Optimizing Robotic Arm Control Using Deep Deterministic Policy Gradient: An Exploration of Hyperparameter Tuning

Yile Ma*

Computing Science, University of Alberta, T6G 2E8 Edmonton, Canada

Abstract. Robotic arms are essential in a wide range of applications, from industrial automation to medical surgeries, where both accuracy and adaptability are critical. Traditional path-planning methods for robotic arms, such as Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM), often suffer from limitations in dynamic environments. Reinforcement Learning (RL) presents a promising alternative for optimizing robotic arm control by enabling adaptive learning through trial and error. This study focuses on the application of Deep Deterministic Policy Gradient (DDPG), a popular RL algorithm, to control a simulated robotic arm following a mouse pointer. The study investigates the impact of three key hyperparameters—learning rate, batch size, and memory capacity—on the performance of the DDPG model. This paper systematically tested multiple values for each parameter and evaluated the model's success rate and average time per goal. Results showed that the optimal combination of parameters was a learning rate of 0.001, a batch size of 50, and a memory capacity of 30,000, yielding a success rate of 76.00% and an average time per goal of 0.07 seconds. These results emphasize the significance of fine-tuning hyperparameters to achieve optimal performance in robotic control tasks. Future work will focus on exploring adaptive hyperparameter tuning strategies and applying these methods to more complex and dynamic robotic environments to further enhance performance and adaptability.

1 Introduction

A robotic arm is a programmable mechanical device resembling a human arm in its structure and function [1]. It's a system powered by both electrical and mechanical components consists of several joints and links that allow for a wide range of motion and manipulation tasks [2]. Robotic arms are pivotal in numerous applications, particularly in fields like manufacturing automation and medical surgery. In industrial settings, robotic arms are employed for tasks that require precision, speed, and consistency, such as assembly, welding, and packaging. In the medical domain, a robotic arm is utilized in situations requiring high precision, where mistakes are not permissible [3], such as minimally invasive surgeries.

*Corresponding author: yile@ualberta.ca

The significance of robotic arms in these applications cannot be overstated. They increase productivity, ensure high-quality outputs, and improve safety by handling hazardous or repetitive tasks that would otherwise endanger human workers. However, optimizing the performance of robotic arms for specific tasks has traditionally been a complex and challenging process. Traditional robotic arm path planning methods primarily include RRT [4], PRM [5], and artificial potential field methods [6]. However, these approaches often suffer from limitations in dynamic planning capabilities and exhibit low levels of intelligence [7]. They are often relying on pre-programmed paths and manual tuning of parameters. Furthermore, these methods are limited in their adaptability and efficiency, particularly when dealing with dynamic environments or tasks requiring high levels of precision and flexibility.

In this situation, RL, which offers a promising alternative for optimizing robotic arm control, can be considered an effective solution. RL is a subset of artificial intelligence (AI) that RL enables machines to find the optimal strategy within the Markov decision process through reward maximization through interaction with the environment [8]. Unlike traditional methods, RL algorithms can continuously improve performance by learning from successes and mistakes. This adaptability is especially beneficial in robotics, where the ability to learn and refine actions based on feedback is critical.

Artificial intelligence has grown rapidly in recent years, with RL becoming a key area of research. RL is critical in fields like robotics, finance, and gaming, with notable algorithms such as Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), and DDPG. These methods allow agents to acquire strategies through trial and error, and in robotics, RL has been particularly impactful. For instance, Kumar used DDPG to enable a bipedal walking robot to learn without prior knowledge of its own or the world's dynamics [9].

Despite progress, most research focuses on algorithm development rather than exploring how hyperparameter choices—like learning rates or discount factors—affect results. This study aims to addresses this gap by investigating the influence of different hyperparameters on the performance of a model based on DDPG controlling a robotic arm. The effectiveness of RL in controlling robotic arms is highly dependent on the careful tuning of various hyperparameters. These include learning rates, discount factors, and other parameters that significantly influence the learning process and final performance. Exploring these parameters and understanding their impact on the model's ability to perform tasks effectively is crucial. This exploration forms the core of this study, as this paper aim to optimize these parameters to enhance the performance of robotic arms in complex environments. The results of this study will offer valuable insights into optimizing reinforcement learning models for robotic applications, offering practical guidance on parameter tuning for improved performance in complex tasks.

2 Methods

2.1 The definition of scenarios

This study focuses on controlling a robotic arm in a simulated environment, where the goal is to follow the movement of a mouse pointer using RL. The simulation is built using Python and TensorFlow, with the robotic arm equipped with multiple joints. The state space consists of the joint positions and velocities, while the action space controls the torque applied to each joint. The primary objective is to minimize the distance between the target (mouse pointer) and the robotic arm's end-effector, while ensuring that the movements are smooth and efficient. The optimization target is to enable the arm to autonomously adapt to different target positions and perform the task with minimal human intervention. This scenario is

common in robotics, where continuous control tasks are critical to the success of autonomous systems. Fig. 1 shows the interface.

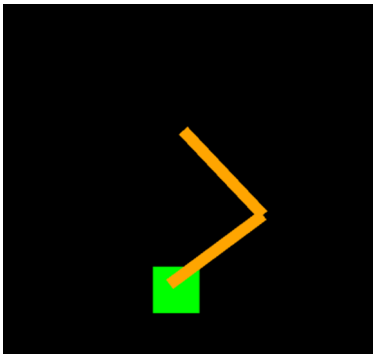

Fig. 1. Interface of the robotic arm (Photo/Picture credit: Original).

2.2 RL-based prediction

RL is a learning framework where an agent interacts with its environment to discover optimal actions that maximize cumulative rewards. RL methods operate in a way akin to how humans learn to perform different tasks, following a trial-and-error process [10]. Unlike traditional supervised learning, RL does not depend on labelled data. Rather, the agent navigates the environment, learning through trial and error, and receives feedback in the form of rewards. The objective is to create a policy—a mapping of states to actions—that maximizes the agent's expected cumulative reward over time.

In this scenario, the robotic arm acts as the RL agent. The environment includes the arm's joint locations, velocities, and the location of the target (mouse pointer). The state is defined by the arm's joint configurations, while the actions correspond to the torques applied to the joints. The reward is determined by the arm's closeness to the target and the fluidity of its movement. This encourages the arm to reach the target efficiently while minimizing unnecessary movements. The RL framework is the DDPG algorithm, which is particularly suited for tasks involving smooth or uninterrupted action spaces, such as robotic control, DDPG could ensure that joint movements remain continuous and smooth [11]. DDPG employs an Actor-Critic framework, where the Actor network produces actions based on the current state, and the Critic network assesses those actions by estimating the expected reward.

The Actor network receives the current state of the robotic arm as input and outputs continuous actions (torques) to be applied to the joints. It is a fully connected neural network consisting of three layers: an input layer that matches the dimensions of the state, a hidden layer with 300 neurons, and an output layer aligned with the number of possible actions. The Critic network receives both the current state and the chosen actions as input, evaluates the anticipated future rewards, and outputs a single Q-value. This network also has three layers, where the state and action inputs are processed in separate branches and merged in the hidden layer, which has 300 neurons. Both networks employ Rectified Linear Unit (ReLU) as the activation function for their hidden layers, while the Actor network utilizes a Tanh activation in the output layer to ensure the actions stay within the permissible range of joint torques.

2.3 Implementation details

This study focuses on testing three key hyperparameters: learning rate, batch size, and experience replay buffer size. For each parameter, range of different values will be tested to assess their impact on the performance of the robotic arm in reaching the target. The

evaluation metrics used are success rate (the percentage of episodes in which the robotic arm successfully reaches the target) and average time per goal (the average amount of steps required to reach the target).

The learning rate is a crucial factor in determining how quickly the model updates its parameters during training. It affects both the Actor and Critic networks, which are tasked with producing actions and evaluating those actions, respectively. To evaluate the influence of this parameter, three learning rates will be tested: 0.0001, 0.001, and 0.01. The Adam optimizer is employed to update the network parameters due to its adaptive learning rate capability, which enhances the robustness of the training process.

The second hyperparameter is batch size, which determines the number of samples drawn from the experience replay buffer during each training update. The batch sizes tested are 16, 32, and 64, allowing a comparison of the trade-off between computational efficiency and the consistency of the learning process.

The third hyperparameter is the experience replay buffer size, which controls how many past experiences the agent stores for sampling during training. To assess its impact, three buffer sizes will be evaluated: 10,000, 20,000, and 30,000. These values will be tested to determine how the memory capacity affects the agent's ability to learn from past experiences and maintain stable learning.

The model will be trained for 1000 epochs, with each epoch comprising a fixed number of steps. During each episode, the robotic arm attempts to minimize the time taken to reach the goal while ensuring smooth and controlled movements. The reward function is designed to penalize deviations from the target, as well as any unnecessary or abrupt movements. This encourages the arm to reach the target efficiently while maintaining stability in its actions.

Multiple runs will be conducted for each combination of hyperparameters to ensure statistically reliable results. The outcomes will be compared to identify the most effective hyperparameter settings for improving the robotic arm's performance. This exploration of learning rate, batch size, and experience replay buffer size aims to identify optimal configurations that enhance the success rate and reduce the time taken to achieve goals in continuous control tasks. This analysis provides valuable insights for tuning RL models in robotic applications.

3 Results and discussion

3.1 The influence of the learning rate of actor and critic on the model performance

The learning rate is a crucial hyperparameter in reinforcement learning that determines how fast an agent adjusts its knowledge from new experiences. This study tested three distinct learning rates: 0.0005, 0.001, and 0.002. The outcomes of these tests are presented in Table 1 below:

Table 1. The performance of tests on different learning rate of Actor and Critic.

Learning Rate	Success Rate (%)	Average Time per Goal (s)
0.0005	58.00	0.10
0.001	76.00	0.07
0.002	60.00	0.10

The findings show that a learning rate of 0.001 delivered the best performance, achieving a success rate of 76.00% and an average time per goal of 0.07 seconds. This suggests that at this learning rate, the model was able to efficiently balance between rapid learning and stability, leading to efficient task execution.

In contrast, when the learning rate increased to 0.002, the success rate decreased to 60.00%, and the average time per goal rose to 0.10 seconds. This decrease in performance may be attributed to the model's tendency to overshoot optimal actions due to aggressive updates, which can lead to oscillations and instability during the learning process. High learning rates often result in erratic behavior, causing the agent to overlook optimal policies.

Conversely, when the learning rate was lowered to 0.0005, the model's success rate further declined to 58.00%, while the average time per goal remained at 0.10 seconds. This outcome highlights the drawbacks of overly conservative learning rates. Although they may provide stability, such rates can significantly slow the learning process, preventing the agent from converging on effective policies in a timely manner. In dynamic environments, this lag can be detrimental, as timely adaptations to new information are crucial.

These results highlight the significance of selecting the learning rate carefully to optimize performance. The results suggest that a learning rate of 0.001 strikes an ideal balance, allowing for both effective learning and stability in action selection. Future experiments might further refine this parameter by exploring values around 0.001 to identify a more precise optimal range.

3.2 The influence of the batch size on the model performance

Batch size is another critical hyperparameter that influences the training dynamics of RL algorithms. This experiment evaluated three batch sizes: 35, 50, and 65. The results of these tests are summarized in the Table 2 below:

Table 2. The performance of tests on different batch size.

Batch Size	Success Rate (%)	Average Time per Goal (s)
35	68.00	0.08
50	76.00	0.07
65	73.00	0.07

The optimal performance was observed at a batch size of 50, resulting in a success rate of 76.00% and an average time per goal of 0.07 seconds. This indicates that this batch size effectively balances the need for stability in learning updates while maintaining responsiveness to new data.

When the batch size was reduced to 35, the success rate fell to 68.00%. This reduction suggests that smaller batch sizes may enhance the adaptability of the agent by allowing for more frequent updates. However, the increased variance in updates can lead to instability and may hinder the agent's ability to reach an optimal policy.

On the other hand, increasing the batch size to 65 resulted in a slight decrease in success rate to 73.00%. While larger batches provide more stable and averaged updates, they may also limit the model's ability to learn rapidly from recent experiences. In this context, a larger batch size can dilute the impact of critical new experiences, which is particularly important in environments that require quick adaptations.

These results suggest that the choice of batch size has significant implications for the agent's performance. A batch size of 50 appears to provide an optimal compromise between

stability and responsiveness, enabling effective learning without introducing excessive noise or inertia in the training process. Further investigations could explore intermediate batch sizes to pinpoint an even finer optimal range.

3.3 The influence of the memory capacity on the model performance

The memory capacity plays a crucial role in RL by determining how much past experience the agent can utilize during training. This study evaluated three memory capacities: 10,000, 30,000, and 50,000. The results are outlined in the Table 3 below:

Table 3. The performance of tests on different memory capacity.

Memory Capacity	Success Rate (%)	Average Time per Goal (s)
10000	21.00	0.17
30000	76.00	0.07
50000	51.00	0.12

The results show that a memory capacity of 30,000 led to the highest success rate of 76.00% and the lowest average time per goal of 0.07 seconds. This suggests that an adequate memory capacity enables the agent to store a diverse array of experiences, enhancing its ability to generalize and learn effectively.

Conversely, reducing the memory capacity to 10,000 resulted in a drastic drop in performance, with a success rate of only 21.00% and an average time per goal of 0.17 seconds. This sharp decline indicates that a limited replay buffer does not provide sufficient historical experiences for the agent to learn from, thereby hindering its ability to make informed decisions. The agent may struggle to understand the environment effectively without access to a variety of past experiences.

Interestingly, when the memory capacity was increased to 50,000, the success rate fell to 51.00%, despite the expectation that more memory would enhance performance. This decline suggests that having an excessive number of stored experiences may introduce noise, making it difficult for the agent to discern which experiences are most relevant. The agent might spend too much time learning from outdated or irrelevant experiences, leading to a slower and less efficient learning process.

Overall, these findings indicate that there exists an optimal range for memory capacity, with 30,000 emerging as the most effective size for enhancing performance. This result highlights the delicate balance between retaining sufficient past experiences and avoiding the pitfalls associated with excessive historical data. Future research might involve exploring dynamic memory capacity strategies that adapt based on the agent's learning progress or the complexity of the task.

3.4 Overall analysis

Given the dynamic nature of the environment, the ability to adaptively tune these parameters could further enhance model performance. Future studies could explore the implementation of adaptive learning rate schedules or dynamic batch sizes that adjust during training. These approaches could lead to more efficient learning processes, particularly in environments characterized by rapid changes and unpredictability.

Additionally, incorporating advanced techniques such as Bayesian optimization for hyperparameter tuning could yield more robust and generalizable models. By systematically exploring the hyperparameter space and considering interactions between parameters, these methods can help identify configurations that significantly enhance the performance of RL agents in complex tasks.

4 Conclusion

This study effectively examined the impact of key hyperparameters—learning rate, batch size, and memory capacity—on the performance of a DDPG-based model controlling a robotic arm. By conducting systematic experiments, this paper determined that a learning rate of 0.001, a batch size of 50, and a memory capacity of 30,000 provided optimal results, achieving a success rate of 76.00% and an average time per goal of 0.07 seconds. These findings underscore the significance of optimizing hyperparameter in improving model efficiency and confirm that specific ranges can dramatically influence performance in continuous control tasks. Despite these contributions, the study has limitations, particularly in the interaction effects between hyperparameters, which warrant further investigation. The results are also context-specific and may not be applicable across diverse environments or robotic applications. Future studies should concentrate on creating adaptive strategies for hyperparameter tuning, such as dynamic adjustments based on instant feedback, and leveraging advanced optimization techniques for improved outcomes. Additionally, exploring more complex robotic scenarios and multi-agent environments may offer greater understanding of the adaptability and robustness of RL methodologies in practical applications. This work lays a foundation for ongoing research aimed at refining robotic control systems through intelligent learning approaches.

References

1. A. A. Awelewa, K. C. Mbanisi, S. O. Majekodunmi, I. A. Odigwe, A. F. Agbetuyi, I. A. Samuel, Development of a Prototype Robot Manipulator for Industrial Pick-and-Place Operations. International Journal of Mechanical Mechatronics Engineering, vol. 5, pp. 20-27 (2013).
2. K. Jahnavi, P. Sivraj, Teaching and learning robotic arm model. In 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), pp. 1570-1575 (2017).
3. V. Patidar, R. Tiwari, Survey of robotic arm and parameters. In 2016 International Conference on Computer Communication and Informatics (ICCCI), pp. 1-6 (2016).
4. C. Peng, S. M. Lavalle, Resolution complete rapidly-exploring random trees. IEEE International Conference on Robotics and Automation, vol. 1, pp. 267-272 (2002).
5. L. E. Kavraki, M. N. Kolountzakis, J. C. Latombe, Analysis of Probabilistic Roadmaps for Path Planning. IEEE Transactions on Robotics and Automation, vol. 14, no. 1, pp. 166-171 (1998).
6. Y. L. Hu, Q. S. Zhang, Multi-Robots Path Planning Based on Improved Artificial Potential Field Method. Advanced Materials Research, vol. 562-564, pp. 937-940 (2012).
7. Z. Li, H. Ma, Y. Ding, C. Wang, Y. Jin, Motion planning of six-dof arm robot based on improved DDPG algorithm. In 2020 39th Chinese Control Conference (CCC), pp. 3954-3959 (2020).
8. R. S. Sutton, A. G. Barto, Reinforcement Learning. A Bradford Book, vol. 15, no. 7, pp. 665-685 (1998).

9. A. Kumar, N. Paul, S. N. Omkar, Bipedal walking robot using deep deterministic policy gradient. arXiv preprint arXiv:1807.05924 (2018).
10. T. Lindner, A. Milecki, Reinforcement Learning-Based Algorithm to avoid obstacles by the anthropomorphic robotic arm. Applied Sciences, vol. 12, no. 13, 6629 (2022).
11. S. Wen, J. Chen, S. Wang, H. Zhang, X. Hu, Path planning of humanoid arm based on deep deterministic policy gradient. In 2018 IEEE International Conference on Robotics and Biomimetics (ROBIO), pp. 1755-1760 (2018).