CSC_52081_EP, Reinforcement Learning Project

Anonymous Authors

***Abstract*—Reinforcement Learning (RL) trains autonomous agents to interact with complex environments. This study examines the CarRacing-v3 environment from Gymnasium, featuring high-dimensional observations and both discrete and continuous actions. We compare RL algorithms: DQN, SARSA, CEM, SAES, PPO, and SAC. Our experiments evaluate the effects of visual variability, action-space design, and hyperparameter tuning on performance. A baseline is established with a default agent. This work analyzes algorithmic trade-offs, offering insights into RL strategies for continuous control in visually complex settings.**

I. Introduction

Reinforcement Learning (RL) has become a powerful paradigm for developing autonomous agents that learn optimal behaviors through interactions with their environments. In this study, we employ the CarRacing-v3 environment provided by Gymnasium [1], which presents a challenging control task in a racing scenario. The environment is characterized by a high-dimensional observation space and two distinct modes for the action space. Specifically, the observation space consists of a top-down 96×96 RGB image capturing both the car and the race track, thus requiring the use of deep convolutional neural networks (CNNs) for effective feature extraction.

Regarding the action space, CarRacing-v3 supports both continuous and discrete control modalities. In the continuous mode, the agent outputs three real-valued commands: steering, where values range from -1 (full left) to $+1$ (full right); gas; and braking. Conversely, in the discrete mode, the action space is reduced to five actions: do nothing, steer left, steer right, gas, and brake. This duality in action representation allows for a comprehensive evaluation of various RL algorithms under different control settings.

The reward structure of the environment underscores the challenge by combining two components: a penalty of -0.1 per frame and a reward of $+\frac{1000}{N}$ for each new track tile visited, where N represents the total number of track tiles. For example, completing the race after visiting all N tiles in 732 frames, results in a reward of $1000 - 0.1 \times 732 = 926.8$ points, as shown in [1]. This scheme incentivize the agent to balance exploration (visiting tiles) with efficiency (minimizing frame usage), aligning its learning objectives with the task's overarching goal.

The primary objective of this project is to investigate and compare different RL policies across both discrete and continuous action modalities. For discrete action control, we implement methods such as Deep Q-Network (DQN) and SARSA. In contrast, for continuous action control, we explore approaches like the Cross-Entropy Method (CEM) and Self-Adaptive Evolution Strategy (SA-ES), and we also consider incorporating policy gradient techniques (e.g., Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC)). This comparative analysis is driven by our interest in understanding the strengths and limitations of each method in handling complex decision spaces.

The dual nature of the action space in CarRacing-v3 presents a significant challenge. When dealing with high-dimensional visual inputs, the necessity of effective feature extraction becomes paramount. To address this, our approach includes the development of a convolutional neural network architecture tailored to process the 96×96 RGB images, reducing their dimensionality while preserving essential spatial features required for decision making. Additionally, transitioning between discrete and continuous representations of actions requires careful algorithmic design and parameter tuning to ensure stable learning and convergence.

While previous studies have applied various RL techniques in simulated environments, many have tended to focus on either discrete or continuous action spaces separately. In our work, we adopt a comparative approach by evaluating different agents within the same CarRacing-v3 environment. This allows us to assess the performance of each method under similar conditions, examining aspects such as learning stability, computational complexity, and overall policy effectiveness.

At this initial stage, this work primarily outlines the methodology and anticipated challenges, rather than presenting final empirical results. Our approach involves designing the CNN-based feature extractor, implementing the chosen RL algorithms, and setting up a robust framework for comparing their performances. While our preliminary findings are yet to be finalized, we expect that this study will offer valuable insights into the practical implications of employing RL in high-dimensional, real-time control tasks.

Specific limitations of the current work include the preliminary nature of our experiments and the need for further tuning and validation. Future work will focus on extensive empirical evaluations, exploring additional policy gradient methods, and refining the network architecture to better handle the complexities of the CarRacing-v3 environment.

The code for this project is available at GitHub, providing a reproducible framework for future investigations and extensions of this work.

II. Background

A. Discrete Action Space

DEEP Q-Network (DQN) and SARSA are powerful reinforcement learning algorithms to solve discrete action control problems, suitable to our first approach to the Car Racing environment. Both approaches are based on the Q-learning algorithm, which is a model-free reinforcement learning algorithm that aims to learn the optimal action-value function $Q(s, a)$, where s is the state and a is the action. However, DQN uses a deep neural network to approximate the Q-function,

while SARSA uses a table to store the Q-values [4]. To account for the limitations of the SARSA approach in limited high-dimensional state spaces (as the Car Racing environment with an observation space of 96x96x3), we will explore a modern approach called Deep SARSA. [5]

Deep SARSA combines the on-policy nature of traditional SARSA with neural network architectures to handle large state spaces effectively. Unlike DQN's off-policy approach, Deep SARSA maintains SARSA's fundamental characteristics while scaling to complex environments.

On-policy learning methods, such as SARSA, updates the Q-values using actions selected by the current policy. The usual update rule follows the equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s', \pi(s')) - Q(s,a) \right] \quad (1)$$

where α is the learning rate, r is the reward, γ is the discount factor, and $\pi(s')$ is the action selected by the policy at the next state s' (usually the ϵ-greedy policy). The ϵ-greedy policy selects a random action (explore) with probability ϵ and the best action (exploit) with probability 1-ϵ.

On the other hand, off-policy learning methods, such as DQN, updates the Q-values using actions selected by a different policy. The update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right] \quad (2)$$

where $\max_{a'} Q(s', a')$ is the maximum Q-value at the next state s' (greedy selection).

On-policy methods tend to be more stable, but they can be less sample-efficient and must actively explore during training. In contrast, off-policy methods can learn from past experiences, which can lead to better exploration and exploitation of the environment, but may require careful tuning to ensure stability.

B. Continuous Action Space

For the second approach to the Car Racing environment, we will explore algorithms that can handle continuous action spaces. Starting from an evolutionary approach, we will test the performance of the Cross-Entropy Method (CEM) and the (1+1) Successive Adaptation Evolution Strategy (SA-ES) in the Car Racing environment. Both algorithms are based on the idea of optimizing a distribution of policies to maximize the expected return in the environment.

The Cross-Entropy Method is a simple optimization algorithm that iteratively samples policies from a Gaussian distribution and updates the distribution parameters to maximize the expected return. It generates multiple candidate solutions (policies), ranks them based on performance, and updates the policy distribution using the best-performing candidates. It is efficient in high-dimensional spaces and can discover complex polices through population diversity. It follows the steps:

1) Sample N policies from a Gaussian distribution.
2) Evaluate the policies in the environment.
3) Select the top M policies.
4) Update the distribution parameters to fit the selected policies.
5) Repeat until convergence.

The (1+1) Successive Adaptation Evolution Strategy is a variant of the Evolution Strategy algorithm that uses a single parent policy to generate a single child policy at each iteration. It is a simple and efficient optimization algorithm that can handle continuous action spaces effectively. It uses a Gaussian mutation operator to perturb the parent policy and keeps the new version if it performs better in the environment. The amount of change (mutation size) is adjusted dynamically to balance exploration and exploitation, making it suitable for continuous updates, and more sample-efficient than other evolutionary algorithms. It follows the steps:

1) Sample a policy from a Gaussian distribution.
2) Evaluate the policy in the environment.
3) Update the parent policy if the child policy performs better.
4) Adjust the mutation size based on the performance.
5) Repeat until convergence.

Secondly, we will explore and compare two policy-based methods designed for continuous control: Proximal Policy Optimization (PPO) [6] and Soft Actor-Critic (SAC) [7] in the Car Racing environment. PPO is an on-policy gradient method, meaning it directly updates the policy (instead of learning a value function like Q-learning). It uses an actor-critic architecture to learn a parameterized policy that maximizes the expected return. It follows the steps:

1) Collect trajectories using the current policy.
2) Compute the advantage function using the critic network.
3) Update the policy using the advantage function and the policy gradient.
4) Repeat until convergence.

SAC is an off-policy (meaning it reuses past experiences for learning) actor-critic method that uses the maximum entropy framework to encourage exploration and improve sample efficiency. It learns a stochastic policy that maximizes the expected return while maximizing the entropy of the policy (avoiding premature convergence to suboptimal policies). It follows the steps:

1) Collect trajectories using the current policy.
2) Compute the advantage function using the critic network.
3) Update the policy using the advantage function and the policy gradient.
4) Update the critic network using the temporal difference error.
5) Repeat until convergence.

C. Objective

In this study, we will compare the performance of Deep SARSA and DQN in the Car Racing environment to understand the trade-offs between on-policy and off-policy learning methods in the context of discrete action space. Conversely, we also aim to compare the performance of CEM, SA-ES, PPO, and SAC to understand the trade-offs between evolutionary algorithms and policy-based methods in continuous action spaces.

III. Methodology / Approach

A. Environment and Agent Implementation

This study employs the CarRacing-v3 environment from Gymnasium [1], a challenging benchmark characterized by high-dimensional visual observations and multiple action modalities. By leveraging this established environment, we reduce the overhead of environment setup and focus on evaluating different RL methodologies for high-dimensional continuous control tasks.

B. Customization and Evaluation Metrics

To ensure robustness and generalization, our experimental methodology rigorously analyses multiple aspects of the learning process.

1) Visual Variability Effects: We assess the stability of the color scheme by evaluating performance under both static and dynamically varying color representations, which simulate changes in lighting and environmental conditions.

2) Action-Space Design: We compare the efficiency of discrete versus continuous action spaces by analyzing convergence rates, stability, and final policy performance across different action representations.

3) Hyperparameter Sensitivity Analysis: We investigate the stability and convergence dynamics under different step-size configurations by adapting the learning rate.

4) Performance Metrics: Our evaluation framework incorporates several key indicators to comprehensively assess the performance of the implemented methodologies. Firstly, cumulative rewards are used to quantify the overall policy efficiency and long-term reward accumulation. Secondly, convergence speed is measured by the number of training episodes required to reach a stable performance threshold. Thirdly, sample efficiency is evaluated by assessing the learning progress per unit of environmental interaction, which helps determine the algorithmic effectiveness. Lastly, robustness evaluation is conducted through perturbation tests under varying environmental conditions to gauge the model's adaptability and resilience.

C. Visual Analysis and Interpretation

To further refine our understanding of agent behavior and policy effectiveness, we employ several visualization techniques:

1) Learning Curves: We analyze episodic reward trends over the course of training to identify key inflection points and phases in the learning process. This involves plotting the cumulative rewards obtained in each episode, which helps in diagnosing the learning stability, convergence behavior, and potential over-fitting or under-fitting issues. By examining these curves, we can infer the efficiency of the learning algorithm and the impact of different hyperparameter settings.

2) Action Heatmaps: Action heatmaps are utilized to map the distribution of actions selected by the agent across different states or observations. This visualization technique helps uncover spatial biases and decision-making tendencies of the agent. By analyzing these heatmaps, we can gain insights into the policy's behavior, such as preferred actions in specific regions of the state space, and detect any anomalies or suboptimal action patterns that may arise during training.

3) Final Policy Comparisons: We conduct a comparative analysis of the final policies learned under different training configurations. This involves contrasting the strategies and behaviors exhibited by the agent after the training process is complete. By systematically comparing these policies, we can infer the impact of various algorithmic choices, such as different exploration strategies, reward structures, or network architectures, on the overall performance and robustness of the learned policies. This comparative analysis provides a deeper understanding of the strengths and limitations of each approach, guiding future improvements and refinements.

D. Reproducibility

All implementations utilie OpenAI Gymnasium, PyTorch/TensorFlow for training, and Stable-Baselines3 for baseline comparisons. The full codebase is available at GitHub to facilitate reproducibility.

E. Future Work

Additionally, we intend to refine our CNN-based feature extractor by integrating advanced architectures such as attention mechanisms or self-supervised learning, which are expected to improve generalization across unseen track configurations. By structuring this study with a comprehensive methodological approach, we aim to contribute meaningful insights into reinforcement learning strategies for complex continuous control tasks.

IV. Results and Discussion

A. Planned Experiments

To rigorously evaluate reinforcement learning methodologies for high-dimensional continuous control, we design a series of structured experiments that investigate different aspects of agent performance and adaptability. Our experiments encompass the following key components:

1) Baseline Performance Assessment: We begin by assessing the performance of a default agent, utilizing an unmodified algorithm with standard hyperparameter. This establishes a reference point against which subsequent optimizations and modifications will be compared. The baseline agent undergoes training in the CarRacing-v3 environment, where its cumulative reward progression, stability, and convergence rate are monitored.

2) Impact of Visual Perturbations: To evaluate robustness to varying environmental conditions, we systematically introduce color shifts during training and testing. The goal is to measure the degradation in policy effectiveness and adaptation capabilities under perturbed visual inputs.

3) Action Representation and Control Granularity: We experiment with both discrete and continuous action representations to analyze differences in learning efficiency, convergence dynamics, and policy smoothness. Additionally, varying the control resolution (i.e., finer vs. coarser action discretization) helps determine the impact of granularity on training stability and generalization.

4) Hyperparameter Optimization: To improve learning dynamics, we conduct a hyperparameter sensitivity analysis, varying key parameters such as learning rate, discount factor, batch size, and replay buffer configuration. This study aims to optimize agent performance while ensuring stability across multiple training runs.

5) Final Policy Comparison: Upon completing the experimental phase, we conduct a comparative analysis of the trained policies, evaluating their decision-making tendencies, behavioral consistency, and generalization to unseen conditions. This is complemented by qualitative visualizations such as action heatmaps and policy trajectories.

B. Preliminary Experiment with a Default Agent

As an initial baseline, we implement a standard reinforcement learning agent using the Stable-Baselines3 PPO algorithm with default hyperparameter. The agent undergoes training in the CarRacing-v3 environment for a fixed number of episodes, providing insight into:

- Initial convergence behavior and training stability.
- Sample efficiency and learning rate under default conditions.
- Performance plateau and limitations of the unoptimized agent.

To further analyze agent behavior, we generate initial learning curves and action heatmaps, highlighting policy inefficiencies and potential areas for improvement. These insights guide subsequent modifications, including fine-tuned hyperparameter, improved feature extraction architectures, and alternative training strategies.

By structuring our experiments in this manner, we establish a systematic framework for iterative enhancements, ultimately leading to a more robust and efficient reinforcement learning agent for complex continuous control tasks.

V. Conclusions

References

[1] Gymnasium. *Car Racing environment*, https://gymnasium.farama.org/environments/box2d/car_racing/, accessed 04 March 2025.

[2] Sutton and Barto. Reinforcement Learning, *MIT Press*, 2020.

[3] Read. Lecture IV - Reinforcement Learning I. In *INF581 Advanced Machine Learning and Autonomous Agents*, 2024.

[4] AI Mind. Popular Reinforcement Learning algorithms and their implementation, 2023 https://pub.aimind.so/popular-reinforcement-learning-algorithms-and-their-implementation-7adf0e092464

[5] van Seijen et al. Deep SARSA: A Novel Approach to Deep Reinforcement Learning, 2014.

[6] Schulman et al. Proximal Policy Optimization Algorithms, 2017.

[7] Haarnoja et al. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, 2018.

Appendix