $p(s_{t+1}|s_t, a_t)$ are transition probabilities and $\gamma \in [0, 1]$ is a discount factor.

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 a_t based on the current state s_t: $a_t = \pi(s_t)$. 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 environment's new state. The discounted sum of future rewards is: $R_t = \sum_{i=T}^{\infty} \gamma^{i-t} r_i$. The agent's 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:

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

2.2 Deep Q-Networks (DQN)

A *Deep Q-Network (DQN)* [44] is defined as a model free reinforcement learner [45], 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 a 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 [44]. Polyak-averaged parameters [46] can also be used.

2.3 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 the 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 a 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.

2.4 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.

2.5 Open Problem Discussion

DDPG + HER suffer from an efficiency problem. The performance of most of the robotic tasks can be improved by using a better set of parameters used in the algorithm. The performance can be measured based on the number of epochs it takes for the learning agent to learn a given robotic task. Further sections of this chapter show how the change in values of various parameters significantly impacts the learning rate of the agent. The solution to this problem is also presented later in this chapter and the supporting experimental results showing that the proposed solution outperforms the existing technique for reinforcement learning.

2.6 DDPG + HER and GA

In this section, we present the primary contribution of our journal: 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 [46]; 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 agent's 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):

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

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 ≥ 0.85
for all chromosome values do
    Initialize DDPG
    Initialize replay buffer R ← φ
    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 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' ∈ 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
        end
        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
    end
    return 1/epochs
end
Perform Uniform Crossover
Perform Flip Mutation at rate 0.1
Repeat for required number of generations to find optimal solution
```

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

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

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

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.

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

Figure **??** 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, 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* [7] 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* [8]. We are also using *flip mutation* [5] 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; 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 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 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.

2.7 Experimental Results

Figure 9, shows the environments used to test robot learning on six different tasks: *FetchPick&Place-v1*, *FetchPush-v1*, *FetchReach-v1*, *FetchSlide-v1*, *DoorOpening*, and *AuboReach*. 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 **??** (a) shows the result of our experiment with *FetchPush-v1*, while Figure **??** (a) shows the results with *FetchSlide-v1*. We let the system run with the GA to find best values of parameters τ and γ. 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 **??** (b), we show one learning run for the original parameter set and the average learning over these 10 different runs of the GA.

Figure **??** (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 **??** and **??** 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.

The GA was then run to optimize all parameters and these results were plotted in Figure 9 for all the tasks. Table **??** 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 9 shows 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 10 runs.

GA-DRL parameters (as in table **??**) were also applied on a custom built gym environment for Aubo-i5 robotic manipulator, as shown in figure 4. 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 ex-

ecution steps were involved in successful completion of each action on AuboReach environment. Unlike other gym environments discussed in this paper, AuboReach could only work with 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 real robot. These factors make this environment time consuming to train.

The GA-DRL parameters were then applied on AuboReach environment, but only with the values of actions. This means that the robot wasn't actually running to perform the action. 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 were applied again. These parameters did not outperform the performance of original parameters. We believe that is because this environment is too complex and different than 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*, *upperarm* 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 upright position, i.e., [0, 0, 0, 0].

For better learning and quick parameters search, in addition to some tweaks to the environment, 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 were 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 was 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 **??**. The difference between success rates of DRL and GA-DRL during training is demonstrated by figure **??**. Clearly, the GA-DRL performs better than DRL.

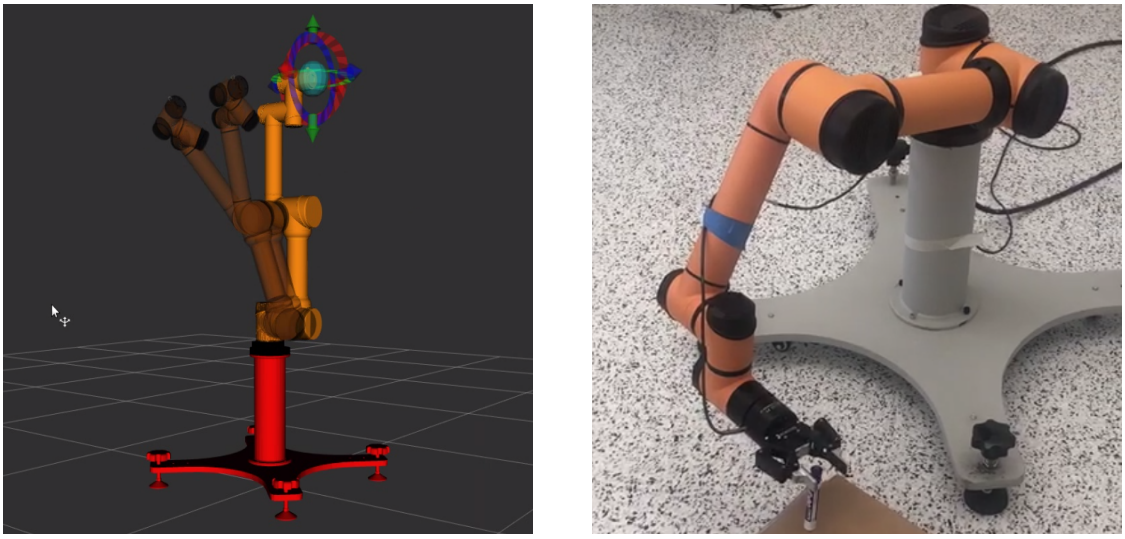
Once the GA-DRL have been found for AuboReach environment, training was run again, using four CPUs, to find the optimal policy. This policy was then applied on the robots in simulation and real experiment. The important thing to note here is that CPU usage was updated to 1 for testing. In both experiments, the robot was successfully able to go from any initial joint states configuration to any target joint states configuration. Since both, DRL and GA-DRL, ultimately reach success rate of 100%, no difference was observed during testing. Main difference lies in how quick can the environment learn using the given set of parameters.

In yet another experiment, AuboReach environment was modified to train and test on random joint states. GA was run on this environment and parameters found by the GA, is as listed in the table **??**. The plot of GA-DRL is still better than DRL, as shown in figure **??**. Figures 6 and **??** show the robot in action as it performs the task or picking the object in real and simulation experiment respectively.

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

2.8 Summary

This chapter discusses about RL, DQN, DDPG and HER. The main contribution, as described in [47] and [48], is also presented in this chapter. Experimental results were compared be-



(a) AuboReach environment - Simulation

(b) AuboReach environment - Real

Figure 4: GA-DRL with Aubo i5 robotic manipulator.

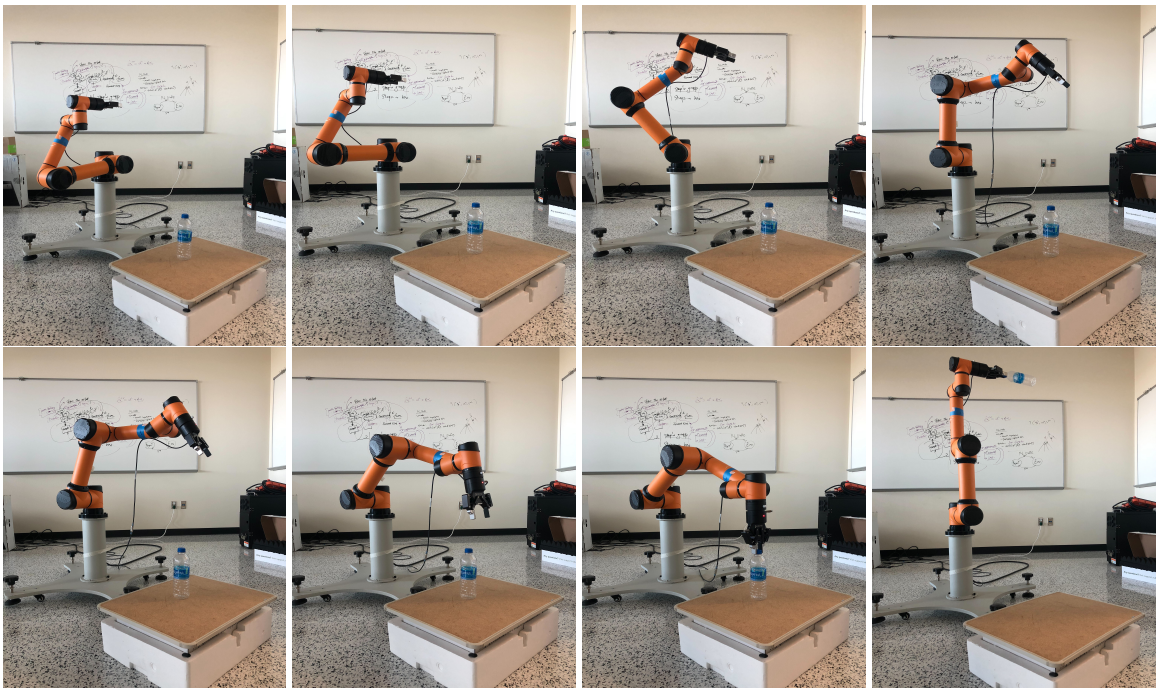


Figure 5: AuboReach environment performing a task in real experiment, using policy learned using GA-DRL.

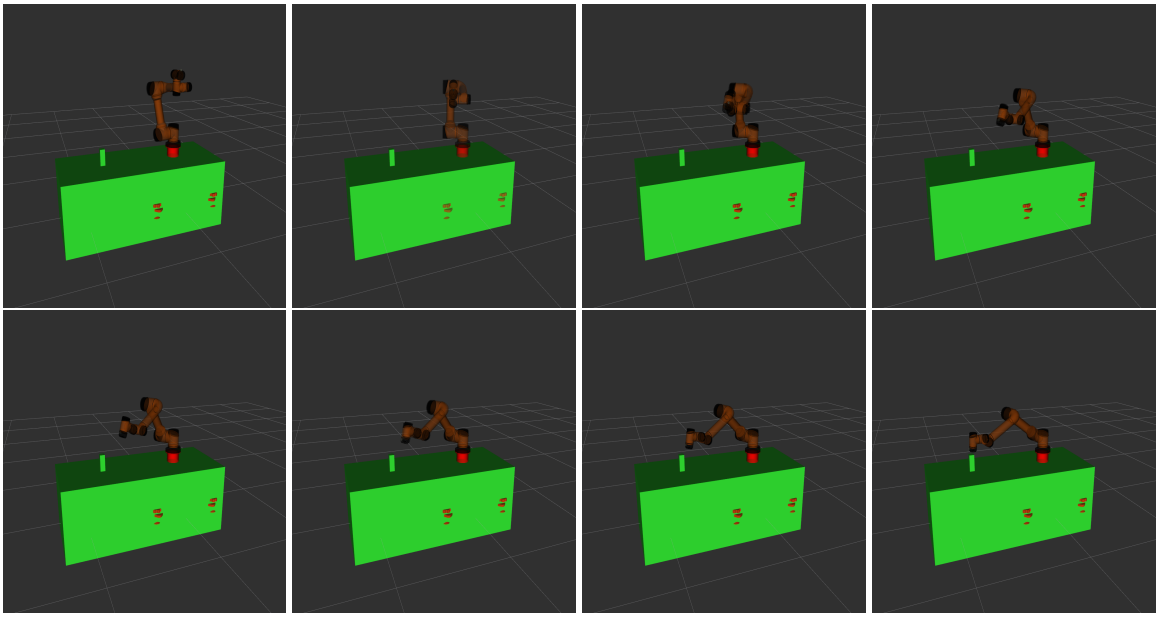


Figure 6: AuboReach environment performing a task in simulation experiment, using policy learned using GA-DRL.

tween DRL [49] and GA-DRL [47] scenarios. It was found that GA found parameters outperformed the original parameters.

3 Conclusion and Future Work

3.1 Conclusion

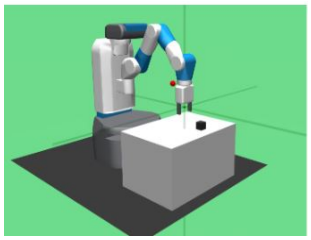
This journal showed initial results that demonstrated that a genetic algorithm can tune reinforcement learning algorithm parameters to achieve better performance, illustrated by faster learning rates at six manipulation tasks. We discussed existing work in reinforcement learning in robotics, presented 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.

3.2 Future Work

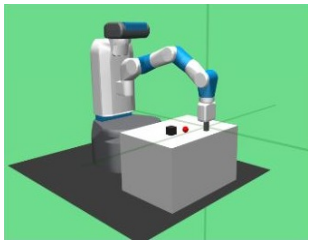
We provided further evidence that heuristic search as performed by genetic and other similar evolutionary computing algorithms are a viable computational tool for optimizing reinforcement learning and sensor odometry performance. Adaptive Genetic Algorithms can also be deployed to have different sets 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

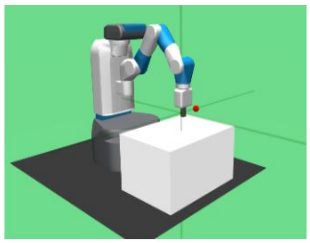
[1] Y. Bar-Cohen, C. Breazeal, Biologically inspired intelligent robots, in: Smart Structures and Materials 2003: Electroactive Polymer Actuators and Devices (EAPAD), Vol. 5051, International Society for Optics and Photonics, 2003, pp. 14–21.



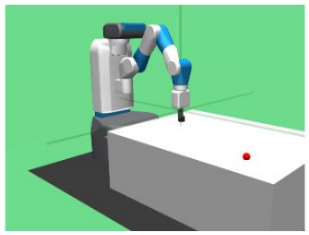
(a) FetchPick&Place environment



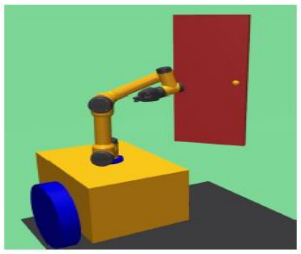
(b) FetchPush environment



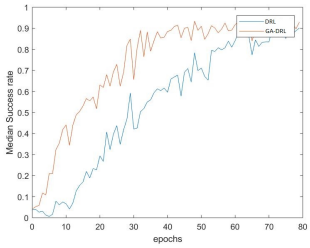
(c) FetchReach environment



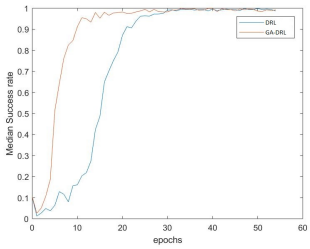
(d) FetchSlide environment



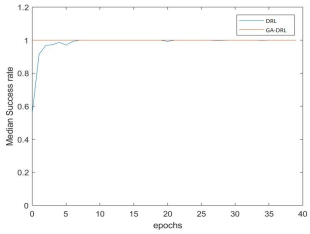
(e) Door Opening environment



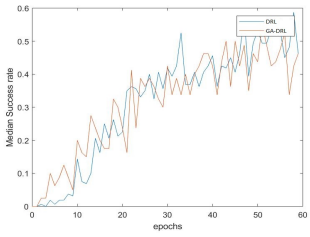
(f) FetchPick&Place plot



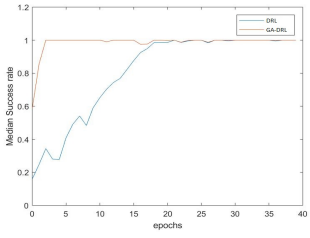
(g) FetchPush plot



(h) FetchReach plot



(i) FetchSlide plot



(j) DoorOpening plot

Figure 9: 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.

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

[3] L. Davis, Handbook of genetic algorithms.

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

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

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

[7] D. E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: Foundations of genetic algorithms, Vol. 1, Elsevier, 1991, pp. 69–93.

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

[9] A. Sehgal, A. Singandhupe, H. M. La, A. Tavakkoli, S. J. Louis, Lidar-monocular visual odometry with genetic algorithm for parameter optimization, arXiv preprint arXiv:1903.02046.

[10] C. J. Watkins, P. Dayan, Q-learning, Machine learning 8 (3-4) (1992) 279–292.

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

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

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

[14] H. V. Hasselt, M. A. Wiering, Reinforcement learning in continuous action spaces.

[15] 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.

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

[17] N. Kohl, 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.

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

[19] J. Peters, K. Mülling, Y. Altun, Relative entropy policy search., in: AAAI, Atlanta, 2010, pp. 1607–1612.

[20] M. Kalakrishnan, L. Righetti, P. Pastor, 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.

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

[22] L. Jin, S. Li, H. M. La, X. Luo, Manipulability optimization of redundant manipulators using dynamic neural networks, IEEE Transactions on Industrial Electronics 64 (6) (2017) 4710–4720. doi:10.1109/TIE.2017.2674624.

[23] C.-K. Lin, H reinforcement learning control of robot manipulators using fuzzy wavelet networks, Fuzzy Sets and Systems 160 (12) (2009) 1765–1786.

[24] Z. Miljković, M. Mitić, M. Lazarević, B. Babić, Neural network reinforcement

learning for visual control of robot manipulators, Expert Systems with Applications 40 (5) (2013) 1721–1736.

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

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

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

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

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

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

[31] H. Nguyen, H. M. La, M. Deans, Deep learning with experience ranking convolutional neural network for robot manipulator, arXiv:1809.05819, cs.ROarXiv:1809.05819.

[32] H. X. Pham, H. M. La, D. Feil-Seifer, L. V. Nguyen, Autonomous uav navigation using reinforcement learning, arXiv:1801.05086, cs.ROarXiv:1801.05086.

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

[34] H. M. La, R. S. Lim, W. Sheng, 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, 2013, pp. 337–342. doi:10.1109/CYBER.2013.6705469.

[35] H. M. La, W. Sheng, J. Chen, Cooperative and active sensing in mobile sensor networks for scalar field mapping, IEEE Transactions on Systems, Man, and Cybernetics: Systems 45 (1) (2015) 1–12. doi:10.1109/TSMC.2014.2318282.

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

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

[38] M. Rahimi, S. Gibb, Y. Shen, 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.

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

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

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

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

[43] S. Mikami, 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.

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

[45] T. Degris, P. M. Pilarski, R. S. Sutton, Model-free reinforcement learning with continuous action in practice, in: 2012 American Control Conference (ACC), IEEE, 2012, pp. 2177–2182.

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

[47] A. Sehgal, H. La, S. Louis, 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.

[48] A. Sehgal, Genetic algorithm as function optimizer in reinforcement learning and sensor odometry, Ph.D. thesis, University of Nevada, Reno (2019).

[49] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, P. Zhokhov, Openai baselines, `https://github.com/openai/baselines` (2017).