Deep Reinforcement Learning using Genetic Algorithm for Parameter Optimization

Adarsh Sehgal, Hung Manh La, Sushil J. Louis, Hai Nguyen
Department of Computer Science & Engineering
University of Nevada, Reno, NV89557, U.S.A.
Email: asehgal@nevada.unr.edu,
hla@unr.edu, sushil@cse.unr.edu,
hainguyen@nevada.unr.edu

***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 for 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) was 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. We are specifically using DDPG combined with Hindsight Experience Replay (HER) [18] for our experiments. A recent work of using experience ranking to improve the learning speed of DDPG + HER was reported in [19].

The main contribution of this paper is a demonstration of better final performance, and this paper shows that we can use a Genetic Algorithm (GA) to find DDPG and HER parameter values that lead more quickly to better performance at our robotic manipulation tasks. Our experiments revealed that values of parameters are not related to performance and speed by any simple linear function. Rather, success rate can vary surprisingly based on the values of 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 GA find good parameter values leading to better task performance, faster.

The paper will be 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 [20], [21] and a multi-robot system [22]–[25]. 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 [26] [27] and the GA operators of crossover and mutation [28], applied to a variety of poorly-understood problems. We use one of such kind of GA for our problem.

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] [18]. Specifically, we use a GA to search for good DDPG + HER algorithm parameters. 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 an learning agent, which interacts with the environment. An environment can be described by variables - S as the set of states, A as the set of actions, $p(s_0)$ as a distribution of initial states, $r : S \times A \rightarrow R$, $p(s_{t+1}|s_t, a_t)$ as transition probabilities and $\gamma \in [0, 1]$ as discount factor.

A deterministic policy maps from sates to actions: $\pi : S \rightarrow A$. 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 environment's new state. A total *return*: $R_t = \sum_{i=T}^{\infty} \gamma^{i-t} r_i$. The agent's goal is to try to maximize its expected return $E[R_t|s_t, a_t]$.

Optimal policy can be denoted by π^*, which 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 same Q-function, is called *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)

Deep Q-Networks (DQN) [29] is defined as the model free RL, designed for discrete action spaces. DQN, a neural network Q is maintained, which approximates Q^*. $\pi_Q(s) = argmax_{a \in A} Q(s, a)$ denotes greedy policy w.r.t. Q. An ϵ-greedy policy takes a random action with probability ϵ and action $\pi_Q(s)$ with probability $1 - \epsilon$.

Episodes are generated during training using ϵ-greedy policy. *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. Loss $\mathcal{L}$ is defined as: $\mathcal{L} = E(Q(s_t, a_t) - y_t)^2$ 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}) being sampled from the replay buffer.

The *target network* changes at a slower pace than 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 [29]. Polyak-averaged parameter [30] 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. Actor neural network is a target policy $\pi : S \rightarrow A$, and critic neural network is action-value function approximator $Q : S \times A \rightarrow R$. Critic network $Q(s, a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ are randomly initialized with weights θ^Q and θ^μ.

Behavioral policy is used to generate the episodes, which is a noisy variant of target policy, $\pi_b(s) = \pi(s) + \mathcal{N}(0, 1)$. The training of critic neural network is done like Q-function in DQN. Rather, 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. 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 mock the human behaviour to learn from failures. The agent learns from all the 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. It also significantly better for sparse rewards than shaped ones.

E. Genetic Algorithm (GA)

Genetic Algorithms (GAs) [26], [31], [32] 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 parameters used in DDPG + HER are found out using GA in order for the algorithm to achieve the close to maximum success rate in minimum number of epochs. We are targeting: discounting factor γ; polyak-averaging coefficient τ [30]; learning rate for critic network α_{critic}; learning rate for actor network α_{actor}; and percent of times a random action is taken ϵ.

Our experiments show that adjusting the values of parameters did not increase or decease the agent's learning in a fixed pattern. So, a simple hill climber won't do any good in finding optimized parameters. This is clearly a poorly understood problem, which needs a GA to find right set of parameters. For example τ, the polyak-averaging coefficient was adjusted to see the trend in learning. τ is used in the algorithm as 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. α denotes the learning rate. Networks are trained based on this update equation.

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

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

Since we have two kinds of networks, we will need two learning rates, one for actor network (α_{actor}), another for critic network (α_{critic}). Equation (5) explains the use of percent of times that a random action is taken, ϵ.

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

Figure 1 shows that when value of τ is modified, there is a change in agent's learning, further emphasizing the need to use GA. The original value of τ in DDPG is set as 0.95, and

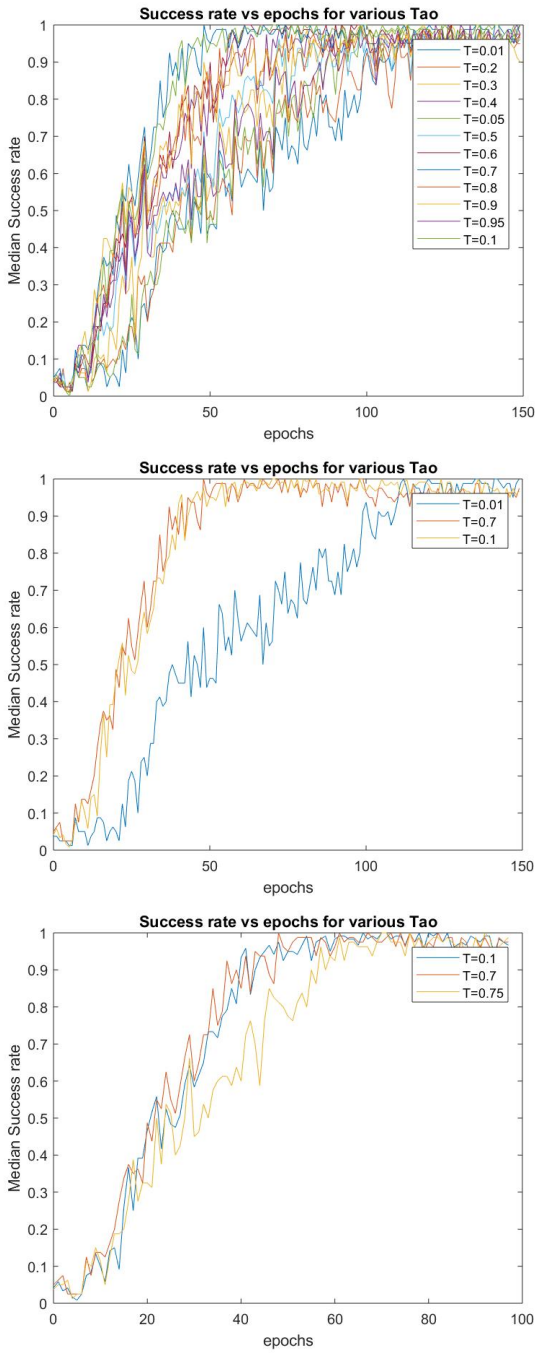


Fig. 1: Success rate vs. epochs for various τ for *FetchPick&Place-v1* task.

we are using number of CPU as 4. 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 algorithm with a GA, which uses a population size of 10 over 100 generations. We are using *ranking selection* [33] to select the parents. The parents are randomly chosen based on the probabilities to get selected, which are in turn decided based on the relative fitness. Children are then generated using *uniform crossover* [34]. We are also using *flip mutation* [32] with probability of mutation to be 0.1. Chromosome is considered in a binary form. The five parameters are arranged in chromosome back to back, with the order: polyak-averaging coefficient; learning rate for critic network; learning rate for actor network; and percent of times a random action is taken. Chromosome assumes a length of 55. Each block of 11 bits represents a parameter with value lying between 0 and 1 (both inclusive) in decimal form. The parameters further considers values up to three decimal places. The fitness for each evaluation 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.

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 $R \leftarrow \phi$
    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

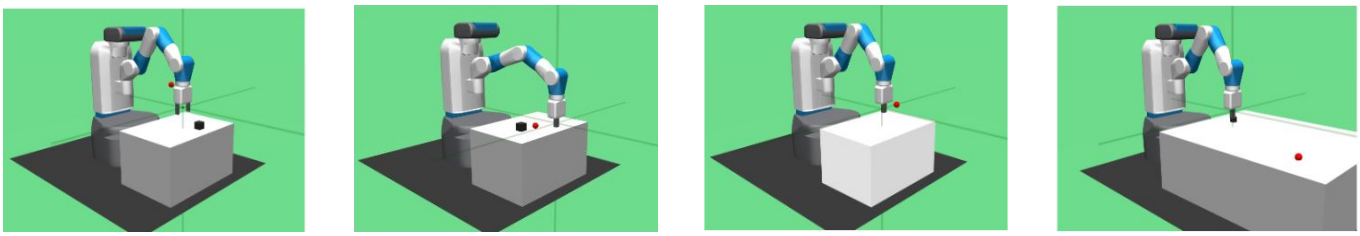


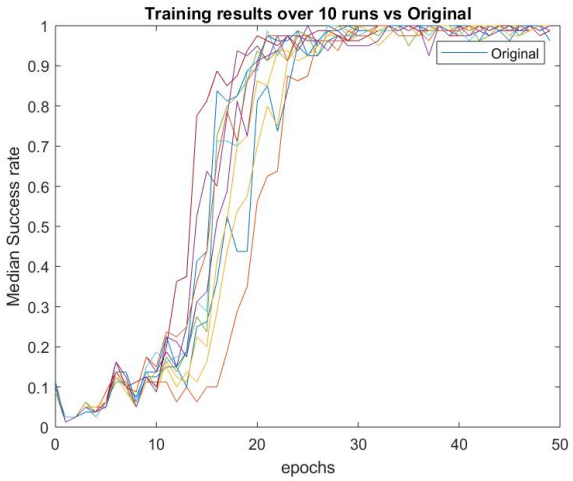
Fig. 2: Fetch Environments: (1) *FetchPick&Place-v1*-Use gripper to pick up a box and move it to a desired goal. (2) *FetchPush-v1*-Push the box to the goal position. (3) *FetchReach-v1*-Take the end effector to the goal position. (4) *FetchSlide-v1*-Slide the box to the desired goal position.

Considering the time it takes to perform each evaluation, the search space turns out to be very large, so exhaustive search is not a viable option. GA offers a promising way to find the parameters.

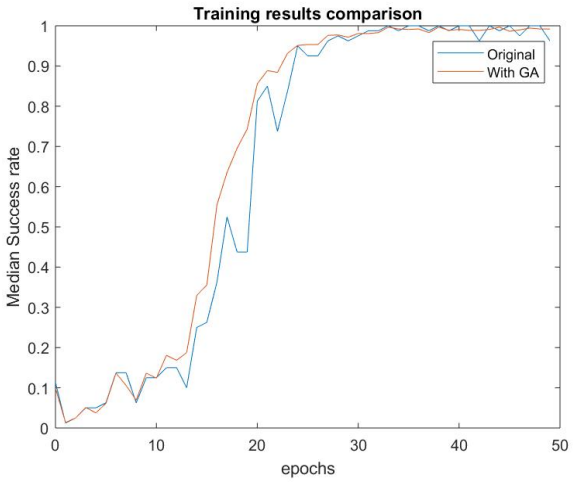
V. Experiment and Results

Figure 2, 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 3 (a) shows the result of our experiment with *FetchPush-v1* environment, while Figure 4 (a) shows the results with *FetchSlide-v1* environment. We let the system run with GA to find the optimal parameters τ and γ. Our results show that the optimized parameters did better than the original. The learning agent was able to run faster, and was able to reach the maximum success rate, faster. In Figure 3 (b), we have considered one run for original and averaged 10 runs for optimal parameters, whereas in Figure 4 (b), we have considered one run for original with averaged 2 runs for optimal parameters



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



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

Fig. 3: Success rate vs. epochs for *FetchPush-v1* task.

VI. Discussion and Future Work

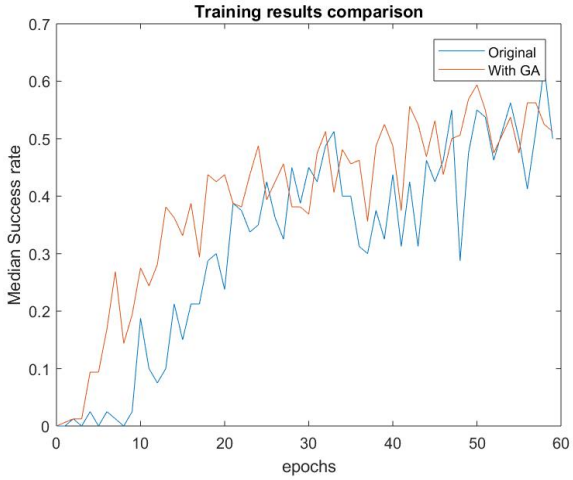
In this paper, we discussed some existing work on reinforcement learning in various fields. We also presented an algorithm, which integrates DDPG + HER with GA to optimize the number of epochs required to achieve close to 1 success rate. We also demonstrated why GA is required. We also showed some initial results, which helped us decide how GA is a good fit in our problem. Plots show that, if a *good* value of parameters is chosen, agent can learn faster with lesser number of epochs and hence, saves resources.

There is definitely a scope of achieving close to optimum success rate in lesser number of epochs. Some other parameters used in DDPG + HER can be optimized to observe any change in success rate of the learning agent.

Acknowledgment

This material is based upon work supported by the National Aeronautics and Space Administration (NASA) Grant No. NNX15AI02H issued through the Nevada NASA Space Grant, sub-award No.: 18-55. 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 NASA.

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



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

Fig. 4: Success rate vs. epochs for *FetchSlide-v1* task.

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; and percent of times a random action is taken, corresponds to $gamma$; $polyak$; Q_lr; pi_lr; $random\_eps$, 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] 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.

[19] 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.

[20] 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.

[21] ——, "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.

[22] 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.

[23] 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.

[24] 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.

[25] M. Rahimi, S. Gibb, Y. Shen, and H. M. La, "A comparison of various approaches to reinforcement learning algorithms for multi-robot box pushing," *arXiv:1809.08337, cs.RO*, 2018.

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

[27] 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.

[28] 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.
[29] 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.
[30] 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.
[31] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.
[32] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.
[33] 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.
[34] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the third international conference on Genetic algorithms*. Morgan Kaufmann Publishers, 1989, pp. 2–9.