Deep Deterministic Policy Gradient and Hindisght Experience Replay for controlling a robotic arm

Hernandez Cancino, J. E.
B.S. in Robotics and Digital Systems Engineering (E.G.D. June, 2024)
Tec de Monterrey
Monterrey, N.L., Mexico
a00827269@tec.mx

Munoz, L. A.
Engineering Sciences Dept.
Tec de Monterrey
Monterrey, N.L., Mexico
amunoz@tec.mx

***Abstract*—This paper presents an approach to trajectory planning for robotic arm manipulation tasks using Deep Reinforcement Learning (DRL), specifically, Deep Deterministic Policy Gradient (DDPG) and Hindsight Experience Replay (HER). This combination of algorithms is utilized to train a policy which allows a robotic arm to compute end-effector poses and all the joint configurations for each time step, required to successfully reach a desired end-effector pose from an random starting position. An additional constraint of this paper's implementation was to ensure a steady orientation of the arm's gripper as to allow the arm to steadely move objects around when acting on the developed policy.**

***Index Terms*—deep reinforcement learning, HER, DDPG, xarm, robotic arm**

I. Introduction

The field of robotics has experienced significant advancements with the application of Deep Reinforcement Learning (DRL) models to accomplish highly intricate tasks. Challenges such as high-dimensional state spaces or a substantial number of constraints often hinder the development of conventional algorithms capable of accurately describing system functionality and exhibiting robustness in guiding a robot's behavior to consistently achieve its goals.

DRL presents a promising approach to tackle this challenge. It empowers a robotic agent to learn from experience by interacting with its environment. This learning process results in the development of a reliable policy, governing the agent's behavior and enabling it to effectively execute goal tasks in the real world.

In this research paper, we demonstrate the efficacy of two fundamental Deep Reinforcement Learning (DRL) algorithms designed for continuous spaces in achieving a specified task: trajectory planning for a robotic arm with 6 degrees of freedom (specifically, the xArm6 robot by UFactory). The objective is to reach goal positions with the gripper, while maintaining a constant orientation of the end effector throughout the entire trajectory.

Trajectory planning involves determining a sequence of joint angles and/or end-effector poses that guide an arm from its initial position to a desired goal state. In our case, an additional constraint is imposed: the rotation changes of the end effector with respect to the x and y planes should be zero throughout the entire trajectory. This constraint ensures a stable trajectory, crucial for navigating around objects that should remain unrotated, such as a cup of water.

The complexity of the problem increases with the number of degrees of freedom and possible reachable states. This escalation in complexity is the primary motivation for employing reinforcement learning as a viable solution to accomplish specific tasks in environments like the one under consideration.

II. Hypothesis

Deep Reinforcement Learning algorithms can be effectively utilized for planning trajectories of a robotic arm to perform manipulation tasks involving moving the arm's gripper from a starting position to a target point in space while maintaining a desired end-effector orientation with respect to the x and y planes.

III. State of the Art

Effectively controlling a multi-joint robot in real-world applications presents a formidable challenge, characterized by the intricacy arising from an increased set of variables and degrees of freedom. The task further demands the formulation of a behavior model and the establishment of a policy capable of navigating a continuous environment. Notably, the capacity to make consistent, deterministic decisions for analogous states also becomes valuable, as to ensure reproducibility and a clear understanding of system behaviors.

Among the diverse existing control strategies, certain deep reinforcement learning algorithms have proven to be useful for this endeavor.

A. Reinforcement Learning for Continuous Action Spaces

Early Deep Reinforcement Learning algorithms, such as the *Deep Q Network* (DQN) Mnih et al. (2013) and *Double DQN* (van Hasselt et al., 2015), have demonstrated efficacy in using Deep Learning networks with reinforcement learning principles, particularly in the context of discrete environments. However, the application of these elemental DRL algorithms encounters challenges when dealing with an environment such as a real 6-degree-of-freedom (6-DOF) robotic arm. The prospect of discretizing the environment space becomes impractical due to the heightened number of degrees of freedom, leading to an exponentially increased number of possible states.

Given the objective of this paper—to demonstrate the implementation of a deep reinforcement learning policy on a real robotic platform—it becomes necessary to employ an algorithm explicitly devised for continuous state spaces.

B. Selected Techniques

The *Deep Deterministic Policy Gradient* (DDPG) stands out as a fundamental Deep Reinforcement Learning algorithm for continuous state space tasks, having demonstrated its capability to successfully address tasks of comparable or even heightened complexity to the one under consideration in this paper. Examples include tasks such as achieving the walking of a cheetah robot or maintaining balance in a Cart Pole environment within a simulated continuous state space (Lillicrap et al., 2019).

Also, given the challenging nature of the environment in use, characterized by sparse rewards, an additional technique known as *Hindsight Experience Replay* is employed. This approach proves fruitful in significantly enhancing the performance of DDPG in environments with sparse rewards, akin to the one described in this paper. The work by (Andrychowicz et al., 2018) shows the advantages of the HER technique in diverse tasks such as *pushing*, *sliding*, and *pick-and-place* by a robotic arm. This showcases the technique's efficacy in overcoming a demanding environment with sparse rewards, wherein positive or useful rewards are granted only upon reaching the goal state.

IV. Methodology

We employ a standard reinforcement learning framework, where an agent (the robotic arm) interacts with its environment in discrete time steps, asumed to be fully observable. The environment is characterized by states s_t, actions a_t, and a reward function $r : s \times a \to \mathbb{R}$.

At each time step t, the agent receives an observation s_t, comprising the joint angle rotations in radians that define the current state. This information is used to compute the action $A \in \mathbb{R}^3$ to be executed by the agent. The decision-making process is governed by the deterministic policy $\pi(s) : s \to a$, wherein the policy involves determining the Cartesian position difference of the end-effector to be achieved from state s_t to s_{t+1}.

A. Deep Deterministic Policy Gradient (DDPG)

The DDPG algorithm is a potent, actor-critic, model-free reinforcement learning (RL) algorithm based on the deterministic policy gradient. It is designed to operate over continuous action spaces and can learn competitive policies for tasks using low-dimensional observations, such as Cartesian coordinates or joint angles (Lillicrap et al., 2019). DDPG maintains two neural networks: a target policy (the actor) and an action-value function approximator (the critic).

The actor selects an action to take given a current state according to $\pi(s)$, and the critic approximates the actor's action-value function Q^π.

As articulated in the original DDPG paper, the actor's policy is updated using mini-batch gradient ascent on the expected return from the start distribution according to the acting policy:

$$\nabla_{\theta^\pi} J \approx E_{s_t}\left[\nabla_{\theta^\pi} Q(s, a|\theta^Q)\Big|_{s=s_t, a=\pi(s_t|\theta^\pi)}\right] \tag{1}$$

Meanwhile, the critic network is updated by minimizing the loss:

$$L(\theta^Q) = E_{s_t, a_t, r_t}\left[(Q(s_t, a_t|\theta^Q) - y_t)^2\right] \tag{2}$$

Here, the targets y_t are computed using the actions outputted by the actor policy:

$$y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})) \tag{3}$$

For both networks, Polyak averaging is employed to update the target actor and critic network weights, enhancing convergence reliability.

B. Hindsight Experience Replay (HER)

HER serves as a valuable technique to address the inherent challenges of sparse rewards in reinforcement learning training environments. In the context of the environment under consideration, the agent consistently receives a negative reward (-1 penalty), except when it reaches its target position, at which point it receives a positive reward (+1). This binary reward scenario poses a challenge for training a new policy using a model-free algorithm like DDPG due to the perpetual lack of informative feedback from the algorithm. This can prolong the training process, requiring an exceedingly high number of episodes for the agent to acquire meaningful insights.

HER presents a sample-efficient technique that in some way, emulates human learning, emphasizing the importance of learning from both desired and undesired outcomes. This is achieved by replaying each training episode with a different goal than the original; specifically, a state that the agent did achieve during the given episode.

There have been several resampling strategies proposed, but in this implementation, the *future* HER strategy is employed, due to its superior performance as demonstrated in the original HER paper. This strategy involves replaying with k random states ”from the same episode as the transition being replayed and were observed after it”. Also the hyperparameter k controls the ratio of HER data to data coming from normal experience replay in the replay buffer (Andrychowicz et al., 2018).

Also an exploration policy is implemented during training to encourage agent exploration in action selection. This policy involves adding Gaussian Noise N to the output of the actor network, introducing variability and randomness. During the training process, the agent takes noise-altered actions with an occurrence percentage of $\epsilon \in [0, 1]$ to help escape sub-optimal policies:

$$\pi'(s_t) = \pi(s_t|\theta_t^{\pi}) + N \tag{4}$$

The code implementation is available at https://github.com/edgarcancinoe/xarm_DDPG_HER.

V. Experimental Design

For the training process, the environment was simulated using the MuJoCo physics engine (Todorov et al., 2012) following the guidelines of OpenAI Gym’s environments. The MuJoCo simulation and Gym environment drew substantial inspiration from a prior implementation that addressed a similar reaching task for the xArm6 robot (Ramírez, 2022).

The key considerations for the experimental design are outlined as follows:

- The environment comprises a world where the training robot (xArm6) is situated, mounted on a custom base, equipped with a custom camera and gripper, and features a red dot indicating the target location.
- The agent incurs a penalty of -1 whenever the arm’s end effector is not in the target position and receives a reward of 1 upon reaching the desired coordinates.
- A state descriptive variables consist of the current observations of the following elements:
  1) End effector x position.
  2) End effector y position
  3) End effector z position
  4) End effector x velocity.
  5) End effector y velocity
  6) End effector z velocity
- An action $a_t \in \mathbb{R}^3$ is output by the actor network, indicating the change in position the end effector should undergo for the next time step.
- The simulation environment manages the necessary arm joint rotations each time a new point is computed according to the policy π. To maintain the gripper orientation throughout the trajectory, it is hard-coded, and the remaining joints are adjusted accordingly to satisfy this constraint.
- Training episodes originate from a constant starting point defined by the initial joint configuration of the xArm6’s joints, starting from the base of the robot $s_0 = \{0, 0, 0, -\pi/2, \pi/2, 0\}$. The target is randomly generated as follows:

$$x_{\text{goal}} = \text{np.random.uniform}(0.5, 0.62, \text{size} = 1)[0]$$
$$y_{\text{goal}} = \text{np.random.uniform}(-0.3, 0.3, \text{size} = 1)[0]$$
$$z_{\text{goal}} = \text{np.random.uniform}(0.15, 0.45, \text{size} = 1)[0]$$

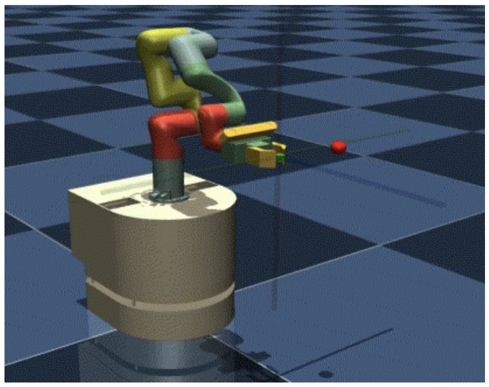

Fig. 1. Snapshot of the training environment in the MuJoCo simulation environment.

A. Algorithm Parameters

The parameters used for training the DDPG algorithm are as follows:

TABLE I
DDPG TRAINING PARAMETERS

Parameter	Value
Actor learning rate	$\alpha = 0.005$
Critic learning rate	$\beta = 0.005$
Discount factor	$\gamma = 0.98$
Noise epsilon	$\epsilon = 0.3$
Gaussian noise std	$\sigma = 0.2$
Batch Size	$B = 256$
Polyak averaging coefficient	$\tau = 0.95$
HER replay k	$k = 4$
Number of training epochs	$N_{\text{epochs}} = 36$
Number of cycles per epoch	$N_{\text{cycles}} = 80$

B. Performance Metrics

Following every epoch, a performance evaluation was conducted on the actor policy. Specifically, $n_{\text{eval}} = 20$ evaluation episodes (e) were played after every N_{cycles} training episodes, during which the actor and critic networks were updated according to their respective loss functions. The success rate was calculated by dividing the number of successful episodes by the total number of played episodes.

VI. Results

Figure 2 illustrates that after the 8^{th} epoch, the agent begins to achieve its goals with a success rate of 0.250, and from then on, it continues to consistently improve the success rate, reaching a final average success rate of 1.0 from epoch 16 up to the last evaluation performed. Figures 3 and 4 display the actor and critic losses over time, respectively, indicating a converging behavior.

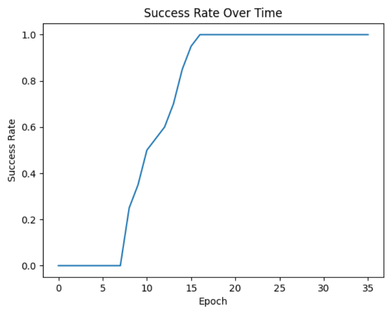

Fig. 2. Evaluation success rate by epoch.

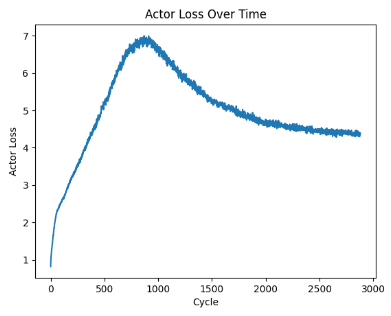

Fig. 3. Actor loss over time, by number of training cycles.

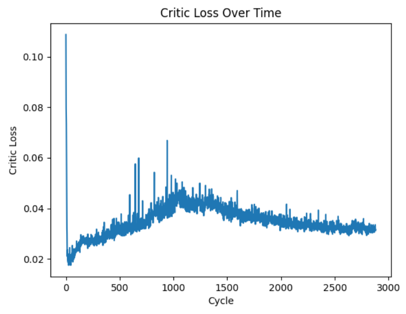

Fig. 4. Critic loss over time, by number of training cycles.

VII. Findings

The employed algorithm, along with the chosen training parameters, has proven to be effective in enabling the robotic agent to learn how to navigate from a starting point to a final target destination. Despite the agent starting each training episode from the same initial position s_0 towards a new random goal, once training is successfully completed, the agent demonstrates the ability to reach targets from changing starting positions within the coordinate space it was trained in while following the resulting policy $\pi(s)$. To validate the usefulness of the policy in a real robot, the simulation code was modified to record each joint configuration and end-effector position reached as the agent moves from a starting position to a given goal. This data was utilized to have an xArm6 robot follow the policy and emulate the desired behavior in a real robot environment. RViz software (Kam et al., 2015), the xArm-Developer repository xArm Developer (2023), and the MoveIt ROS planning tools Planning (2023) were employed for this purpose.

The following repositories contain the application of policy $\pi(s)$ for both an original xArm6 robot and for a custom robot with the xArm6 mounted in ROS Noetic:

- https://github.com/edgarcancinoe/xArm6_DDPG_ROS
- https://github.com/edgarcancinoe/home_DDPG_ROS

Additionally, a video of the custom robot following the computed trajectories can be found at:
https://www.youtube.com/watch?v=VB5VkgwCG1A

This demonstrates the applicability of DDPG in understanding the dynamics of a 6-DOF arm to act as a controlling policy for reaching target destinations. In this implementation, the resulting policy could be utilized to perform tasks such as moving objects where end-effector rotation should remain stable, particularly in controlled environments where the target positions fall within the space where the policy has been trained.

VIII. Conclusions

This research underscores the efficacy of Deep Reinforcement Learning in enabling robotic agents to learn and adapt in continuous state spaces. The significance lies in the capacity of these advanced techniques to provide a viable alternative for addressing intricate tasks within complex systems—a challenge that may be formidable using traditional approaches. While computational power is a requisite and the need for precise observations, state descriptions, and meticulous reward engineering grows with the complexity of the environment, Deep RL algorithms offer a notable advantage: They can significantly simplify the development of control strategies for complex systems, a task where traditional methods might struggle to accommodate high numbers of variables and possibilities, demanding immense efforts. This is achieved through leveraging the agent's own experience.

In this research paper and its accompanying code implementation, the goal of employing DDPG and HER to plan stable, reliable trajectories for a 6-DOF arm, reaching points within its accessible space, has been successfully achieved.

Acknowledgment

I would like to express my sincere gratitude to Professor Luis Alberto Muñoz for his guidance and mentorship throughout the course of this research. Professor Muñoz's insights and expertise have been invaluable, shaping the trajectory of this work from the beginning and contributing significantly to its quality.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2018.

Hyeong Ryeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, 60:337–345, 2015.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

ROS Planning. Ros planning github organization. https://github.com/ros-planning, 2023. Accessed: November, 2023.

Julio Cesar Ramírez. xArm6-Gym-Env: A Gym Environment for xArm6 Robot in OpenAI Gym. https://github.com/julio-design/xArm6-Gym-Env, 2022. Accessed: November, 2023.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

xArm Developer. xarm_ros: Ros driver for xarm robots. https://github.com/xArm-Developer/xarm_ros, 2023. Accessed: November, 2023.