Reinforcement Learning Project

Anonymous Authors (recall: the entire document should be anonymous)

***Abstract*—This is a rough guide to producing the project report. The structure outlined here is a suggestion, but you must use this template[1] (of course, replace these hints/instructions/examples with your own text); a limit of 5 pages *not including* references and an optional appendix.**

I. Introduction ($\approx$ 1 page)

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 incentivizes 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 (SAES), 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.

[1]There are also Word templates for this format (https://www.ieee.org/conferences/publishing/templates.html) if you wish; but don't forget you will submit a pdf

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 96×966 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 `[TEM QUE COLOCAR O NOSSO LINK AQUI, NÃO ESQUECER]`, 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-dimentional 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-dimentional spaces and can discover complex plocies 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 (≈ 1-2 pages)

As this report is about reinforcement learning, it should involve discussion of at least one environment, and at least one agent. Since you already introduced the general background and concepts you are using in Section II, here you can get straight into the specific details of *your* approach; basically, providing details on item **??** in Section I. What did you implement, what experiments did you do (and why); including details of your approach to hyper-parameterization.

For example, if you propose a new environment, you would define the state space, action space, reward function, transition function; here.

As elsewhere in the report, don't hesitate to use diagrams, figures, and screenshots wherever they can be useful. And precisely and ambiguously credit any work (code, theory, or otherwise) that you are using, building from, or comparing to; via an appropriate reference.

IV. Results and Discussion (≈ 1-2 pages)

Show the results (tables, plots, ...), and – most importantly – discuss and *interpret* the results. Do not just narrate what you did and report results in the tables, but also *discuss the implications of the results*. Method A beats method B; but: Why? How? In which contexts?

Negative results are also results. Your agent performed poorly or not as expected? If you can explain why then this is indeed an important result.

Always discuss limitations, whether observed in your results or suspected in different scenarios.

Make use of plots, e.g., Fig. **??**, tables, etc.; anything that illustrates the performance of your agent in the environment under different configurations. Make sure to clearly indicate the parametrization behind each result (e.g., γ, or whatever is relevant to your experiments).

V. Conclusions

Summarize the project briefly (one paragraph will do). Main outcome, lessons learned, suggestions of hypothetical future work. Reflect upon, but don't needlessly repeat, material from the conclusion.

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

This is the place to put work that you did but is not essential to understand the main outcome of your work. For example, additional results and tables, lengthy proofs or derivations. Material here does not count towards page limit (but also it will be optional for the reviewer/teacher to work through).