Enhancing Game Control Through Hybrid Reinforcement Learning

Danhua Yan [1]

1. Introduction

Training Reinforcement Learning (RL) agents usually requires a substantial amount of data and exploration to find an optimal policy. In many complex games, the challenges of high-dimensional state spaces, sparse rewards, and complex dynamics make training agents using pure exploration particularly inefficient. Moreover, in cases where exploration opportunities are limited or costly, the RL agent might fail to learn any usable policies (Coletti et al., 2023). Such inefficiency not only slows down learning but also increases the risk of converging to policies that are far from optimal.

The research field of bootstrapping an RL agent's policy from demonstrations or imitation learning shows significant promise. Various hybrid paradigms that combine human guidance as offline RL and agent exploration as online RL have shown they can accelerate policy learning and achieve above-demonstration performance (Hester et al., 2017; Nair et al., 2018; Song et al., 2023; Ren et al., 2024; Coletti et al., 2023).

This project investigates how hybrid RL can effectively enhance game control through guided explorations of the agent. It aims to evaluate the potential for achieving performance that surpasses the demonstration level.

2. Approach

2.1. Environement

In this project, we leverage the `stable-retro` library to create an OpenAI Gym environment for training an agent to play the NES game Super Mario Bros, level 1-1. The default integration of the environment encapsulates the game into the in-game visual frame as a matrix $I \in \mathbb{R}^{H\times W\times 3}$, where each element takes an integer value between 0 and 255, representing the RGB channels of the frame. The action of pressing 9 buttons on NES controllers is represented as a vector $\mathbf{a} = (a_1, a_2, \ldots, a_9) \in \{0, 1\}^9$, where each button can be toggled independently, resulting in a total of 512 discrete action spaces. The default reward is the change in the x-axis position Δx moved by Mario. In-game metadata, including scores, time left, and positions of Mario, can also be retrieved for each timestep t.

To record human demonstrations, we implemented scripts that save the gameplay through interactions with the environment, controlled via a game controller. The trajectory data of each episode i is saved as $\tau^i_{hd}\{(s_t, a_t, r_t, d_t, m_t)\}^T_{t=0}$, where each element in the tuple represents the state, action, reward, termination boolean, and metadata. We recorded five gameplays by amateur players, each successfully finishing Level 1-1 without losing a life of Mario.

To frame the game into a proper RL problem that can be solved within a reasonable time, we made below custom modifications to the default integration of the game:

Action Space The default 512 discrete action space captures all possible joystick button combinations. However, most of these combinations are not meaningful for controlling Mario. From the human demonstration trajectories, we narrowed down the action space to 10 common used button combinations (see Appendix A.1).

Termination States The default termination of the game occurs either when Mario has exhausted all his lives (3 to start with) or when the time limit of 400 seconds for Level 1-1 is reached. This setting poses challenges for RL exploration, as it may take too long to wait for the game to finish if Mario gets stuck at some point. We employ a stricter termination state: 1) Mario has only one life, and the game terminates immediately if he loses it; 2) If Mario remains stuck at the same position without moving to the right for 10 seconds, the game is terminated.

Reward Function The default reward function is simply $\mathcal{R} = \min(0, \Delta x)$, which awards Mario for moving right and does not penalize for moving left. To incorporate other elements of the game, such as collecting coins, power-ups, defeating enemies, and to penalize Mario for not moving or losing the game, we adjust the reward function as follows: $\mathcal{R} = \min(0, \Delta x) + \Delta k - \Delta \mathrm{t} - \mathbf{1}[d_t = 1]p$, where Δk is the score earned since the last state, Δt is the time spent in seconds since the last state, and p is the penalty for terminating the game without successfully reaching the destination.

[1]Department of Computer Science, Stanford University. Correspondence to: Danhua Yan <dhyan@stanford.edu>.

CS234: Reinforcement Learning Winter 2025 - Final Project

2.2. Baselines

Offline-only and online-only approaches are used for comparisons:

2.2.1. Deep Q-Learning (DQN)

Online-only RL: Train an agent with online exploration only, leveraging Deep Q-Learning (DQN) with ϵ-greedy explorations.

2.2.2. Behavior Cloning

Imitation Learning Only: Train a policy via behavioral cloning (BC) using human demonstrations.

2.3. Hybrid Reinforcement Learning (HRL)

2.3.1. Deep Q-Learning from Demonstrations (DQfD)

Following the DQfD (Deep Q-Learning from Demonstrations) framework by (Hester et al., 2017), which incorporates expert demonstrations into the replay buffer of DQN to control explorations.

3. Experiements

4. Next Steps

Leveraging behavioral cloning (BC) as a warm-start, then further leveraging PPO (Proximal Policy Optimization) for policy fine-tuning. This approach is inspired by (Coletti et al., 2023).

References

Coletti, C. T., Williams, K. A., Lehman, H. C., Kakish, Z. M., Whitten, D., and Parish, J. Effectiveness of warm-start ppo for guidance with highly constrained nonlinear fixed-wing dynamics. *2023 American Control Conference (ACC)*, pp. 3288–3295, 2023. URL https://api.semanticscholar.org/CorpusID:259338376.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J. Z., and Gruslys, A. Deep Q-learning from Demonstrations, November 2017. URL http://arxiv.org/abs/1704.03732. arXiv:1704.03732 [cs].

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming Exploration in Reinforcement Learning with Demonstrations, February 2018. URL http://arxiv.org/abs/1709.10089. arXiv:1709.10089 [cs].

Ren, J., Swamy, G., Wu, Z. S., Bagnell, J. A., and Choudhury, S. Hybrid Inverse Reinforcement Learning, June 2024. URL http://arxiv.org/abs/2402.08848. arXiv:2402.08848 [cs].

Song, Y., Zhou, Y., Sekhari, A., Bagnell, J. A., Krishnamurthy, A., and Sun, W. Hybrid RL: Using Both Offline and Online Data Can Make RL Efficient, March 2023. URL http://arxiv.org/abs/2210.06718. arXiv:2210.06718 [cs].

A. Appendix

A.1. Custom Environment

The default 512 discrete action space captures all possible joystick button combinations. However, most of these combinations are not meaningful for controlling Mario. From the human demonstration trajectories, we narrowed down the action space to 10 common used button combinations. Then the action vector is labeled as integers (0-9, following below orders) as discrete action space for the environment.

```
# List of meaningful button combinations used in gameplay
meaningful_actions = [
      [0, 0, 0, 0, 0, 0, 0, 0, 0],  # No action
      [0, 0, 0, 0, 0, 0, 0, 0, 1],  # A (Jump)
      [0, 0, 0, 0, 0, 0, 0, 1, 0],  # Right
      [0, 0, 0, 0, 0, 0, 0, 1, 1],  # Right + A (Jump)
      [0, 0, 0, 0, 0, 0, 1, 0, 0],  # Left
      [0, 0, 0, 0, 0, 0, 1, 0, 1],  # Left + A (Jump)
      [1, 0, 0, 0, 0, 0, 0, 0, 0],  # B (Run)
      [1, 0, 0, 0, 0, 0, 0, 0, 1],  # A + B (Jump + Run)
      [1, 0, 0, 0, 0, 0, 0, 1, 0],  # Right + B (Run)
      [1, 0, 0, 0, 0, 0, 0, 1, 1]   # Right + A + B (Jump + Run)
]
```