CSE 151/251B Final Project: Car Racing Reinforcement Learning

Dmitri Demler
ddemler@ucsd.edu

Jason Weitz
jdweitz@ucsd.edu

Alec Digirolamo
adigirolamo@ucsd.edu

Yanbo Yu
yay023@ucsd.edu

Andrew Du
adu@ucsd.edu

Hariz Megat Zariman
mqmegatz@ucsd.edu

1 Abstract

This project explores the application of Reinforcement Learning (RL) techniques to optimize performance in a racing game environment provided by Gymnasium, focusing specifically on the CarRacing game. We employ Deep-Q Networks (DQN) and Proximal Policy Optimization (PPO) models, leveraging the Stable Baselines3 library Raffin et al. [2021] for implementation. Our experiments reveal that a Convolutional Neural Network (CNN) architecture paired with PPO exhibits promising results, achieving significant progress in race completion with high speed within a few hours of training. To enhance training efficiency and model performance, we explore various augmentation strategies, including reward structure modifications and model architectural adjustments. Our work includes the implementation of reward augmentations such as grass detection, speed, and acceleration. This project demonstrates the intricate balance between model architecture, training methodologies, and augmentation strategies in achieving optimal RL performance in complex environments. Using the PPO approach alongside grass detection and acceleration augmentation, we yielded our best model achieving a mean reward of 916.80273.

2 Introduction

We apply Reinforcement Learning techniques to the CarRacing environment from Gymnasium. For our benchmarks we focus on using Deep-Q Networks (DQN) and Proximal Policy Optimization (PPO) to train a virtual car to race effectively.

Reinforcement Learning is a computational approach where an agent learns to make decisions by interacting with an environment. It aims to maximize a cumulative reward through trial and error, adapting its strategy based on feedback from its actions. RL is highly effective in teaching machines how to perform complex tasks without explicit programming for each step.

Deep Q Learning (DQN) is an advanced RL technique that integrates deep neural networks to approximate the optimal action-value function, known as Q. This method allows the agent to evaluate the potential future rewards of its actions, enabling more informed decision-making. DQN has proven effective in environments where the state and action spaces are large, making it a robust choice for complex tasks like driving simulations.

Proximal Policy Optimization (PPO) is another RL strategy, known by its approach to updating policies. It optimizes the policy directly, aiming to improve performance while ensuring that the new policy does not deviate too drastically from the old one. This balance helps in maintaining stable and consistent learning progress, particularly beneficial in environments where action consequences are significant and varied.

3 Related Work

Deep Q-Learning (DQN) stands as a cornerstone in reinforcement learning, integrating deep neural networks to approximate the optimal action-value function. This integration enables agents to evaluate the potential future rewards of their actions, thereby facilitating informed decision-making in complex environments. DQN was first pioneered by Mnih et al. [2013], used for multiple Atari games.

Proximal Policy Optimization (PPO) [Schulman et al., 2017] is a newer type of policy optimization methods that have gained prominence for their effective balance between performance enhancement and policy stability. PPO's methodology, which directly optimizes the policy while ensuring minimal deviation from previous policies, has proven beneficial in various challenging RL scenarios.

One augmentation we considered, but ultimately were not able to implement, was a constrained RL algorithm aiming to minimize a cost function associated with driving off the track. The algorithm that we decided on was PPO-Lagrangian [Achiam and Amodei, 2019], a mode of constrained learning that adds a cost term multiplied by a Lagrangian multiplier to the advantage during rollouts. We also considered POAR (Policy Optimization via Online Abstract Representation Learning). POAR [Chen et al., 2021] for potential improvements in ease of hyperparamater tuning, but ultimately found it unsuitable due to the lack of a continuous cost variable.

4 Methods

Our GitHub repository can be seen and ran here Learning [2024].

4.1 Training

For each policy, we trained the models until they seemed to converge. Because the number of epochs required to converge are not comparable between models, we subjectively determined whether or not the models converged based on the current time spent training and seeing if the model has not improved significantly over a portion of the training time.

The baseline reward follows the structure such that there is +1000/N for every new track tile that is visited with N being the amount of tiles explored, and -0.1 every frame. Additionally, an episode ends when the track is completed or if it exits the environment, getting -100 and the game ends.

4.2 Deep Q-Learning

We used the Stable-Baselines3 implementation of Deep-Q learning with a discrete action space. Stable-Baselines3 takes images as inputs from the gym environment to determine state, therefore, it is necessary to have both a feature extraction network component and a reinforcement learning network component.

The feature extraction component, shared across all Stable-Baselines3 RL policies, is the convolutional network present in Mnih et al. [2013], consisting of two convolutional layers and two full-connected layers. The Deep-Q policy also implements a replay buffer and target network for state value estimation.

The training parameters we used for DQN are:

1. Learning rate: 0.0001
2. Batch size: 32
3. Optimizer: Adam

4.3 Proximal Policy Optimization

We use the Stable-Baselines3 implementation of the PPO function, using the default Stable-Baselines3CNN outlined prior for feature extraction. PPO iteratively improves the policy by using multiple epochs of stochastic gradient ascent to optimize a surrogate objective.

The training parameters we used for PPO are:

1. Learning rate: 0.003
2. Batch size: 64
3. Optimizer: Adam

Raffin et al. [2021].

In fifty epochs of five thousand training steps each, the model becomes capable of completing tracks, but does so slowly, and experiences difficulty with recovery if it goes off track and loses orientation.

4.3.1 Reward Augmentation

The current reward function adds a reward for each road tile the car moves over and punishes a small amount for every time step. To change the reward function, we implemented two augmentations.

4.3.2 Grass Punishment

The first is to heavily punish the car for going into the grass instead of just rewarding getting new road tiles. We implemented this because in most types of racing, a driver is punished if they go off the track. For this, we subtract from the reward a small value times the number of tires in the grass. We detect if the tire is in the grass by looking if the pixels around the tires are green as shown in Figure 1

4.3.3 Speed Reward

While the grass reward is effective at incentivizing the agent to stay on the track, it does little emphasize the importance for the car to maximize its speed. To address this limitation, we implemented two additional reward augmentations focused on encouraging higher velocities: a speed reward and an acceleration reward. The acceleration reward grants extra positive rewards when the agent applies gas, directly promoting acceleration. Complementing this, the speed reward provides a reward proportional to the car's current speed, pushing the agent to attain and maintain higher speeds. One of the key challenges we faced was tuning the constant scaling factors for these two rewards. Improper scaling could lead to the agent prioritizing reckless acceleration over safe racing lines, diminishing overall performance. We experimented with various constants before converging on values that struck an effective balance, augmenting speed and acceleration without destabilizing the baseline rewards responsible for keeping the car on the track. With these reward augmentations, the agent can learn to navigate the course with heightened speeds while respecting the track boundaries.

4.3.4 Two-Stage Training Approach for Speed Optimization

For the acceleration reward augmentation, we trained it in conjunction with the grass edge detection reward component. This allowed the agent to simultaneously learn to accelerate while still prioritizing staying on the track and avoiding collisions with the grass edges.

However, for the speed reward augmentation, we adopted a different approach by splitting the learning phase into two distinct stages. In the first stage, the agent focused solely on learning to stay on the road and complete the track, with the only reward component being the grass edge detection. Once the agent achieved a satisfactory reward level over a certain number of epochs, indicating its proficiency in navigating the track, we transitioned to the second phase.

During this second phase, the emphasis on strictly adhering to the track boundaries was reduced, and the speed reward augmentation was introduced. By decreasing the weight of the grass edge detection reward and incorporating the speed reward, the agent was encouraged to prioritize achieving higher velocities while still maintaining a reasonable level of track adherence.

5 Results

The way in which we quantify our models is the mean reward and time to complete a given seeded track, that it has not been trained on. The mean reward is a viable metric, as it has a maximum value of 1000 across all implementations. It is also a clear measurement of the success of the car given the reward structure. The time taken to complete a track is a more realistic metric, as it quantifies the true purpose of the game: to race with the goal of completing quickly. See Table 2 for reward comparisons and Table 1 for time comparisons.

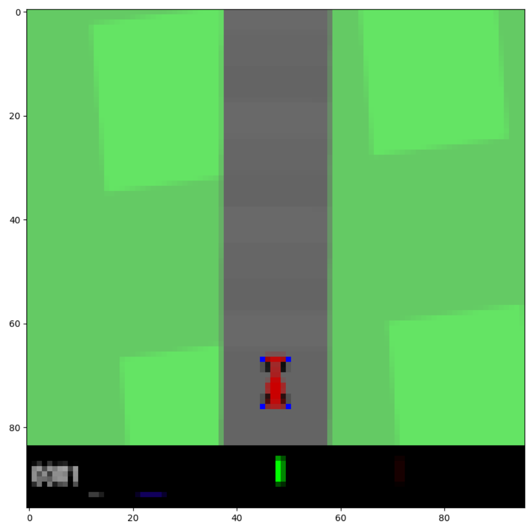

Figure 1: The grass detection pixels highlighted in blue.

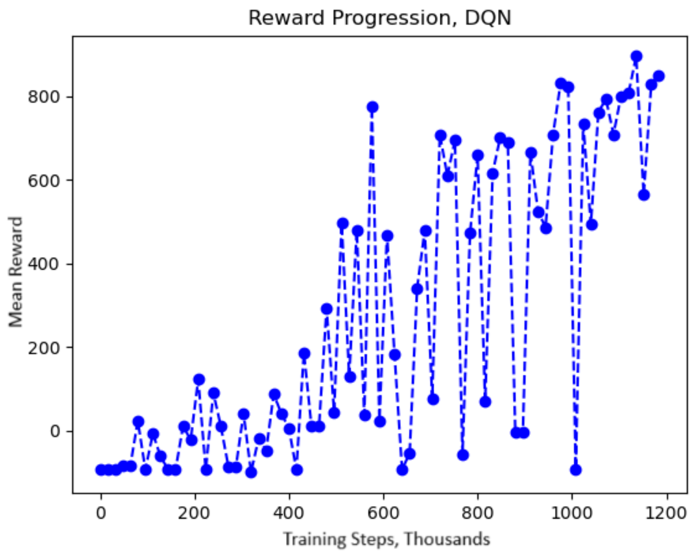

Figure 2: DQN baseline reward performance

5.1 Deep Q-Learning Baseline

See Figure 2, Table 2, and Table 1 for results. This method took too long to converge and also has sub-optimal performance.

5.2 Proximal Policy Optimization Baseline

See Figure 3, Table 2, and Table 1 for results. This method converges relatively quickly, but does not outperform its augmentations.

5.2.1 Reward Augmentation: Grass Punishment

See Figure 4, Table 2, and Table 1 for results. This augmented method performs well, but can be further improved.

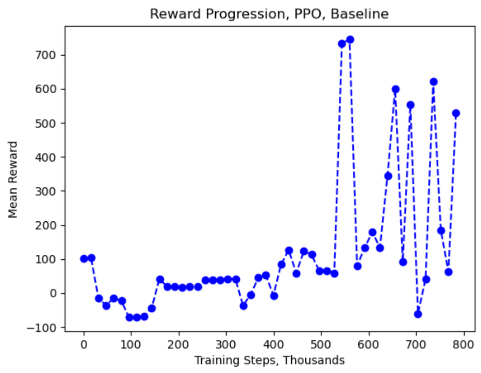

Figure 3: CNN PPO baseline reward performance

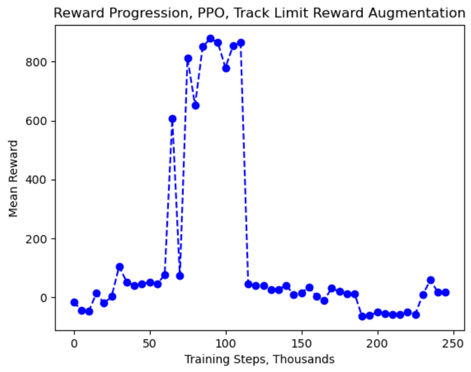

Figure 4: Grass Detection reward performance

5.2.2 Reward Augmentation: Grass Detection + Speed Reward

See Table 2, and Table 1 for results. This augmented method performs very well, as it leads multiple trials for time, but the mean reward is not optimal.

5.2.3 Reward Augmentation: Grass Detection + Acceleration Reward

See Table 2, and Table 1 for results. This augmented method performs very well, the main leader in time trials and leader in mean reward.

Model	Seed 41	Seed 50	Seed 51	Seed 52	Seed 53	Seed 54	Seed 55	Mean Time
PPO Baseline	26.01	22.93	22.02	23.27	19.29	19.80	18.99	21.76
PPO + Grass Detection	19.23	19.04	20.48	20.60	19.04	17.55	19.06	19.29
PPO + Grass Detection + Acceleration	17.31	15.45	**16.13**	**19.55**	19.52	**15.29**	**16.27**	**17.07**
PPO + Grass Detection + Speed	**15.55**	**14.08**	20.49	19.92	**18.05**	19.89	20.61	18.37

Table 1: Comparison of time taken by different RL models for various seeds. The best times for each seed are highlighted in bold.

6 Discussion

6.1 DQN

The Deep Q network was eventually able to complete the track, however, its training time and memory usage greatly exceeded that of PPO's: whereas for PPO and all augmented PPO policies we trained for 50 epochs of 5000 steps and found such a regimen sufficient for producing models that could successfully complete the circuit, for DQN, we needed to train for 75 epochs of 16000 steps - a

Method	Mean Reward
DQN	896.46
PPO	744.52
PPO + Grass Detection	906.10
PPO + Grass Detection + Acceleration	**916.80**
PPO + Grass Detection + Speed	801.97

Table 2: Comparison of Methods and Rewards

fivefold increase in training time. At the point at which PPO model could complete the track, the DQN model could not complete more than two turns. For these reasons, we did not proceed with DQN augmentations.

6.2 PPO

6.2.1 Baseline

Compared to DQN, PPO's training process is more stable and faster. The result of the baseline doesn't seem to be significantly different than DQN baseline. It faces similar issue with DQN baseline that the car can still find it hard to handle sharp turn and the speed is relatively slow.

6.2.2 Grass Augmentation

We implement grass detection by checking if the corners of the car are on the grass, the more corners on the grass, the harder the punishment. By implementing this feature, the car tries to stay in the middle of the road and it reaches a higher rewards during training. However, the method seems to go slowly to ensure this is the case. This persuaded us to implement additional reward augmentation functions that ensure that the speed is also prioritized by the model during training.

6.2.3 Grass and Speed Augmentations

The performance of the different reward augmentation strategies was evaluated across multiple random seeds, as shown in Table 1. The results indicate that incorporating reward augmentations significantly improves the agent's performance compared to the baseline PPO model without any augmentations.

Among the augmented models, the combination of grass detection and speed reward (PPO + Grass Detection + Speed) demonstrated the best overall performance, achieving the fastest mean time of 18.37 seconds across all seeds. However, the grass detection and acceleration reward (PPO + Grass Detection + Acceleration) model closely followed with a mean time of 17.07 seconds, outperforming the speed reward model on several individual seeds.

Interestingly, upon closer inspection of the simulations, we noticed that the grass detection and speed reward model often reached high velocities, leading to occasional oversteering and drifting (donut) behavior, particularly during sharp turns. This characteristic made the model excel on simpler tracks but struggle on more challenging configurations with tighter turns.

In contrast, the grass detection and acceleration reward model displayed a more controlled and consistent driving behavior, making it better suited for handling complex track layouts. While it may not have achieved the highest top speeds, its ability to navigate intricate sections more effectively resulted in superior performance on the more demanding tracks.

Even the grass detection reward augmentation alone (PPO + Grass Detection) showed a significant improvement over the baseline model, highlighting the importance of incentivizing the agent to stay within the track boundaries.

It is extremely difficult to find optimal reward augmentations for a model. While the grass detection and speed reward model excelled in terms of overall time, the grass detection and acceleration reward model demonstrated greater versatility and robustness across a wider range of track configurations.

6.3 Constrained RL

One characteristic that we identified for all models was difficulty in recovery once control and orientation are lost; this was observed even when negative rewards was assigned to any instance of off-track driving. We therefore wanted to explore constrained RL algorithms, which would be able to strictly enforce track limits without incurring possible reward-hacking behaviors, such as the car not moving at all with too large a negative reward assigned to going off track.

We ultimately decided on PPO-Lagrangian for ease of implementation with a discrete cost function that would penalize off-track driving at a constant rate per frame, however, the task of adding the additional dimension of cost to the Stable-Baselines3 environment proved too challenging to complete in the project timeframe. Any future work on constrained RL would likely first involve confirming the hypothesized reward-hacking behavior.

7 Team Member Acknowledgement

Dmitri Demler: Co-wrote the Stable-Baselines3 baseline implementations and worked on reward augmentation (specifically speed).

Andrew Du: Wrote the Stable-Baselines3 implementation of DQN and experimented with implementation of PPO-Lag.

Yanbo Yu: Helped setup the environment, developed baseline DQN model, and experimented with constraint optimization for the PPO model.

Alec DiGirolamo: Helped develop baseline DQN model and created reward augmentations for the PPO model.

Jason Weitz: Helped develop and code the PPO baseline. Contributed to the development, coding, and logic of the track detection reward augmentation.

Hariz Megat Zariman: Helped setting up the gymnasium environment and worked on the baseline PPO model.

References

Joshua Achiam and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning, 2019. URL `https://api.semanticscholar.org/CorpusID:208283920`.

Zhaorun Chen, Siqi Fan, Yuan Tan, Liang Gong, Binhao Chen, Te Sun, David Filliat, Natalia Díaz-Rodríguez, and Chengliang Liu. Poar: Efficient policy optimization via online abstract state representation learning. *arXiv preprint arXiv:2109.08642*, 2021.

Shallow Learning. Rl-car-racing. `https://github.com/jdweitz/RL-Car-Racing`, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.