PPO vs. DDPG: A Critical Analysis of Reinforcement Learning Performance

Benyamain Yacoob, Caitlin Snyder

Department of Electrical & Computer Engineering & Computer Science
University of Detroit Mercy
Detroit, MI, United States
{yacoobby, snydercr}@udmercy.edu

***Abstract*—This paper closely examines two important reinforcement learning methods: Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG). These methods solve different problems in training AI agents. PPO is great for making reliable updates in various situations, while DDPG is effective for handling tasks with continuous possible actions. We explain how each algorithm works, its main parts, and things to consider when using them, backing it up with math and findings from other research. We also compare their designs, how they work, and their performance to see when one is better than the other. The findings show that PPO is more stable and efficient in tasks with clear-cut choices and simulated environments. DDPG works better in complicated situations with continuous choices that need fine-tuned actions.**

***Index Terms*—reinforcement learning, proximal policy optimization, deep deterministic policy gradient, continuous control, policy optimization**

I. Introduction

Reinforcement learning (RL) is now a primary method for training AI to make a series of decisions by learning from an environment. Two algorithms, Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG), are especially good at solving specific RL problems. PPO, an on-policy method, balances stability and performance in policy updates, while DDPG, an off-policy approach, performs well in continuous action spaces by combining Q-learning with policy gradients. This paper critically analyzes these algorithms, focusing on their methodologies, practical considerations, and comparative performance.

The goal is to clarify how PPO and DDPG function, compare their algorithmic designs, and identify which delivers better performance in specific contexts. The analysis draws on foundational papers, mathematical formulations, and code implementations to offer a thorough evaluation, advancing the understanding of RL techniques in neural network applications.

II. PPO Description

A. Methodology and Key Algorithms

Proximal Policy Optimization (PPO), introduced by Schulman et al. [1], is an on-policy reinforcement learning algorithm that enhances the stability of earlier policy gradient methods, such as Trust Region Policy Optimization (TRPO). PPO achieves this by limiting policy updates to prevent significant deviations from the previous policy, utilizing a clipped surrogate objective function. The algorithm alternates between gathering experience through environment interactions and optimizing this objective over multiple epochs with stochastic gradient ascent.

The core objective in PPO-Clip is defined as:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{1}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, $\hat{A}_t$ is the advantage estimate, and ϵ (typically 0.2) restricts the policy update size. This clipping mechanism helps to maintain the new policy close to the old one, improving training stability.

PPO also includes a value function $V_\phi(s)$ to estimate expected returns, optimized via:

$$L^{\text{VF}}(\phi) = \hat{\mathbb{E}}_t \left[(V_\phi(s_t) - \hat{R}_t)^2 \right], \tag{2}$$

where $\hat{R}_t$ represents rewards-to-go. The combined loss, with an entropy bonus for exploration, is:

$$L^{\text{CLIP+VF+S}}(\theta, \phi) = \hat{\mathbb{E}}_t \left[L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\phi) + c_2 S[\pi_\theta](s_t) \right], \tag{3}$$

with coefficients c_1 and c_2 balancing the terms.

B. Practical Implementation Considerations

The implementation of PPO, as shown in the PyTorch code [2], uses a buffer to store trajectories, applying Generalized Advantage Estimation (GAE) with parameters $\gamma = 0.99$ and $\lambda = 0.97$ to calculate advantages:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{4}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$. This method reduces variance in advantage estimates, boosting learning efficiency.

Key practical features include early stopping based on KL-divergence (target 0.01) to limit excessive policy shifts, and parallelization via MPI for scalability. The policy and value networks are typically multilayer perceptrons (MLPs) with hidden layers (e.g., 64 units each), optimized using the Adam optimizer with learning rates 3×10^{-4} for the policy and 1×10^{-3} for the value function.

C. Performance Insights

Schulman et al. [1] demonstrated the effectiveness of PPO on benchmarks such as MuJoCo tasks (e.g., HalfCheetah) and Atari games, achieving average normalized scores of 0.82 with $\epsilon = 0.2$. Its simplicity and sample efficiency make it well-suited for both discrete and continuous action spaces, outperforming methods like A2C in terms of sample complexity.

III. DDPG Description

A. Methodology and Key Components

Deep Deterministic Policy Gradient (DDPG), proposed by Lillicrap et al. [3], is an off-policy actor-critic algorithm designed for continuous action spaces. It combines Q-learning with deterministic policy gradients, simultaneously learning a Q-function $Q_\phi(s, a)$ and a deterministic policy $\mu_\theta(s)$.

$$Q_\phi(s_t, a_t) = \mathbb{E}\left[r(s_t, a_t) + \gamma Q_\phi(s_{t+1}, \mu_\theta(s_{t+1}))\right], \quad (5)$$

and the policy is optimized to maximize the Q-value:

$$\nabla_\theta J \approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_\theta Q_\phi(s, a)|_{s=s_t, a=\mu_\theta(s_t)}\right]. \quad (6)$$

DDPG tackles instability in neural network training with two features: a replay buffer to store transitions (s, a, r, s', d), supporting off-policy learning, and target networks $Q_{\phi_{\text{targ}}}$ and $\mu_{\theta_{\text{targ}}}$, updated via Polyak averaging:

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi, \quad \theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta, \quad (7)$$

where $\rho = 0.995$ is typical.

B. Network Architecture and Exploration Strategy

The PyTorch implementation [4] uses an MLP actor-critic architecture with hidden layers (e.g., 256 units each). The Q-function loss is:

$$L(\phi) = \mathbb{E}\left[\left(Q_\phi(s, a) - (r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s')))\right)^2\right], \quad (8)$$

optimized with a Q-learning rate of 1×10^{-3}, while the policy loss is:

$$L(\theta) = -\mathbb{E}\left[Q_\phi(s, \mu_\theta(s))\right], \quad (9)$$

optimized with a policy learning rate of 1×10^{-3}.

To encourage exploration during training, Gaussian noise (standard deviation 0.1) is added to actions, and a uniform random action phase is implemented for the first 10,000 steps. At test time, deterministic actions are used to assess performance.

C. Performance Metrics

Lillicrap et al. [3] tested DDPG on over 20 MuJoCo tasks, achieving normalized scores exceeding 1.0 in cases like HardCheetah (1.311 average), often outperforming planning-based solvers like iLQG. Its capacity to learn from pixels and manage high-dimensional continuous control tasks underscores its strengths.

IV. Critical Analysis

A. Algorithm Design Differences

PPO and DDPG differ fundamentally in their RL approaches. PPO's on-policy nature requires fresh data per update, using a clipped objective to maintain stability, whereas DDPG's off-policy method uses a replay buffer for data efficiency, relying on target networks to stabilize Q-learning. PPO's stochastic policy supports exploration via entropy regularization, while DDPG's deterministic policy requires external noise for exploration.

Mathematically, PPO limits policy updates within a trust region ($1-\epsilon$ to $1+\epsilon$), contrasting with DDPG's unconstrained gradient ascent on the Q-function. This design makes PPO less likely to experience drastic updates but potentially slower to converge in complex tasks, whereas DDPG adapts quickly to continuous domains but may face instability without careful tuning.

B. Execution and Implementation

In execution, PPO's multi-epoch optimization per trajectory batch improves sample efficiency, as seen in its strong Atari performance (30 games won in training average) [1]. DDPG's single-step updates per batch, paired with a large replay buffer (1 million transitions), allow it to learn from varied experiences, excelling in tasks like locomotion (e.g., Cheetah) [3].

Implementation-wise, PPO's simplicity (requiring fewer hyperparameters and no separate target networks) contrasts with DDPG's complexity, involving dual networks and noise tuning. PPO's early stopping based on KL-divergence enhances stability, while DDPG's reliance on Polyak averaging requires precise ρ selection.

C. Performance Comparison

Performance depends on the task domain. In discrete action spaces like Atari, PPO outperforms DDPG (not applicable due to its continuous focus) and other methods like A2C, achieving higher sample efficiency [1]. In continuous control tasks (e.g., MuJoCo), PPO performs well (0.82 normalized) but is surpassed by DDPG in complex scenarios (e.g., HardCheetah: 1.311 vs. PPO's typical < 1.0), where precise action control is vital [3].

PPO's stability suits environments needing consistent progress, such as robotic simulation with discrete elements. DDPG's strong performance in high-dimensional continuous tasks, like humanoid locomotion, supports its use where fine-tuned actions are more critical than stability.

V. Conclusion

This analysis shows that PPO and DDPG address different RL needs. PPO provides a stable, sample-efficient solution for diverse environments, performing well in tasks with moderate complexity. DDPG, in contrast, delivers strong performance in continuous control, outperforming PPO in scenarios requiring precise action modulation. The choice between them depends

on task demands: PPO for stability and broad applicability, DDPG for complex continuous domains.

REFERENCES

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[2] S. Up, "Ppo pytorch implementation." https://github.com/openai/spinningup/tree/master/spinup/algos/pytorch/ppo, 2018. OpenAI.

[3] 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 *International Conference on Learning Representations (ICLR)*, 2016.

[4] S. Up, "Ddpg pytorch implementation." https://github.com/openai/spinningup/tree/master/spinup/algos/pytorch/ddpg, 2018. OpenAI.