12-6-2023

Comparative Study of Reinforcement Learning Algorithms: Deep Q-Networks, Deep Deterministic Policy Gradients and Proximal Policy Optimization

Haoyi Wang
Washington University in St. Louis, haoyi@wustl.edu

Comparative Study of Reinforcement Learning Algorithms: Deep Q-Networks, Deep Deterministic Policy Gradients and Proximal Policy Optimization

Haoyi Wang
haoyi@wustl.edu
Department of Electrical & System Engineering, Washington University in St. Louis, St. Louis, MO, USA
Research Supervisor: Bruno Sinopoli

Washington University in St. Louis
James McKelvey School of Engineering

Introduction

Background

Artificial Intelligence (AI) has dramatically advanced through Reinforcement Learning (RL), particularly with the development of algorithms like Deep Q-Networks (DQN), Deep Deterministic Policy Gradients (DDPG), and Proximal Policy Optimization (PPO). These algorithms have been pivotal in diverse RL challenges.

Purpose of the Project

This study compares DQN, DDPG, and PPO, aiming to clarify their mechanics, efficiencies, and practical applications. We assess their theoretical foundations, performance in standard benchmarks, and adaptability to various environments. The goal is to offer insights for choosing the right algorithms for different environments.

Theoretical Background

Deep Q-Networks (DQN)

- Given state information, output the best policy based on the Q-value function, which estimates the expected cumulative rewards for each action in each state.
- Integrates deep neural networks with Q-learning.
- Experience Replay method is introduced.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
  end for
end for

Deep Deterministic Policy Gradients (DDPG)

- Given state and action information, output the best policy based on the estimation of Action value and Q-value.
- Actor-critic approach: Actor learns the policy; Critic learns the Q-value function.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Proximal Policy Optimization (PPO)

- Given current state, output action probabilities or specific actions and a value estimate for those actions.
- Use the Clipping mechanism to prevent large policy updates.

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

Algorithm 1 PPO, Actor-Critic Style

for iteration$=1, 2, \ldots$ **do**
  for actor$=1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Analysis & Results

Experiment Setup

The study utilizes classic control environments from OpenAI Gym, specifically the Mountain Car (Discrete Action Space), Mountain Car Continuous (Continuous Action Space), and Cart Pole (Discrete Action Space).

The experiment is implemented by code projects running locally and is divided into two parts:

- Mountain Car for Q-learning and Mountain Car Continuous for DDPG. Compare the performance by measuring the time it takes for the car to reach the top of the hill in the simulation.
- Cart Pole for DQN and PPO. Compare the performance of the two algorithms' rewards and the holding time for the inverted pendulum in the simulation.

Results

Both Q-learning and DDPG successfully solved the related problems. Since the environments used are different, we cannot directly compare the rewards of the algorithms.

Q-learning trained for 48 seconds for 5000 episodes, while DDPG trained for 7 minutes and 29 seconds for 200 episodes.

In terms of performance in simulations of the trained cars, DDPG outperforms Q-learning, using only 1 second compared to the 5 seconds required by Q-learning.

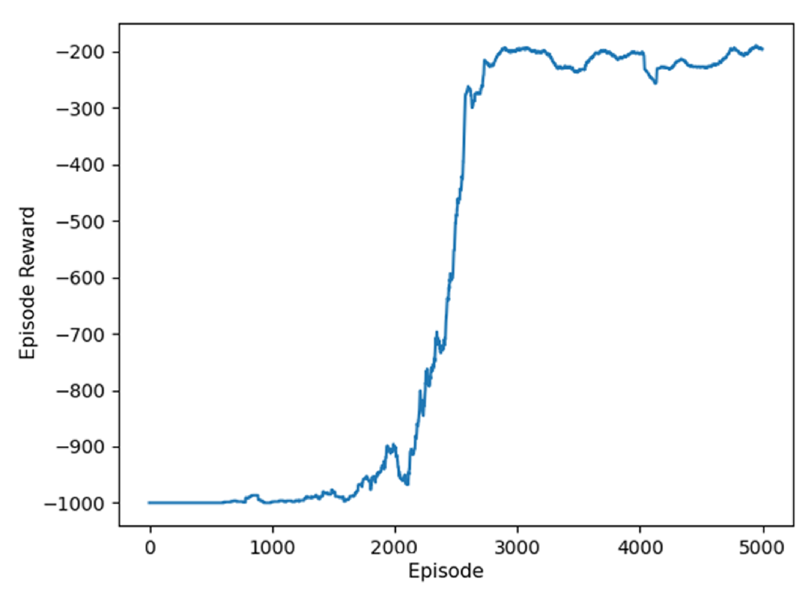

Figure 1. Training Reward for Mountain Car Q-learning

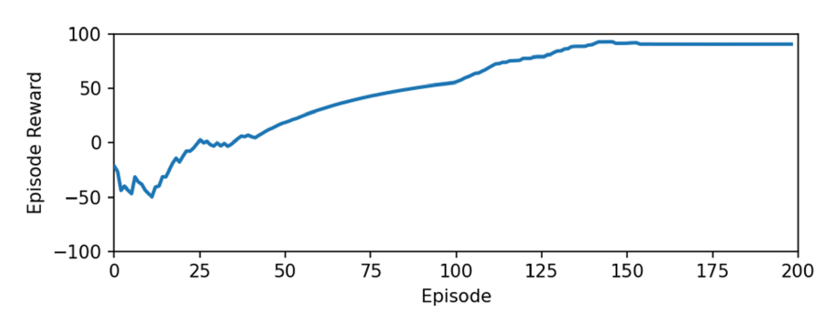

Figure 2. Training Reward for Mountain Car Continuous DDPG

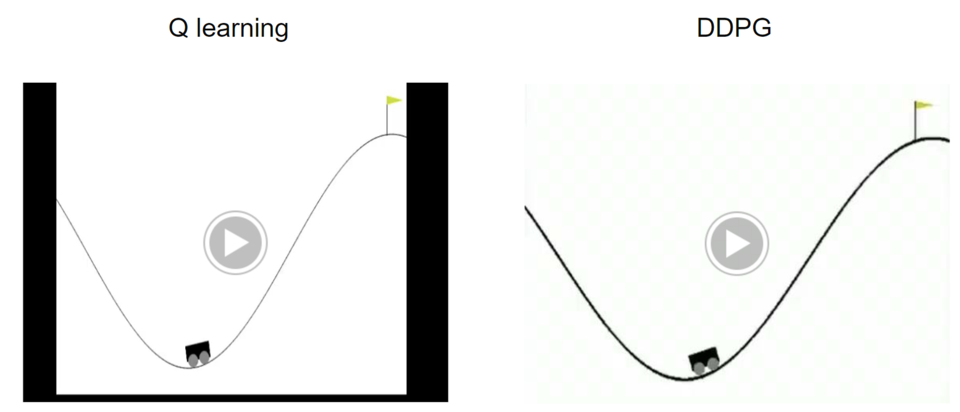

Figure 3: Simulation videos of Q-learning and DDPG

By comparing the reward graphs, we can conclude that PPO outperforms DQN.

DQN trained for 3 hours and 5 minutes, while PPO trained for 7 minutes and 34 seconds.

In terms of performance in simulations of the trained pendulums, PPO outperforms DQN by stabilizing the inverted pendulum for a longer time, e.g., at the 120th episode, PPO holds for 54 seconds while DQN holds for only 14 seconds.

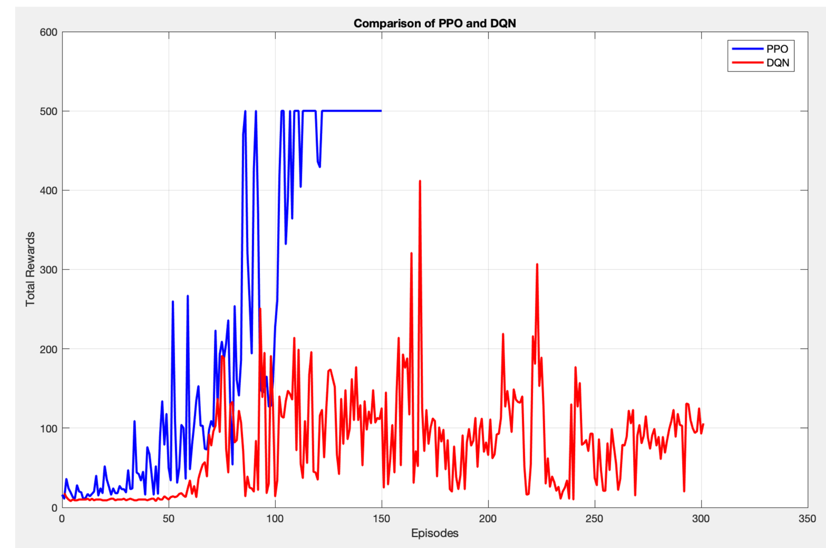

Figure 4. Training Reward for CartPole DQN and PPO

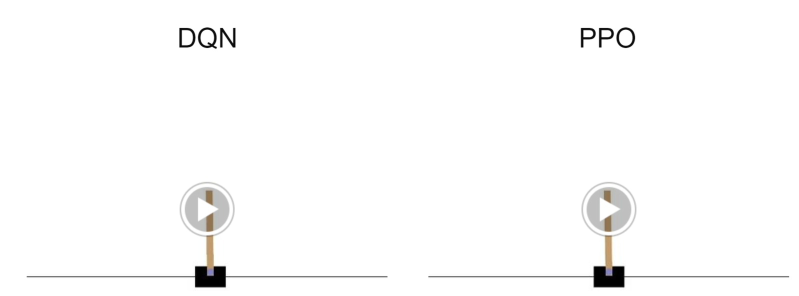

Figure 5: Simulation videos of DQN and PPO

Conclusion

Our analysis has demonstrated that while DQN excels in discrete action spaces, DDPG is more suited for continuous control tasks, and PPO shows consistent performance offering versatility and ease of tuning. This research not only aids in informed algorithm selection but also sets a foundation for future exploration in this area of study.

Related Publications

- [1] V. Mnih et al. "Playing atari with deep reinforcement learning". In: arXiv preprint arXiv:1312.5602 (2013).
- [2] T. P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: arXiv preprint arXiv:1509.02971 (2015).
- [3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms". In: arXiv preprint arXiv:1707.06347 (2017).

Acknowledgment

Carmel Fiscko, Haoyu Yin and Chengyu Li