Analysis and Empirical Evaluation of Deep Deterministic Policy Gradient in a Continuous Environment

Andrew Garcia, Kayleigh Movalli, Christopher Frutos
Virginia Polytechnic Institute and State University
ECE 5984 - Deep Reinforcement Learning
agarcia1296@vt.edu, kayleighm@vt.edu, cfrutos@vt.edu

Abstract

With companies like Boston Dynamics and Engineered Arts gaining attention in the public eye, it's evident that robotics, mainly humanoid or real-world ready, has been on the rise. Despite the many breakthroughs these companies have had, one of the most common problems found in this type of field is how difficult it is to teach fine motor skills like balance, walking, and interaction within the real world. Even though it may appear like these things have been done before, in actuality, these problems are persistent and require further research before they can operate reliably [1]. In our project, we aim to emulate real-world interaction to test and analyze a reinforcement learning algorithm that can teach a robotic arm a core fine motor skill by having it learn to move small objects. We will be using pre-placed blocks along with the robotic arm from the PyBullet environment to have the arm learn where the blocks are in the environment and how to interact with them. The reinforcement learning (RL) approach we will be analyzing and testing is Deep Deterministic Policy Gradient (DDPG) to measure how it performs in a continuous 3D space.

1. Introduction

The application of reinforcement learning (RL) methods for continuous control application is a topic of great interest in the field of research. The incorporation of factors such as noise, which is not typically considered in RL, makes discovering optimal policies for tasks considerably more challenging. However, with the recent advancements in RL, we are now on the brink of witnessing the profound impact of robotics and artificial intelligence (AI) on our daily lives. The objective of our project is to study the application and contribute to the field's progression by evaluating a reinforcement learning algorithm to see if it is a suitable candidate for robotics in a 3D environment that closely mimics our physical world. This project is an excellent exercise for evaluating the potential of intelligent robotics in real-world applications.

We chose to implement Deep Deterministic Policy Gradient (DDPG) [2], a model-free deep RL actor-critic algorithm that combines policy-based and value-based methods to learn and update policies to create a robust optimal policy in a high-dimensional action-space. We selected DDPG due to its success in robotic environments, including real-world robotic benchmarks [3]. In our study, we present a detailed implementation of DDPG in the Methods section. DDPG operates so well in continuous environments because of its unique setup.

We designed a 3D environment based on the robotic arm environment in PyBullet, which we will test the DDPG algorithm. Our environment consists of a table with three Jenga blocks placed vertically in front of the robotic arm as seen in Figure 1. The objective is for the DDPG-powered robotic arm to learn to interact with only the center block without touching the other two, simulating fine motor skills in a simplified version of a complex 3D environment. PyBullet was our choice of simulation environment due to its open-source nature, ease of use, and realistic physics engine that enables us to achieve real-world-like results.

1.1. DDPG Background

DDPG is a reinforcement learning algorithm that is well-suited for continuous control tasks. It combines ideas from deep learning and actor-critic methods to learn policies that can map observations to actions in an efficient and robust manner.

The core idea behind DDPG is to use a deterministic policy, instead of a stochastic policy, to improve the stability and convergence of the learning process. The policy is represented by a neural network called the actor network, which takes the current observation as input and outputs the corresponding action. The actor network is trained using the policy gradient method, which involves computing the gradient of a performance measure with respect to the parameters of the network and updating them in the direction

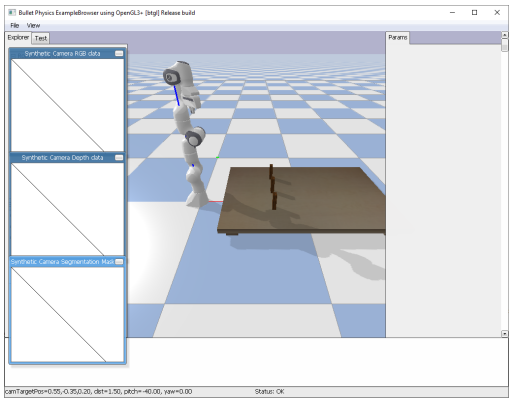

Figure 1. Initial position of the environment, the robot arm is in stow and the Jenga blocks are in line standing up on the table.

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Figure 2. Pseudocode of the DDPG algorithm. DDPG trains a deterministic policy in an off-policy way, and to enhance exploration, noise is added to their actions at training time.

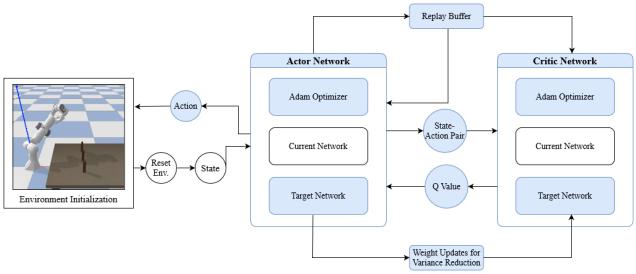

Figure 3. Flowchart of DDPG algorithm implemented in this project.

of the gradient.

To improve the quality of the action predictions and reduce the variance of the gradients, DDPG also uses a critic network, which estimates the expected return from a given state and action. The critic network is trained using the Bellman equation, which defines the optimal value function as the maximum expected return achievable from each state. The critic network is trained using the temporal difference learning method, which involves minimizing the difference between the estimated value and the actual reward received.

To improve the exploration and exploitation trade-off, DDPG adds noise to the actions selected by the actor network. The noise is sampled from a Gaussian distribution with zero mean and a fixed variance and is annealed over time to reduce its contribution to the actions as learning progresses. The following pseudocode in Figure 2 explains the DDPG process [4].

In our study, we used the DDPG algorithm to learn policies for a simulated robot arm task. The details of the task and the experimental setup are described in the following sections.

2. Methods

2.1. DDPG Implementation

The Deep Deterministic Policy Gradient (DDPG) algorithm implemented in our project is based on the code provided by Phil Tabor's Github repository [5]. However, his original code was intended for the Lunar Lander gym environment so it was a challenge to implement our own custom environment to fit the object inputs and outputs. By studying the implementation in-depth, we gained a better understanding of the hyperparameters used in DDPG and how they affect the algorithm's performance. The use of Tabor's code as a starting point provided a strong foundation for our implementation and allowed us to build on existing knowledge to improve our results. Using this deep understanding, we created our own flowchart for visualizing the agent processes shown in Figure 3

2.2. Custom Jenga Block Environment

We created a custom OpenAI Gym environment for a Jenga block manipulation task, which consists of a simulation of the commercially available Franka Panda robot arm that needs to knock over the middle Jenga block while avoiding knocking over the two other Jenga blocks. The environment is implemented using the PyBullet physics engine and provides a continuous observation space and a continuous action space [6].

The observation space includes the positions and velocities of the three Jenga blocks, the position and orientation of the robot arm, and the joint angles and velocities of the robot arm. The action space consists of the joint torques that the robot arm can apply to its joints.

To incentivize the robot arm to knock over the middle

Jenga block while avoiding the other blocks, we defined a reward function that reduces the negative reward if the gripper gets closer to the middle block and a negative reward if any of the other blocks two blocks are moved from their original position. The following equation represents the reward function:

$$Reward = -d_{GM} - d_{LD} - d_{RD} \quad (1)$$

, where d_{GM} is the distance of the robot grippers to the middle block, and d_{LD} and d_{LD} are the distances of the left and right block displacement from its original position. The structure of this reward computation incentives the agent to minimize the distance of the gripper to the middle block and punishes for moving the left and right block. The episode reaches its done state if the robot grippers touch the middle block or if a maximum timestep value of 3000 is reached.

We also implemented a visual rendering of the environment, which displays the Jenga blocks and the robot arm using a simple OpenGL graphics engine. This allows for easy visualization of the task and the performance of the robot arm.

In the experiments reported in this paper, we used this custom Jenga block manipulation environment to evaluate the performance of our proposed reinforcement learning algorithms for robotic manipulation tasks.

3. Experimental Results

We conducted three different experiments to evaluate the performance of our DDPG algorithm. Each experiment took on average 8 to 14 hours to complete, so in order to complete these experiments in a reasonable time, each experiment was performed simultaneously across each member's computer. The first experiment varied the learning rates of the actor and critic to help determine an optimal value. The second experiment involved training neural networks of varying sizes to determine the impact of network size on performance. In the third experiment, we extended the number of episodes used during training to determine the effect of longer training periods on the algorithm's performance. Our results showed that a learning rate of 0.001 and a larger neural network with 128 hidden units performed best while extending the number of training episodes led to diminishing returns in performance gains. Overall, our experiments demonstrated the effectiveness of our proposed approach and provided insights into the impact of various hyperparameters on the algorithm's performance.

3.1. Various Learning Rates

The plot in Figure 4 depicts the performance of our reinforcement learning algorithm for varying learning rates of the actor and critic components. Our experiments showed that a learning rate of 0.0001 for the actor and 0.001 for the critic produced the best performance on the task. This finding is consistent with our hypothesis and studies in class, which have suggested that low learning rates can lead to more stable and effective training of deep reinforcement learning models.

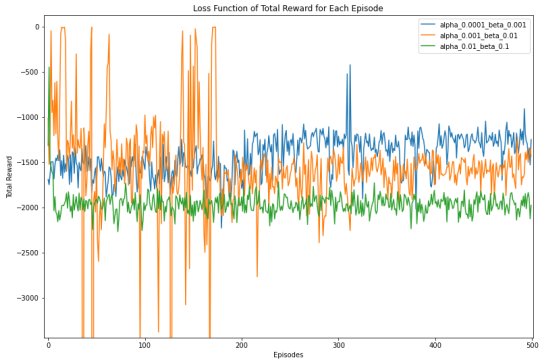

Figure 4. Plot of various learning rates for the Actor and Critic networks and their resulting total reward for each episode.

3.2. Various Neural Networks

This section will provide an overview of the results from varying the size of the hidden layers of the nerual networks of the actor and critic. In this experiment, we observed in Figure 5 that the 16 and 54 neuron networks produced riskier attempts, sometimes resulting in highly negative rewards but also occasionally achieving highly positive rewards. While these attempts were outliers, the 16 and 54 neuron networks exhibited slightly lower overall performance compared to the 128 neuron network. In contrast, the 128 neuron network displayed greater stability and higher overall performance than the other two networks. Based on these results, we concluded that the 128 neuron network would be the best choice for this experiment if the complexity were to increase in the future.

3.3. Extended Episodes

To further evaluate the performance of our DDPG algorithm, we conducted an experiment with extended episodes. We ran the algorithm for 1000 episodes (Figure 7), which took approximately 8 hours to complete. The purpose of this experiment was to determine if the performance of the algorithm would plateau after a certain number of episodes. We found that after approximately 500 episodes, we began to see diminishing returns for the total reward collected. The results suggest that while the algorithm may continue to learn beyond this point, the rate of improvement is significantly reduced. These findings could inform future ex-

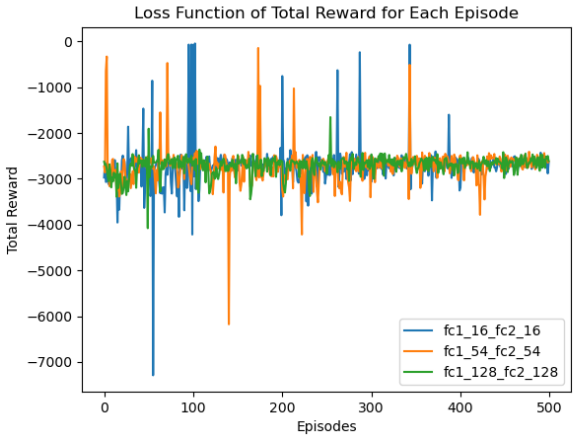

Figure 5. Plot of various neural network sizes for the Actor and Critic networks and their resulting total reward for each episode.

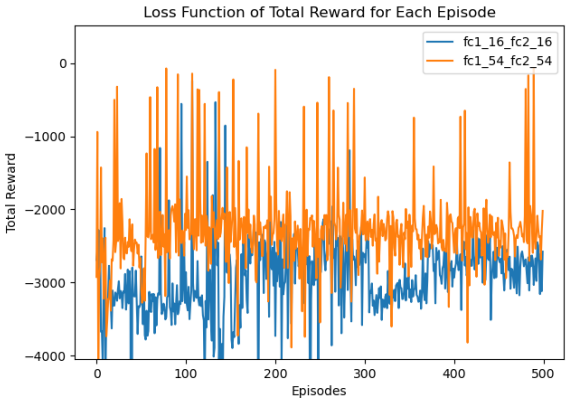

Figure 6. Plot of 16 and 54 neural networks per hidden layer showing that increasing the neural network count improved performance. 128 was being tested, however due to time constraints, was not able to complete in time.

periments and potentially optimize the run-time of the algorithm.

4. Discussion

In this study, we employed the Deep Deterministic Policy Gradient (DDPG) algorithm to teach the Franka Panda robotic arm to interact with a 3D environment of Jenga blocks to demonstrate the algorithm's ability to complete a continuous control task shown in Figure 8. Our results show that DDPG is a powerful algorithm for robotic control tasks, as it was able to successfully complete the task after extensive training. One interesting observation is that the robot arm never learned to unfold its top joint to move the gripper directly to the middle Jenga block. This could be due to

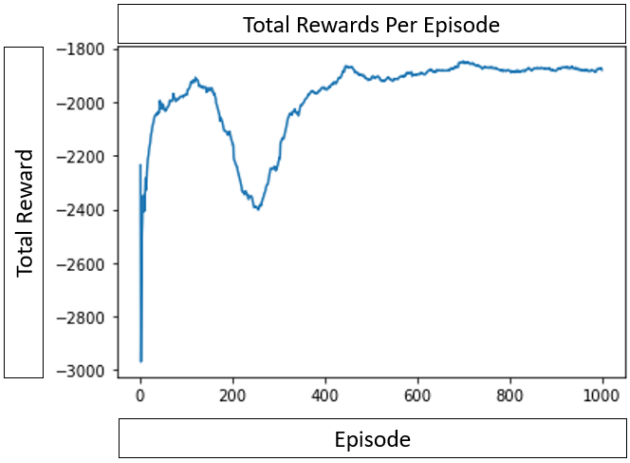

Figure 7. Plot of total rewards collected for each episode over 1000 episodes.

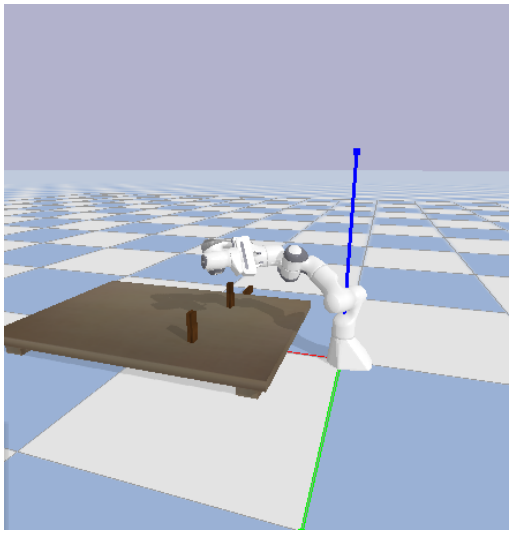

Figure 8. Example of completed done state when the robot arm touches the middle Jenga block, which was pushed off the table to the right.

a missed dimension in the action space that allows control over this joint. However, upon code inspection, our implementation uses all of the joints to create the action space so it is still unknown what is causing the issue.

Our study demonstrates the potential of DDPG as a valuable tool for teaching robotic arms to interact with complex 3D environments. Going forward, further research is needed to explore the use of DDPG for other robotic control tasks, as well as to do a deeper investigation of the impact of different hyperparameters and training regimes on the algorithm's performance. It would also be great to see more complicated tasks being performed, such as picking up and moving the jenga block to a desired location. Over-

all, the results of this study suggest that DDPG can play a crucial role in advancing the field of robotics and artificial intelligence, with exciting implications for a wide range of real-world applications.

5. Contributions

As a key contributor to the project, I developed and created a custom OpenAI Gym environment using PyBullet, which involved designing all the functions of the RobotArmEnv object to enable resetting, stepping, computing reward and done signals, as well as rendering the OpenGL GUI. The creation of this environment posed a significant challenge, requiring a comprehensive understanding of both PyBullet and the OpenAI Gym framework. Additionally, I implemented a dynamic input feature that allowed users to specify the target block for the robot arm to manipulate. Although I added a reward function for moving the target block to a specific location, due to time constraints and limitations, we were unable to carry out this advanced task. Collaborating with my teammates, I ensured that the environment was fully functional and properly integrated with the DDPG algorithm. While performing the experiments, I executed the various learning rates experiment on my personal computer. Overall, the successful implementation of this custom environment was a critical step in developing a unique platform for testing the application of DDPG to a continuous control problem.

References

[1] W. D. Heaven, "Forget Boston Dynamics. This Robot Taught Itself to Walk." With AI, MIT Technology Review, Apr. 09, 2021. Accessed: Apr. 03, 2023. [Online]. Available: https://www.technologyreview.com/2021/04/08/1022176/boston-dynamics-cassie-robot-walk-reinforcement-learning-ai/amp/.

[2] 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," in Proceedings of the 4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, May 2016.

[3] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma and J. Bergstra, "Benchmarking Reinforcement Learning Algorithms on Real-World Robots," in Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, May 2018, pp. 1-8.

[4] Deep deterministic policy gradient¶. Deep Deterministic Policy Gradient - Spinning Up documentation. (n.d.). Retrieved May 6, 2023, from https://spinningup.openai.com/en/latest/algorithms/ddpg.html

[5] P. Tabor, "Actor-Critic-Methods-Paper-To-Code," GitHub. [Online]. Available: https://github.com/philtabor/Actor-Critic-Methods-Paper-To-Code/tree/master/DDPG. [Accessed: May 6, 2023].

[6] Bullet physics SDK. GitHub. (n.d.). Available: https://github.com/bulletphysics [Accessed: May 6, 2023].