Enhancing Game Control Through Hybrid Reinforcement Learning

Danhua Yan [1]

Abstract

This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. Gross violations will trigger corrections at the camera-ready phase.

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 (HRL) 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. Related Work

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

CS234: Reinforcement Learning Winter 2025 - Final Project

3. Data and 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.

3.1. Human Demonstration Data

To record human demonstrations, we implemented scripts to save gameplay interactions with the environment via a game controller. 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 represents the observation, action, reward, termination boolean, and metadata. We recorded five gameplays by amateur players, each successfully completing Level 1-1 without losing a life. Additionally, a single trajectory of game emulator states is saved every 50 steps, used for resetting RL agents to start from a state along the human demonstrated trajectory.

3.2. Customized Game Environement

To frame the game as a solvable RL problem within a reasonable time, we made the following custom modifications to the default game integration:

Action Space The default 512 discrete action space includes all possible joystick button combinations, most of which are not meaningful for controlling Mario. We reduced the action space to 3 commonly used button combinations (see Appendix A.1).

Termination States The default game termination occurs when Mario exhausts all lives or the 400 second time limit for Level 1-1 is reached. We employ stricter termination conditions: 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 right for 10 seconds, the game is terminated.

Reward Function The game's scoring system provides sparse rewards for defeating enemies, collecting coins or power-ups, and successfully completing the level. We modify the reward function to provide dense rewards, incorporating scores, encouraging rightward movement with milestones, and penalizing time consumption and termination without success:

$$\begin{aligned}\mathcal{R} =& \Delta s + \beta_x \Delta x + \beta_t \Delta t + \mathbf{1}[d_{\text{milestones}} = 1]M \\ &+ \mathbf{1}[d_{\text{timeout}} = 1]T_t + \mathbf{1}[d_{\text{death}} = 1]T_d\end{aligned}$$

where Δs is the score earned since the last state, Δx is the movement, Δt is the time spent, β are coefficients, M is the milestone score at 10%, 20%, etc., of the level, and T_t and T_d are penalties for timeout and death terminations.

Sampling Rate To ensure smooth rendering, the game runs at 60 fps. However, consecutive frames exhibit minimal differences. Following (PyTorch, 2024), we reduced the sampling rate to 15 fps to enhance efficiency.

4. Approach

In this section, we describe baseline and three different hybrid reinforcement learning approaches in controlling Mario for completing level 1-1.

4.1. Policy Network Archetectures

For comparisons, all baselines and HRL approaches will leverage the same architecture of a convolutional neural network $\pi(s, a; \theta)$ that maps an RGB image of dimensions $H \times W \times 3$ to a discrete action space of 10 actions. The architecture consists of three convolutional layers, with 16, 32, and 64 hidden channels respectively. Each convolution uses a 3×3 kernel with stride 1 and padding 1, followed by a ReLU activation and 2×2 max pooling that halves the spatial dimensions. The output feature maps are flattened into a vector, then processed by a fully connected hidden layer with 512 units and ReLU activation. Finally, a linear output layer projects the 512-dimensional representation to 10 logits corresponding to the available actions. All models are trained using the AdamW optimizer with default settings from PyTorch, and a learning rate of 10^{-4} unless otherwise specified.

4.2. Baselines

Here we shall establish performance of offline-only and online-only RL approaches as baselines to compare against future HRL approaches and human demonstration trajectories.

4.2.1. Deep Q-Learning (DQN)

Here we use the DQN architecture with a linearly decaying ϵ-greedy exploration as a baseline for the online RL approach through pure explorations. We implemented the DQN architecture from scratch to facilitate future modifications for the HRL approach DQfD.

We set the DQN parameters as follows: the discount factor $\gamma = 0.99$. The replay buffer has a capacity of $|\mathcal{D}| = 10000$, where new states overwrite old states in a FIFO manner. The policy is trained for 2500 episodes, with the target policy network updated after each episode.

We made a minor modification to the classic DQN algorithm: instead of sampling a minibatch of samples and updating the policy network $\pi(s, a; \theta)$ for every t, we set a parameter ω that allows the agent to interact with the environment for ω steps before updating $\pi(s, a; \theta)$. The rationale is that in Super Mario Bros, after an action is played, especially for a jump, there is a brief period during which no actions can be controlled. Thus, adding such frames to the replay buffer becomes inefficient for updates without meaningful new information. ω is empirically set to 50, which is approximately under 1 second in a 60 fps game setting. ϵ is decayed from 1 to 0.01 linearly, scaled by episodes interacted during the training process.

4.2.2. Behavior Cloning (BC)

Behavioral cloning (BC) is an offline-only approach that uses supervised learning to map state-action pairs from human demonstration trajectories. We recorded five successful trajectories from amateur players, and each episode i is saved as $\tau^i_{hd}\{(s_t, a_t, r_t, d_t, m_t)\}_{t=0}^{T}$. All (s_t, a_t) pairs are randomly split into `train` and `dev` sets in a 7:3 ratio. With a batch size of 32 and cross-entropy loss function, the policy network $\pi(s, a; \theta)$ is trained for 500 epochs, with early termination after 50 epochs based on `dev` data accuracy.

5. Experiements

In this section, we present experimental results for the aforementioned approaches. All experiments are trained on a single Nvidia GeForce RTX 4070 Ti SUPER 16GB GPU.

5.1. Baselines

DQN policy π_{DQN} and BC policy π_{BC} are learned through the aforementioned setups. It took about 8 hours to train π_{DQN} and 1 hour for π_{BC} on the GPU. After training, we rolled out 50 game plays for each policy to evaluate their performance based on cumulative rewards at terminated states and game completion percentage, defined as the total x-scrolling distance compared to the end of the level.

Since the trained policies are deterministic, we introduce some stochasticity by adding a small $\epsilon = 0.01$ where the policy will act randomly. This evaluates the robustness under perturbations, especially in complex game settings where encountering enemies and obtaining power-ups can significantly change the best actions to take. For all gameplays, the initial action is set to move right regardless of the policies to avoid Mario being stalled.

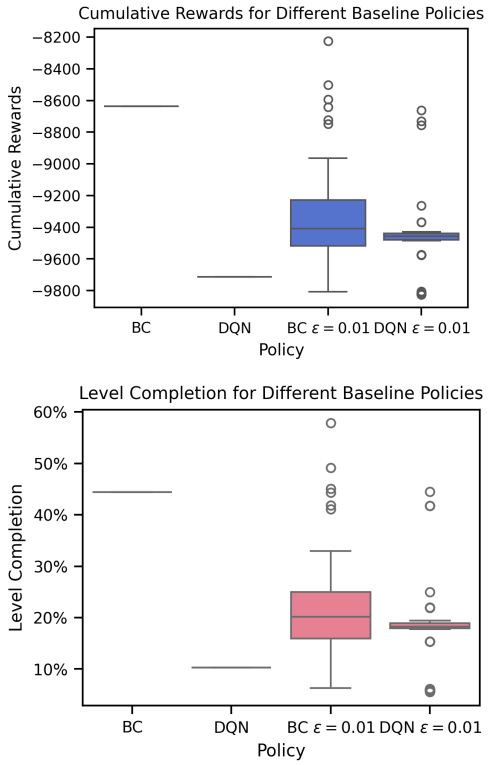

Figure 1. Baseline policies in-game performance statistics for 50 episodes each.

Both policies failed to complete the game level successfully. π_{BC} demonstrates significantly better performance in terms of cumulative rewards and game completions under the deterministic setup, despite taking only 1/8 of the time to train compared to π_{DQN} and using only 5 human-demonstrated trajectories, as shown in Figure 1. However, π_{BC} suffers from greater performance degradation when random perturbations are introduced. It exhibits higher variance in performance in stochastic setups. π_{DQN}, on the other hand, is rather difficult to train. In the complex game environment, though rewards are not sparse, encountering enemies and power-ups makes the Markov assumptions less valid, and the pipes in the game create challenges for the agent to explore states effectively. Thus, the explorations are rarely sufficient to learn good Q value estimations even after 2500 episodes (see Appendix A.2). The trained π_{DQN} does not compare with π_{BC} in deterministic settings but shows superior stability in terms of variance when randomness is introduced, with increased performance when acting stochastically. This illustrates that the current exploration is sub-optimal, and when exploring guided by human demonstrations, it could surpass π_{BC} in performance.

6. Next Steps

Above initial results show the expected performance of baselines. The remaining work of the project involves finishing the implementation of the HRL algorithms and evaluating the performance of each approach.

6.1. Deep Q-Learning from Demonstrations (DQfD)

Following the DQfD (Deep Q-Learning from Demonstrations) framework by (Hester et al., 2017), we incorporate human demonstrations into the replay buffer $\mathcal{D}$ of DQN to control explorations. This builds on the already implemented DQN framework, where the loss functions and replay buffer trajectories are modified to include human demonstrations.

6.2. PPO with Behavior Cloning Warm-start

Inspired by (Coletti et al., 2023), this approach leverages π_{BC} trained model weights as a warm-start, then further leverages PPO for policy fine-tuning. PPO's property of prohibiting significant updates to the policies ensures that explorations will be around human demonstrations, with the possibility to improve and surpass human performance.

6.3. Evaluations

We will evaluate the approaches using both quantitative and qualitative metrics. Quantitatively, performance will be measured via cumulative reward, level completion rate, and distance traversed per episode, plotted as learning curves against training episodes or timesteps. Multiple independent runs will ensure statistical significance. Sample efficiency will be analyzed by measuring interactions required to reach performance thresholds and wall-clock training time. Qualitatively, gameplay visualizations and trajectory overlays will provide insights into behavioral strategies.

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].

PyTorch. Reinforcement learning (ppo) with super mario bros, 2024. URL https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html. Accessed: 2025-03-15.

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 3 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, 1, 0],  # Right
      [0, 0, 0, 0, 0, 0, 0, 1, 1],  # Right + A (Jump)
]
```

A.2. DQN Training

π_{DQN} is rather difficult to train. In the complex Super Mario Bros game environment, though rewards are not sparse, encountering enemies and power-ups makes the Markov assumptions less valid, and the pipes in the game create challenges for the agent to explore states effectively.

As seen in Figure A1, the loss has increased as random explorations reduced. The rewards grow very slowly and drastically drop at the end. We observed the policy was stuck in a reward-hacking phase where Mario moves only to the left until time-out termination. This reduces the risks of getting stuck at further pipes with lower rewards and encountering enemies. Such behaviors might be addressed by tuning reward functions and exploring if HRL can help with this situation.

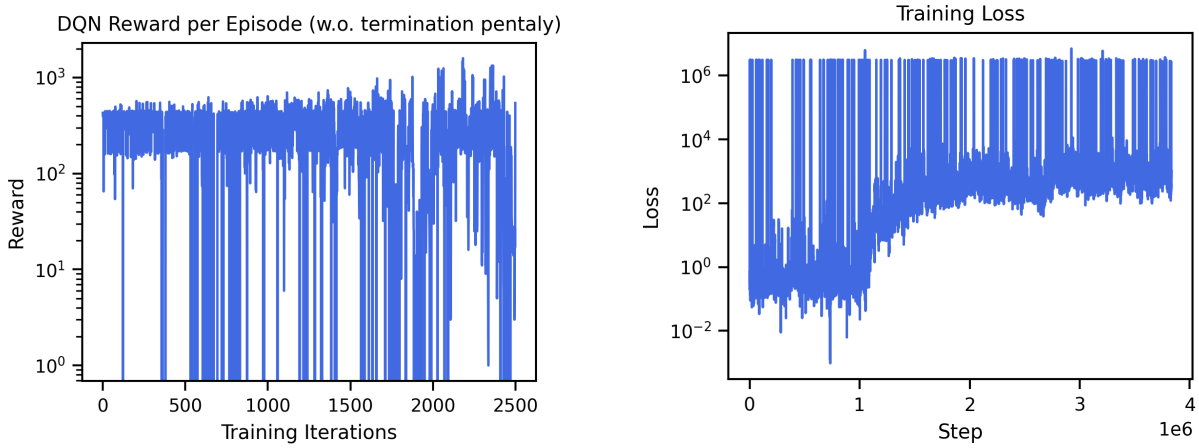

Figure A1. DQN training loss and rewards