Accelerated Reward Policy (ARP) For Robotics Deep Reinforcement Learning

Harry Li, Chee Vang, Shifabanu Shaikh, Yusuke Yakuwa †, Allen Lee †
Nitin Patil †, Nisarg Vadher, Zhixuan Zhou †, Shuwen Zheng †
Computer Engineering Department, San Jose State University, San Jose, CA 95192, USA
† CTI One Corporation, Santa Clara, CA 95051, hua.li@sjsu.edu, † yusuke.yakuwa.ctione.com, † allenlee@usc.edu

***Abstract*—Reward policy is a crucial part for Deep Reinforcement Learning (DRL) applications in Robotics. The challenges for autonomous systems with "human-like" behavior have posed significant need for a better, faster, and more robust training based on optimized reward function. Inspired by the Berkeley and Google's work, this paper addresses our recent development in reward policy/function design. In particular, we have formualated an accelerated reward policy (ARP) based on a non-linear functions. We have applied this reward function to SAC (Soft Actor Critic) algorithm for 6 DoF (Degree of Freedom) robot training in simulated environment using Unity Gaming platform and a 6 DoF robot. This nonlinear ARP function gives bigger reward to accelerate the robot's positive behavior during the trainning. Comparing to the existing algorithm our experimental results demonstrated faster convergence and bigger, better accumulative reward. With limited experimental data, the results show improved accumulative reward function as much as 2 times of the previous results.**[1]

I. Introduction

The challenges for autonomous systems with "human-like" behavior have posed significant need for a better, faster, and more robust training based on optimized reward function. Inspired by the work [3,4,5,6,18,19,20] and [1,8].

Reward policy is a cruicial part for Deep Reinforcement Learning (DRL) applications in Robotics. This paper addresses our recent development in reward policy/function design. In particular, we have formualted an accelerated reward policy (ARP) based on a non-linear function. We have applied this reward function to SAC (Soft Actor Critic) algorithm [3,4,5,6] for 6 DoF (Degree of Freedom) robot training in simulated environment using Unity Gaming platform. This nonlinear ARP function gives bigger reward to accelerate the robot's positive behavior during the trainning.

The soft actor critic algorithm (SAC) [3] is the technique widely adoption for deep reinforcement learning (DRL). The scope of this algritm is illustrated in Figure 1. Note that:

1 A Markov Decision Process (MDP) which serves as the foundation for the discussion of all algorithms here;
2 Deep-Q (DQ) and double Deep-Q (DDQ) algorithms for the discrete control actions based DRL; and
3 Stochastic policy gradient (SPG) algorithm and SAC algorithm for continuous control action based DRL.

[1]This paper has been submitted to the International Conference: the Future of Information and Communication Conference (FICC), San Francisco, 2022

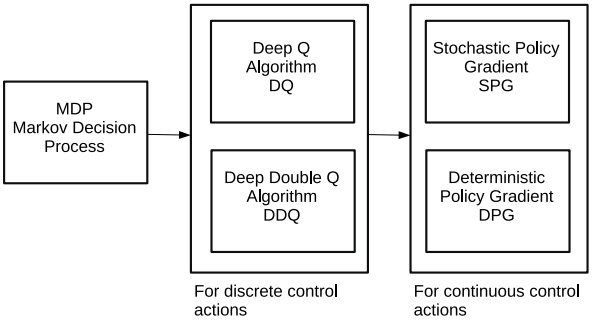

Fig. 1. The scope of SGP algrithm with relaship to the related algorithms [3,4,5,6,10].

II. Background on DRL Architecture

In DRL archictecture as illustrated in Figure 2, depending on the applications, we can use one of the Deep Neural Networks (DNN) models, e.g., SPG or DPG in the DRL system design. In our case, SPG is employed to deal with the stochastic nature of a 6 Degree-of-Freedom (DoF) robot control.

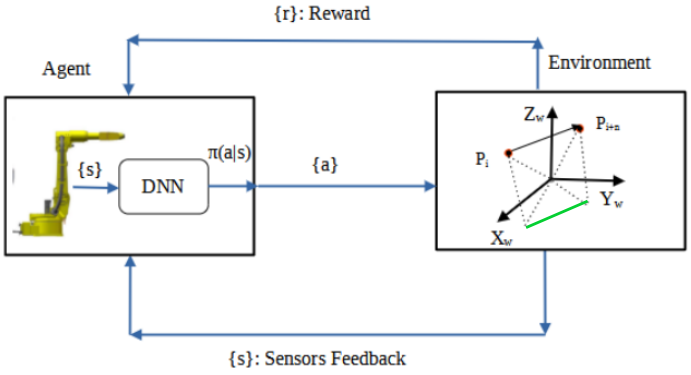

Fig. 2. Use SPG in the DRL system design to deal with the stochastic nature of a 6 Degree-of-Freedom (DoF) robot control.

Note in the DRL architecture, there are 2 feedbacks, one feedback is for the feedback of the agent state, or, system state denoted as S. This feedback is common in control systems. For

example in PID (Proportional, Integral, Derivative) controller design, we have feedback of those error functions, or in modern control we have state feedback. However, the second feedback is quite unique, which is reward feedback. The reward feedback forms the core of reinforcement learning where the maximization of long term reward is the objective of deep learning process.

In the context of robotic operations, a robot makes sequential movement, e.g., control actions in a stochastic environment. A stochastic environment makes robot action not necessarily bring to the desired position. The action only makes the robot move to a desired position with certain likelihood, which is described as probability distribution $Prob(a_{t+1}|s_t, a_t)$.

DRL mathematical formulation consists of

1. a set S of states, denoted as $S = \{s_1, s_2, ..., s_N\}$, and
2. a set A of actions, denoted as $A = \{a_1, a_2, ..., a_M\}$ as well as
3. a transition function T and
4. a reward function $R(s_t, a_t)$.

We define a tuple $< S, A, T, R >$ for the above defined sets and functions. This tuple describes the relationship of the agents and its interaction with an environment via state feedback and reward functions.

A. Markov Decision Process

Robot motion by the mathematical nature is a sequential decision process modeled as a Markov Decision Process (MDP) which consists of the tuple $< S, A, T, R >$, and has the following "short-memory" characteristics for its states S:

$$Prob(s_{t+1}|s_t) = Prob(s_{t+1}|s_t, s_{t-1}, ..., s_2, s_1). \quad (1)$$

Now, let's take a look at the reward function $R(s_t, a_t)$ which define an one step reward at current state s_t per the action a_t. Since the environment is stochastic, the agnet has its next state s_t stochastic as well, because the action a_t may not lead the agent to the desired state s_t as it is intended for. Therefore, the reward at this step $r(s_t, a_t)$ is stochastic as well. Define a policy function

$$\pi(a|s) = Prob(a|s, \theta), \quad (2)$$

where θ is a set of agent (DNN) parameters. The policy π is defined as a conditional probability of an action a given a state s with the agent's DNN parameter θ.

Choose Gaussian distribution for the policy function π. So to describe the entire process (e.g., an episode), for all rewards, we will have to take an average of the rewards. This is the statistical expecation of the rewards as follows: $E_{s\sim P_\pi, a\sim\pi_\Theta}[r(s, a)]$.

Denote this expectation as an objective function $J(\theta)$,

$$J(\theta) = E_{s\sim P_\pi, a\sim\pi_\Theta}[r(s, a)] \quad (3)$$

e.g.,

$$J(\theta) = \sum_{s\sim S} Prob(s) \sum_{a\sim A} \pi_\theta(s, a) R(s, a). \quad (4)$$

When in any state $s \in S$, an action $a \in A$ will lead to a new state with a transition probability $P_T(s, a, s\prime)$, and a reward R(s, a) function. In case of discrete actions a_t, we can have a_t as left, or right, up or down in the case of driving a simulated car in a computer game.

In the case of 6 degree-of-freedom (DoF) robot arm movement, we will have to operate with continued actions a_t. We use Stochastic Policy Gradient (SPG) technique. Given in this figure is a FD100 robot and it is operated in a stochastic environment and whose actions are continuous.

Fig. 3. An experimental platform for testing of the proposed ARP reward polic. An end effector (a set of three in this case) from CTI's FD100 robot, whose control action is continuous.

B. Stochastic Policy

The objective for DRL is to maximize long term rewards, e.g., to maximize $J(\theta)$. Substitute equation 2, into equation 4, so

$$J(\theta) = \sum_{s\sim S} Prob(s) \sum_{a\sim A} Prob(a|s, \theta) R(s, a). \quad (5)$$

We define policy by using Gaussian probability distribution in equation 2 as a stochastic policy.

C. Stochastic Policy Gradient

Now, a gradient of $J(\theta)$,

$$\bigtriangledown J(\theta) = \sum_{s\sim S} Prob(s) \sum_{a\sim A} \bigtriangledown Prob(a|s, \theta) R(s, a). \quad (6)$$

Note:

$$dx = x d(\log x), \quad (7)$$

so is

$$\bigtriangledown Prob(a|s, \theta) = Prob(a|s, \theta) \bigtriangledown log(Prob(a|s, \theta)). \quad (8)$$

Therefore, we have

$$\bigtriangledown J(\theta) = \sum_{s\sim S} Prob(s) \sum_{a\sim A} Prob(a|s, \theta) \bigtriangledown log(Prob(a|s, \theta)) R(s, a). \quad (9)$$

which forms the foundation in this work.

III. Analysis of Reward Policy

A trajectory of a robot motion defines a sequence of state-action pairs in time sequence $t_1, t_2, ..., t_N$, denoted as $\tau_N = (s_1, a_1, s_2, a_2, ..., s_N, a_N)$, defines the robot end effector's motion history in 3D space, whicd defines performance and is tied to a reward function.

For the case of continuous actions in our study, the design of reward function $R(s_t, a_t)$ is defined based on the general guidelines from Unity AI Robot github site [8,13]:

1 Define the starting end effector's position as $distance(0)$, or in short notation $d(0)$,

2 Define the target position as "NearestComponent" as in the code implementation. The NearestComponent is a constant throughout each training episod.

3 Define distance $d(t)$ as the distance from the position of end effector to the NearestComponent, e.g., the target position [13]). $d(t) = ||P(t) - P_{tgt}||_2$, where $P(t)$ is the position of the end effector at time t, and P_{tgt} is the target position, which appears in the Unity program as a position of the "NearestComponent". The target is fixed for each experiment. The distance $d(t)$ will be changing for each time index t during the training. Note we use time index t here in the mathematical formulation, and index i or k in the program.

4 Define $\Delta d(t)$ from the base line code derived from the RobotControllerAgent.c program [13].

$$\Delta d(t) = d(t) - PrevBest(t). \tag{10}$$

5 Denote PrevBest(t) as $P_b(t)$, so

$$P_b(t) = \begin{cases} d(t), & \text{if } d(t) < P_b(t) \\ P_b(t-1), & \text{o/w.} \end{cases} \tag{11}$$

So, we have

$$P_b(t) = \min(d(t), P_b(t-1)). \tag{12}$$

Now, let's take a look at the base line algorithm for the reward function.

Description of the base line algorithm

1 For the extrem case I: when the arm hits the ground, a Hefty Penalty (-1) is given, e.g.,

$$R(s_t, a_t) = -1 \tag{13}$$

where $s_t =$ (ground state), then the training episode is terminated. Note each pisode is defined as one full cycle of training.

2 For the extrem case II: when the arm reaches the target, a Hefty Reward (1) is given,

$$R(s_t, a_t) = 1 \tag{14}$$

where $s_{tgt} =$ (target state), then the training episode is terminated. [2]

[2]Or the training episod is ended if a catastrophic event, e.g., robot end effector hits the ground occurs.

The target state can be detected when the end effector $P(t)$ reaches the object(target) P_{tgt}, e.g.,

$$||P(t) - P_{tgt}|| \leq \epsilon \tag{15}$$

3 For the general cases, the base line reward function is defined with an independent variable of $\Delta d(t)$ which forms the horizontal axis, the positive value of $\Delta d(t)$ reflects the increases of the distance of the robot end effector to its target position in a single time step, therefore it is not desirable behavior, or the penalty (negative reward).

$$R(s_t, a_t) = \begin{cases} -\Delta d(t), & \text{if } \Delta d(t) \geq 0, \\ Bd - Pb - \Delta d(t), & \Delta d(t) < 0. \end{cases} \tag{16}$$

where

$$Bd = BeginDistance(0), \tag{17}$$

which is a distance set at time $t = 0$ at the beginning of each episod and it will be a constant during the entire episod, and

$$Pb(t) = PrevBest(t). \tag{18}$$

Suppose it takes N steps for the robot end effector to move from its current position $P(t)$ to P_{tgt}, e.g., (s_t, a_t) for $t = 1, 2, ..., N$, if $P(t) = P_{tgt}$, the program will end the training episode if

$$||P(t) - P_{tgt}|| \leq \epsilon \tag{19}$$

To analyze the current reward function from the Unity AI code [13], we introduce time index for each of the following 5 parameters to keep track the short past, current and possible near future values:

1 $beginDistance[i]$ of the end effector;
2 $prevBest[i]$ of the end effector;
3 $distance[i]$;
4 $Delta_distance[i]$;
5 $reward[i]$.

IV. Accelerated Reward Policy (ARP)

The reward function from Unity implementation based on the published work on [8, 13] adopt the distance based reward function, which is a linear reward function with the bigger reward for larger distance of the robot end effector traveled towards the target. We define this reward function as Type I reward 20.

To encourage positive behavior during the DRL training, for the bigger distance traveled towards the target by the end effector, we design an accelerated reward policy (ARP) by a new reward function as follows and illustrated in the following figure as well:

$$R(s_t, a_t) = \begin{cases} -K_1 \Delta d(t), & \text{if } \Delta d(t) < 0, \\ K_2(Bd - Pb - \Delta d(t)), & \Delta d(t) \geq 0. \end{cases} \tag{20}$$

As you can see the new reward function with K_2 gain coefficient on the nagative axis of $\Delta d(t)$ together with the base line existing slop of the reward function, denoted as K_1, form a non-linear reward function (pice-wise linear).

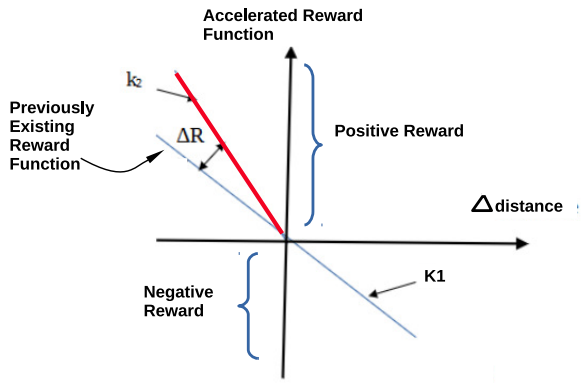

Fig. 4. The new reward function with K_2 gain coefficient on the nagative axis of $\Delta d(t)$ together with the base line existing reward function.

V. Experiment Design

A. ARP Experiments

To test the proposed ARP function, we conducted 20 training experiments separated into 2 groups with each of 10. Group 1 is for the base line algorithm and Group 2 is for the ARP function.

Each experiment is conducted on Unity as shown below. The program is coded in CS (C#), the DRL engine is in Python, and each experiment starts with the same fixed starting position and same fixed target position for the robot end effector. We choose the experiments with training upto 5,5000 steps. The code of our proposed algorithm was placed in the github [14,15,16].

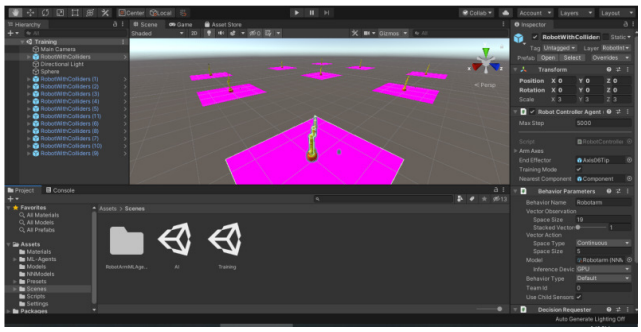

Fig. 5. The experiments are conducted on Unity and the algorithm is coded in CS (C#). We choose each experiment with training upto 5,5000 steps.

B. Re-write The Base Line Code

We define the following parameters explicitly.

1 *EndEff[i]* which defines the robot end effector's position in $x_w - y_w - z_w$ space. Clearly define *EndEff[i]* in $x_w - y_w - z_w$ space, e.g., $P_{end} = EndEff[i] = (x_{ef}(i), y_{ef}(i), z_{ef}(i))$,

2 Seperate relative movement from absolute movement. See figure below for illustration. The relative movement is the movement from *EndEff[i-1]* to *EndEff[i]*, while the absolute movement is the movement defined with reference in $x_w - y_w - z_w$ space, simply stated, it is *EndEff[i]*.

VI. Performance Evaluations

Compute the averaged reward from the base-line algorithm and the proposed ARP algorithm. Each of these 2 accumulated rewards plotted with steps as an independent variable. We illustrated one of the accumulated reward function in the following figure. Note we define the steps which give the negative reward as region Ω_N while the steps give the positive reward as region Ω_P, the cross-over point is the point to separate these two regions.

1 For a better performance, we are expecting to have a smaller accumulated negative reward, and

2 a bigger accumulated positive reward, e.g., the bigger area in the region of Ω_P.

A. Index for Accumulated Reward

Use the averaged outpout from both base-line algorithm and the proposed algorithm to calculate the accumulated reward as follows:

$$\int_{\Omega_N} r_1(t; s_t, a_t)\, dt + \int_{\Omega_P} r_1(t; s_t, a_t)\, dt \tag{21}$$

Computationally, we have

$$I_N = \sum_{t=1}^{t_k} r_1(t; s_t, a_t),\ I_P = \sum_{t=t_{k+1}}^{t_N} r_1(t; s_t, a_t). \tag{22}$$

So we have $I_{N,K2}$ as the index covering the negative accumulated reward for the proposed ARP algorithm and $I_{N,B}$ as the index of the accumulated reward for the base-line algorithm. Therefore, the indices are defined to compare the performance of two different reward functions as

$$\eta_N = \frac{I_{N,K2}}{I_{N,B}},\ \eta_P = \frac{I_{P,K2}}{I_{P,B}}. \tag{23}$$

B. Performance Comparison

To compare the performance, we have the following guide-line.

If $\eta_N < 1$, then the accumulated negative reward of the proposed ARP algorithm performs better in the sense of generating lesser negative penalty. And if If $\eta_P > 1$ then the accumulated positive reward of the proposed ARP algorithm performs better in generating more positive reward.

VII. Analysis and Conclusion

The experimental result as shown in the following figure. The Illustration of the base line reward function result (above) vs. the accelerated reward policy (ARP) result. Note the very visible accumulative reward function increases for the ARP shown as a black color curve in the lower half of this figure.

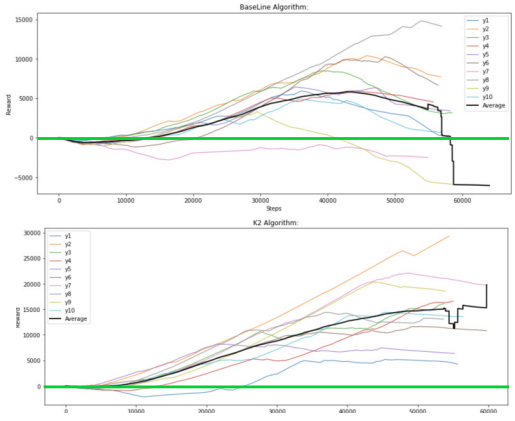

Fig. 6. Illustration of base line reward function result vs. the accelerated reward policy (ARP) result. Note very visible accumulative reward function increases for the ARP.

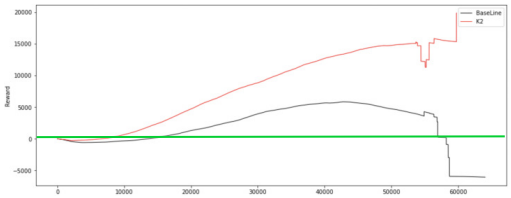

Fig. 7. Comparison of base line reward function result (gray, lower) vs. the accelerated reward policy (ARP), (red, upper), with clear incease in accumulative reward function for the ARP.

There are 11 plots in each group, which are 10 different experiments and 1 calculated average as a mean value for the comparison to the other group. Note from step 0 to around 11,000 steps for Group 1 in Fig 6, the agent (robot) has negative reward. Then it crosses over to the positive side of the Reward. The agent starts to receive a positive reward more after 11,000 steps, the cross-over point. This trend of elevation continues gradually until around 55,000 steps. We neglect the steps after 55,000 steps as there are some missing training points due to there are a few curves with exactly 60,000 training steps on the graph. Now, for the ARP algorithm evaluation, we did same calculation of average curve from all 10 curves. For the 10 experiments of ARP with $K_1 = 1.0$ and $K_2 = 1.1$, we did not observe much visible difference from the experiments graph. Once we adjust the gain coefficients with $K_1 = 1.0$ and $K_2 = 2.0$, large visible change has occurved as shown in the lower half of this figure.

From these 20 experiments with each of over 55K steps, we compute the reward and have the results listed in the following table.

TABLE I
Experiment Results Comparison

Category	Base line	Accelerated Reward Policy
Cross over	13,635 Steps	6,529 Steps
I_N	-5.21^6	-1.1^6
I_P	1.53^8	3.97^8

Compute the performance based on equation 23,

$$\eta_N = 0.212 \text{ and } \eta_P = 2.595. \tag{24}$$

These experiments results demonstrated

1 The proposed ARP reward policy/function has converged faster, e.g., from 13,635 steps of the base line algorithm to 6,529 steps of the new algorithm on average.
2 The proposed ARP reward policy/function reduces the negative behavior including the catastrophic event (e.g., the end effector hitting the ground) by significant reduction, from -5.21^6 of the base line to -1.1^6, about 21% of less negative behavior. So the ARP has reduced the negative behavior of the robot agent by as much as 79% from the experimental data results.
3 The ARP accelerates the robot learning/gainning desired movement from 1.53^8 of the base line to 3.97^8, about 2.59 times bigger in accumulative reward. Hence the ARP has accelerated the robot agent learning in a very visible way.
4 Since 20 experiments were conducted and each with steps upto 55K, it is needed to take this preliminary results here to move to large experiment population with each exmperiments going up to as many as 500K steps or even higher if needed for further verification. The code of our ARP algorithm was placed in the github [14,15,16].

Acknowledgment

I would like to thank CTI One engineers and intern team for coding and carrying out robot parts design (by Allen Lee).

References

[1] Andrea Franceschetti, Elisa Tosello, Nicola Castaman, and Stefano Ghidoni, "Robotic Arm Control and Task Training through Deep Reinforcement Learning", https://arxiv.org/pdf/2005.02632.pdf, May 2020.
[2] https://github.com/Unity-Technologies/ml-agents 2020.
[3] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, Kristian Hartikainen, Vikash Kumar, Pieter Abbeel, Henry Zhu, George Tucker, Abhishek Gupta, Sergey Levine UC Berkeley, and Google Brain, "Soft Actor-Critic Algorithms and Applications", https://arxiv.org/pdf/1812.05905.pdf, Jan. 2019.
[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, Berkeley Artificial Intelligence Research, University of California, Berkeley, USA., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", haarnoja@berkeley.edu, http://proceedings.mlr.press/v80/haarnoja18b/haarnoja18b.pdf, 2018.
[5] Petros Christodoulou, Imperial College London, "SOFT ACTOR-CRITIC FOR DISCRETE ACTION SETTINGS", petros.christodoulou18@imperial.ac.uk, https://arxiv.org/pdf/1910.07207.pdf, Oct. 2019.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, Berkeley Artificial Intelligence Research, University of California, Berkeley, USA., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", haarnoja@berkeley.edu, http://proceedings.mlr.press/v80/haarnoja18b, Aug. 2018.

[7] Russ Salakhutdinov, Machine Learning Department, CMU, "10703 Deep Reinforcement Learning and Control", www.cs.cmu.edu, 2018.

[8] The Unity Machine Learning Agents Toolkit, software, https://github.com/Unity-Technologies/ml-agents.

[9] The experiments were conducted on Linux x86 machine with GPU acceleration, and Mono environment for CS script is established for programming and testing CS script which is utilized to modify the existing ML algorithm on the Unity ML platform.
Mono, CS-Script engine file on Linux, www.cs-script.net.
Q-Learning and difficulties with continuous action space Value-Based Methods like DQN have achieved remarkable breakthroughs in the domain of Reinforcement Learning. However, their success is bound to problems with discrete action spaces, like Atari games.

[10] Takuma Seno, NAF: Normalized Advantage Function DQN for Continuous Control Tasks, google search: DQN_NAF+deep+reinforcement+learning, Oct. 2017.

[11] Robot modeling on unity, https://blogs.unity3d.com/2020/11/19/robotics-simulation-in-unity-is-as-easy-as-1-2-3

[12] 3D model of Robot arm is from this link. The Look and feel of the model and hierarchy of components altered to fit for ML training purpose.
Joel, 3D Robot Modeling, esp. for the bottle pick and place model, /sketchfab.com/3d-models, https://sketchfab.com/3d-models/robot-arm-22d9367c8d2f4457b3a4e74193e86ac9.

[13] in our training and simulation, we have employed the software for the training of 6 axis robot arm inverse kinematics using Unity ML Agents R. Kandas, RobotArmMLAgentUnity, https://github.com/rkandas.

[14] Harry Li, Chee Vang, Shifa S., Base-line algorithm, for the Deep Reinforcement Learning based on the baseline reward function https://github.com/hualili/robotics...fd100/105g-1RobotControllerAgent-1Base.cs

[15] Harry Li, Chee Vang, Shifa S., Accelerated reward policy (ARP) algorithm, for the Deep Reinforcement Learning based on the ARP reward function with K=2, https://github.com/hualili/robotics- ... fd100/105g-2RobotControllerAgent-2timeInd.cs

[16] Harry Li, Chee Vang, Shifa S., Accelerated reward policy (ARP) algorithm, for the Deep Reinforcement Learning based on the ARP reward function with bigger K, https://github.com/hualili/robotics- ... fd100/105g-3RobotControllerAgent-3K2.cs

[17] Unity ML, the training configuration file and the definition of the NN architecture, https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md

[18] Mohammad Taghi Saffar, Mohammad Babaeizadeh, Danijar Hafner, Harini Kannan, Sergey Levine, Chelsea Finn, Dumitru Erhan, Google Brain, msaffar@google.com, mbz@google.com, danijar@google.com, hkannan@google.com, slevine@google.com, chelseaf@google.com, dumitru@google.com, "MODELS , PIXELS , AND REWARDS: EVALUATING DESIGN TRADE-OFFS IN VISUAL MODEL BASED REINFORCEMENT LEARNING", https://arxiv.org/abs/2012.04603, Dec. 2020.

[19] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra, "Continuous control with deep reinforcement learning", https://arxiv.org/abs/1509.02971, Sept. 2015.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Google DeepMind Technologies, "Playing Atari with Deep Reinforcement Learning" vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller@deepmind.com, https://www.cs.toronto.edu/ vmnih/docs/dqn.pdf, 2015.

[21] Volodymyr Mnih1, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis "Human-level control through deep reinforcement learning", LETTER doi:10.1038/nature14236, Nature, pp. 529., Feb. 2015,