Investigation of Reinforcement Learning Methods for Raceline Optimization

Joseph Hadidjojo, Tung Pham

December 17, 2024

Abstract

1 Introduction

Formula 1 (F1) racing is renowned for the intense precision, skill, and strategy that its drivers bring to the track. In each race, drivers must determine the fastest path around the circuit, or the "racing line," which enables them to maintain optimal speed while negotiating corners and straights. This process is not merely a physical feat; it requires an in-depth understanding of the vehicle's capabilities and a sharp sense of timing. Talented drivers often master the racing line in only a few laps of practice, fine-tuning it to suit their unique driving style and the distinct characteristics of their machinery.

Inspired by this mastery, the objective of this project is to explore the potential of reinforcement learning (RL) to approximate an optimal racing line on a simplified F1 track. The core idea is to use RL, where an agent learns through experience, to find the fastest path around a track by optimizing two key actions: acceleration and rotation. By framing the problem in this way, we aim to develop a model that could eventually serve as a foundational tool for amateur racers with limited experience.

However, this problem presents unique challenges compared to more conventional RL problems:

- **High-dimensional, continuous state and action spaces**: Unlike simpler tasks with discrete or limited action spaces, racing involves continuous control of speed, position, and rotation, all of which must be adjusted dynamically to adapt to the track layout.
- **Balancing competing objectives**: Achieving the optimal racing line requires the agent to balance maximizing speed with avoiding crashes or leaving the track boundaries. High speeds, especially around tight corners, increase the risk of the agent veering off course, demanding precise control over acceleration and rotation to maintain the ideal path within the track's limits.

- **Delayed rewards**: The consequences of an action, like setting up for a corner, may not be immediately evident. The agent must learn to associate actions taken several turns ago with the eventual outcome, making the learning process more complex.
- **Track variability and precision demands**: Unlike simpler environments, the race track's varying curvature and the need for fine-grained control make it harder for the agent to generalize its learned strategy across different segments of the track.

These factors contribute to making the problem of finding an optimal race line particularly challenging for RL, setting it apart from simpler RL applications.

While F1 and professional racing teams likely already utilize advanced tools and systems to estimate the optimal racing line, these applications are typically proprietary and remain restricted to private use. Due to the competitive nature of the sport, insights and strategies derived from such systems are rarely disclosed to the public. Consequently, amateur racers and enthusiasts have limited access to data-driven methods for learning and improving their racing lines. By exploring this problem with reinforcement learning, we aim to create a foundation for a more accessible tool that can help individuals develop their skills and gain insights into racing line optimization without requiring the high costs or exclusivity of professional-grade systems. This project not only contributes to the field of RL in complex, continuous control tasks but also has the potential to democratize access to advanced racing analytics for a broader audience.

2 Background and Related Works

Reinforcement learning (RL) has seen significant advancements in recent years, especially in applications requiring continuous control and dynamic decision-making, such as autonomous driving and racing. In racing, RL agents must optimize not only for speed but also for safety, as they navigate complex tracks with tight turns and variable conditions. This section reviews notable works in RL and autonomous driving that have informed the development of racing agents, highlighting the techniques and challenges relevant to this project.

2.1 Deep Q-Network (DQN) for Autonomous Driving

A foundational work in deep reinforcement learning is the introduction of the Deep Q-Network (DQN) by Mnih et al. (2015) [3], which achieved human-level performance on Atari games. Although the DQN algorithm primarily focuses on discrete action spaces, it inspired adaptations for continuous and complex environments, such as autonomous driving and racing, where agents must navigate through tracks while balancing speed and control. This approach laid the groundwork for subsequent RL algorithms that could handle the continuous control demands of racing tasks.

2.2 Racing Agent with Reinforcement Learning

In racing, agents often benefit from continuous-action reinforcement learning algorithms, which allow finer control over movement variables such as speed and rotation. The Soft Actor-Critic (SAC) algorithm, introduced by Haarnoja et al. (2018) [1], is well-known for its ability to handle continuous action spaces with entropy regularization. SAC's exploration techniques are possibly quite effective for racing environments, where agents must strike a balance between fast exploration of new actions and reliable, safe driving.

2.3 Using RL for Track Navigation in Self-Driving Cars

Another significant contribution to continuous-action reinforcement learning is the Deep Deterministic Policy Gradient (DDPG) algorithm by Lillicrap et al. (2016) [2]. DDPG has shown promise in applications requiring high-precision control. By enabling continuous state and action spaces, DDPG provides a foundation for control in dynamic and high-stakes environments, which we believe makes it well-suited for racing agent applications.

By drawing on these established techniques, this project seeks to develop an RL-based racing agent capable of navigating a simplified F1 track while balancing speed and safety, making it accessible as a foundational tool for aspiring racers.

3 Technical Approach / Methodology / Theoretical Framework

A detailed description of your problem (with math, notation, algorithms, figures, etc.). Use footnotes to cite links to your code or videos

3.1 Problem Formulation

3.1.1 Formulation

The agent operates within a discrete racing track environment represented as a 2D space. At each time step t, the agent observes the current state s_t, which includes:

- Position on the track (x, y),
- Velocity components (v_x, v_y),
- Orientation (θ).
- Angular momentum (ω).
- Absolute distance to Left and Right track limits

The agent selects an action $a_t = [a_{\text{acc}}, a_{\text{rot}}]$, where:

- a_{acc} controls acceleration or deceleration ($a \in \{-1, 0, 1\}$),
- a_{rot} controls the steering angle of the car ($\Delta\theta \in \{-1, 0, 1\}$)

The environment updates the agent's state based on these actions and the track's physics, such as track boundaries. Most real-life environmental factors such as friction, elevation, and grip condition will be ignored. The agent receives a reward r_t at each step, formulated as $-T + \alpha\Delta v_max - \beta O$, where:

1. T: the lap time.
2. Δv_max: The max speed change over a run.
3. O: The number of track excursion.
4. α,β: weight parameters.

The agent learns to balance the objectives of speed and control, gradually improving its policy over training episodes.

3.1.2 Roadblock

Initially, Gymnasium's Car Racing environment was a candidate for our agents testing environment. However, we discovered that there is a bug in the discrete version of the environment [1] which crashed our program whenever the environment is created. By committing to this environment, it would result in extensive effort to debug, fix and adapt to our current implementation and we had underestimate the amount of work required.

As an alternative, we found that Gymnasium's Cart Pole has a similar state-space representation and underlying physics formulation as the CarRacing environment. Therefore, we identified that Cart Pole would be a good replacement for CarRacing in the interim.

3.1.3 CartPole Environment

In this environment, the agent operates within a discrete space where at each time step t, the agent observes the current state s_t, which includes the followings:

- Cart Position, p, on the x-axis ($-4.8 < p < 4.8$).
- Cart Velocity, v.
- Pole Angle, θ.
- Pole angle velocity, ω.

The actions that the agent can took at any given t is to push the cart with a fixed amount of force in the either left or right direction ($a \in \{0, 1\}$) where:

[1] The issue was later resolved by this PR https://github.com/Farama-Foundation/Gymnasium/pull/1253

- 0: push the cart to the left.
- 1: push the cart to the right.

The environment will update the state based on the action taken with any underlying physics calculation needed such as the velocity impact based on the angle of the pole. Most real-life factors such as frictions are ignored and the agent receives 1 unit of reward for every timestep that it keeps the pole upright.

3.2 Methodologies

3.2.1 Deep Q Network

Q-Learning algorithm in Sutton and Barton is a value-based reinforcement learning algorithm that seeks to learn the optimal action-value function $Q^*(s, a)$ using the Bellman equation:

$$Q^*(s, a) = r + \gamma \max_a Q^*(s', a),$$

where s' is the next state, and γ is the discount factor.

The algorithm iteratively updates the Q-values using [4]:

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

where α is the learning rate. This simple, table-based approach works well in discrete environments with small state and action spaces. However, Q-learning has significant limitations when applied to complex environments like racing tracks:

- **High-dimensional state spaces**: Representing continuous states (e.g., positions, velocities, and orientations) as discrete states leads to a combinatorial explosion in the size of the Q-table.
- **Continuous action spaces**: Q-learning requires discrete actions, making it unsuitable for tasks requiring fine-grained control over continuous variables like acceleration and rotation.
- **Inefficient learning**: In large state-action spaces, learning becomes computationally expensive, and generalization across similar states is not possible.

These limitations necessitate scalable approaches like Deep Q-Networks (DQN) for handling high-dimensional state spaces. DQN extends Q-learning by approximating the Q-function $Q(s, a)$ with a deep neural network, enabling it to handle large, high-dimensional state spaces. Instead of maintaining a Q-table, DQN uses a neural network parameterized by θ to predict $Q(s, a)$ for each possible action. This makes DQN effective for high-dimensional problems, though it remains limited to discrete action spaces.

DQN introduces two key techniques to improve stability and efficiency:

- **Experience replay**: The agent stores past transitions (s, a, r, s') in a replay buffer. During training, mini-batches of transitions are sampled randomly, reducing correlations between consecutive updates and improving data efficiency.
- **Target networks**: A separate target network $Q(s', a'; \theta^-)$ is used for calculating target Q-values. The parameters θ^- are updated periodically to reflect the weights of the current Q-network, reducing the risk of divergence during training.

To train the Q-network, DQN minimizes the temporal difference (TD) loss:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right],$$

where:

- θ: Represents the parameters (weights) of the current Q-network used to approximate $Q(s, a)$.
- θ^-: Represents the parameters of the target Q-network, which are periodically updated to match θ. The use of θ^- ensures stability by decoupling the target computation from the current network's rapidly changing parameters.

While DQN is effective for high-dimensional discrete action problems, it still maintains one of the issue of overestimation bias that Q-Learning algorithm encountered. To overcome this issue, Double Deep Q-Network is utilized and will be discussed in further details in the next section.

3.2.2 Double Deep Q Network

4 Experimental Results / Technical Demonstration

A description of how you evaluated or demonstrated your solution.[2]

5 Conclusion and Future Work

A high level summary of what was accomplished, along with a discussion on limitations and avenues for future work (typically 2 to 3 paragraphs).

[2] a video of the robot doing x y z is available at...

References

[1] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[2] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.