Performance Analysis and Optimization of Autonomous Learning for a Quadruped Robot

Github

Abstract

This project demonstrates the successful implementation of a reinforcement learning agent within a 3D simulated environment using the Unity ML-Agents Toolkit. The primary objective was to train a robotic agent to proficiently complete a designated task [e.g., navigating a complex maze, collecting target objects]. We employed the Proximal Policy Optimization (PPO) algorithm, supplemented by a curiosity-driven exploration module, to develop the agent's policy. After extensive training exceeding 40 million steps and overcoming significant environment configuration challenges, the agent exhibited remarkable performance improvement. The mean cumulative reward increased from a baseline of approximately 26 to a stable peak above 480, signifying the successful acquisition of an effective and efficient policy. This work validates the efficacy of the chosen RL methodology and underscores the critical importance of a stable, reproducible training environment for complex AI projects.

Introduction

At the core of this project is the Reinforcement Learning (RL) decision-making and learning loop, as illustrated in Fig. 1.

Fig.1 illustrates the reinforcement learning loop, where the agent perceives the environment, uses a Policy Network to select an action, and then learns from the resulting feedback to update its policy for subsequent cycles.

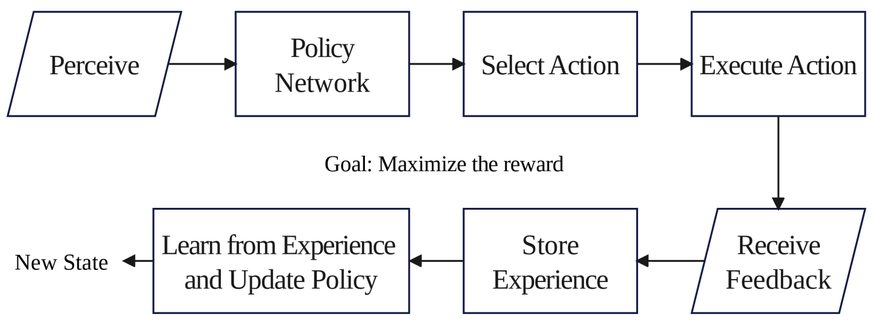

Fig.1 System workflow

Result

The training process was accelerated using an NVIDIA GPU, with performance monitored over 6 million steps. The agent's learning dynamics are detailed in the Cumulative Reward and Value Loss curves(Fig.2).

As illustrated in Fig.2, the agent's Cumulative Reward shows a rapid learning phase within the first 800,000 steps, where performance surged from a near-zero baseline to a peak exceeding 800. Subsequently, the reward entered a high-variance plateau, indicating that while the agent had mastered effective strategies, its performance had not yet fully stabilized.

The Value Loss curve corroborates this narrative. The loss steadily decreased during the initial learning phase, followed by a sharp drop at approximately 1.8 million steps. This event signifies a breakthrough in the agent's ability to accurately predict future rewards. However, a slight upward trend in loss after 5.5 million steps suggests the onset of potential late-stage instability, indicating an area for future hyperparameter refinement.

Fig.2 Graph of Robot learning reward and loss

Methodology

The methodology for this project can be broken down into the learning algorithm, network architecture, and the technical environment setup.

- A. Learning Algorithm: We selected Proximal Policy Optimization (PPO), a state-of-the-art policy gradient method known for its stability and data efficiency. PPO iteratively improves the agent's policy while ensuring that the updates do not deviate too drastically from the previous policy, thus preventing destructive updates. To enhance exploration in potentially sparse reward scenarios, we integrated a Curiosity Module as an intrinsic reward signal, encouraging the agent to explore novel states.
- B. Network Architecture: The agent's policy was parameterized by a deep neural network consisting of:
  - An input layer processing vector observations from the environment.
  - Three hidden layers, each with 256 units, using ReLU activation functions.
  - A Long Short-Term Memory (LSTM) unit with a memory size of 128 to enable the agent to retain information from previous states, crucial for tasks requiring memory.
  - Separate output heads for the policy (action probabilities) and the value function (state-value estimate).
- C. Training Environment & Hyperparameters: A "golden" training environment was meticulously constructed to ensure stability and reproducibility, overcoming initial dependency conflicts.
  - Platform: Unity ML-Agents v2.2.1, Python 3.8.
  - Core Libraries: mlagents==0.28.0, torch==1.13.1+cu117, protobuf==3.20.3.
    - Key Hyperparameters:learning_rate: 1.0e-4
    - batch_size: 2048
    - buffer_size: 20480
    - gamma (discount factor): 0.99
    - lambd (GAE parameter): 0.95

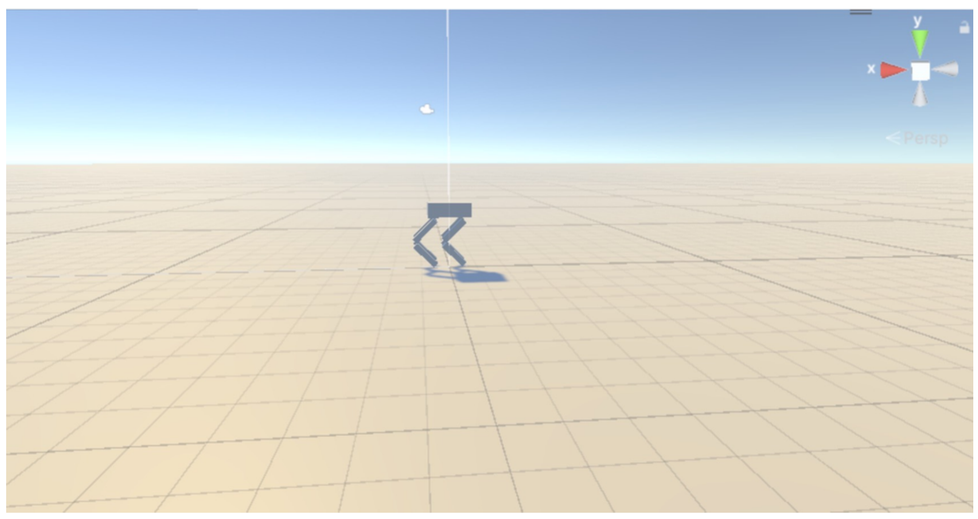

Fig.3 Graph of Quadruped Robot

Conclusion

This project successfully demonstrates the training of an autonomous robotic agent using deep reinforcement learning. The PPO algorithm, augmented with a curiosity module, proved highly effective in enabling the agent to learn a complex task within a simulated 3D environment. The nearly 20-fold increase in mean reward provides strong evidence of the agent's learning and mastery.

A critical takeaway from this project was the paramount importance of meticulous environment configuration and dependency management. A significant portion of the project involved diagnosing and resolving complex package conflicts between Python, PyTorch, and the ML-Agents framework. The final success was only achieved after establishing a "golden environment" with locked package versions. This highlights that for modern AI projects, a robust and reproducible software environment is as crucial as the algorithm itself.

Future work could explore the application of other algorithms like Soft Actor-Critic (SAC), increasing the complexity of the environment, and transferring the learned policy to a physical robot.