Algorithmic Experiments on the MuJoCo Ant Robot

YAORUI YIN, YEKUN XU, and RUI ZHAO, Department of Computer Science and Technology, Tsinghua University, China

Abstract: This paper presents a series of algorithmic experiments conducted on the MuJoCo Ant robot. We have implemented the AC, TRPO, PPO, and DDPG algorithms based on existing physical simulation environment interfaces. During the process of tuning the hyperparameters, we gained insights into the unique logic of each algorithm, and based on the cumulative reward results during training, we have drawn some conclusions regarding the strengths and weaknesses of these algorithms. Moreover, we made some intuitive attempts on the neural network models built on top of the basic algorithms and obtained the corresponding performance curves.

1 Introduction

Reinforcement learning (RL) has become a dominant approach in solving complex control tasks, particularly in robotics. In this paper, we focus on a series of algorithmic experiments conducted on the MuJoCo Ant robot, a popular platform for evaluating RL algorithms in continuous control problems. We explore several well-known RL algorithms. These algorithms are implemented using existing physical simulation environment interfaces, which allow us to study their performance in the context of robotic control.

Throughout the experimentation process, we carefully tune the hyperparameters of each algorithm, gaining valuable insights into their unique operational characteristics. Based on the cumulative rewards obtained during training, we draw conclusions regarding the relative strengths and weaknesses of each algorithm in the context of robotic control tasks. In addition to the basic implementations, we also draw inspiration from the AlphaGo Zero paper [5] to explore modifications to the neural network models, aiming to improve the performance of the algorithms.

2 Preliminary Knowledge

This section provides the necessary background knowledge for the MuJoCo Ant robot(Shown in Figure 1), which is especially helpful for understanding the experiments conducted in this paper.

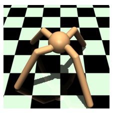

Fig. 1. MuJoCo Ant robot

2.1 MuJoCo Ant

Environment: The MuJoCo Ant environment[1] simulates a 3D quadruped robot, with each leg consisting of two body segments. The primary objective is to control the robot's locomotion by coordinating the movement of its four legs. The robot's motion is governed by applying torques to eight hinge joints that connect the two segments of each leg. The action space in this environment corresponds to the torques applied to these hinge joints, which is modeled as a continuous control space. The observation space includes several key elements, such as *qpos* (representing the positions

Authors' Contact Information: Yaorui Yin; Yekun Xu; Rui Zhao, Department of Computer Science and Technology, Tsinghua University, Beijing, China.

of the robot's body parts), *qvel* (denoting the velocities of the body parts), and *cfrc_ext* (which captures the external forces acting on the body parts). Specifically, the observation space consists of 13 elements for *qpos*, 14 elements for *qvel*, and 78 elements for *cfrc_ext*.

Reward: The reward function in this environment is designed to incentivize the ant to move efficiently in the rightward direction, while simultaneously minimizing excessive actuation and external contact forces. The total reward is a combination of the following components:

- *Healthy Reward:* A positive reward is granted when the robot maintains valid spatial values and when the torso height remains within a predefined range, ensuring the robot's stability.
- *Forward Reward:* A positive reward is awarded for advancing the robot in the rightward direction, promoting efficient locomotion.
- *Ctrl Cost:* A negative reward is imposed to penalize the robot for exerting excessive torques on its joints, discouraging unnecessary energy expenditure.
- *Contact Cost:* A negative reward is given when external contact forces between the robot's body and the environment exceed a specified threshold, encouraging the robot to avoid unwanted collisions.

The total reward is computed as follows:

$$R = R_{healthy} + R_{forward} - C_{ctrl} - C_{contact}.$$

This reward structure is intended to promote efficient locomotion while encouraging stability, minimizing energy consumption, and reducing the likelihood of undesirable interactions with the environment.

3 Methods

We have made the source code for this work publicly available, which can be accessed at the following GitHub repository: GitHub Repository

3.1 Neural Network Structure

In our code framework, the basic neural network layers are designed as follows: The action mean vector μ and standard deviation vector σ in the PolicyNet, as well as the value v in the ValueNet, are all obtained through two linear layers (MLP). The mean vector is mapped to the $[-1, 1]$ range required by the environment using the atanh activation function, while the standard deviation vector is mapped to the positive range through appropriate truncation with clamp and the softplus function (Shown in Figure 2). We apply this network structure to various reinforcement learning algorithms to obtain their performance results.

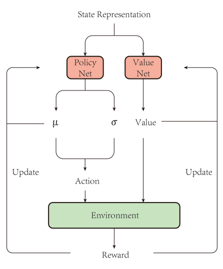

Fig. 2. Network Structures

3.2 Implemented Algorithms

In the Policy Gradient algorithm, based on Reinforcement Learning, we replace the actual sampled value with a one-step temporal difference $r + V(s_{t+1}) - \gamma V(s_t)$, which forms the AC algorithm [6].

TRPO [3], in order to prevent large fluctuations in each policy update that might lead to gradients pointing in the wrong direction, introduces a constraint on the KL divergence between the old and new policies:

$$\mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[\text{KL}\left(\pi_{\theta_{\text{old}}}(\cdot|s)\|\pi_\theta(\cdot|s)\right)\right] \le \delta.$$

The optimization objective is:

$$\max_\theta \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}(s,a)\right],$$

where conjugate gradient methods are used during the optimization process to save the computation of large matrix inversions.

Due to the computational complexity of TRPO, an alternative approach called PPO-clip [4] was introduced. This method simplifies the optimization process by directly clipping the importance weights between the old and new policies. The optimization objective function becomes:

$$\mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[\min\left(r_\theta(s,a)\hat{A}(s,a), \text{clip}(r_\theta(s,a), 1-\epsilon, 1+\epsilon)\hat{A}(s,a)\right)\right].$$

The DDPG [2] algorithm realizes off-policy learning through deterministic policy gradients:

$$\nabla_\theta J(\theta) = \mathbb{E}_s\left[\nabla_\theta \mu_\theta(s)\nabla_a Q^\mu(s,a)\Big|_{a=\mu_\theta(s)}\right],$$

and also draws from the Double-DQN network for slow policy updates.

3.3 Single MLP

Inspired by the success of AlphaGo Zero, where the neural network takes the raw board representation s of the game position along with its history as input, and outputs both the move probabilities and a value, $(p, v) = f_\theta(s)$ [5], we hypothesize that similar correlations between policy and value may exist in relatively simpler environments, such as the MuJoCo Ant. AlphaGo Zero, which defeated AlphaGo and became the world's best Go player, demonstrated the potential of such an approach. Additionally, we recognize the importance of time efficiency in real-world applications, where robots must respond to environmental stimuli in real time. While more complex models offer superior performance, there is also a strong need for smaller models that prioritize faster inference speeds.

In light of these considerations, our team aims to use a single neural network architecture to jointly represent both the policy and value networks within the PPO framework. Specifically, we employ a linear layer to encode the state from a 27-dimensional input to a hidden representation, which is then decoded by three separate linear layers to produce the policy parameters μ, σ, and the value output (Shown in Figure 3).

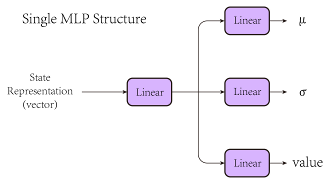

Fig. 3. Single MLP

3.4 CNN

Another potential approach to enhance the performance of the MuJoCo Ant environment is to improve the representation of the environment itself. While the 27-dimensional vector representation is effective, we hypothesize that 480×480 RGB images could provide richer and more detailed information. This is especially relevant given that real-world robots often rely on cameras to perceive their environment. Motivated by this, our team considers replacing the multi-layer perceptron (MLP) architecture traditionally used in PPO with a convolutional neural network (CNN).

We opt for a single-network structure due to its favorable performance and computational efficiency. Specifically, our network architecture consists of three blocks of 2D convolutional layers, batch normalization, and max-pooling operations, followed by a linear layer for encoding. The decoded outputs are produced by three distinct linear layers, corresponding to the policy parameters μ, σ, and the value output (Shown in Figure 4).

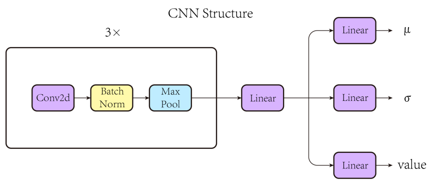

Fig. 4. CNN

4 Experiments

In our team's experiments, we adjust the hyperparameters of each algorithm based on the final performance curves and the rendered Ant movement videos to achieve approximately optimal results.

4.1 Performance of Different Algorithms

Table 1 presents the performance of all the implemented algorithms. The Actor-Critic algorithm exhibits unstable cumulative rewards in the later stages, and its convergence value is relatively low. We speculate that this may be due to the lack of control over the gap between the old and new policies. In contrast, PPO, which uses a clipped objective function, performs as expected. Although TRPO, which follows a similar approach, provides a more rigorous control of the policy gap, the increased computational complexity does not seem to lead to a significant improvement in performance.

It is worth noting that the DDPG algorithm shows an exceptionally ideal performance in terms of reward curves. However, upon observing the actual behavior of the Ant robot during experiments, we found that the strategy learned by the Ant tends to stagnate. We initially suspect this may be related to the healthy reward setting in the framework. Thus, we reduced its weight, but the Ant still failed to learn a forward-moving strategy. Even after further reducing the weight, it lost the ability to maintain balance. We further hypothesize that this could be due to the inability of the deterministic policy to adequately explore the action space. As a result, we increased the action standard deviation function in DDPG (which is hardcoded as a hyperparameter, not learned by the neural network) and also attempted to introduce the training trajectories obtained from the PPO algorithm into DDPG for learning, but the results were still unsatisfactory.

In summary, among the various on-policy algorithms, PPO shows the best performance, which is consistent with its widespread use in contemporary robotic control tasks. On the other hand, the off-policy DDPG algorithm failed to learn useful information during training. We hypothesize that this issue may be related to the challenges in training the Q function and the rigid exploration strategy inherent in deterministic policies.

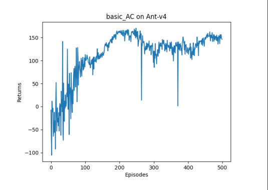	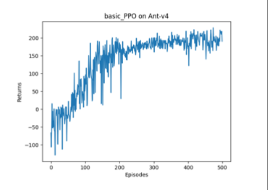	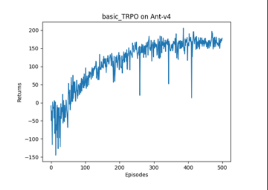	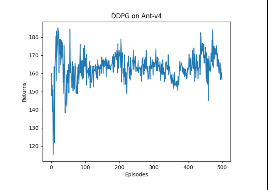
AC	PPO	TRPO	DDPG

Table 1. Performance of Different Algorithms

4.2 Advantage of GAE

In our experiments, we also explored the impact of Generalized Advantage Estimation (GAE) on the performance of PPO. GAE is a technique that aims to reduce the variance of the policy gradient estimator while introducing minimal bias. It is particularly useful for improving stability and performance in reinforcement learning tasks. GAE introduces a more generalized way of estimating the advantage function by combining multiple-step returns.

We tested different values of the lambda parameter, which controls the trade-off between bias and variance in the advantage estimation. The key formula for GAE is:

$$A^{\mathrm{GAE}}(s_t, a_t) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_t^l,$$

where δ_t is the TD error at time step t, and is given by:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

We ran three experiments (Shown in Table 2) with different lambda values: 0 (no GAE), 0.6 (more stable), and 0.95 (too dependent on others). The cumulative reward results show that a lambda value of 0.6 yields the best performance, balancing the trade-off between bias and variance. In contrast, setting lambda to 0 results in higher variance and less stability, while a value of 0.95 leads to excessive dependence on future rewards, which negatively impacts the performance. These findings suggest that a moderate lambda value is key to optimizing the stability and effectiveness of RL algorithms in robotic control tasks.

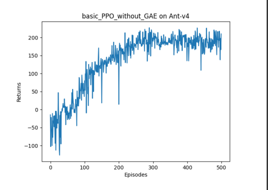	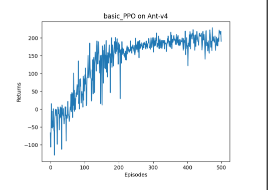	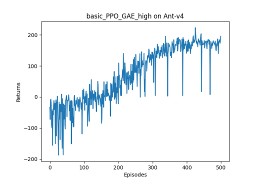
lambda=0	lambda=0.6	lambda=0.95

Table 2. Performance on different lambdas

4.3 Single MLP

As shown in Figure 5, PPO with single MLP slightly outperforms Normal PPO and also trains faster. These results suggest that PPO with single MLP is more effective at capturing the correlations between policy and value while incurring lower computational costs. Additionally, the training of the single-net algorithm requires a smaller ϵ for clipping, likely because it tends to update more frequently with larger gradients in each iteration.

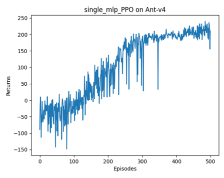

Fig. 5. Single MLP

4.4 CNN

Due to resource and environmental constraints, the training process has faced challenges. (Insufficient information in the pictures results in NaN value in model outputs) However, we observe a significant upward trend in the return curve, which indicates that our approach holds promise. Moreover, adding the original state vector as non-graphic input may make the model multimodal, thereby improving its applicability to real-world tasks, which will be tried in our future works.

5 Conclusion

In this work, we have explored various reinforcement learning algorithms and their performance in the MuJoCo Ant environment. Through a series of experiments, we have identified key factors that contribute to the success of these algorithms and propose improvements based on our findings.

Our experiments demonstrated that among the on-policy algorithms, PPO outperformed others, showing stable and reliable performance. On the other hand, DDPG, an off-policy algorithm, struggled to learn effective strategies, primarily due to limitations in exploration and deterministic policy constraints. Furthermore, we observed that the choice of the lambda parameter in Generalized Advantage Estimation (GAE) plays a crucial role in balancing bias and variance, with a value of 0.6 yielding the best performance in our experiments.

Additionally, we introduced a single-net structure in PPO, inspired by the success of AlphaGo Zero, which combines both policy and value functions into a single network. This approach not only reduced computational costs but also enhanced the ability to capture correlations between policy and value. The results showed that PPO with a single MLP outperforms the traditional PPO and converges faster, which validates our approach.

Moreover, we explored the potential of using convolutional neural networks (CNNs) for processing image inputs in reinforcement learning tasks. Although resource limitations hindered the full exploration of this approach, the significant improvement in the return curve indicates that using CNNs may lead to better performance in more complex, multimodal environments. This direction offers promising opportunities for future work.

In the current project, our use of CNNs has not fully leveraged their potential. In future work, we will explore the multi-modal approach mentioned in the Experiments section. Additionally, we will further investigate other off-policy algorithms, including SAC, among others.

References

[1] Farama. 2023. MuJoCo Ant Environment. https://gymnasium.farama.org/environments/mujoco/ant/.

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tomer Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[3] John Schulman, Filip Wolski, Puneet Dhariwal, Alec Radford, and Oleg Klimov. 2015. Trust region policy optimization. *arXiv preprint arXiv:1502.05477* (2015).

[4] John Schulman, Filip Wolski, Puneet Dhariwal, Alec Radford, Xue Qiu, Ilya Sutskever, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[5] David Silver, Thomas Hubert, Julian Schrittwieser, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354–359.

[6] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3 (1992), 229–256.