Reinforcement Learning for Rocket League (RL^2)

Anshuman Dash, Phillip Peng, Matthew Shen
Department of Computer Science
University of California, Santa Barbara
Email: {anshumandash, plp, matthewshen}@ucsb.edu

***Abstract*—We aim to investigate the use of Reinforcement Learning in the context of the game of Rocket League. Through the uses of PPO and Curriculum Learning through self-play, we train bots on a variety of reward structures and hyperparameters. To do an empirical comparison of the different models, we evaluate the bots against each other in a tournament setting of 1v1 matches to determine model ELO. We analyze the results of this tournament as well as individual models and discuss the implications of our findings.**

I. Introduction

Reinforcement Learning (RL) is a powerful tool for training agents to perform tasks in a variety of environments. We aim to investigate the use of RL in the context of the game of Rocket League. Rocket League is a popular video game that combines soccer with rocket-powered cars. The game is played in a 3D environment, and players control their cars to hit a ball into the opposing team's goal. The game itself holds a large number of states making it a challenging environment for RL agents to learn in.

II. Related Work

The use of RL to play games has been a popular research topic in recent years, most famously with AlphaGo. The application of RL in other games such as Dota 2 and Starcraft II has also been explored.

A. Proximal Policy Optimization (PPO)

We use

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

III. Implenetation

We implement and train the agents through a package called RLgym which allows the training of Rocket League agents. It does so in a

IV. Model and Methods

Through this work we largely explore how a change in reward structure as well as model hyperparameters affect the performance of trained agents. All models are trained using the Proximal Policy Optimization (PPO) algorithm.

Algorithm 1 Pseudocode for the Proximal Policy Optimization (PPO) algorithm. The algorithm iteratively updates the policy and value function parameters to maximize the expected reward while ensuring the updates do not deviate too much from the previous policy in the positive direction.

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for k = 0,1,2, ... **do**
  Collect set of trajectories $\mathcal{D}_k$ by running policy $\pi_k = \pi(\theta_k)$ in the environment
  Compute rewards-to-go $\hat{R}_t$
  Compute advantage estimates $\hat{A}_t$ for current value function V_{ϕ_k}
  Update policy by maximizing PPO-Clip objective:
    $\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{\text{CLIP}}$
  Fit value function by regression on mean-squared error:
    $\phi_{k+1} = \arg\min_{\phi} \phi \hat{\mathbb{E}}_t \left\{ (V_\phi(s_t) - \hat{R}_t)^2 \right\}$
end for

A. Reward Scaling

B. Reward Structures

C. Curriculum Learning

Curriculum learning is a training strategy where the learning process is organized in a way that gradually increases in complexity. We first attempted to implement this in the context of Rocket League by starting with a simple reward structure and gradually increasing the complexity of the reward structure as the agents learn.

We find that this doesn't perform well for a variety of reasons. The increasing sparsification of rewards makes it difficult for the agents to learn effectively. In order to address this, we propose a modified approach that balances exploration and exploitation.

We then attempted to implement an automatically updating curriculum learning strategy, which would automatically scale rewards based on the number of training steps that have occurred.

$$ifr_t = \frac{1}{n}$$

D. Common Pitfalls

In the beginning of the project, we encountered a number of pitfalls that made training difficult. One such pitfall was

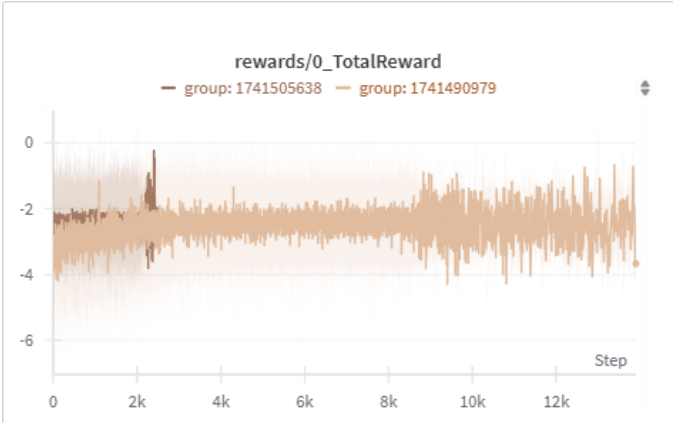

Fig. 1. **Total reward over time for a model trained with a constant negative reward:** Constant Negative Reward at every timestep resulted in an overall negative reward. This resulted in a form of reward hacking where the agent would allow the opponent to score to stop incurring negative reward.

the use of a constant negative reward for the agent. This is intutive as it would encourage the agent to score goals sooner to prevent the occurance of the negative reward. However, in practice, we found that this more often than not led the agent to simply allow the opponent to score a goal to stop incurring the negative reward. This can be seen as a form of reward hacking, where the agent finds a way to exploit the reward structure to maximize its reward. This could theoretically be fixed by scaling this constant reward down, however we decided to remove it entirely from our reward structure.

The thought of using a pure goal oriented reward structure may also seem intutive. This can be especially seen in the reward structure of AlphaGo for . However, in the context of Rocket League, this is not effective as the state-action space is humongous. Additionally without a method such as Monte Carlo Tree Search (MCTS) as in AlphaGo, it makes each state exceptionally sparse in terms of reward, making it difficult for the agent to learn effectively. We train a model with a pure goal oriented reward structure for 700 million timesteps to show the ineffectiveness of this reward structure.

V. Results

We evaluate the performance of the trained agents by having them play against each other in a Placket Luce tournament

VI. Future Work

Due to time constraints of the project most of the models were trained in the order of tens to hundreds of million timesteps, while most high level play bots, are trained in the order of billions of timesteps. Additionally, the majority of bots are not incentivised to use some inherent features of the space, such as boost and the space in the air. Exploring whether these features still apply to models trained for longer periods of time uses these features would be an interesting analysis.

Additionally we initially aimed to explore the use of methods such as Hierarchical Reinforcement Learning, Model distillation, and Network Distillation and their effect on model performance. Hierarchical Learning requires a good low-level policy to be able to be effective before training higher level policies which would require having a good low-level policy early on. Model distillation suffers from the same drawback of needing some initial good policy to distill from. On the case of Network Distillation, we just ran out of time lmao.

Lastly implementing a more accurate form of self-play woudl be beneficial. Currently models are trained only against its current self, which can lead to a lack of diversity in training data. Implementing self-play where the model plays against a variety of past versions of itself would improve this significantly.

VII. Conclusion

[1]
dfadsf [2]
adsfasd [3]
asdf [4]
[5]
[6]
[7]

References

[1] A. Sigal, H.-C. Lin, and A. Moon, "Improving generalization in reinforcement learning training regimes for social robot navigation," *arXiv preprint arXiv:2308.14947*, 2023.

[2] Z. Zhang, H. Li, L. Zhang, T. Zheng, T. Zhang, X. Hao, X. Chen, M. Chen, F. Xiao, and W. Zhou, "Hierarchical reinforcement learning for multi-agent moba game," *arXiv preprint arXiv:1901.08004*, 2019.

[3] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," *arXiv preprint arXiv:1912.10944*, 2019.

[4] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[5] M. Pleines, K. Ramthun, Y. Wegener, H. Meyer, M. Pallasch, S. Prior, J. Drögemüller, L. Büttinghaus, T. Röthemeyer, A. Kaschwig, O. Chmurzynski, F. Rohkrähmer, R. Kalkreuth, F. Zimmer, and M. Preuss, "On the verge of solving rocket league using deep reinforcement learning and sim-to-sim transfer," pp. 253–260, 2022.

[6] M. Misaros, D. l. Gota, D. F. Niste, A. Stan, A. Ciobotaru, and L. Miclea, "Mastering rocket league with reinforcement learning a proximal policy optimization approach," pp. 1–6, 2024.

[7] A. R. Albuainain and C. Gatzoulis, "Reinforcement learning for physics-based competitive games," pp. 1–6, 2020.