Reinforcement Learning Project: Car Racing

Artur César Araújo Alves, Arina Ivanova, Ksenia Khmelnikova, Polina Ovsiannikova, Alexander Tarasov

***Abstract*—In this project we implement and evaluate various reinforcement learning methods, including Proximal Policy Optimization (PPO), Deep Q-learning (DQN), Deep Deterministic Policy Gradient (DDPG) and the Cross Entropy Method (CEM) to train the agent for OpenAI-Car Racing-v2 game environment.**[1]

I. Introduction

Reinforcement Learning (RL) is at the forefront of artificial intelligence, presenting promising avenues for training agents to make decisions in complex environments. One particularly compelling application of RL lies in the domain of car racing, where autonomous vehicles must navigate dynamic tracks, make split-second decisions, and optimize their performance to succeed.

We will explore prominent RL techniques such as Deep Q-Networks (DQN)[1], Proximal Policy Optimization (PPO)[2], Deep Deterministic Policy Gradient (DDPG)[3], and the Cross Entropy Method (CEM)[4] and study their performance and nuances in a specific car racing environment. Each algorithm has its own strengths and weaknesses, providing unique insights into the complexities of autonomous racing.

The environment chosen, although a very simplified version of an autonomous race (which, for example, does not consider other vehicles on the track and potential collisions), still demonstrates certain complexity at the level of modeling in RL. Throughout the project, we encountered certain "limitations" in the used environment that sometimes hindered the learning process or restricted the performance potential of the models used. For some of the models, certain modifications were made to the original environment to try to enhance the performance of each model.

The modifications were mainly in the observation space and the action space in order to make the environment more adapted to each algorithm. This analysis is quite interesting, not only with the aim of trying to enhance the performance of the algorithms but also to explore how they are affected by minor modifications to the environment. The instability character of certain models, the famous "falling off the cliff", can be more easily observed or mitigated according to the implemented changes.

The implementation of all aforementioned algorithms and analysis is accessible **here**.

II. Background

A. DQN

The DQN (Deep Q-Network) algorithm [1], developed by DeepMind in 2015, marked a significant breakthrough in artificial intelligence. At its core, the DQN algorithm builds upon the foundation of Q-Learning, a classic reinforcement learning technique. However, it enhances Q-Learning by leveraging deep neural networks and introducing a new concept known as experience replay. During the training phase of Q-learning, the Q-value of a state-action pair directly is updated this way:

$$Q^*_{k+1}(s,a) \leftarrow Q^*_k(s,a) + \alpha(r + \gamma \max_{a'} Q^*_k(s',a') - Q^*_k(s,a))$$

For most problems, it is impractical to represent the Q-function as a table containing values for each combination of s and a. Instead, we train a function approximator, such as a neural network with parameters θ, to estimate the Q-values, i.e. $Q(s,a;\theta) \approx Q^*(s,a)$ This can done by minimizing the following loss at each step :

$$L_i(\theta_i) = (y_i - Q(s,a;\theta_i))^2$$

where

$$y_i = r + \gamma \max_{a'} Q_k(s',a';\theta_i).$$

Experience replay memory serves as a crucial component in training the DQN model. It functions by storing transitions observed by the agent during gameplay, allowing for the reuse of this valuable data in subsequent training iterations. This approach significantly enhances the stability and effectiveness of the DQN training procedure, enabling more robust and efficient learning.

The Deep Q-Learning training algorithm has two phases:

- Sampling: we perform actions and store the observed experience tuples in a replay memory.
- Training: Select a small batch of tuples randomly and learn from this batch using a gradient descent update step.

B. PPO

Proximal Policy Optimization is a reinforcement learning algorithm developed by OpenAI in 2017 [2] that is part of the policy gradient methods family. PPO follows the same principle as its predecessor, TRPO [5], in striving to maintain a conservative approach to policy updates. The major contribution of this algorithm is evidenced by the use of a clipped surrogate objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right]$$

Where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ represents the ratio of the probability of taking action a_t at time t, given state s, under the current policy π_θ to the probability under the old policy $\pi_{\theta_{old}}$, and $\hat{A}_t = Q(s,a_t) - V(s)$ represents the advantage of choosing action a_t in state s over other actions. The clipping is applied to disregard excessively large updates in the policy,

[1] https://gymnasium.farama.org/environments/box2d/car_racing/

and the *min* function is used to select this clipping in cases where the update is in a non-conservative direction.

Building upon the clipping technique, PPO introduces additional components to refine the training process further. One crucial component is the value function loss, L^{VF}, which aims to reduce the error in value estimation:

$$L^{VF}(\theta) = \left(V_\theta(s_t) - V_t^{target}\right)^2$$

Here, $V_\theta(s_t)$ is the estimated value of state s_t under policy parameters θ, and V_t^{target} is the target value for state s_t. This value function loss component ensures the value function remains closely aligned with the expected returns, improving policy evaluation.

Additionally, PPO penalizes significant deviations from the old policy to maintain a stable learning process. This is achieved through an entropy bonus, $S(\pi_\theta)$, which encourages exploration by rewarding more diverse action selections:

$$S(\pi_\theta) = \hat{\mathbb{E}}_t \left[-\sum \pi_\theta(a_t|s_t) \log \pi_\theta(a_t|s_t) \right]$$

The final objective function of PPO combines these components, aiming to maximize the expected reward while keeping the policy updates conservative and encouraging exploration. The combined objective can be expressed as:

$$L^{PPO}(\theta) = \hat{\mathbb{E}}_t \left[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S(\pi_\theta) \right]$$

where c_1 and c_2 are coefficients balancing the importance of the value function loss and the entropy bonus relative to the clipped surrogate objective.

The PPO algorithm, by integrating these components, facilitates efficient and stable policy learning by balancing between exploitation of the current policy and exploration of new actions, ensuring that the updates are substantial enough to improve performance yet conservative enough to prevent performance collapse.

C. DDPG

Deep Deterministic Policy Gradient is a model-free, off-policy algorithm used in reinforcement learning that combines the concepts of Deep Q-Learning (DQN) and Policy Gradients. Developed by Timothy P. Lillicrap et al. [3], DDPG is particularly effective in continuous action spaces.

DDPG borrows the idea of using a deep neural network to approximate the policy (the actor) and the value function (the critic) from actor-critic methods. It also incorporates techniques from DQN to stabilize the training process, such as experience replay and target networks. In DDPG, the actor network takes the current state as input and outputs a deterministic action, representing the policy. The critic network evaluates the quality of actions chosen by the actor, estimating the state-action value function (Q-value). The actor is trained to maximize the Q-value predicted by the critic, which corresponds to the expected return.

One of the key features of DDPG is its use of off-policy learning, allowing it to learn from experiences collected by the policy while still improving the policy. This is achieved by storing experiences (state-action-reward-next state tuples) in a replay buffer and sampling mini-batches of experiences during training. Experience replay helps break temporal correlations in the data and reduces the variance of updates, leading to more stable training. To further stabilize training, DDPG utilizes target networks for both the actor and the critic. These target networks are copies of the main networks that are updated slowly over time using a soft update mechanism. This helps in reducing the variance in Q-value estimates and makes the training process more stable.

D. CEM

The Cross Entropy Method is an optimization technique that originates from the field of evolutionary algorithms. It was initially proposed by Reuven Rubinstein [4] in the late 1990s as a method for solving difficult combinatorial optimization problems. Since then, it has found applications in various domains, including reinforcement learning. In this context, CEM is applied to solve complex control problems, such as the OpenAI Car Racing environment. Unlike deep reinforcement learning algorithms like DQN or PPO, which rely on gradient-based optimization methods, CEM is a gradient-free optimization technique. This characteristic makes CEM particularly suitable for problems with high-dimensional action spaces or environments that are noisy and stochastic.

The core idea behind CEM is to iteratively refine a probability distribution $P_\theta(*)$ over the parameters of a policy or solution space to find optimal solutions. In each iteration, a set of candidate solutions is **sampled** from the current distribution, and their performance is **evaluated** according to the objective function . Based on the performance of these samples, the distribution $P_\theta(*)$ is **updated** to favor solutions that performed well. This process of sampling, evaluation, and distribution update is repeated iteratively until convergence criteria are met.

A big advantage of CEM is its ability to explore diverse solutions and effectively deal with uncertainty and noise in the environment. By sampling multiple candidate solutions and evaluating their performance over multiple trials, CEM can identify robust policies that perform well on average across different scenarios. Moreover, it is relatively ease to implement unlike some deep reinforcement learning algorithms that require complex neural network architectures and extensive tuning of hyperparameters.

III. Methodology/Approach

A. Original Car Racing's environment

We utilized the second version of the Car Racing environment from Gymnasium [6] as our foundational environment. The state is represented by a 96x96 pixel RGB image. The reward structure is designed as follows: a penalty of -0.1 for each frame and a bonus of +1000/N for each track tile visited, where N denotes the total number of visited tiles on the entire track.

The observation imagery delineates three distinct areas: the track in gray, the off-track grass in green, and the boundary barriers in black, with the car prominently displayed in red.

Additionally, a minor panel at the image's base presents the accumulated reward alongside vehicle control data.

The action space is modeled as a 3D vector, dictating the car's steering, acceleration, and braking. Simplified, the action space encompasses five primary actions: idle, steer left, steer right, accelerate, and brake, catering to intuitive vehicle control:

$$\begin{cases} \mathbf{I} = (0,0,0), & \text{idle} \\ \mathbf{S_l} = (-1,0,0), & \text{steer left} \\ \mathbf{S_r} = (1,0,0), & \text{steer right} \\ \mathbf{G} = (0,0.2,0), & \text{gas} \\ \mathbf{B} = (0,0,0.8), & \text{brake} \end{cases}$$

Episodes within this environment are concluded under two principal conditions: crossing 95% of the track, which is quantified by the proportion of track tiles visited, or upon the vehicle breaching the boundary barriers, leading to an immediate episode termination with a -100 penalty. Furthermore, although not documented, the maximum duration for an episode is capped at 1000 frames.

B. Technical details

In order to better support the algorithms we selected, we made several adjustments to the original environment, varying according to the algorithm used.

1) DQN: As part of the preprocessing routine, we developed a technique to convert the observation images into grayscale and remove the bottom panel. This process yields a simpler, single-channel image with dimensions of 84 by 84 pixels. We undertook these modifications primarily to optimize the system by decreasing the amount of data being processed in general and because we assumed that the bottom panel would act more like noise, given the relatively small size of the informational content.

Another important consideration is that we receive only one current game frame for each step. This observation setting fails to satisfy the Markov property. With just one frame, it's impossible to discern whether the car is moving forward or backward, thus making it challenging to predict the subsequent frame based solely on the current one. To address this limitation, we need to stack the previous, for example $k = 4$ frames, to provide the necessary context for better decision-making.

Fig. 1. Scaled images served as input for the DQN algorithm.

Finally, we also attempted to penalize high speeds to decrease the likelihood of the car straying off the track. To achieve this, we implemented reward clipping at 1.0 for each frame, thereby constraining the benefits of higher velocities.

2) PPO: We followed a similar approach to the previous model for the observation space, formatting and stacking the frames at every step. Similarly, we also examined the effects of reward clipping.

As for the action space, we tried to expand the original one by integrating linear combinations of the original action vectors, introducing actions that blend steering with either brake or gas in varying intensities. Specifically, we added 16 new vectors represented as:

- For gas and steering combinations: $\lambda_1 \mathbf{G} + \lambda_2 \mathbf{S_{l,r}}$
- For brake and steering combinations: $\lambda_1 \mathbf{B} + \lambda_2 \mathbf{S_{l,r}}$

Here, λ_1 and λ_2 are hardness coefficients set to $\{0.5, 1\}$, allowing for soft and hard action intensities.

The idea here was to give the car more flexibility on curves, the natural critical spot of the track, and maybe get to use the corners efficiently in order to preserve momentum.

Lastly, we also bypassed the maximum number of episode steps in some experiments to give the agent more time to complete the lap or even learn to recover after getting off the track.

3) DDPG: The key features of our implementation of DDPG innclude:

Actor and Critic Networks: The actor network is implemented as a feedforward neural network. It takes the state as input and outputs a deterministic action. Our actor network consists of three fully connected layers with ReLU activation functions, followed by a tanh activation function to ensure the output is within the action bounds. The critic network is also a feedforward NN. It takes both the state and action as input and outputs the estimated Q-value. It concatenates the state and action tensors and passes them through three fully connected layers with ReLU activation functions.

The loss function for the critic network is the mean squared error (MSE) between the estimated Q-value and the target Q-value:

$$MSE(Q, target) = \frac{1}{N} \sum_i (Q_i - target_i)^2$$

Experience replay is implemented using a replay buffer that stores experience tuples (state, action, next state, reward, done). During training, mini-batches of experiences are sampled from the replay buffer to update the actor and critic networks.

Target networks are used to stabilize training by slowly updating the target critic and actor networks. To update target networks we use a soft update mechanism, where the parameters of the target networks are updated as a weighted average of the main networks' parameters.

To encourage exploration noise is added to the actions predicted by the actor network. We sample the noise from a noise process, such as the Ornstein-Uhlenbeck process, which provides temporally correlated noise that helps in exploration without excessive randomness.

4) CEM: To implement the Cross Entropy Method, several key components need to be defined: an objective function to minimize, a parameterized family of distributions for sampling, and a policy to guide the agent's actions.

In our approach, the objective function is defined by the reward obtained by the agent as it takes certain actions in the environment. This reward is extracted from the environment after each action is executed.

We chose a multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$ to represent the family of distribution probabilities. In each iteration of the method, both the mean μ and the covariance matrix Σ are updated based on the top policies sampled from the distribution. New values are calculated according to the follwoing formulas:

$$\mu_{new} = \frac{1}{M}\sum_{i=1}^{M}\theta_i$$

$$\Sigma_{new} = \frac{1}{M}\sum_{i=1}^{M}(\theta_i - \mu_{new})(\theta_i - \mu_{new})^T$$

where θ_i are the parameters of the top M policies.

The policy in our implementation is a neural network that maps the current state (in our case, an image of the car on the track) to an action. The policy network consists of three convolutional layers to extract features from the image, followed by two linear layers to produce the final action output. It's important to note that linear regression cannot be used as a policy in this case, as the space is non-linear and requires the use of convolutional layers to extract meaningful features.

IV. Results and Discussion

A. DQN

In our survey, we conducted experiments to assess the impacts of the following elements on DQN: reward clipping and the total number of interactions.

1) Reward clipping: As discussed earlier, we conducted experiments testing two methods: one with reward clipping and the other without reward clipping. The Figure 2 clearly illustrates that training without reward clipping has exhibited greater stability compared to training with reward clipping.

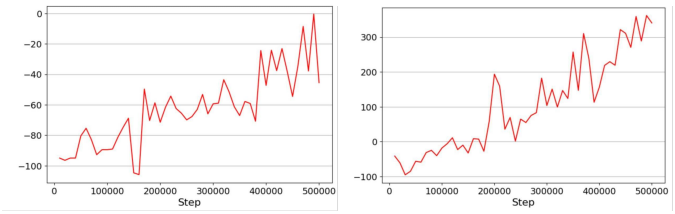

Fig. 2. Average reward during DQN training with reward clipping (left) and without reward clipping (right)

Reward clipping does, indeed, have the potential to restrict the exploration of diverse policies, limiting the model's ability to discover optimal strategies. In the case of DQN, it appears that reward clipping impedes the exploration of the most effective policy, emphasizing the importance of enabling the model to explore a broader range of possibilities during training. For further insights, please refer to the provided link in the beginning, where GIF images of the trained agents are presented.

Our attempts to increase the training steps are depicted in Figure 3. Despite these efforts, the average rewards did not exhibit significant increases, thereby supporting our assertion regarding the impact of reward clipping.

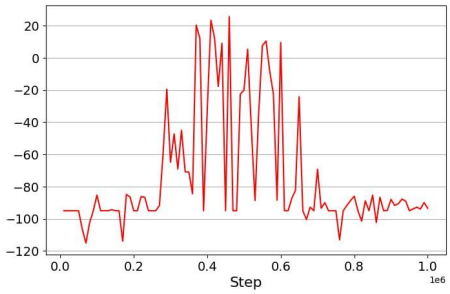

Fig. 3. Average reward during DQN training with reward clipping

2) Total number of interactions: DQN's performance is indeed sensitive to hyperparameter settings, such as learning rate, exploration rate, and network architecture, necessitating meticulous tuning. Our findings indicate that increasing the total number of interactions leads to enhanced stability, confidence, and speed of the agent. This suggests that allocating more training steps allows the agent to better learn and adapt to the environment, resulting in improved performance. Additionally, the GIF results of the training with 500,000 training steps and 1 million steps provide visual insights into the training process, further supporting our conclusions.

B. PPO

For PPO we separate the results in 3 parts, each evaluating the impact of the following elements as discussed: clip reward, max episode length and the discretization of the action space.

The main observation reveals that the most sensitive element in every training test is the value function estimation. Small errors such as going off the track and missing tiles have a significant impact on reward estimation.

1) Episode length: After a certain number of training steps, we observed a significant decline in the agent's performance, accompanied by a notable decrease in the final mean reward. Empirical investigation revealed a negative behavior: the agents tended to favor very slow movement or complete cessation until the episode's conclusion. This behavior became more pronounced as we increased the maximum length of the episodes.

We hypothesized that this behavior allowed the agents to better estimate the value function under such a policy, as the final reward became easier to predict. The L^{VF} plot gives strong support to this hypothesis.

2) Reward clipping: Clipping the reward generally had a negative impact on the best model obtained. However, we observed two positive effects: (1) it significantly improved the stability of the learning process up to a certain point, and (2) it resulted in slower agents but less prone to error.

The first observation is supported by examining the clip fraction and approximate KL divergence plots. These measures directly relate to how much the policies are updated during training.

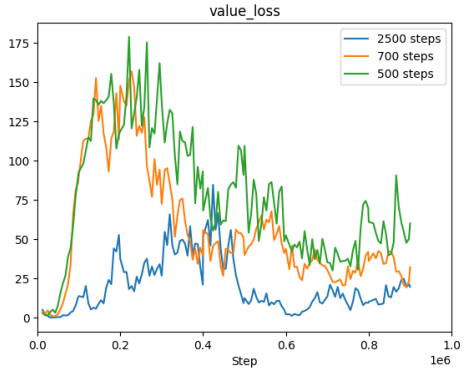

Fig. 4. Value loss function decreases as we increase the maximum episode length

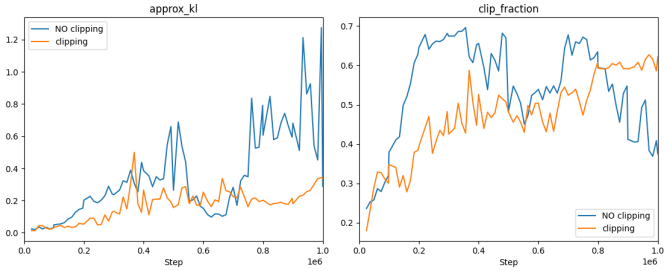

Fig. 5. Clipping x No clipping comparison using: approximate KL (left) and clip fraction (right)

Regarding the second observation, we can analyze the mean episode length: without clipping, the agents more frequently complete laps, i.e., they are able to visit 95% of the tiles. Conversely, with clipping, they tend to deviate off-track in certain areas.

3) Action space discretization: We found that expanding the action space had a positive effect on the car's turning behavior as anticipated. The car began to prepare better for curves by adjusting speed or positioning before entering them. A positive impact on recovering was also observed, with a larger bag of tools the agent could now faster get on track.

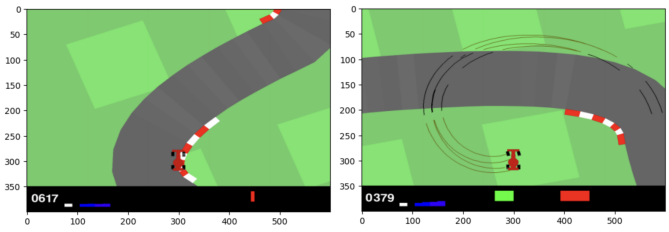

Fig. 6. Agent better uses corner (left) and quickly recovers after getting off track in high speed (right), thanks to training with an expanded action space

However, we noticed a quicker onset of the same slow-down behavior observed before and here again attributed to the value loss. As a result, the agents seldom completed laps and eventually hindered the learning process.

C. DDPG

So far, the algorithm has not produced satisfactory results due to the limited training duration. With only 100 episodes, each comprising 50 steps, the models have not had sufficient exposure to the environment to learn effective policies. As a consequence, the car remains stationary, occasionally attempting to turn left, but without significant progress.

Moving forward, our plan is to delve deeper into understanding the underlying reasons behind this behavior. One hypothesis is that the training duration is insufficient for the models to grasp the complexities of the environment and learn meaningful driving strategies. In addition, there may be problems with the model architecture, hyperparameters, or the training process that need to be addressed.

To improve results, we intend to extend the training duration significantly, allowing the models to accumulate more experience and refine their policies over time. Furthermore, we will explore various techniques to enhance training efficiency, such as adjusting the model architecture, fine-tuning hyperparameters.

In summary, while the current results are suboptimal, we are committed to addressing the challenges and optimizing the algorithm to achieve better performance in future iterations.

D. CEM

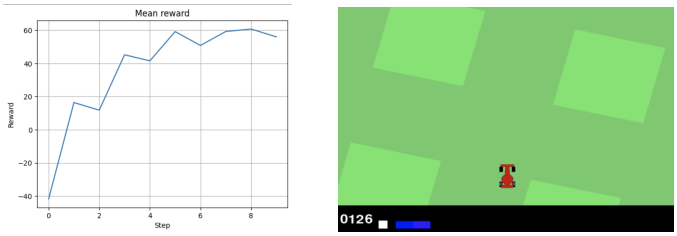

Fig. 7. CEM the first version

The observations made from the graphs indicate a discrepancy with actual performance, emphasizing the need for more nuanced evaluation methods, such as rendering the car's movement in gifs. This would vividly illustrate behaviors like the car gaining excessive speed and veering off track, often triggered by straight road sections at the track's start.

Moreover, the significant computational complexity of the algorithm, stemming from the sequential reward computation for each sample through a CNN comprising three convolutional layers and two linear layers, is highlighted. Given that each state is represented by a 96x96 RGB image, this architecture is logically sound for transforming the visual state into a vector representation, which is then translated into an action.

However, the primary concern is identified as the CNNPolicy architecture's simplicity, where the convolutional layers' weights are iteratively updated using the same Cross-Entropy Method (CEM) which is not sufficient. To address this, several hypotheses are proposed for testing:

1. **Utilizing a Pre-trained Model:** By adopting a pre-trained model with convolutional layers, freezing these layers to prevent changes during training, it's anticipated that more informative embeddings could be obtained, enhancing the model's ability to discern effective actions.

2. **Exploring Different Distributions:** A different distribution, such as a beta distribution, might better suit the discrete action space of this task. Yet, this approach requires thorough validation and hypothesis testing to confirm its efficacy.
3. **Covariance Instead of Variance:** Transitioning from variance to covariance computation in the algorithm could potentially offer a more refined solution to handling continuous space. This adjustment has shown promise during our practical sessions.
4. **Momentum Mechanism for Negative Rewards:** To complement the existing *reward_threshold* that curtails the evaluation of environments underperforming against a set minimum score, we suggest a dynamic approach to managing consecutive negative rewards. Specifically, if the accumulated reward remains negative over a predetermined number of iterations, it would be adjusted by a scaling factor that increases in magnitude the longer the policy continues to underperform. This momentum mechanism aims to penalize prolonged poor performance more severely, encouraging the CEM to explore alternative strategies that might yield positive outcomes more quickly.

V. Conclusions

In conclusion, our study has delved into prominent reinforcement learning techniques, including DQN, PPO, DDPG and CEM. DQN demonstrated improved stability as interactions increased, whereas PPO's performance was affected by episode length and reward clipping. Clipping the reward for PPO improved stability but resulted in slower agents. However, it negatively impacted the best model obtained. In case of DQN, reward clipping restricts policy exploration, hindering the discovery of optimal strategies. Meanwhile, DDPG encountered challenges due to its limited training duration.

These findings underscore the necessity of thoughtful consideration and optimization tailored to the unique requirements of specific tasks and environments across reinforcement learning methods. By comprehensively evaluating these algorithms, we have laid the groundwork for future research endeavors aimed at advancing the capabilities of autonomous racing systems.

References

[1] Read. Lecture VI - Reinforcement Learning III. In *INF581 Advanced Machine Learning and Autonomous Agents*, 2024.

[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv:1707.06347.

[3] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv:1509.02971.

[4] Rubinstein, R. (1999). The cross-entropy method for combinatorial and continuous optimization. Methodology and computing in applied probability, 1, 127-190.

[5] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust Region Policy Optimization. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 1889-1897).

[6] *Solving Car Racing with Proximal Policy Optimisation*, https://notanymike.github.io/Solving-CarRacing/, accessed 10 March 2024.

[7] *Car Racing - Gymnasium Documentation*, https://gymnasium.farama.org/environments/box2d/car_racing/, accessed 10 March 2024.